NAME

File::Glob - Perl extension for BSD glob routine

SYNOPSIS

```
use File::Glob ':bsd_glob';

@list = bsd_glob('*.[ch]');
$homedir = bsd_glob('~gnat', GLOB_TILDE | GLOB_ERR);

if (GLOB_ERROR) {
  # an error occurred reading $homedir
}

## override the core glob (CORE::glob() does this automatically
## by default anyway, since v5.6.0)
use File::Glob ':globally';
my @sources = <*.{c,h,y}>;

## override the core glob, forcing case sensitivity
use File::Glob qw(:globally :case);
my @sources = <*.{c,h,y}>;

## override the core glob forcing case insensitivity
use File::Glob qw(:globally :nocase);
my @sources = <*.{c,h,y}>;

## glob on all files in home directory
use File::Glob ':globally';
my @sources = <~gnat/*>;
```

DESCRIPTION

The glob angle-bracket operator <> is a pathname generator that implements the rules for file name pattern matching used by Unix-like shells such as the Bourne shell or C shell.

File::Glob::bsd_glob() implements the FreeBSD glob(3) routine, which is a superset of the POSIX glob() (described in IEEE Std 1003.2 "POSIX.2"). bsd_glob() takes a mandatory `pattern` argument, and an optional `flags` argument, and returns a list of filenames matching the pattern, with interpretation of the pattern modified by the `flags` variable.

Since v5.6.0, Perl's CORE::glob() is implemented in terms of bsd_glob(). Note that they don't share the same prototype--CORE::glob() only accepts a single argument. Due to historical reasons, CORE::glob() will also split its argument on whitespace, treating it as multiple patterns, whereas bsd_glob() considers them as one pattern. But see `:bsd_glob` under *EXPORTS*, below.

META CHARACTERS

```
\       Quote the next metacharacter
[]      Character class
{}      Multiple pattern
*       Match any string of characters
?       Match any single character
~       User name home directory
```

The metanotation `a{b,c,d}e` is a shorthand for `abe ace ade`. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. As a special case `{`, `}` , and `{}` are passed undisturbed.

EXPORTS

See also the *POSIX FLAGS* below, which can be exported individually.

:bsd_glob

The `:bsd_glob` export tag exports bsd_glob() and the constants listed below. It also overrides glob() in the calling package with one that behaves like bsd_glob() with regard to spaces (the space is treated as part of a file name), but supports iteration in scalar context; i.e., it preserves the core function's feature of returning the next item each time it is called.

:glob

The `:glob` tag, now discouraged, is the old version of `:bsd_glob`. It exports the same constants and functions, but its glob() override does not support iteration; it returns the last file name in scalar context. That means this will loop forever:

```
    use File::Glob ':glob';
    while (my $file = <* copy.txt>) {
 ...
    }
```

bsd_glob

This function, which is included in the two export tags listed above, takes one or two arguments. The first is the glob pattern. The second, if given, is a set of flags ORed together. The available flags and the default set of flags are listed below under *POSIX FLAGS*.

Remember that to use the named constants for flags you must import them, for example with `:bsd_glob` described above. If not imported, and `use strict` is not in effect, then the constants will be treated as bareword strings, which won't do what you what.

:nocase and :case

These two export tags globally modify the default flags that bsd_glob() and, except on VMS, Perl's built-in `glob` operator use. `GLOB_NOCASE` is turned on or off, respectively.

csh_glob

The csh_glob() function can also be exported, but you should not use it directly unless you really know what you are doing. It splits the pattern into words and feeds each one to bsd_glob(). Perl's own glob() function uses this internally.

POSIX FLAGS

If no flags argument is give then `GLOB_CSH` is set, and on VMS and Windows systems, `GLOB_NOCASE` too. Otherwise the flags to use are determined solely by the flags argument. The POSIX defined flags are:

`GLOB_ERR`

Force bsd_glob() to return an error when it encounters a directory it cannot open or read. Ordinarily bsd_glob() continues to find matches.

`GLOB_LIMIT`

Make bsd_glob() return an error (GLOB_NOSPACE) when the pattern expands to a size bigger than the system constant `ARG_MAX` (usually found in limits.h). If your system does not define this constant, bsd_glob() uses `sysconf(_SC_ARG_MAX)` or `_POSIX_ARG_MAX` where available (in that order). You can inspect these values using the standard `POSIX` extension.

`GLOB_MARK`

Each pathname that is a directory that matches the pattern has a slash appended.

`GLOB_NOCASE`

By default, file names are assumed to be case sensitive; this flag makes bsd_glob() treat case differences as not significant.

`GLOB_NOCHECK`

If the pattern does not match any pathname, then bsd_glob() returns a list consisting of only the pattern. If `GLOB_QUOTE` is set, its effect is present in the pattern returned.

`GLOB_NOSORT`

By default, the pathnames are sorted in ascending ASCII order; this flag prevents that sorting (speeding up bsd_glob()).

The FreeBSD extensions to the POSIX standard are the following flags:

`GLOB_BRACE`

Pre-process the string to expand `{pat,pat,...}` strings like csh(1). The pattern '{}' is left unexpanded for historical reasons (and csh(1) does the same thing to ease typing of find(1) patterns).

`GLOB_NOMAGIC`

Same as `GLOB_NOCHECK` but it only returns the pattern if it does not contain any of the special characters "*", "?" or "[". `NOMAGIC` is provided to simplify implementing the historic csh(1) globbing behaviour and should probably not be used anywhere else.

`GLOB_QUOTE`

Use the backslash ('\') character for quoting: every occurrence of a backslash followed by a character in the pattern is replaced by that character, avoiding any special interpretation of the character. (But see below for exceptions on DOSISH systems).

`GLOB_TILDE`

Expand patterns that start with '~' to user name home directories.

`GLOB_CSH`

For convenience, `GLOB_CSH` is a synonym for `GLOB_BRACE | GLOB_NOMAGIC | GLOB_QUOTE | GLOB_TILDE | GLOB_ALPHASORT`.

The POSIX provided `GLOB_APPEND`, `GLOB_DOOFFS`, and the FreeBSD extensions `GLOB_ALTDIRFUNC`, and `GLOB_MAGCHAR` flags have not been implemented in the Perl version because they involve more complex interaction with the underlying C structures.

The following flag has been added in the Perl implementation for csh compatibility:

`GLOB_ALPHASORT`

If `GLOB_NOSORT` is not in effect, sort filenames is alphabetical order (case does not matter) rather than in ASCII order.

DIAGNOSTICS

bsd_glob() returns a list of matching paths, possibly zero length. If an error occurred, &File::Glob::GLOB_ERROR will be non-zero and `$!` will be set. &File::Glob::GLOB_ERROR is guaranteed to be zero if no error occurred, or one of the following values otherwise:

`GLOB_NOSPACE`

An attempt to allocate memory failed.

`GLOB_ABEND`

The glob was stopped because an error was encountered.

In the case where bsd_glob() has found some matching paths, but is interrupted by an error, it will return a list of filenames **and** set &File::Glob::ERROR.

Note that bsd_glob() deviates from POSIX and FreeBSD glob(3) behaviour by not considering `ENOENT` and `ENOTDIR` as errors - bsd_glob() will continue processing despite those errors, unless the `GLOB_ERR` flag is set.

Be aware that all filenames returned from File::Glob are tainted.

NOTES

- If you want to use multiple patterns, e.g. `bsd_glob("a* b*")`, you should probably throw them in a set as in `bsd_glob("{a*,b*}")`. This is because the argument to bsd_glob() isn't subjected to parsing by the C shell. Remember that you can use a backslash to escape things.
- On DOSISH systems, backslash is a valid directory separator character. In this case, use of backslash as a quoting character (via GLOB_QUOTE) interferes with the use of backslash as a directory separator. The best (simplest, most portable) solution is to use forward slashes for directory separators, and backslashes for quoting. However, this does not match "normal practice" on these systems. As a concession to user expectation, therefore, backslashes (under GLOB_QUOTE) only quote the glob metacharacters '[', ']', '{', '}', '-', '~', and backslash itself. All other backslashes are passed through unchanged.
- Win32 users should use the real slash. If you really want to use backslashes, consider using Sarathy's File::DosGlob, which comes with the standard Perl distribution.

SEE ALSO

"glob" in perlfunc, glob(3)

AUTHOR

The Perl interface was written by Nathan Torkington <gnat@frii.com>, and is released under the artistic license. Further modifications were made by Greg Bacon <gbacon@cs.uah.edu>, Gurusamy Sarathy <gsar@activestate.com>, and Thomas Wegner <wegner_thomas@yahoo.com>. The C glob code has the following copyright:

Copyright (c) 1989, 1993 The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Guido van Rossum.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.