



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

Rocky Enterprise Linux 9.2 Manual Pages on command 'linkat.2'

\$ man linkat.2

LINK(2) Linux Programmer's Manual LINK(2)

NAME

link, linkat - make a new name for a file

SYNOPSIS

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);

#include <fcntl.h>           /* Definition of AT_* constants */

#include <unistd.h>

int linkat(int olddirfd, const char *oldpath,
           int newdirfd, const char *newpath, int flags);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

linkat():

Since glibc 2.10:

```
  _POSIX_C_SOURCE >= 200809L
```

Before glibc 2.10:

```
  _ATFILE_SOURCE
```

DESCRIPTION

link() creates a new link (also known as a hard link) to an existing

file.

If `newpath` exists, it will not be overwritten.

This new name may be used exactly as the old one for any operation; both names refer to the same file (and so have the same permissions and ownership) and it is impossible to tell which name was the "original".

`linkat()`

The `linkat()` system call operates in exactly the same way as `link()`, except for the differences described here.

If the `pathname` given in `oldpath` is relative, then it is interpreted relative to the directory referred to by the file descriptor `olddirfd` (rather than relative to the current working directory of the calling process, as is done by `link()` for a relative `pathname`).

If `oldpath` is relative and `olddirfd` is the special value `AT_FDCWD`, then `oldpath` is interpreted relative to the current working directory of the calling process (like `link()`).

If `oldpath` is absolute, then `olddirfd` is ignored.

The interpretation of `newpath` is as for `oldpath`, except that a relative `pathname` is interpreted relative to the directory referred to by the file descriptor `newdirfd`.

The following values can be bitwise ORed in flags:

`AT_EMPTY_PATH` (since Linux 2.6.39)

If `oldpath` is an empty string, create a link to the file referenced by `olddirfd` (which may have been obtained using the `open(2)` `O_PATH` flag). In this case, `olddirfd` can refer to any type of file except a directory. This will generally not work if the file has a link count of zero (files created with `O_TMPFILE` and without `O_EXCL` are an exception). The caller must have the `CAP_DAC_READ_SEARCH` capability in order to use this flag. This flag is Linux-specific; define `_GNU_SOURCE` to obtain its definition.

`AT_SYMLINK_FOLLOW` (since Linux 2.6.18)

By default, `linkat()`, does not dereference `oldpath` if it is a symbolic link (like `link()`). The flag `AT_SYMLINK_FOLLOW` can be

specified in flags to cause oldpath to be dereferenced if it is a symbolic link. If procfs is mounted, this can be used as an alternative to AT_EMPTY_PATH, like this:

```
linkat(AT_FDCWD, "/proc/self/fd/<fd>", newdirfd,  
       newname, AT_SYMLINK_FOLLOW);
```

Before kernel 2.6.18, the flags argument was unused, and had to be specified as 0.

See `openat(2)` for an explanation of the need for `linkat()`.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and `errno` is set appropriately.

ERRORS

EACCES Write access to the directory containing newpath is denied, or search permission is denied for one of the directories in the path prefix of oldpath or newpath. (See also `path_resolution(7)`.)

EDQUOT The user's quota of disk blocks on the filesystem has been exhausted.

EEXIST newpath already exists.

EFAULT oldpath or newpath points outside your accessible address space.

EIO An I/O error occurred.

ELOOP Too many symbolic links were encountered in resolving oldpath or newpath.

EMLINK The file referred to by oldpath already has the maximum number of links to it. For example, on an `ext4(5)` filesystem that does not employ the `dir_index` feature, the limit on the number of hard links to a file is 65,000; on `btrfs(5)`, the limit is 65,535 links.

ENAMETOOLONG

oldpath or newpath was too long.

ENOENT A directory component in oldpath or newpath does not exist or is a dangling symbolic link.

ENOMEM Insufficient kernel memory was available.

ENOSPC The device containing the file has no room for the new directory entry.

ENOTDIR

A component used as a directory in oldpath or newpath is not, in fact, a directory.

EPERM oldpath is a directory.

EPERM The filesystem containing oldpath and newpath does not support the creation of hard links.

EPERM (since Linux 3.6)

The caller does not have permission to create a hard link to this file (see the description of `/proc/sys/fs/pro? tected_hardlinks` in `proc(5)`).

EPERM oldpath is marked immutable or append-only. (See `ioctl_iflags(2)`.)

EROFS The file is on a read-only filesystem.

EXDEV oldpath and newpath are not on the same mounted filesystem.

(Linux permits a filesystem to be mounted at multiple points, but `link()` does not work across different mount points, even if the same filesystem is mounted on both.)

The following additional errors can occur for `linkat()`:

EBADF `olddirfd` or `newdirfd` is not a valid file descriptor.

EINVAL An invalid flag value was specified in `flags`.

ENOENT `AT_EMPTY_PATH` was specified in `flags`, but the caller did not have the `CAP_DAC_READ_SEARCH` capability.

ENOENT An attempt was made to link to the `/proc/self/fd/NN` file corresponding to a file descriptor created with `open(path, O_TMPFILE | O_EXCL, mode)`; See `open(2)`.

ENOENT An attempt was made to link to a `/proc/self/fd/NN` file corresponding to a file that has been deleted.

ENOENT oldpath is a relative pathname and `olddirfd` refers to a directory that has been deleted, or newpath is a relative pathname and `newdirfd` refers to a directory that has been deleted.

ENOTDIR

oldpath is relative and olddirfd is a file descriptor referring to a file other than a directory; or similar for newpath and newdirfd

EPERM AT_EMPTY_PATH was specified in flags, oldpath is an empty string, and olddirfd refers to a directory.

VERSIONS

linkat() was added to Linux in kernel 2.6.16; library support was added to glibc in version 2.4.

CONFORMING TO

link(): SVr4, 4.3BSD, POSIX.1-2001 (but see NOTES), POSIX.1-2008.

linkat(): POSIX.1-2008.

NOTES

Hard links, as created by link(), cannot span filesystems. Use sym? link(2) if this is required.

POSIX.1-2001 says that link() should dereference oldpath if it is a symbolic link. However, since kernel 2.0, Linux does not do so: if oldpath is a symbolic link, then newpath is created as a (hard) link to the same symbolic link file (i.e., newpath becomes a symbolic link to the same file that oldpath refers to). Some other implementations behave in the same manner as Linux. POSIX.1-2008 changes the specification of link(), making it implementation-dependent whether or not oldpath is dereferenced if it is a symbolic link. For precise control over the treatment of symbolic links when creating a link, use linkat().

Glibc notes

On older kernels where linkat() is unavailable, the glibc wrapper function falls back to the use of link(), unless the AT_SYMLINK_FOLLOW is specified. When oldpath and newpath are relative pathnames, glibc constructs pathnames based on the symbolic links in /proc/self/fd that correspond to the olddirfd and newdirfd arguments.

BUGS

On NFS filesystems, the return code may be wrong in case the NFS server

performs the link creation and dies before it can say so. Use `stat(2)` to find out if the link got created.

SEE ALSO

`ln(1)`, `open(2)`, `rename(2)`, `stat(2)`, `symlink(2)`, `unlink(2)`, `path_resolu?`
`tion(7)`, `symlink(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/>.

Linux

2020-12-21

LINK(2)