

**NAME**

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

**SYNOPSIS**

```
#include <unistd.h>
```

```
int unlink(const char *pathname);
```

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

```
#include <unistd.h>
```

```
int unlinkat(int dirfd, const char *pathname, 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 the *pathname* given in *pathname* 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 the *pathname* given in *pathname* is relative and *dirfd* is the special value `AT_FDCWD`, then *pathname* is interpreted relative to the current working directory of the calling process (like `unlink()` and `rmdir(2)`).

If the *pathname* given in *pathname* 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 *pathname*. If the `AT_REMOVEDIR` flag is specified, then performs the equivalent of `rmdir(2)` on *pathname*.

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 appropriately.

**ERRORS****EACCES**

Write access to the directory containing *pathname* is not allowed for the process's effective UID, or one of the directories in *pathname* did not allow search permission. (See also [path\\_resolution\(7\)](#).)

**EBUSY**

The file *pathname* 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**

*pathname* points outside your accessible address space.

**EIO** An I/O error occurred.

**EISDIR**

*pathname* refers to a directory. (This is the non-POSIX value returned by Linux since 2.1.132.)

**ELOOP**

Too many symbolic links were encountered in translating *pathname*.

**ENAMETOOLONG**

*pathname* was too long.

**ENOENT**

A component in *pathname* does not exist or is a dangling symbolic link, or *pathname* is empty.

**ENOMEM**

Insufficient kernel memory was available.

**ENOTDIR**

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

**EPERM**

The system does not allow unlinking of directories, or unlinking of directories requires privileges that the calling process doesn't have. (This is the POSIX prescribed 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 *pathname* has the sticky bit (**S\_ISVTX**) set and the process's effective 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 **ioctl\_iflags(2)**.)

**EROFS**

*pathname* 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**

*dirfd* is not a valid file descriptor.

**EINVAL**

An invalid flag value was specified in *flags*.

**EISDIR**

*pathname* refers to a directory, and **AT\_REMOVEDIR** was not specified in *flags*.

**ENOTDIR**

*pathname* is relative and *dirfd* is a file descriptor referring to a file other than a directory.

**VERSIONS**

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

**CONFORMING TO**

**unlink()**: SVr4, 4.3BSD, POSIX.1-2001, POSIX.1-2008.

**unlinkat()**: POSIX.1-2008.

**NOTES****Glibc notes**

On older kernels where **unlinkat()** is unavailable, the glibc wrapper function falls back to the use of **unlink()** or **rmdir(2)**. When *pathname* is a relative pathname, 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)**, **remove(3)**, **path\_resolution(7)**, **symlink(7)**

**COLOPHON**

This page is part of release 5.05 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/>.