



Linux Ubuntu 26.04 Manual Pages on command 'mkfifoat.3'

\$ man mkfifoat.3

mkfifo(3) Library Functions Manual mkfifo(3)

NAME

mkfifo, mkfifoat - make a FIFO special file (a named pipe)

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

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

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

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

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

mkfifoat():

Since glibc 2.10:

`_POSIX_C_SOURCE` $\geq$ 200809L

Before glibc 2.10:

`_ATFILE_SOURCE`

DESCRIPTION

mkfifo() makes a FIFO special file with name path. mode specifies the FIFO's permissions.

It is modified by the process's umask in the usual way: the permissions of the created file are (mode & ~umask).

A FIFO special file is similar to a pipe, except that it is created in a different way. In?

stead of being an anonymous communications channel, a FIFO special file is entered into the filesystem by calling `mkfifo()`.

Once you have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa. See `fifo(7)` for nonblocking handling of FIFO special files.

`mkfifoat()`

The `mkfifoat()` function operates in exactly the same way as `mkfifo()`, 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 `mkfifo()` 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 `mkfifo()`).

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

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

RETURN VALUE

On success `mkfifo()` and `mkfifoat()` return 0. On error, -1 is returned and `errno` is set to indicate the error.

ERRORS

EACCES One of the directories in `path` did not allow search (execute) permission.

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

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

EEXIST `path` already exists. This includes the case where `path` is a symbolic link, dangling or not.

ENAMETOOLONG

Either the total size of `path` is greater than `PATH_MAX`, or an individual filename component has a length greater than `NAME_MAX`. In the GNU system, there is no imposed limit on overall filename length, but some filesystems may place limits on the length of a component.

ENOENT A directory component in `path` does not exist or is a dangling symbolic link.

ENOSPC The directory or filesystem has no room for the new file.

ENOTDIR

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

ENOTDIR

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

EROFS path refers to a read-only filesystem.

ATTRIBUTES

For an explanation of the terms used in this section, see `attributes(7)`.

[illegible]

?	Interface	?	Attribute	?	Value	?
---	-----------	---	-----------	---	-------	---

[illegible]

? mkfifo(), mkfifoat() ? Thread safety ? MT-Safe ?

[illegible]

VERSIONS

It is implemented using `mknodat(2)`.

STANDARDS

POSIX.1-2008.

HISTORY

mkfifo()

POSIX.1-2001.

mkfifoat()

glibc 2.4. POSIX.1-2008.

SEE ALSO

mkfifo(1), close(2), open(2), read(2), stat(2), umask(2), write(2), fifo(7)