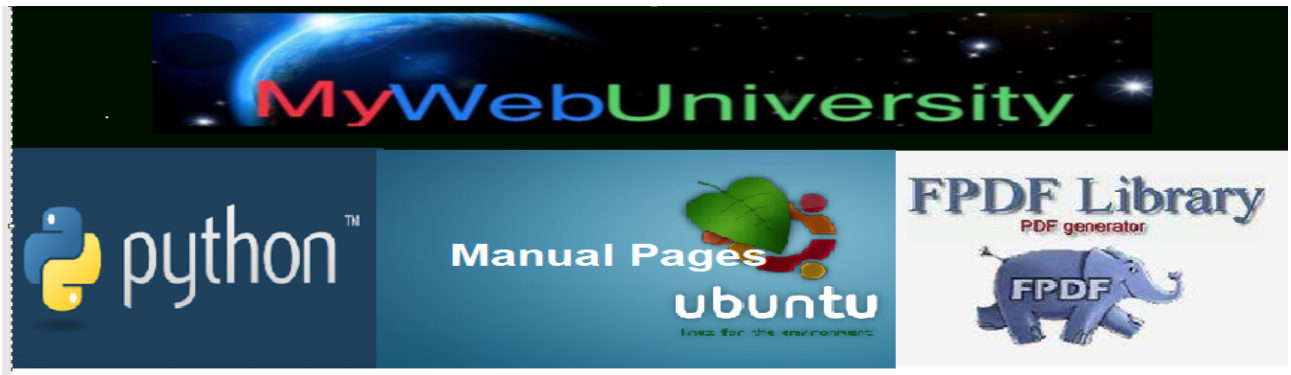

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'unlinkat.2'

$ man unlinkat.2

unlink(2) System Calls Manual unlink(2)

NAME

unlink, unlinkat - delete a name and possibly the file it refers to

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <unistd.h>

int unlink(const char *path);

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

int unlinkat(int dirfd, const char *path, int flags);
```

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

unlinkat():

Since glibc 2.10:

_POSIX_C_SOURCE >= 200809L

Before glibc 2.10:

_ATFILE_SOURCE

DESCRIPTION

unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed.

If the name referred to a symbolic link, the link is removed.

If the name referred to a socket, FIFO, or device, the name for it is removed but processes which have the object open may continue to use it.

unlinkat()

The unlinkat() system call operates in exactly the same way as either unlink() or rmdir(2) (depending on whether or not flags includes the AT_REMOVEDIR flag) except for the differences described here.

If path is relative, then it is interpreted relative to the directory referred to by the file descriptor dirfd (rather than relative to the current working directory of the calling process, as is done by unlink() and rmdir(2) for a relative pathname).

If path is relative and dirfd is the special value AT_FDCWD, then path is interpreted relative to the current working directory of the calling process (like unlink() and rmdir(2)).

If path is absolute, then dirfd is ignored.

flags is a bit mask that can either be specified as 0, or by ORing together flag values that control the operation of unlinkat(). Currently, only one such flag is defined:

AT_REMOVEDIR

By default, unlinkat() performs the equivalent of unlink() on path. If the AT_REMOVEDIR flag is specified, it performs the equivalent of rmdir(2) on path.

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

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set to indicate the error.

ERRORS

EACCES Write access to the directory containing path is not allowed for the process's effective UID, or one of the directories in path did not allow search permission. (See also path_resolution(7).)

EBUSY path cannot be unlinked because it is being used by the system or another process; for example, it is a mount point or the NFS client software created it to represent an active but otherwise nameless inode ("NFS silly renamed").

EFAULT path points outside your accessible address space.

EIO An I/O error occurred.

EISDIR path refers to a directory. (This is the non-POSIX value returned since Linux 2.1.132.)

ELOOP Too many symbolic links were encountered in translating path.

ENAMETOOLONG

path was too long.

ENOENT A component in path does not exist or is a dangling symbolic link, or path is empty.

ENOMEM Insufficient kernel memory was available.

ENOTDIR

A component used as a directory in path is not, in fact, a directory.

EPERM The system does not allow unlinking of directories, or unlinking of directories re?

quires privileges that the calling process doesn't have. (This is the POSIX pre?

scribed error return; as noted above, Linux returns EISDIR for this case.)

EPERM (Linux only)

The filesystem does not allow unlinking of files.

EPERM or EACCES

The directory containing path has the sticky bit (S_ISVTX) set and the process's ef?

fective UID is neither the UID of the file to be deleted nor that of the directory

containing it, and the process is not privileged (Linux: does not have the CAP_FOWNER

capability).

EPERM The file to be unlinked is marked immutable or append-only. (See FS_IOC_SET?

FLAGS(2const).)

EROFS path refers to a file on a read-only filesystem.

The same errors that occur for unlink() and rmdir(2) can also occur for unlinkat(). The

following additional errors can occur for unlinkat():

EBADF path is relative but dirfd is neither AT_FDCWD nor a valid file descriptor.

EINVAL An invalid flag value was specified in flags.

EISDIR path refers to a directory, and AT_REMOVEDIR was not specified in flags.

ENOTDIR

path is relative and dirfd is a file descriptor referring to a file other than a di?

rectory.

STANDARDS

POSIX.1-2024.

HISTORY

unlink()

SVr4, 4.3BSD, POSIX.1-2001.

unlinkat()

POSIX.1-2008. Linux 2.6.16, glibc 2.4.

glibc

On older kernels where unlinkat() is unavailable, the glibc wrapper function falls back to the use of unlink() or rmdir(2). When path is relative, glibc constructs a pathname based on the symbolic link in /proc/self/fd that corresponds to the dirfd argument.

BUGS

Infelicities in the protocol underlying NFS can cause the unexpected disappearance of files which are still being used.

SEE ALSO

rm(1), unlink(1), chmod(2), link(2), mknod(2), open(2), rename(2), rmdir(2), mkfifo(3), re? move(3), path_resolution(7), symlink(7)