```
**********************************************************
   11248 Mon Jun 15 18:59:41 2015
new/usr/src/man/man5/fnmatch.5
3768 fnmatch(5) is worded poorly
**********************************************************
'\" te
.\"  Copyright (c) 1992, X/Open Company Limited
.\"  All Rights Reserved  Portions Copyright (c) 1995, Sun Microsystems, Inc.
.\"  All Rights Reserved
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\"  This notice shall appear on any product containing this material.
.\" The contents of this file are subject to the terms of the Common Development
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
.\" When distributing Covered Code, include this CDDL HEADER in each file and in
.TH FNMATCH 5 "Jun 14, 2015"
.TH FNMATCH 5 "Mar 28, 1995"
.SH NAME
fnmatch \- file name pattern matching
.SH DESCRIPTION
.sp
.LP
The pattern matching notation described below  is used to specify patterns for
matching strings in the shell. Historically, pattern matching notation is
related to, but slightly different from, the regular expression notation. For
this reason, the description of the rules for this pattern matching notation is
based on the description of regular expression notation described on the
\fBregex\fR(5) manual page.
.SS "Patterns Matching a Single Character"
.sp
.LP
The following patterns match a single character: \fIordinary characters\fR,
\fIspecial pattern characters\fR and \fIpattern bracket expressions\fR. The patt
bracket expression will also match a single collating element.
The following \fIpatterns matching a single character\fR match a single
character: \fIordinary characters\fR, \fIspecial pattern characters\fR and
\fIpattern bracket expressions\fR. The pattern bracket expression will also
match a single collating element.
.sp
.LP
An ordinary character is a pattern that matches itself. It can be any character
in the supported character set except for \fINUL\fR, those special shell
characters that require quoting, and the following three special pattern
characters. Matching is based on the bit pattern used for encoding the
character, not on the graphic representation of the character. If any character
(ordinary, shell special, or pattern special) is quoted, that pattern will
match the character itself. The shell special characters always require
quoting.
.sp
.LP
When unquoted and outside a bracket expression, the following three characters
will have special meaning in the specification of patterns:
.sp
.ne 2
.na
\fB\fB?\fR \fR
.ad
.RS 6n
A question-mark is a pattern that will match any character.
.RE

.sp
.ne 2
.na
\fB\fB*\fR \fR
```



```
.ad
.RS 6n
An asterisk is a pattern that will match multiple characters, as described in
\fBPatterns Matching Multiple Characters\fR, below.
.RE

.sp
.ne 2
.na
\fB\fB[\fR \fR
.ad
.RS 6n
The open bracket will introduce a pattern bracket expression.
.RE

.sp
.LP
The description of basic regular expression bracket expressions on the
\fBregex\fR(5) manual page also applies to the pattern bracket expression,
except that the exclamation-mark character \fB(\fR \fB!\fR \fB)\fR replaces the
circumflex character (\fB^\fR) in its role in a \fInon-matching list\fR in the
regular expression notation. A bracket expression starting with an unquoted
circumflex character produces unspecified results.
.sp
.LP
The restriction on a circumflex in a bracket expression is to allow
implementations that support pattern matching using the circumflex as the
negation character in addition to the exclamation-mark. A portable application
must use something like \fB[\e^!\fR] to match either character.
.sp
.LP
When pattern matching is used where shell quote removal is not performed (such
as in the argument to the \fBfind\fR \fB-name\fR primary when \fBfind\fR is
being called using one of the  \fBexec\fR functions, or in the \fIpattern\fR
argument to the \fBfnmatch\fR(3C) function, special characters can be escaped
to remove their special meaning by preceding them with a backslash character.
This escaping backslash will be discarded. The sequence \fB\e\e\fR represents
one literal backslash. All of the requirements and effects of quoting on
ordinary, shell special and special pattern characters will apply to escaping
in this context.
.sp
.LP
Both quoting and escaping are described here because pattern matching must work
in three separate circumstances:
.RS +4
.TP
.ie t \(bu
.el o
Calling directly upon the shell, such as in pathname expansion or in a
\fBcase\fR statement. All of the following will match the string or file
\fBabc\fR:
.sp

.sp
.TS
l l l l l
l l l l l .
\fBabc\fR       \fB"abc"\fR     \fBa"b"c\fR     \fBa\ebc\fR     \fBa[b]c\fR
\fBa["b"]c\fR   \fBa[\eb]c\fR   \fBa["\eb"]c\fR \fBa?c\fR       \fBa*c\fR
.TE

The following will not:
.sp

.sp
.TS
```

```
l l l .
\fB"a?c"\fR     \fBa\e*c\fR     \fBa\e[b]c\fR
.TE

.RE
.RS +4
.TP
.ie t \(bu
.el o
Calling a utility or function without going through a shell, as described for
\fBfind\fR(1) and the function \fBfnmatch\fR(3C)
.RE
.RS +4
.TP
.ie t \(bu
.el o
Calling utilities such as \fBfind\fR, \fBcpio\fR, \fBtar\fR or \fBpax\fR
through the shell command line. In this case, shell quote removal is performed
before the utility sees the argument.  For example, in:
.sp
find /bin -name e\ec[\eh]o -print
.sp
after quote removal, the backslashes are presented to \fBfind\fR and it treats
them as escape characters. Both precede ordinary characters, so the \fBc\fR and
\fBh\fR represent themselves and \fBecho\fR would be found on many historical
systems (that have it in \fB/bin\fR). To find a file name that contained shell
special characters or pattern characters, both quoting and escaping are
required, such as:
.sp
\fBpax -r .\|.\|. "*a\e\|(\|\e?"\fR
.sp
to extract a filename ending with \fBa(?\fR.
.RE
.sp
.LP
Conforming applications are required to quote or escape the shell special
characters (sometimes called metacharacters). If used without this protection,
syntax errors can result or implementation extensions can be triggered. For
example, the KornShell supports a series of extensions based on parentheses in
patterns; see  \fBksh\fR(1)
.SS "Patterns Matching Multiple Characters"
.sp
.LP
The following rules are used to construct \fIpatterns matching multiple
characters\fR from \fIpatterns matching a single character\fR:
.RS +4
.TP
.ie t \(bu
.el o
The asterisk (*) is a pattern that will match any string, including the null
string.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The concatenation of \fIpatterns matching a single character\fR is a valid
pattern that will match the concatenation of the single characters or collating
elements matched by each of the concatenated patterns.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The concatenation of one or more \fIpatterns matching a single character\fR
with one or more asterisks is a valid pattern. In such patterns, each asterisk
```



```
will match a string of zero or more characters, matching the greatest possible
number of characters that still allows the remainder of the pattern to match
the string.
.RE
.sp
.LP
Since each asterisk matches zero or more occurrences, the patterns \fBa*b\fR
and  \fBa**b\fR have identical functionality.
.sp
.LP
Examples:
.sp
.ne 2
.na
\fB\fBa[bc]\fR \fR
.ad
.RS 10n
matches the strings \fBab\fR and \fBac\fR.
.RE

.sp
.ne 2
.na
\fB\fBa*d\fR \fR
.ad
.RS 10n
matches the strings \fBad\fR, \fBabd\fR and \fBabcd\fR, but not the string
\fBabc\fR.
.RE

.sp
.ne 2
.na
\fB\fBa*d*\fR \fR
.ad
.RS 10n
matches the strings \fBad\fR, \fBabcd\fR, \fBabcdef\fR, \fBaaaad\fR and
\fBadddd\fR.
.RE

.sp
.ne 2
.na
\fB\fB*a*d\fR \fR
.ad
.RS 10n
matches the strings \fBad\fR, \fBabcd\fR, \fBefabcd\fR, \fBaaaad\fR and
\fBadddd\fR.
.RE

.SS "Patterns Used for Filename Expansion"
.sp
.LP
The rules described so far in \fBPatterns\fR \fBMatching\fR \fBMultiple\fR
\fBCharacters\fR and \fBPatterns\fR \fBMatching\fR \fBa\fR \fBSingle\fR
\fBCharacter\fR are qualified by the following rules that apply when pattern
matching notation is used for filename expansion.
.RS +4
.TP
1.
The slash character in a pathname must be explicitly matched by using one
or more slashes in the pattern; it cannot be matched by the asterisk or
question-mark special characters or by a bracket expression. Slashes in the
pattern are identified before bracket expressions; thus, a slash cannot be
included in a pattern bracket expression used for filename expansion. For
example, the pattern \fBa[b/c]d\fR will not match such pathnames as \fBabd\fR
```

```
251 or \fBa/d\fR. It will only match a pathname of literally \fBa[b/c]d\fR.
252 .RE
253 .RS +4
254 .TP
255 2.
256 If a filename begins with a period (.), the period must be explicitly
257 matched by using a period as the first character of the pattern or immediately
258 following a slash character. The leading period will not be matched by:
259 .sp
260 \(bu the asterisk or question-mark special characters
261 .sp
262 \(bu a bracket expression containing a non-matching list, such as:
263 .sp
264 \fB[!a]\fR
265 .sp
266 a range expression, such as:
267 .sp
268 \fB[%\(mi0]\fR
269 .sp
270 or a character class expression, such as:
271 .sp
272 \fB[[:punct:]]\fR
273 .sp
274 It is unspecified whether an explicit period in a bracket expression matching
275 list, such as:
276 .sp
277 \fB[.abc]\fR
278 .sp
279 can match a leading period in a filename.
280 .RE
281 .RS +4
282 .TP
283 3.
284 Specified patterns are matched against existing filenames and pathnames,  as
285 appropriate.  Each component that contains a pattern character requires read
286 permission in the directory containing that component. Any component, except
287 the last, that does not contain a pattern character requires search permission.
288 For example, given the pattern:
289 .sp
290 \fB/foo/bar/x*/bam\fR
291 .sp
292 search permission is needed for directories \fB/\fR and \fBfoo\fR, search and
293 read permissions are needed for directory \fBbar\fR, and search permission is
294 needed for each \fBx*\fR directory.
295 .sp
296 If the pattern matches any existing filenames or pathnames, the pattern will be
297 replaced with those filenames and pathnames, sorted according to the collating
298 sequence in effect in the current locale. If the pattern contains an invalid
299 bracket expression or does not match any existing filenames or pathnames, the
300 pattern string is left unchanged.
301 .RE
302 .SH SEE ALSO
308 .sp
303 .LP
304 \fBfind\fR(1), \fBksh\fR(1), \fBfnmatch\fR(3C), \fBregex\fR(5)
```