Rocky Enterprise Linux 9.2 Manual Pages on command 'symlink.7'

$ man symlink.7

SYMLINK(7) Linux Programmer's Manual SYMLINK(7)

NAME

symlink - symbolic link handling

DESCRIPTION

Symbolic links are files that act as pointers to other files. To un?
derstand their behavior, you must first understand how hard links work.
A hard link to a file is indistinguishable from the original file be?
cause it is a reference to the object underlying the original filename.
(To be precise: each of the hard links to a file is a reference to the
same inode number, where an inode number is an index into the inode ta?
ble, which contains metadata about all files on a filesystem. See
stat(2).) Changes to a file are independent of the name used to refer?
ence the file. Hard links may not refer to directories (to prevent the
possibility of loops within the filesystem tree, which would confuse
many programs) and may not refer to files on different filesystems (be?
cause inode numbers are not unique across filesystems).
A symbolic link is a special type of file whose contents are a string
that is the pathname of another file, the file to which the link

refers. (The contents of a symbolic link can be read using read? link(2).) In other words, a symbolic link is a pointer to another name, and not to an underlying object. For this reason, symbolic links may refer to directories and may cross filesystem boundaries.

There is no requirement that the pathname referred to by a symbolic link should exist. A symbolic link that refers to a pathname that does not exist is said to be a dangling link.

Because a symbolic link and its referenced object coexist in the filesystem name space, confusion can arise in distinguishing between the link itself and the referenced object. On historical systems, com? mands and system calls adopted their own link-following conventions in a somewhat ad-hoc fashion. Rules for a more uniform approach, as they are implemented on Linux and other systems, are outlined here. It is important that site-local applications also conform to these rules, so that the user interface can be as consistent as possible.

Magic links

There is a special class of symbolic-link-like objects known as "magic links", which can be found in certain pseudofilesystems such as proc(5) (examples include /proc/[pid]/exe and /proc/[pid]/fd/*). Unlike normal symbolic links, magic links are not resolved through pathname-expan? sion, but instead act as direct references to the kernel's own repre? sentation of a file handle. As such, these magic links allow users to access files which cannot be referenced with normal paths (such as un? linked files still referenced by a running program).

Because they can bypass ordinary mount_namespaces(7)-based restric? tions, magic links have been used as attack vectors in various ex? ploits.

Symbolic link ownership, permissions, and timestamps

The owner and group of an existing symbolic link can be changed using lchown(2). The only time that the ownership of a symbolic link matters is when the link is being removed or renamed in a directory that has the sticky bit set (see stat(2)).

The last access and last modification timestamps of a symbolic link can

be changed using utimensat(2) or lutimes(3).

On Linux, the permissions of an ordinary symbolic link are not used in any operations; the permissions are always 0777 (read, write, and exe? cute for all user categories), and can't be changed.

However, magic links do not follow this rule. They can have a non-0777 mode, though this mode is not currently used in any permission checks.

Obtaining a file descriptor that refers to a symbolic link

Using the combination of the O_PATH and O_NOFOLLOW flags to open(2) yields a file descriptor that can be passed as the dirfd argument in system calls such as fstatat(2), fchownat(2), fchmodat(2), linkat(2), and readlinkat(2), in order to operate on the symbolic link itself (rather than the file to which it refers).

By default (i.e., if the AT_SYMLINK_FOLLOW flag is not specified), if name_to_handle_at(2) is applied to a symbolic link, it yields a handle for the symbolic link (rather than the file to which it refers). One can then obtain a file descriptor for the symbolic link (rather than the file to which it refers) by specifying the O_PATH flag in a subse? quent call to open_by_handle_at(2). Again, that file descriptor can be used in the aforementioned system calls to operate on the symbolic link itself.

Handling of symbolic links by system calls and commands

Symbolic links are handled either by operating on the link itself, or by operating on the object referred to by the link. In the latter case, an application or system call is said to follow the link. Sym? bolic links may refer to other symbolic links, in which case the links are dereferenced until an object that is not a symbolic link is found, a symbolic link that refers to a file which does not exist is found, or a loop is detected. (Loop detection is done by placing an upper limit on the number of links that may be followed, and an error results if this limit is exceeded.)

There are three separate areas that need to be discussed. They are as follows:

1. Symbolic links used as filename arguments for system calls.

2. Symbolic links specified as command-line arguments to utilities that are not traversing a file tree.

3. Symbolic links encountered by utilities that are traversing a file tree (either specified on the command line or encountered as part of the file hierarchy walk).

Before describing the treatment of symbolic links by system calls and commands, we require some terminology. Given a pathname of the form a/b/c, the part preceding the final slash (i.e., a/b) is called the dirname component, and the part following the final slash (i.e., c) is called the basename component.

Treatment of symbolic links in system calls

The first area is symbolic links used as filename arguments for system calls.

The treatment of symbolic links within a pathname passed to a system call is as follows:

1. Within the dirname component of a pathname, symbolic links are al? ways followed in nearly every system call. (This is also true for commands.) The one exception is openat2(2), which provides flags that can be used to explicitly prevent following of symbolic links in the dirname component.

2. Except as noted below, all system calls follow symbolic links in the basename component of a pathname. For example, if there were a sym? bolic link slink which pointed to a file named afile, the system call open("slink" ...) would return a file descriptor referring to the file afile.

Various system calls do not follow links in the basename component of a pathname, and operate on the symbolic link itself. They are: lchown(2), lgetxattr(2), llistxattr(2), lremovexattr(2), lsetxattr(2), lstat(2), readlink(2), rename(2), rmdir(2), and unlink(2).

Certain other system calls optionally follow symbolic links in the basename component of a pathname. They are: faccessat(2), fchownat(2), fstatat(2), linkat(2), name_to_handle_at(2), open(2), openat(2), open_by_handle_at(2), and utimensat(2); see their manual pages for de?

tails. Because remove(3) is an alias for unlink(2), that library func?
tion also does not follow symbolic links. When rmdir(2) is applied to
a symbolic link, it fails with the error ENOTDIR.
link(2) warrants special discussion. POSIX.1-2001 specifies that
link(2) should dereference oldpath if it is a symbolic link. However,
Linux does not do this. (By default, Solaris is the same, but the
POSIX.1-2001 specified behavior can be obtained with suitable compiler
options.) POSIX.1-2008 changed the specification to allow either be?
havior in an implementation.

Commands not traversing a file tree

The second area is symbolic links, specified as command-line filename
arguments, to commands which are not traversing a file tree.
Except as noted below, commands follow symbolic links named as command-
line arguments. For example, if there were a symbolic link slink which
pointed to a file named afile, the command cat slink would display the
contents of the file afile.
It is important to realize that this rule includes commands which may
optionally traverse file trees; for example, the command chown file is
included in this rule, while the command chown -R file, which performs
a tree traversal, is not. (The latter is described in the third area,
below.)
If it is explicitly intended that the command operate on the symbolic
link instead of following the symbolic link?for example, it is desired
that chown slink change the ownership of the file that slink is,
whether it is a symbolic link or not?then the -h option should be used.
In the above example, chown root slink would change the ownership of
the file referred to by slink, while chown -h root slink would change
the ownership of slink itself.
There are some exceptions to this rule:

* The mv(1) and rm(1) commands do not follow symbolic links named as
arguments, but respectively attempt to rename and delete them.
(Note, if the symbolic link references a file via a relative path,
moving it to another directory may very well cause it to stop work?

ing, since the path may no longer be correct.)

* The ls(1) command is also an exception to this rule. For compatibil? ity with historic systems (when ls(1) is not doing a tree walk?that is, -R option is not specified), the ls(1) command follows symbolic links named as arguments if the -H or -L option is specified, or if the -F, -d, or -l options are not specified. (The ls(1) command is the only command where the -H and -L options affect its behavior even though it is not doing a walk of a file tree.)

* The file(1) command is also an exception to this rule. The file(1) command does not follow symbolic links named as argument by default. The file(1) command does follow symbolic links named as argument if the -L option is specified.

Commands traversing a file tree

The following commands either optionally or always traverse file trees: chgrp(1), chmod(1), chown(1), cp(1), du(1), find(1), ls(1), pax(1), rm(1), and tar(1).

It is important to realize that the following rules apply equally to symbolic links encountered during the file tree traversal and symbolic links listed as command-line arguments.

The first rule applies to symbolic links that reference files other than directories. Operations that apply to symbolic links are per? formed on the links themselves, but otherwise the links are ignored. The command rm -r slink directory will remove slink, as well as any symbolic links encountered in the tree traversal of directory, because symbolic links may be removed. In no case will rm(1) affect the file referred to by slink.

The second rule applies to symbolic links that refer to directories. Symbolic links that refer to directories are never followed by default. This is often referred to as a "physical" walk, as opposed to a "logi? cal" walk (where symbolic links that refer to directories are fol? lowed).

Certain conventions are (should be) followed as consistently as possi? ble by commands that perform file tree walks:

* A command can be made to follow any symbolic links named on the com?
mand line, regardless of the type of file they reference, by specify?
ing the -H (for "half-logical") flag. This flag is intended to make
the command-line name space look like the logical name space. (Note,
for commands that do not always do file tree traversals, the -H flag
will be ignored if the -R flag is not also specified.)
For example, the command chown -HR user slink will traverse the file
hierarchy rooted in the file pointed to by slink. Note, the -H is
not the same as the previously discussed -h flag. The -H flag causes
symbolic links specified on the command line to be dereferenced for
the purposes of both the action to be performed and the tree walk,
and it is as if the user had specified the name of the file to which
the symbolic link pointed.
* A command can be made to follow any symbolic links named on the com?
mand line, as well as any symbolic links encountered during the tra?
versal, regardless of the type of file they reference, by specifying
the -L (for "logical") flag. This flag is intended to make the en?
tire name space look like the logical name space. (Note, for com?
mands that do not always do file tree traversals, the -L flag will be
ignored if the -R flag is not also specified.)
For example, the command chown -LR user slink will change the owner
of the file referred to by slink. If slink refers to a directory,
chown will traverse the file hierarchy rooted in the directory that
it references. In addition, if any symbolic links are encountered in
any file tree that chown traverses, they will be treated in the same
fashion as slink.
* A command can be made to provide the default behavior by specifying
the -P (for "physical") flag. This flag is intended to make the en?
tire name space look like the physical name space.

For commands that do not by default do file tree traversals, the -H,
-L, and -P flags are ignored if the -R flag is not also specified. In
addition, you may specify the -H, -L, and -P options more than once;
the last one specified determines the command's behavior. This is in?

tended to permit you to alias commands to behave one way or the other, and then override that behavior on the command line.

The ls(1) and rm(1) commands have exceptions to these rules:

* The rm(1) command operates on the symbolic link, and not the file it references, and therefore never follows a symbolic link. The rm(1) command does not support the -H, -L, or -P options.

* To maintain compatibility with historic systems, the ls(1) command acts a little differently. If you do not specify the -F, -d or -l options, ls(1) will follow symbolic links specified on the command line. If the -L flag is specified, ls(1) follows all symbolic links, regardless of their type, whether specified on the command line or encountered in the tree walk.

SEE ALSO

chgrp(1), chmod(1), find(1), ln(1), ls(1), mv(1), namei(1), rm(1), lchown(2), link(2), lstat(2), readlink(2), rename(2), symlink(2), un? link(2), utimensat(2), lutimes(3), path_resolution(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.