



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

\$ man fchmodat.2

chmod(2) System Calls Manual chmod(2)

NAME

chmod, fchmod, fchmodat - change permissions of a file

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/stat.h>

int chmod(const char *path, mode_t mode);

int fchmod(int fd, mode_t mode);

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

#include <sys/stat.h>

int fchmodat(int dirfd, const char *path, mode_t mode, int flags);
```

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

fchmod():

Since glibc 2.24:

 _POSIX_C_SOURCE >= 199309L

glibc 2.19 to glibc 2.23

 _POSIX_C_SOURCE

glibc 2.16 to glibc 2.19:

 _BSD_SOURCE || _POSIX_C_SOURCE

glibc 2.12 to glibc 2.16:

 _BSD_SOURCE || _XOPEN_SOURCE >= 500

 || _POSIX_C_SOURCE >= 200809L

glibc 2.11 and earlier:

```
_BSD_SOURCE || _XOPEN_SOURCE >= 500
```

fchmodat():

Since glibc 2.10:

```
_POSIX_C_SOURCE >= 200809L
```

Before glibc 2.10:

```
_ATFILE_SOURCE
```

DESCRIPTION

The `chmod()` and `fchmod()` system calls change a file's mode bits. (The file mode consists of the file permission bits plus the set-user-ID, set-group-ID, and sticky bits.) These system calls differ only in how the file is specified:

? `chmod()` changes the mode of the file specified whose pathname is given in `path`, which is dereferenced if it is a symbolic link.

? `fchmod()` changes the mode of the file referred to by the open file descriptor `fd`.

The new file mode is specified in `mode`, which is a bit mask created by ORing together zero or more of the following:

`S_ISUID` (04000) set-user-ID (set process effective user ID on `execve(2)`)

`S_ISGID` (02000) set-group-ID (set process effective group ID on `execve(2)`; mandatory locking, as described in `fcntl(2)`; take a new file's group from parent directory, as described in `chown(2)` and `mkdir(2)`)

`S_ISVTX` (01000) sticky bit (restricted deletion flag, as described in `unlink(2)`)

`S_IRUSR` (00400) read by owner

`S_IWUSR` (00200) write by owner

`S_IXUSR` (00100) execute/search by owner ("search" applies for directories, and means that entries within the directory can be accessed)

`S_IRGRP` (00040) read by group

`S_IWGRP` (00020) write by group

`S_IXGRP` (00010) execute/search by group

`S_IROTH` (00004) read by others

`S_IWOTH` (00002) write by others

`S_IXOTH` (00001) execute/search by others

The effective UID of the calling process must match the owner of the file, or the process must be privileged (Linux: it must have the `CAP_FOWNER` capability).

If the calling process is not privileged (Linux: does not have the `CAP_FSETID` capability), and the group of the file does not match the effective group ID of the process or one of its supplementary group IDs, the `S_ISGID` bit will be turned off, but this will not cause an error to be returned.

As a security measure, depending on the filesystem, the set-user-ID and set-group-ID execution bits may be turned off if a file is written. (On Linux, this occurs if the writing process does not have the `CAP_FSETID` capability.) On some filesystems, only the superuser can set the sticky bit, which may have a special meaning. For the sticky bit, and for set-user-ID and set-group-ID bits on directories, see `inode(7)`.

On NFS filesystems, restricting the permissions will immediately influence already open files, because the access control is done on the server, but open files are maintained by the client. Widening the permissions may be delayed for other clients if attribute caching is enabled on them.

`fchmodat()`

The `fchmodat()` system call operates in exactly the same way as `chmod()`, 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 `chmod()` 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 `chmod()`).

If `path` is absolute, then `dirfd` is ignored.

`flags` can either be 0, or include the following flags:

`AT_EMPTY_PATH` (since Linux 6.6)

If `path` is an empty string, operate on the file referred to by `dirfd` (which may have been obtained using the `open(2)` `O_PATH` flag). In this case, `dirfd` can refer to any type of file, not just a directory. If `dirfd` is `AT_FDCWD`, the call operates on the current working directory. This flag is Linux-specific; define `_GNU_SOURCE` to obtain its definition.

`AT_SYMLINK_NOFOLLOW`

If `path` is a symbolic link, do not dereference it: instead operate on the link itself.

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

RETURN VALUE

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

ERRORS

Depending on the filesystem, errors other than those listed below can be returned.

The more general errors for `chmod()` are listed below:

`EACCES` Search permission is denied on a component of the path prefix. (See also `path_resolution(7)`.)

`EBADF` (`fchmod()`) The file descriptor `fd` is not valid.

`EBADF` (`fchmodat()`) `path` is relative but `dirfd` is neither `AT_FDCWD` nor a valid file descriptor.

`EFAULT` `path` points outside your accessible address space.

`EINVAL` (`fchmodat()`) Invalid flag specified in `flags`.

`EIO` An I/O error occurred.

`ELOOP` Too many symbolic links were encountered in resolving `path`.

`ENAMETOOLONG`

`path` is too long.

`ENOENT` The file does not exist.

`ENOMEM` Insufficient kernel memory was available.

`ENOTDIR`

A component of the path prefix is not a directory.

`ENOTDIR`

(`fchmodat()`) `path` is relative and `dirfd` is a file descriptor referring to a file other than a directory.

`ENOTSUP`

(`fchmodat()`) `flags` specified `AT_SYMLINK_NOFOLLOW`, which is not supported.

`EPERM` The effective UID does not match the owner of the file, and the process is not privileged (Linux: it does not have the `CAP_FOWNER` capability).

`EPERM` The file is marked immutable or append-only. (See `FS_IOC_SETFLAGS(2)`.)

`EROFS` The named file resides on a read-only filesystem.

VERSIONS

C library/kernel differences

The GNU C library `fchmodat()` wrapper function implements the POSIX-specified interface described

scribed in this page. This interface differs from the underlying Linux system call, which does not have a flags argument.

glibc notes

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

STANDARDS

POSIX.1-2024.

HISTORY

`chmod()`

SVr4, POSIX.1-1988, 4.4BSD.

`fchmod()`

SVr4, 4.4BSD, SUSv1, POSIX.1-1996.

`fchmodat()`

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

`AT_SYMLINK_NOFOLLOW`

POSIX.1-2008, glibc 2.32, Linux 6.5.

SEE ALSO

`chmod(1)`, `chown(2)`, `execve(2)`, `open(2)`, `stat(2)`, `inode(7)`, `path_resolution(7)`, `symlink(7)`

Linux man-pages 6.17

2026-02-08

`chmod(2)`