



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

\$ man fanotify_mark.2

fanotify_mark(2) System Calls Manual fanotify_mark(2)

NAME

fanotify_mark - add, remove, or modify an fanotify mark on a filesystem object

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

int fanotify_mark(int fanotify_fd, unsigned int flags,
                  uint64_t mask, int dirfd,
                  const char *_Nullable path);
```

DESCRIPTION

For an overview of the fanotify API, see fanotify(7).

fanotify_mark() adds, removes, or modifies an fanotify mark on a filesystem object. The caller must have read permission on the filesystem object that is to be marked.

The fanotify_fd argument is a file descriptor returned by fanotify_init(2).

flags is a bit mask describing the modification to perform. It must include exactly one of the following values:

FAN_MARK_ADD

The events in mask will be added to the mark mask (or to the ignore mask). mask must be nonempty or the error EINVAL will occur.

FAN_MARK_REMOVE

The events in argument mask will be removed from the mark mask (or from the ignore mask). mask must be nonempty or the error EINVAL will occur.

FAN_MARK_FLUSH

Remove either all marks for filesystems, all marks for mounts, or all marks for directories and files from the fanotify group. If flags contains FAN_MARK_MOUNT, all marks for mounts are removed from the group. If flags contains FAN_MARK_FILESYSTEM, all marks for filesystems are removed from the group. Otherwise, all marks for directories and files are removed. No flag other than, and at most one of, the flags FAN_MARK_MNTNS, FAN_MARK_MOUNT, or FAN_MARK_FILESYSTEM can be used in conjunction with FAN_MARK_FLUSH. mask is ignored.

If none of the values above is specified, or more than one is specified, the call fails with the error EINVAL.

In addition, zero or more of the following values may be ORed into flags:

FAN_MARK_DONT_FOLLOW

If path is a symbolic link, mark the link itself, rather than the file to which it refers. (By default, fanotify_mark() dereferences path if it is a symbolic link.)

FAN_MARK_ONLYDIR

If the filesystem object to be marked is not a directory, the error ENOTDIR shall be raised.

FAN_MARK_MNTNS (since Linux 6.14)

Mark the mount namespace specified by path. If path does not represent a mount namespace (e.g. /proc/pid/ns/mnt), the call fails with the error EINVAL. A fanotify group that was initialized with flag FAN_REPORT_MNT is required.

FAN_MARK_MOUNT

Mark the mount specified by path. If path is not itself a mount point, the mount containing path will be marked. All directories, subdirectories, and the contained files of the mount will be monitored. The events which require that filesystem objects are identified by file handles, such as FAN_CREATE, FAN_ATTRIB, FAN_MOVE, and FAN_DELETE_SELF, cannot be provided as a mask when flags contains FAN_MARK_MOUNT. Attempting to do so will result in the error EINVAL being returned. Use of this flag requires the CAP_SYS_ADMIN capability.

FAN_MARK_FILESYSTEM (since Linux 4.20)

Mark the filesystem specified by path. The filesystem containing path will be marked. All the contained files and directories of the filesystem from any mount point will be monitored. Use of this flag requires the CAP_SYS_ADMIN capability.

FAN_MARK_IGNORED_MASK

The events in mask shall be added to or removed from the ignore mask. Note that the flags `FAN_ONDIR`, and `FAN_EVENT_ON_CHILD` have no effect when provided with this flag. The effect of setting the flags `FAN_ONDIR`, and `FAN_EVENT_ON_CHILD` in the mark mask on the events that are set in the ignore mask is undefined and depends on the Linux kernel version. Specifically, prior to Linux 5.9, setting a mark mask on a file and a mark with ignore mask on its parent directory would not result in ignoring events on the file, regardless of the `FAN_EVENT_ON_CHILD` flag in the parent directory's mark mask. When the ignore mask is updated with the `FAN_MARK_IGNORED_MASK` flag on a mark that was previously updated with the `FAN_MARK_IGNORE` flag, the update fails with `EEXIST` error.

FAN_MARK_IGNORE (since Linux 6.0, 5.15.154, and 5.10.220)

This flag has a similar effect as setting the `FAN_MARK_IGNORED_MASK` flag. The events in mask shall be added to or removed from the ignore mask. Unlike the `FAN_MARK_IGNORED_MASK` flag, this flag also has the effect that the `FAN_ONDIR`, and `FAN_EVENT_ON_CHILD` flags take effect on the ignore mask. Specifically, unless the `FAN_ONDIR` flag is set with `FAN_MARK_IGNORE`, events on directories will not be ignored. If the flag `FAN_EVENT_ON_CHILD` is set with `FAN_MARK_IGNORE`, events on children will be ignored. For example, a mark on a directory with combination of a mask with `FAN_CREATE` event and `FAN_ONDIR` flag and an ignore mask with `FAN_CREATE` event and without `FAN_ONDIR` flag, will result in getting only the events for creation of sub-directories. When using the `FAN_MARK_IGNORE` flag to add to an ignore mask of a mount, filesystem, or directory inode mark, the `FAN_MARK_IGNORED_SURV_MODIFY` flag must be specified. Failure to do so will result with `EINVAL` or `EISDIR` error.

FAN_MARK_IGNORED_SURV_MODIFY

The ignore mask shall survive modify events. If this flag is not set, the ignore mask is cleared when a modify event occurs on the marked object. Omitting this flag is typically used to suppress events (e.g., `FAN_OPEN`) for a specific file, until that specific file's content has been modified. It is far less useful to suppress events on an entire filesystem, or mount, or on all files inside a directory, until some file's content has been modified. For this reason, the `FAN_MARK_IGNORE` flag requires the `FAN_MARK_IGNORED_SURV_MODIFY` flag on a mount, filesystem, or directory inode mark. This flag cannot be removed from a mark once set. When the ignore mask is up?

dated without this flag on a mark that was previously updated with the `FAN_MARK_IG?`

`NORE` and `FAN_MARK_IGNORED_SURV_MODIFY` flags, the update fails with `EEXIST` error.

`FAN_MARK_IGNORE_SURV`

This is a synonym for `(FAN_MARK_IGNORE|FAN_MARK_IGNORED_SURV_MODIFY)`.

`FAN_MARK_EVICTABLE` (since Linux 5.19, 5.15.154, and 5.10.220)

When an inode mark is created with this flag, the inode object will not be pinned to the inode cache, therefore, allowing the inode object to be evicted from the inode cache when the memory pressure on the system is high. The eviction of the inode object results in the evictable mark also being lost. When the mask of an evictable inode mark is updated without using the `FAN_MARK_EVICTABLE` flag, the marked inode is pinned to inode cache and the mark is no longer evictable. When the mask of a non-evictable inode mark is updated with the `FAN_MARK_EVICTABLE` flag, the inode mark remains non-evictable and the update fails with `EEXIST` error. Mounts and filesystems are not evictable objects, therefore, an attempt to create a mount mark or a filesystem mark with the `FAN_MARK_EVICTABLE` flag, will result in the error `EINVAL`. For example, inode marks can be used in combination with mount marks to reduce the amount of events from noninteresting paths. The event listener reads events, checks if the path reported in the event is of interest, and if it is not, the listener sets a mark with an ignore mask on the directory. Evictable inode marks allow using this method for a large number of directories without the concern of pinning all inodes and exhausting the system's memory.

mask defines which events shall be listened for (or which shall be ignored). It is a bit mask composed of the following values:

`FAN_ACCESS`

Create an event when a file or directory (but see `BUGS`) is accessed (read).

`FAN_MODIFY`

Create an event when a file is modified (write).

`FAN_CLOSE_WRITE`

Create an event when a writable file is closed.

`FAN_CLOSE_NOWRITE`

Create an event when a read-only file or directory is closed.

`FAN_OPEN`

Create an event when a file or directory is opened.

FAN_OPEN_EXEC (since Linux 5.0)

Create an event when a file is opened with the intent to be executed. See NOTES for additional details.

FAN_ATTRIB (since Linux 5.1)

Create an event when the metadata for a file or directory has changed. An fanotify group that identifies filesystem objects by file handles is required.

FAN_CREATE (since Linux 5.1)

Create an event when a file or directory has been created in a marked parent directory. An fanotify group that identifies filesystem objects by file handles is required.

FAN_DELETE (since Linux 5.1)

Create an event when a file or directory has been deleted in a marked parent directory. An fanotify group that identifies filesystem objects by file handles is required.

FAN_DELETE_SELF (since Linux 5.1)

Create an event when a marked file or directory itself is deleted. An fanotify group that identifies filesystem objects by file handles is required.

FAN_MOVED_FROM (since Linux 5.1)

Create an event when a file or directory has been moved from a marked parent directory. An fanotify group that identifies filesystem objects by file handles is required.

FAN_MOVED_TO (since Linux 5.1)

Create an event when a file or directory has been moved to a marked parent directory. An fanotify group that identifies filesystem objects by file handles is required.

FAN_RENAME (since Linux 5.17, 5.15.154, and 5.10.220)

This event contains the same information provided by events FAN_MOVED_FROM and FAN_MOVED_TO, however is represented by a single event with up to two information records. An fanotify group that identifies filesystem objects by file handles is required. If the filesystem object to be marked is not a directory, the error ENOTDIR shall be raised.

FAN_MOVE_SELF (since Linux 5.1)

Create an event when a marked file or directory itself has been moved. An fanotify group that identifies filesystem objects by file handles is required.

FAN_MNT_ATTACH

FAN_MNT_DETACH (both since Linux 6.14)

Create an event when a mount was attached to or detached from a marked mount namespace, respectively. An attempt to set this flag on an inode, mount, or filesystem mark will result in the error EINVAL. A fanotify group that was initialized with flag FAN_REPORT_MNT and the mark flag FAN_MARK_MNTNS are required. An additional information record of type FAN_EVENT_INFO_TYPE_MNT is returned with the event. See fanotify(7) for additional details.

FAN_FS_ERROR (since Linux 5.16, 5.15.154, and 5.10.220)

Create an event when a filesystem error leading to inconsistent filesystem metadata is detected. An additional information record of type FAN_EVENT_INFO_TYPE_ERROR is returned for each event in the read buffer. A fanotify group that identifies filesystem objects by file handles is required. Events of such type are dependent on support from the underlying filesystem. At the time of writing, only the ext4 filesystem reports FAN_FS_ERROR events. See fanotify(7) for additional details.

FAN_OPEN_PERM

Create an event when a permission to open a file or directory is requested. A fanotify file descriptor created with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required.

FAN_OPEN_EXEC_PERM (since Linux 5.0)

Create an event when a permission to open a file for execution is requested. A fanotify file descriptor created with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required. See NOTES for additional details.

FAN_ACCESS_PERM

Create an event when a permission to read a file or directory is requested. A fanotify file descriptor created with FAN_CLASS_PRE_CONTENT or FAN_CLASS_CONTENT is required.

FAN_PRE_ACCESS (since Linux 6.14)

Create an event before read or write access to a file range, that provides an opportunity for the event listener to modify the content of the file before access to the content in the specified range. An additional information record of type FAN_EVENT_INFO_TYPE_RANGE is returned for each event in the read buffer. A fanotify file descriptor created with FAN_CLASS_PRE_CONTENT is required.

FAN_ONDIR

Create events for directories?for example, when `opendir(3)`, `readdir(3)` (but see BUGS), and `closedir(3)` are called. Without this flag, events are created only for files. In the context of directory entry events, such as `FAN_CREATE`, `FAN_DELETE`, `FAN_MOVED_FROM`, and `FAN_MOVED_TO`, specifying the flag `FAN_ONDIR` is required in order to create events when subdirectory entries are modified (i.e., `mkdir(2)`/`rmdir(2)`).

FAN_EVENT_ON_CHILD

Events for the immediate children of marked directories shall be created. The flag has no effect when marking mounts and filesystems. Note that events are not generated for children of the subdirectories of marked directories. More specifically, the directory entry modification events `FAN_CREATE`, `FAN_DELETE`, `FAN_MOVED_FROM`, and `FAN_MOVED_TO` are not generated for any entry modifications performed inside subdirectories of marked directories. Note that the events `FAN_DELETE_SELF` and `FAN_MOVE_SELF` are not generated for children of marked directories. To monitor complete directory trees it is necessary to mark the relevant mount or filesystem.

The following composed values are defined:

FAN_CLOSE

A file is closed (`FAN_CLOSE_WRITE|FAN_CLOSE_NOWRITE`).

FAN_MOVE

A file or directory has been moved (`FAN_MOVED_FROM|FAN_MOVED_TO`).

The filesystem object to be marked is determined by the file descriptor `dirfd` and the path name specified in `path`:

- ? If `path` is `NULL`, `dirfd` defines the filesystem object to be marked.
- ? If `path` is `NULL`, and `dirfd` takes the special value `AT_FDCWD`, the current working directory is to be marked.
- ? If `path` is absolute, it defines the filesystem object to be marked, and `dirfd` is ignored.
- ? If `path` is relative, and `dirfd` does not have the value `AT_FDCWD`, then the filesystem object to be marked is determined by interpreting `path` relative to the directory referred to by `dirfd`.
- ? If `path` is relative, and `dirfd` has the value `AT_FDCWD`, then the filesystem object to be marked is determined by interpreting `path` relative to the current working directory.

(See `openat(2)` for an explanation of why the `dirfd` argument is useful.)

RETURN VALUE

On success, fanotify_mark() returns 0. On error, -1 is returned, and errno is set to indi-

cate the error.

ERRORS

EBADF An invalid file descriptor was passed in fanotify_fd.

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

EEXIST The filesystem object indicated by dirfd and path has a mark that was updated without the FAN_MARK_EVICTABLE flag, and the user attempted to update the mark with FAN_MARK_EVICTABLE flag.

EEXIST The filesystem object indicated by dirfd and path has a mark that was updated with the FAN_MARK_IGNORE flag, and the user attempted to update the mark with FAN_MARK_IGNORED_MASK flag.

EEXIST The filesystem object indicated by dirfd and path has a mark that was updated with the FAN_MARK_IGNORE and FAN_MARK_IGNORED_SURV_MODIFY flags, and the user attempted to update the mark only with FAN_MARK_IGNORE flag.

EINVAL An invalid value was passed in flags or mask, or fanotify_fd was not an fanotify file descriptor.

EINVAL The fanotify file descriptor was opened with FAN_CLASS_NOTIF or the fanotify group identifies filesystem objects by file handles and mask contains a flag for permission events (FAN_OPEN_PERM or FAN_ACCESS_PERM).

EINVAL The group was initialized without FAN_REPORT_FID but one or more event types specified in the mask require it.

EINVAL flags contains FAN_MARK_IGNORE, and either FAN_MARK_MOUNT or FAN_MARK_FILESYSTEM, but does not contain FAN_MARK_IGNORED_SURV_MODIFY.

EISDIR flags contains FAN_MARK_IGNORE, but does not contain FAN_MARK_IGNORED_SURV_MODIFY, and dirfd and path specify a directory.

ENODEV The filesystem object indicated by dirfd and path is associated with a filesystem that reports zero fsid (e.g., fuse(4)). This error can be returned only with an fanotify group that identifies filesystem objects by file handles. Since Linux 6.8, this error can be returned when trying to add a mount or filesystem mark.

ENOENT The filesystem object indicated by dirfd and path does not exist. This error also occurs when trying to remove a mark from an object that is not marked.

ENOMEM The necessary memory could not be allocated.

ENOSPC The number of marks for this user exceeds the limit and the FAN_UNLIMITED_MARKS flag

was not specified when the fanotify file descriptor was created with fan?

otify_init(2). See fanotify(7) for details about this limit.

ENOSYS This kernel does not implement fanotify_mark(). The fanotify API is available only if the kernel was configured with CONFIG_FANOTIFY.

ENOTDIR

flags contains FAN_MARK_ONLYDIR, and dirfd and path do not specify a directory.

ENOTDIR

mask contains FAN_RENAME, and dirfd and path do not specify a directory.

ENOTDIR

flags contains FAN_MARK_IGNORE, or the fanotify group was initialized with flag FAN_REPORT_TARGET_FID, and mask contains directory entry modification events (e.g., FAN_CREATE, FAN_DELETE), or directory event flags (e.g., FAN_ONDIR, FAN_EVENT_ON_CHILD), and dirfd and path do not specify a directory.

EOPNOTSUPP

The object indicated by path is associated with a filesystem that does not support the encoding of file handles. This error can be returned only with an fanotify group that identifies filesystem objects by file handles. Calling name_to_handle_at(2) with the flag AT_HANDLE_FID (since Linux 6.5) can be used as a test to check if a filesystem supports reporting events with file handles.

EPERM The operation is not permitted because the caller lacks a required capability.

EXDEV The filesystem object indicated by path resides within a filesystem subvolume (e.g., btrfs(5)) which uses a different fsid than its root superblock. This error can be returned only with an fanotify group that identifies filesystem objects by file handles. Since Linux 6.8, this error will be returned when trying to add a mount or filesystem mark on a subvolume, when trying to add inode marks in different subvolumes, or when trying to add inode marks in a btrfs(5) subvolume and in another filesystem. Since Linux 6.8, this error will also be returned when trying to add marks in different filesystems, where one of the filesystems reports zero fsid (e.g., fuse(4)).

STANDARDS

Linux.

HISTORY

Linux 2.6.37.

NOTES

FAN_OPEN_EXEC and FAN_OPEN_EXEC_PERM

When using either FAN_OPEN_EXEC or FAN_OPEN_EXEC_PERM within the mask, events of these types will be returned only when the direct execution of a program occurs. More specifically, this means that events of these types will be generated for files that are opened using `execve(2)`, `execveat(2)`, or `uselib(2)`. Events of these types will not be raised in the situation where an interpreter is passed (or reads) a file for interpretation.

Additionally, if a mark has also been placed on the Linux dynamic linker, a user should also expect to receive an event for it when an ELF object has been successfully opened using `execve(2)` or `execveat(2)`.

For example, if the following ELF binary were to be invoked and a FAN_OPEN_EXEC mark has been placed on /:

```
$ /bin/echo foo
```

The listening application in this case would receive FAN_OPEN_EXEC events for both the ELF binary and interpreter, respectively:

```
/bin/echo
```

```
/lib64/ld-linux-x86-64.so.2
```

BUGS

The following bugs were present in before Linux 3.16:

- ? If flags contains FAN_MARK_FLUSH, dirfd, and path must specify a valid filesystem object, even though this object is not used.
- ? `readdir(2)` does not generate a FAN_ACCESS event.
- ? If `fanotify_mark()` is called with FAN_MARK_FLUSH, flags is not checked for invalid values.

SEE ALSO

`fanotify_init(2)`, `fanotify(7)`

Linux man-pages 6.17

2025-10-06

`fanotify_mark(2)`