NAME

fnmatch – match filename or pathname

SYNOPSIS

#include <fnmatch.h>

int fnmatch(const char * *pattern***, const char *** *string***, int** *flags***);**

DESCRIPTION

The **fnmatch**() function checks whether the *string* argument matches the *pattern* argument, which is a shell wildcard pattern.

The *flags* argument modifies the behavior; it is the bitwise OR of zero or more of the following flags:

FNM_NOESCAPE
If this flag is set, treat backslash as an ordinary character, instead of an escape character.

FNM_PATHNAME
If this flag is set, match a slash in *string* only with a slash in *pattern* and not by an asterisk (*) or a question mark (?) metacharacter, nor by a bracket expression ([]) containing a slash.

FNM_PERIOD
If this flag is set, a leading period in *string* has to be matched exactly by a period in *pattern*. A period is considered to be leading if it is the first character in *string*, or if both **FNM_PATHNAME** is set and the period immediately follows a slash.

FNM_FILE_NAME
This is a GNU synonym for **FNM_PATHNAME**.

FNM_LEADING_DIR
If this flag (a GNU extension) is set, the pattern is considered to be matched if it matches an initial segment of *string* which is followed by a slash. This flag is mainly for the internal use of glibc and is only implemented in certain cases.

FNM_CASEFOLD
If this flag (a GNU extension) is set, the pattern is matched case-insensitively.

RETURN VALUE

Zero if *string* matches *pattern*, **FNM_NOMATCH** if there is no match or another nonzero value if there is an error.

CONFORMING TO

POSIX.2. The **FNM_FILE_NAME**, **FNM_LEADING_DIR**, and **FNM_CASEFOLD** flags are GNU extensions.

NAME

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf – formatted output conversion

SYNOPSIS

#include <stdio.h>

int printf(const char * *format***, ...);**
int fprintf(FILE * *stream***, const char *** *format***, ...);**
int sprintf(char * *str***, const char *** *format***, ...);**
int snprintf(char * *str***, size_t** *size***, const char *** *format***, ...);**
...

DESCRIPTION

The functions in the **printf**() family produce output according to a *format* as described below. The function **printf**() writes output to *stdout*, the standard output stream; **fprintf**() writes output to the given output *stream*; **sprintf**() and **snprintf**(), write to the character string *str*.

The function **snprintf**() writes at most *size* bytes (including the trailing null byte ('\0')) to *str*.

These functions write the output under the control of a *format* string that specifies how subsequent arguments (or arguments accessed via the variable-length argument facilities of **stdarg**(3)) are converted for output.

RETURN VALUE

Upon successful return, these functions return the number of characters printed (not including the trailing '\0' used to end output to strings).

The functions **snprintf**() and **vsnprintf**() do not write more than *size* bytes (including the trailing '\0'). If the output was truncated due to this limit then the return value is the number of characters (not including the trailing '\0') which would have been written to the final string if enough space had been available. Thus, a return value of *size* or more means that the output was truncated.

If an output error is encountered, a negative value is returned.

FORMAT OF THE FORMAT STRING

The format string is a character string, beginning and ending in its initial shift state, if any. The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character **%**, and ends with a *conversion specifier*. In between there may be (in this order) zero or more *flags*, an optional minimum *field width*, an optional *precision* and an optional *length modifier*.

The conversion specifier

A character that specifies the type of conversion to be applied. An example for a conversion specifier is:

d, **i** The *int* argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.

o, **u**, **x**, **X**
The *unsigned int* argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal (**x** and **X**) notation.

s The *const char* * argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating null byte ('\0'); if a precision is specified, no more than the number specified are written. If a precision is given, no null byte need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating null byte.

NAME

scandir, alphasort, versionsort – scan a directory for matching entries

SYNOPSIS

#include <dirent.h>

int scandir(const char ****dirp*, struct dirent ****namelist*,
int (* filter*)(const struct dirent *),**
int (* compar*)(const struct dirent **, const struct dirent **));**

int alphasort(const struct dirent ** a*, **const struct dirent **** b*);**

int versionsort(const struct dirent ** a*, **const struct dirent **** b*);**

DESCRIPTION

The **scandir**() function scans the directory *dirp*, calling *filter*() on each directory entry. Entries for which *filter*() returns nonzero are stored in strings allocated via **malloc**(3), sorted using **qsort**(3) with the comparison function *compar*(), and collected in array *namelist* which is allocated via **malloc**(3). If *filter* is NULL, all entries are selected.

The comparison function *compar*() must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

The **alphasort**() and **versionsort**() functions can be used as the comparison function *compar*(). The former sorts directory entries using **strcoll**(3), the latter using **strverscmp**(3)
on the strings *(*a)->d_name* and *(*b)->d_name*.

The **dirent** structure includes the following members:
ino_t d_ino File serial number.
char d_name[] Filename string of entry.

RETURN VALUE

The **scandir**() function returns the number of directory entries selected. On error, −1 is returned, with *errno* set to indicate the cause of the error.

The **alphasort**() and **versionsort**() functions return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

ERRORS

ENOENT
The path in *dirp* does not exist.

ENOMEM
Insufficient memory to complete the operation.

ENOTDIR
The path in *dirp* is not a directory.

NAME

stat, lstat – get file status

SYNOPSIS

int stat(const char * path*, **struct stat *** buf*);**
int lstat(const char * path*, **struct stat *** buf*);**

DESCRIPTION

These functions return information about a file. No permissions are required on the file itself, but — in the case of **stat**() and **lstat**() — execute (search) permission is required on all of the directories in *path* that lead to the file.

stat() stats the file pointed to by *path* and fills in *buf* .

lstat() is identical to **stat**(), except that if *path* is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file  */
    ino_t     st_ino;     /* inode number                  */
    mode_t    st_mode;    /* protection                    */
    ...
};
```

The *st_dev* field describes the device on which this file resides.

The following POSIX macros are defined to check the file type using the *st_mode* field:

S_ISREG(m) is it a regular file?

S_ISDIR(m) directory?

S_ISCHR(m) character device?

S_ISBLK(m) block device?

S_ISFIFO(m) FIFO (named pipe)?

S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)

S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

EACCES
Search permission is denied for one of the directories in the path prefix of *path*. (See also **path_resolution**(7).)

EBADF
fd is bad.

EFAULT
Bad address.

ELOOP
Too many symbolic links encountered while traversing the path.

ENAMETOOLONG
File name too long.

...