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 'glob.3p' command

$ man glob.3p

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

glob, globfree ? generate pathnames matching a pattern

SYNOPSIS

```
#include <glob.h>
int glob(const char *restrict pattern, int flags,
    int(*errfunc)(const char *epath, int eerrno),
    glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

DESCRIPTION

The glob() function is a pathname generator that shall implement the rules defined in the Shell and Utilities volume of POSIX.1?2017, Sec? tion 2.13, Pattern Matching Notation, with optional support for rule 3 in the Shell and Utilities volume of POSIX.1?2017, Section 2.13.3, Pat? terns Used for Filename Expansion.

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

??

?Member Type ? Member Name ? Description ?

??

?size_t ?gl_pathc ? Count of paths matched by pattern. ?

?char ** ?gl_pathv ? Pointer to a list of matched pathnames. ?

?size_t ?gl_offs ? Slots to reserve at the beginning of ?

? ? ? gl_pathv. ?

??

The argument pattern is a pointer to a pathname pattern to be expanded.

The glob() function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, glob() requires search permission on every component of a path except the last, and read permission on each direc? tory of any filename component of pattern that contains any of the fol? lowing special characters: '*', '?', and '['.

The glob() function shall store the number of matched pathnames into pglob->gl_pathc and a pointer to a list of pointers to pathnames into pglob->gl_pathv. The pathnames shall be in sort order as defined by the current setting of the LC_COLLATE category; see the Base Definitions volume of POSIX.1?2017, Section 7.3.2, LC_COLLATE. The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of pglob->gl_pathv are implementation-defined.

It is the caller's responsibility to create the structure pointed to by pglob. The glob() function shall allocate other space as needed, in? cluding the memory pointed to by gl_pathv. The globfree() function shall free any space associated with pglob from a previous call to glob().

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 follow? ing constants, which are defined in <glob.h>:

GLOB_APPEND Append pathnames 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 shall point to pglob->gl_offs null pointers, followed by pglob->gl_pathc pathname pointers, followed by a null pointer.

GLOB_ERR Cause glob() to return when it encounters a directory that it cannot open or read. Ordinarily, glob() contin? ues to find matches.

GLOB_MARK Each pathname that is a directory that matches pattern shall have a <slash> appended.

GLOB_NOCHECK Supports rule 3 in the Shell and Utilities volume of POSIX.1?2017, Section 2.13.3, Patterns Used for Filename Expansion. If pattern does not match any pathname, then glob() shall return a list consisting of only pattern, and the number of matched pathnames is 1.

GLOB_NOESCAPE Disable backslash escaping.

GLOB_NOSORT Ordinarily, glob() sorts the matching pathnames according to the current setting of the LC_COLLATE category; see the Base Definitions volume of POSIX.1?2017, Section 7.3.2, LC_COLLATE. When this flag is used, the order of pathnames returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to glob(). The following rules apply to applications 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 shall not set GLOB_APPEND. All subsequent calls shall set it.
2. All the calls shall set GLOB_DOOFFS, or all shall 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 pathnames that were in the pglob->gl_pathv list

before the call, in the same order as before.

c. Pointers to the new pathnames generated by the second call, in the specified order.

4. The count returned in pglob->gl_pathc shall be the total number of pathnames from the two calls.

5. The application can change any of the fields after a call to glob(). If it does, the application shall reset them to the origi? nal value before a subsequent call, using the same pglob value, to globfree() or glob() with the GLOB_APPEND flag.

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 opendir(), readdir(), or stat(). (Other values may be used to report other errors not explicitly documented for those functions.)

If (()*errfunc) is called and returns non-zero, or if the GLOB_ERR flag is set in flags, glob() shall stop the scan and return 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 0, the error shall be ignored.

The glob() function shall not fail because of large files.

RETURN VALUE

Upon successful completion, glob() shall return 0. The argument pglob->gl_pathc shall return the number of matched pathnames and the argument pglob->gl_pathv shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if pglob->gl_pathc is 0, the content of pglob->gl_pathv is undefined.

The globfree() function shall not return a value.

If glob() terminates due to an error, it shall return one of the non-zero constants defined in <glob.h>. The arguments pglob->gl_pathc and pglob->gl_pathv are still set as defined above.

ERRORS

The glob() function shall fail and return the corresponding value if:

GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (()*er?

rfunc) returned non-zero.

GLOB_NOMATCH The pattern does not match any existing pathname, and

GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE An attempt to allocate memory failed.

The following sections are informative.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argu?

ment list for use with execv(), execve(), or execvp(). 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);
...
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to

perform pathname 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

pathname 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 pathname matches a given pattern, it can use fnmatch().

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. How? ever, if gl_pathc is 0, gl_pathv is unspecified even if glob() did not return an error.

The GLOB_NOCHECK option could be used when an application wants to ex? pand a pathname if wildcards are specified, but wants to treat the pat? tern as just a string otherwise. The sh utility might use this for op? tion-arguments, for example.

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

Applications that need tilde and parameter expansion should use word? exp().

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
      * sizeof(char *));
(void) memcpy(new+n, pglob->gl_pathv,
      pglob->gl_pathc * sizeof(char *));
(void) memset(new, 0, n * sizeof(char *));
free(pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by gl_pathv is a block that was separately created using malloc(). This is not necessarily the case. An application should make no assumptions about how the mem? ory referenced by fields in pglob was allocated. It might have been ob? tained from malloc() in a large chunk and then carved up within glob(),

or it might have been created using a different memory allocator. It is not the intent of the standard developers to specify or imply how the memory used by glob() is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, fdopendir(), fnmatch(), fstatat(), readdir(), Section 2.6, Word Expansions

The Base Definitions volume of POSIX.1?2017, Section 7.3.2, LC_COLLATE, <glob.h>

COPYRIGHT