NAME

glob, globfree – generate path names matching a pattern

SYNOPSIS

#include <glob.h>

int glob(const char **pattern*, **int** *flags*, **int(****errfunc*)**(const char** **epath*, **int** *eerrno*)**,**
glob_t **pglob*);

void globfree(glob_t **pglob*);

MT-LEVEL

MT-Safe

DESCRIPTION

The **glob( )** function is a path name generator.

The **globfree( )** function frees any memory allocated by **glob( )** associated with *pglob*.

pattern **Argument**

The argument *pattern* is a pointer to a path name pattern to be expanded. The **glob( )** function matches all accessible path names against this pattern and develops a list of all path names that match. In order to have access to a path name, **glob( )** requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters:

* ? [

pglob **Argument**

The structure type **glob_t** is defined in the header **<glob.h>** and includes at least the following members:

size_t gl_pathc	Count of paths matched by *pattern*.
char **gl_pathv	Pointer to a list of matched path names.
size_t gl_offs	Slots to reserve at the beginning of **gl_pathv**.

The **glob( )** function stores the number of matched path names into *pglob*–>**gl_pathc** and a pointer to a list of pointers to path names into *pglob*–>**gl_pathv**. The path names are in sort order as defined by the current setting of the **LC_COLLATE** category. The first pointer after the last path name is a **NULL** pointer. If the pattern does not match any path names, the returned number of matched paths is set to zero, and the contents of *pglob*–>**gl_pathv** are implementation-dependent.

It is the caller's responsibility to create the structure pointed to by *pglob*. The **glob( )** function allocates other space as needed, including the memory pointed to by **gl_pathv**. The **globfree( )** function frees any space associated with *pglob* from a previous call to **glob( )**.

flags **Argument**

The *flags* argument is used to control the behavior of **glob( )**. The value of *flags* is a bitwise inclusive OR of zero or more of the following constants, which are defined in the header **<glob.h>**:

GLOB_APPEND Append path names generated to the ones from a previous call to **glob( )**.

GLOB_DOOFFS Make use of *pglob*–>**gl_offs**. If this flag is set, *pglob*–>**gl_offs** is used to specify how many **NULL** pointers to add to the beginning of *pglob*–>**gl_pathv**. In other words, *pglob*–>**gl_pathv** will point to *pglob*–>**gl_offs** **NULL** pointers, followed by *pglob*–>**gl_pathc** path name pointers, followed by a **NULL** pointer.

GLOB_ERR Causes **glob( )** to return when it encounters a directory that it cannot open or read. Ordinarily, **glob( )** continues to find matches.

GLOB_MARK Each path name that is a directory that matches *pattern* has a slash appended.

GLOB_NOCHECK If *pattern* does not match any path name, then **glob( )** returns a list consisting of only *pattern*, and the number of matched path names is 1.

GLOB_NOESCAPE Disable backslash escaping.

GLOB_NOSORT Ordinarily, **glob( )** sorts the matching path names according to the current setting of the **LC_COLLATE** category. When this flag is used the order of path names returned is unspecified.

The **GLOB_APPEND** flag can be used to append a new set of path names to those found in a previous call to **glob( )**. The following rules apply when two or more calls to **glob( )** are made with the same value of *pglob* and without intervening calls to **globfree( )**:

1. The first such call must not set **GLOB_APPEND**. All subsequent calls must set it.
2. All the calls must set **GLOB_DOOFFS**, or all must not set it.
3. After the second call, *pglob*–>**gl_pathv** points to a list containing the following:
   a. Zero or more **NULL** pointers, as specified by **GLOB_DOOFFS** and *pglob*–>**gl_offs**.
   b. Pointers to the path names that were in the *pglob*–>**gl_pathv** list before the call, in the same order as before.
   c. Pointers to the new path names generated by the second call, in the specified order.
4. The count returned in *pglob*–>**gl_pathc** will be the total number of path names from the two calls.
5. The application can change any of the fields after a call to **glob( )**. If it does, it must reset them to the original value before a subsequent call, using the same *pglob* value, to **globfree( )** or **glob( )** with the **GLOB_APPEND** flag.

errfunc and *epath* Arguments

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not a **NULL** pointer, **glob( )** calls **(**∗*errfunc*) with two arguments:

1. The *epath* argument is a pointer to the path that failed.
2. The *eerrno* argument is the value of *errno* from the failure, as set by the **opendir**(3C), **readdir**(3C) or **stat**(2) functions. (Other values may be used to report other errors not explicitly documented for those functions.)

The following constants are defined as error return values for **glob( )**:

GLOB_ABORTED The scan was stopped because **GLOB_ERR** was set or **(∗errfunc)** returned non-zero.

GLOB_NOMATCH The pattern does not match any existing path name, and **GLOB_NOCHECK** was not set in flags.

GLOB_NOSPACE An attempt to allocate memory failed.

If **(***errfunc***)** is called and returns non-zero, or if the **GLOB_ERR** flag is set in *flags*, **glob( )** stops the scan and returns **GLOB_ABORTED** after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If **GLOB_ERR** is not set and either *errfunc* is a **NULL** pointer or **(***errfunc***)** returns zero, the error is ignored.

RETURN VALUES

The following values are returned by **glob( )**:

0 successful completion. The argument *pglob*–>**gl_pathc** returns the number of matched path names and the argument *pglob*–>**gl_pathv** contains a pointer to a null-terminated list of matched and sorted path names. However, if *pglob*–>**gl_pathc** is zero, the content of *pglob*–>**gl_pathv** is undefined.

non-zero an error has occurred. Non-zero constants are defined in **<glob.h>**. The arguments *pglob*–>**gl_pathc** and *pglob*–>**gl_pathv** are still set as defined above.

The **globfree( )** function returns no value.

USAGE

This function is not provided for the purpose of enabling utilities to perform path name expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do path name expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a path name matches a given pattern, it can use **fnmatch**(3C).

Note that **gl_pathc** and **gl_pathv** have meaning even if **glob( )** fails. This allows **glob( )** to report partial results in the event of an error. However, if **gl_pathc** is zero, **gl_pathv** is unspecified even if **glob( )** did not return an error.

The **GLOB_NOCHECK** option could be used when an application wants to expand a path name if wildcards are specified, but wants to treat the pattern as just a string otherwise.

The new path names generated by a subsequent call with **GLOB_APPEND** are not sorted together with the previous path names. This mirrors the way that the shell handles path name expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use the **wordexp( )** function.

EXAMPLES

One use of the **GLOB_DOOFFS** flag is by applications that build an argument list for use with the **execv**(2), **execve( )** or **execvp( )** functions. Suppose, for example, that an application wants to do the equivalent of:

```
ls -l *.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using **GLOB_APPEND** as follows:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS | GLOB_APPEND, NULL, &globbuf);
. . .
```

SEE ALSO

execv(2), **stat**(2), **fnmatch**(3C), **opendir**(3C), **readdir**(3C), **wordexp**(3C)