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 'rename.3p' command

$ man rename.3p

RENAME(3P) POSIX Programmer's Manual RENAME(3P)

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

rename, renameat ? rename file

SYNOPSIS

```
#include <stdio.h>
int rename(const char *old, const char *new);
#include <fcntl.h>
int renameat(int oldfd, const char *old, int newfd,
    const char *new);
```

DESCRIPTION

For rename(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The rename() function shall change the name of a file. The old argument points to the pathname of the file to be renamed. The new argument points to the new pathname of the file. If the new argument does not resolve to an existing directory entry for a file of type directory and

the new argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters after all symbolic links have been processed, rename() shall fail.

If either the old or new argument names a symbolic link, rename() shall operate on the symbolic link itself, and shall not resolve the last component of the argument. If the old argument and the new argument re? solve to either the same existing directory entry or different direc? tory entries for the same existing file, rename() shall return success? fully and perform no other action.

If the old argument points to the pathname of a file that is not a di? rectory, the new argument shall not point to the pathname of a direc? tory. If the link named by the new argument exists, it shall be removed and old renamed to new. In this case, a link named new shall remain visible to other threads throughout the renaming operation and refer either to the file referred to by new or old before the operation be? gan. Write access permission is required for both the directory con? taining old and the directory containing new.

If the old argument points to the pathname of a directory, the new ar? gument shall not point to the pathname of a file that is not a direc? tory. If the directory named by the new argument exists, it shall be removed and old renamed to new. In this case, a link named new shall exist throughout the renaming operation and shall refer either to the directory referred to by new or old before the operation began. If new names an existing directory, it shall be required to be an empty direc? tory.

If either pathname argument refers to a path whose final component is either dot or dot-dot, rename() shall fail.

If the old argument points to a pathname of a symbolic link, the sym? bolic link shall be renamed. If the new argument points to a pathname of a symbolic link, the symbolic link shall be removed.

The old pathname shall not name an ancestor directory of the new path? name. Write access permission is required for the directory containing old and the directory containing new. If the old argument points to

the pathname of a directory, write access permission may be required for the directory named by old, and, if it exists, the directory named by new.

If the link named by the new argument exists and the file's link count becomes 0 when it is removed and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before rename() re? turns, but the removal of the file contents shall be postponed until all references to the file are closed.

Upon successful completion, rename() shall mark for update the last data modification and last file status change timestamps of the parent directory of each file.

If the rename() function fails for any reason other than [EIO], any file named by new shall be unaffected.

The renameat() function shall be equivalent to the rename() function except in the case where either old or new specifies a relative path. If old is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor oldfd instead of the current working directory. If new is a relative path, the same hap? pens only relative to the directory associated with newfd. If the ac? cess mode of the open file description associated with the file de? scriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If renameat() is passed the special value AT_FDCWD in the oldfd or newfd parameter, the current working directory shall be used in the de? termination of the file for the respective path parameter.

RETURN VALUE

Upon successful completion, the rename() function shall return 0. Oth? erwise, it shall return -1, errno shall be set to indicate the error, and neither the file named by old nor the file named by new shall be

changed or created.

Upon successful completion, the renameat() function shall return 0.

Otherwise, it shall return -1 and set errno to indicate the error.

ERRORS

The rename() and renameat() functions shall fail if:

EACCES A component of either path prefix denies search permission; or one of the directories containing old or new denies write per? missions; or, write permission is required and is denied for a directory pointed to by the old or new arguments.

EBUSY The directory named by old or new is currently in use by the system or another process, and the implementation considers this an error.

[EEXIST] or [ENOTEMPTY]

The link named by new is a directory that is not an empty directory.

EINVAL The old pathname names an ancestor directory of the new pathname, or either pathname argument contains a final com? ponent that is dot or dot-dot.

EIO A physical I/O error has occurred.

EISDIR The new argument points to a directory and the old argument points to a file that is not a directory.

ELOOP A loop exists in symbolic links encountered during resolu? tion of the path argument.

EMLINK The file named by old is a directory, and the link count of the parent directory of new would exceed {LINK_MAX}.

ENAMETOOLONG

The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT The link named by old does not name an existing file, a component of the path prefix of new does not exist, or ei? ther old or new points to an empty string.

ENOSPC The directory that would contain new cannot be extended.

ENOTDIR A component of either path prefix names an existing file

that is neither a directory nor a symbolic link to a direc?
tory; or the old argument names a directory and the new ar?
gument names a non-directory file; or the old argument con?
tains at least one non-<slash> character and ends with one
or more trailing <slash> characters and the last pathname
component names an existing file that is neither a direc?
tory nor a symbolic link to a directory; or the old argu?
ment names an existing non-directory file and the new argu?
ment names a nonexistent file, contains at least one
non-<slash> character, and ends with one or more trailing
<slash> characters; or the new argument names an existing
non-directory file, contains at least one non-<slash> char?
acter, and ends with one or more trailing <slash> charac?
ters.

EPERM or EACCES

The S_ISVTX flag is set on the directory containing the file referred to by old and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1?2017, Section 4.3, Directory Protection with re? spect to old; or new refers to an existing file, the S_ISVTX flag is set on the directory containing this file, and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1?2017, Section 4.3, Directory Protection with respect to this file.

EROFS The requested operation requires writing in a directory on a read-only file system.

EXDEV The links named by new and old are on different file sys? tems and the implementation does not support links between file systems.

In addition, the renameat() function shall fail if:

EACCES The access mode of the open file description associated with oldfd or newfd is not O_SEARCH and the permissions of the direc? tory underlying oldfd or newfd, respectively, do not permit di?

rectory searches.

EBADF The old argument does not specify an absolute path and the oldfd argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching, or the new argument does not specify an absolute path and the newfd argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR

The old or new argument is not an absolute path and oldfd or newfd, respectively, is a file descriptor associated with a non-directory file.

The rename() and renameat() functions may fail if:

EBUSY The file named by the old or new arguments is a named STREAM.

ELOOP More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path argument.

ENAMETOOLONG

The length of a pathname exceeds {PATH_MAX}, or pathname resolu? tion of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ETXTBSY

The file named by new exists and is the last directory entry to a pure procedure (shared text) file that is being executed.

The following sections are informative.

EXAMPLES

Renaming a File

The following example shows how to rename a file named /home/cnd/mod1 to /home/cnd/mod2.

```
#include <stdio.h>
int status;
...
status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

APPLICATION USAGE

Some implementations mark for update the last file status change time? stamp of renamed files and some do not. Applications which make use of

the last file status change timestamp may behave differently with re?
spect to renamed files unless they are designed to allow for either be?
havior.

RATIONALE

This rename() function is equivalent for regular files to that defined by the ISO C standard. Its inclusion here expands that definition to include actions on directories and specifies behavior when the new pa? rameter names a file that already exists. That specification requires that the action of the function be atomic.

One of the reasons for introducing this function was to have a means of renaming directories while permitting implementations to prohibit the use of link() and unlink() with directories, thus constraining links to directories to those made by mkdir().

The specification that if old and new refer to the same file is in? tended to guarantee that:

```
rename("x", "x");
```

does not remove the file.

Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in rmdir() and [EBUSY] in unlink(). For a discussion of [EXDEV], see link().

The purpose of the renameat() function is to rename files in directo? ries other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to rename(), resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the renameat() function it can be guaranteed that that renamed file is lo? cated correctly and the resulting file is in the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

link(), rmdir(), symlink(), unlink()

The Base Definitions volume of POSIX.1?2017, Section 4.3, Directory

Protection, <fcntl.h>, <stdio.h>

COPYRIGHT