NAME

fnmatch – file name pattern matching

DESCRIPTION

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 **regex**(5) manual page.

Patterns Matching a Single Character

The following *patterns matching a single character* match a single character: *ordinary characters*, *special pattern characters* and *pattern bracket expressions*. The pattern bracket expression will also match a single collating element.

An ordinary character is a pattern that matches itself. It can be any character in the supported character set except for NUL, 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.

When unquoted and outside a bracket expression, the following three characters will have special meaning in the specification of patterns:

? A question-mark is a pattern that will match any character.

* An asterisk is a pattern that will match multiple characters, as described in **Patterns Matching Multiple Characters**, below.

[The open bracket will introduce a pattern bracket expression.

The description of basic regular expression bracket expressions on the **regex**(5) manual page also applies to the pattern bracket expression, except that the exclamation-mark character **(!)** replaces the circumflex character (^) in its role in a *non-matching list* in the regular expression notation. A bracket expression starting with an unquoted circumflex character produces unspecified results.

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 **[\^!]** to match either character.

When pattern matching is used where shell quote removal is not performed (such as in the argument to the **find** –**name** primary when **find** is being called using one of the **exec** functions, or in the *pattern* argument to the **fnmatch**(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 \\ 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.

Both quoting and escaping are described here because pattern matching must work in three separate circumstances:

- Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All of the following will match the string or file **abc**:

abc **"abc"** **a"b"c** **a\bc** **a[b]c**
a["b"]c **a[\b]c** **a["\b"]c** **a?c** **a*c**

The following will not:

"a?c" **a*c** **a\[b]c**

- Calling a utility or function without going through a shell, as described for **find**(1) and the function **fnmatch**(3C)**.**
- Calling utilities such as **find**, **cpio**, **tar** or **pax** through the shell command line. In this case, shell quote removal is performed before the utility sees the argument. For example, in:

  find /bin -name e\c[\h]o -print

  after quote removal, the backslashes are presented to **find** and it treats them as escape characters. Both precede ordinary characters, so the **c** and **h** represent themselves and **echo** would be found on many historical systems (that have it in **/bin**). To find a file name that contained shell special characters or pattern characters, both quoting and escaping are required, such as:

  pax -r . . . "*a\ (\?"

  to extract a filename ending with **a(?**.

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 **ksh**(1).

Patterns Matching Multiple Characters

The following rules are used to construct *patterns matching multiple characters* from *patterns matching a single character*:

- The asterisk (*) is a pattern that will match any string, including the null string.
- The concatenation of *patterns matching a single character* is a valid pattern that will match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.
- The concatenation of one or more *patterns matching a single character* 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.

Since each asterisk matches zero or more occurrences, the patterns **a*b** and **a**b** have identical functionality.

Examples:

a[bc] matches the strings **ab** and **ac**.

a*d matches the strings **ad**, **abd** and **abcd**, but not the string **abc**.

a*d* matches the strings **ad**, **abcd**, **abcdef**, **aaaad** and **adddd**.

***a*d** matches the strings **ad**, **abcd**, **efabcd**, **aaaad** and **adddd**.

Patterns Used for Filename Expansion

The rules described so far in **Patterns Matching Multiple Characters** and **Patterns Matching a Single Character** are qualified by the following rules that apply when pattern matching notation is used for filename expansion.

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 **a[b/c]d** will not match such pathnames as **abd** or **a/d**. It will only match a pathname of literally **a[b/c]d**.
2. If a filename begins with a period (.), the period must be explicitly matched by using a period as the first character of the pattern or immediately following a slash character. The leading period will not be matched by:
   - the asterisk or question-mark special characters
   - a bracket expression containing a non-matching list, such as :

     [!a]

     a range expression, such as:

     [%−0]

     or a character class expression, such as:

     [[:punct:]]

   It is unspecified whether an explicit period in a bracket expression matching list, such as:

   [.abc]

   can match a leading period in a filename.
3. Specified patterns are matched against existing filenames and pathnames, as appropriate. Each component that contains a pattern character requires read permission in the directory containing that component. Any component, except the last, that does not contain a pattern character requires search permission. For example, given the pattern:

   /foo/bar/x*/bam

   search permission is needed for directories **/** and **foo**, search and read permissions are needed for directory **bar**, and search permission is needed for each **x*** directory. If the pattern matches any existing filenames or pathnames, the pattern will be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale. If the pattern contains an invalid bracket expression or does not match any existing filenames or pathnames, the pattern string is left unchanged.

SEE ALSO

find(1), **ksh**(1), **fnmatch**(3C), **regex**(5)