NAME

symlink - symbolic link handling

SYMBOLIC LINK HANDLING

Symbolic links are files that act as pointers to other files. To understand their behavior, you must first understand how hard links work. A hard link to a file is indistinguishable from the original file because it is a reference to the object underlying the original file name. Changes to a file are independent of the name used to reference the file. Hard links may not refer to directories and may not reference files on different file systems. A symbolic link contains the name of the file to which it is linked, i.e., it is a pointer to another name, and not to an underlying object. For this reason, symbolic links may reference directories and may span file systems.

Because a symbolic link and its referenced object coexist in the file system name space, confusion can arise in distinguishing between the link itself and the referenced object. Historically, commands and system calls have adopted their own link following conventions in a somewhat ad-hoc fashion. Rules for more a uniform approach, as they are implemented in this system, are outlined here. It is important that local applications conform to these rules, too, so that the user interface can be as consistent as possible.

Symbolic links are handled either by operating on the link itself, or by operating on the object referenced by the link. In the latter case, an application or system call is said to "follow" the link. Symbolic links may reference other symbolic links, in which case the links are dereferenced until an object that is not a symbolic link is found, a symbolic link which references 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 file name 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).

System calls.

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

Except as noted below, all system calls follow symbolic links. For example, if there were a symbolic link "slink" which pointed to a file named "afile", the system call "open("slink" ...)" would return a file descriptor to the file "afile".

There are thirteen system calls that do not follow links, and which operate on the symbolic link itself. They are: lchflags(2), lchmod(2), lchown(2), lpathconf(2), lstat(2), lutimes(2), readlink(2), readlinkat(2), rename(2), renameat(2), rmdir(2), unlink(2), and unlinkat(2). Because remove(3) is an alias for unlink(2), it also does not follow symbolic links. When rmdir(2) or unlinkat(2) with the AT_REMOVEDIR flag is applied to a symbolic link, it fails with the error ENOTDIR.

The linkat(2) system call does not follow symbolic links unless given the AT_SYMLINK_FOLLOW flag.

The following system calls follow symbolic links unless given the AT_SYMLINK_NOFOLLOW flag: chflagsat(2), fchmodat(2), fchownat(2), fstatat(2) and utimensat(2).

The owner and group of an existing symbolic link can be changed by means of the lchown(2) system call. The flags, access permissions, owner/group and modification time of an existing symbolic link can be changed by means of the lchflags(2), lchmod(2), lchown(2), and lutimes(2) system calls, respectively. Of these, only the flags and ownership are used by the system; the access permissions are ignored.

The 4.4BSD system differs from historical 4BSD systems in that the system call chown(2) has been changed to follow symbolic links. The lchown(2) system call was added later when the limitations of the new chown(2) became apparent.

Commands not traversing a file tree.

The second area is symbolic links, specified as command line file name 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, e.g. the command "chown file" is included in this rule, while the command "chown -R file" 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, e.g., it is desired that "chown slink" change the ownership of the file that "slink" is, whether it is a symbolic link or not, the **-h** option should be used. In the above example, "chown root slink" would change the ownership of the file referenced by "slink", while "chown -h root slink" would change the ownership of "slink" itself.

There are five 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 working, since the path may no longer be correct.)

The ls(1) command is also an exception to this rule. For compatibility with historic systems (when **ls** is not doing a tree walk, i.e., the **-R** option is not specified), the **ls** 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** 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) and stat(1) commands are also exceptions to this rule. These commands do not follow symbolic links named as argument by default, but do follow symbolic links named as argument if the **-L** option is specified.

The 4.4BSD system differs from historical 4BSD systems in that the **chown** and **chgrp** commands follow symbolic links specified on the command line.

Commands traversing a file tree.

The following commands either optionally or always traverse file trees: chflags(1), chgrp(1), chmod(1), cp(1), du(1), find(1), ls(1), pax(1), rm(1), tar(1) and chown(8).

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 that are not of type directory. Operations that apply to symbolic links are performed 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** affect the file which "slink" references in any way.

The second rule applies to symbolic links that reference files of type directory. Symbolic links which reference files of type directory are never "followed" by default. This is often referred to as a "physical" walk, as opposed to a "logical" walk (where symbolic links referencing directories are followed).

As consistently as possible, you can make commands doing a file tree walk follow any symbolic links named on the command line, regardless of the type of file they reference, by specifying 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 both for the purposes of 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.

As consistently as possible, you can make commands doing a file tree walk follow any symbolic links named on the command line, as well as any symbolic links encountered during the traversal, regardless of the type of file they reference, by specifying the **-L** (for "logical") flag. This flag is intended to make the entire name space look like the logical name space. (Note, for commands 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 referenced by "slink". If "slink" references 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".

As consistently as possible, you can specify the default behavior by specifying the **-P** (for "physical") flag. This flag is intended to make the entire 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 intended 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** command operates on the symbolic link, and not the file it references, and therefore never follows a symbolic link. The **rm** command does not support the **-H**, **-L** or **-P** options.

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

SEE ALSO

chflags(1), chgrp(1), chmod(1), cp(1), du(1), find(1), ln(1), ls(1), mv(1), pax(1), rm(1), tar(1), lchflags(2), lchmod(2), lchown(2), lstat(2), lutimes(2), readlink(2), rename(2), symlink(2), unlink(2), fts(3), remove(3), chown(8)