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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'nftw.3p' command

$ man nftw.3p

NFTW(3P) POSIX Programmer's Manual NFTW(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nftw ? walk a file tree

SYNOPSIS

```
#include <ftw.h>
int nftw(const char *path, int (*fn)(const char *,
    const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

DESCRIPTION

The nftw() function shall recursively descend the directory hierarchy rooted in path. The nftw() function has a similar effect to ftw() ex? cept that it takes an additional argument flags, which is a bitwise-in? clusive OR of zero or more of the following flags:

FTW_CHDIR If set, nftw() shall change the current working directory to each directory as it reports files in that directory. If clear, nftw() shall not change the current working direc? tory.

FTW_DEPTH If set, nftw() shall report all files in a directory before reporting the directory itself. If clear, nftw() shall re?

port any directory before reporting the files in that di?
rectory.

FTW_MOUNT If set, nftw() shall only report files in the same file system as path. If clear, nftw() shall report all files encountered during the walk.

FTW_PHYS If set, nftw() shall perform a physical walk and shall not follow symbolic links.

If FTW_PHYS is clear and FTW_DEPTH is set, nftw() shall follow links instead of reporting them, but shall not report any directory that would be a descendant of itself. If FTW_PHYS is clear and FTW_DEPTH is clear, nftw() shall follow links instead of reporting them, but shall not report the contents of any directory that would be a descendant of itself.

At each file it encounters, nftw() shall call the user-supplied func? tion fn with four arguments:

* The first argument is the pathname of the object.
* The second argument is a pointer to the stat buffer containing in? formation on the object, filled in as if fstatat(), stat(), or lstat() had been called to retrieve the information.
* The third argument is an integer giving additional information. Its value is one of the following:

  FTW_D The object is a directory.

  FTW_DNR The object is a directory that cannot be read. The fn function shall not be called for any of its descendants.

  FTW_DP The object is a directory and subdirectories have been visited. (This condition shall only occur if the FTW_DEPTH flag is included in flags.)

  FTW_F The object is a non-directory file.

  FTW_NS The stat() function failed on the object because of lack of appropriate permission. The stat buffer passed to fn is undefined. Failure of stat() for any other reason is considered an error and nftw() shall return -1.

  FTW_SL The object is a symbolic link. (This condition shall only

occur if the FTW_PHYS flag is included in flags.)

FTW_SLN The object is a symbolic link that does not name an ex? isting file. (This condition shall only occur if the FTW_PHYS flag is not included in flags.)

* The fourth argument is a pointer to an FTW structure. The value of base is the offset of the object's filename in the pathname passed as the first argument to fn. The value of level indicates depth relative to the root of the walk, where the root level is 0.

The results are unspecified if the application-supplied fn function does not preserve the current working directory.

The argument fd_limit sets the maximum number of file descriptors that shall be used by nftw() while traversing the file tree. At most one file descriptor shall be used for each directory level.

The nftw() function need not be thread-safe.

RETURN VALUE

The nftw() function shall continue until the first of the following conditions occurs:

* An invocation of fn shall return a non-zero value, in which case nftw() shall return that value.
* The nftw() function detects an error other than [EACCES] (see FTW_DNR and FTW_NS above), in which case nftw() shall return -1 and set errno to indicate the error.
* The tree is exhausted, in which case nftw() shall return 0.

ERRORS

The nftw() function shall fail if:

EACCES Search permission is denied for any component of path or read permission is denied for path, or fn returns -1 and does not re? set errno.

ELOOP A loop exists in symbolic links encountered during resolution of the path argument.

ENAMETOOLONG

The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT A component of path does not name an existing file or path is an empty string.

ENOTDIR A component of path names an existing file that is neither a directory nor a symbolic link to a directory.

EOVERFLOW A field in the stat structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The nftw() function may fail if:

ELOOP More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path argument.

EMFILE All file descriptors available to the process are currently open.

ENAMETOOLONG The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE Too many files are currently open in the system.

In addition, errno may be set if the function pointed to by fn causes errno to be set.

The following sections are informative.

EXAMPLES

The following program traverses the directory tree under the path named in its first command-line argument, or under the current directory if no argument is supplied. It displays various information about each file. The second command-line argument can be used to specify characters that control the value assigned to the flags argument when calling nftw().

```
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <stdint.h>
static int
display_info(const char *fpath, const struct stat *sb,
    int tflag, struct FTW *ftwbuf)
{
    printf("%-3s %2d %7jd   %-40s %d %s\n",
        (tflag == FTW_D) ?   "d"   : (tflag == FTW_DNR) ? "dnr" :
        (tflag == FTW_DP) ?  "dp"  : (tflag == FTW_F) ?
            (S_ISBLK(sb->st_mode) ? "f b" :
             S_ISCHR(sb->st_mode) ? "f c" :
             S_ISFIFO(sb->st_mode) ? "f p" :
             S_ISREG(sb->st_mode) ? "f r" :
             S_ISSOCK(sb->st_mode) ? "f s" : "f ?") :
        (tflag == FTW_NS) ?  "ns"  : (tflag == FTW_SL) ?  "sl" :
        (tflag == FTW_SLN) ? "sln" : "?",
        ftwbuf->level, (intmax_t) sb->st_size,
        fpath, ftwbuf->base, fpath + ftwbuf->base);
    return 0;           /* To tell nftw() to continue */
}
int
main(int argc, char *argv[])
{
    int flags = 0;
    if (argc > 2 && strchr(argv[2], 'd') != NULL)
        flags |= FTW_DEPTH;
    if (argc > 2 && strchr(argv[2], 'p') != NULL)
        flags |= FTW_PHYS;
    if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
    {
        perror("nftw");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
```

```
}
```

APPLICATION USAGE

The nftw() function may allocate dynamic storage during its operation. If nftw() is forcibly terminated, such as by longjmp() or siglongjmp() being executed by the function pointed to by fn or an interrupt rou? tine, nftw() does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by fn return a non-zero value at its next invocation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir(), fstatat(), readdir()

The Base Definitions volume of POSIX.1?2017, <ftw.h>

COPYRIGHT