NAME

regex – internationalized basic and extended regular expression matching

DESCRIPTION

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings. The Internationalized Regular Expressions described below differ from the Simple Regular Expressions described on the **regexp**(5) manual page in the following ways:

- both Basic and Extended Regular Expressions are supported
- the Internationalization features—character class, equivalence class, and multi-character collation—are supported.

The Basic Regular Expression (BRE) notation and construction rules described in the **BASIC REGULAR EXPRESSIONS** section apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in the **EXTENDED REGULAR EXPRESSIONS** section; any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interfaces **regcomp**(3C) and **regexec**(3C).

BASIC REGULAR EXPRESSIONS

BREs Matching a Single Character

A BRE ordinary character, a special character preceded by a backslash, or a period matches a single character. A bracket expression matches a single character or a single collating element. See **RE Bracket Expression**, below.

BRE Ordinary Characters

An ordinary character is a BRE that matches itself: any character in the supported character set, except for the BRE special characters listed in **BRE Special Characters**, below.

The interpretation of an ordinary character preceded by a backslash (\) is undefined, except for:

1. the characters), (, {, and }
2. the digits 1 to 9 inclusive (see **BREs Matching Multiple Characters**, below)
3. a character inside a bracket expression.

BRE Special Characters

A *BRE special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character will be a BRE that matches the special character itself. The BRE special characters and the contexts in which they have their special meaning are:

. [\ The combination of period, left-bracket and backslash is special except when used in a bracket expression (see **RE Bracket Expression**, below). An expression containing a [that is not preceded by a backslash and is not part of a bracket expression produces undefined results.

* The asterisk is special except when used:

- in a bracket expression
- as the first character of an entire BRE (after an initial ^, if any)
- as the first character of a subexpression (after an initial ^, if any); see **BREs Matching Multiple Characters**, below.

^ The circumflex is special when used:

- as an anchor (see **BRE Expression Anchoring**, below).
- as the first character of a bracket expression (see **RE Bracket Expression**, below).

$ The dollar sign is special when used as an anchor.

Periods in BREs

A period (**.**), when used outside a bracket expression, is a BRE that matches any character in the supported character set except NUL.

RE Bracket Expression

A bracket expression (an expression enclosed in square brackets, [ ]) is an RE that matches a single collating element contained in the non-empty set of collating elements represented by the bracket expression.

The following rules and definitions apply to bracket expressions:

1. A *bracket expression* is either a matching list expression or a non-matching list expression. It consists of one or more expressions: collating elements, collating symbols, equivalence classes, character classes, or range expressions (see rule 7 below). Portable applications must not use range expressions, even though all implementations support them. The right-bracket (]) loses its special meaning and represents itself in a bracket expression if it occurs first in the list (after an initial circumflex (^), if any). Otherwise, it terminates the bracket expression, unless it appears in a collating symbol (such as [.].]) or is the ending right-bracket for a collating symbol, equivalence class, or character class. The special characters:

   . * [\

   (period, asterisk, left-bracket and backslash, respectively) lose their special meaning within a bracket expression.

   The character sequences:

   [. [= [:

   (left-bracket followed by a period, equals-sign, or colon) are special inside a bracket expression and are used to delimit collating symbols, equivalence class expressions, and character class expressions. These symbols must be followed by a valid expression and the matching terminating sequence .], =] or :], as described in the following items.

2. A *matching list* expression specifies a list that matches any one of the expressions represented in the list. The first character in the list must not be the circumflex. For example, **[abc]** is an RE that matches any of the characters **a**, **b** or **c**.

3. A *non-matching list* expression begins with a circumflex (ˆ), and specifies a list that matches any character or collating element except for the expressions represented in the list after the leading circumflex. For example, [ˆabc] is an RE that matches any character or collating element except the characters **a, b** or **c**. The circumflex will have this special meaning only when it occurs first in the list, immediately following the left-bracket.

4. A *collating symbol* is a collating element enclosed within bracket-period ([. .]) delimiters. Multi-character collating elements must be represented as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multi-character collating element. For example, if the string **ch** is a collating element in the current collation sequence with the associated collating symbol <ch>, the expression [[.ch.]] will be treated as an RE matching the character sequence **ch,** while [ch] will be treated as an RE matching **c** or **h.** Collating symbols will be recognized only inside bracket expressions. This implies that the RE **[[.ch.]]*c** matches the first to fifth character in the string chchch. If the string is not a collating element in the current collating sequence definition, or if the collating element has no characters associated with it, the symbol will be treated as an invalid expression.

5. An *equivalence class expression* represents the set of collating elements belonging to an equivalence class. Only primary equivalence classes will be recognised. The class is expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal ([= =]) delimiters. For example, if a, à and â belong to the same equivalence class, then [[=a=]b], [[=à=]b] and [[=â=]b] will each be equivalent to [aàâb]. If the collating element does not belong to an equivalence class, the equivalence class expression will be treated as a *collating symbol* .

6. A *character class expression* represents the set of characters belonging to a character class, as defined in the **LC_CTYPE** category in the current locale. All character classes specified in the current locale will be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon ([: :]) delimiters.

   The following character class expressions are supported in all locales:

[:alnum:]	**[:cntrl:]**	**[:lower:]**	**[:space:]**
[:alpha:]	**[:digit:]**	**[:print:]**	**[:upper:]**
[:blank:]	**[:graph:]**	**[:punct:]**	**[:xdigit:]**

   In addition, character class expressions of the form:

   [:name:]

   are recognized in those locales where the *name* keyword has been given a **charclass** definition in the **LC_CTYPE** category.

7. A *range expression* represents the set of collating elements that fall between two elements in the current collation sequence, inclusively. It is expressed as the starting point and the ending point separated by a hyphen (–).

Range expressions must not be used in portable applications because their behavior is dependent on the collating sequence. Ranges will be treated according to the current collating sequence, and include such characters that fall within the range based on that collating sequence, regardless of character values. This, however, means that the interpretation will differ depending on collating sequence. If, for instance, one collating sequence defines ä as a variant of **a,** while another defines it as a letter following **z,** then the expression [ä–z] is valid in the first language and invalid in the second.

In the following, all examples assume the collation sequence specified for the POSIX locale, unless another collation sequence is specifically defined.

The starting range point and the ending range point must be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. For example, the unspecified expression [[=e=]–f] should be given as [[=e=]e–f]. The ending range point must collate equal to or higher than the starting range point; otherwise, the expression will be treated as invalid. The order used is the order in which the collating elements are specified in the current collation definition. One-to-many mappings (see **locale**(5)) will not be performed. For example, assuming that the character eszet (β) is placed in the collation sequence after **r** and **s,** but before **t,** and that it maps to the sequence **ss** for collation purposes, then the expression [r–s] matches only **r** and **s,** but the expression [s–t] matches **s,** β or **t.**

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for instance [a–m–o]) is undefined.

The hyphen character will be treated as itself if it occurs first (after an initial ^, if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions [–ac] and [ac–] are equivalent and match any of the characters **a, c,** or –**;** [^–ac] and [^ac–] are equivalent and match any characters except **a, c,** or –; the expression [%– –] matches any of the characters between % and – inclusive; the expression [– –@] matches any of the characters between – and @ inclusive; and the expression [a– –@] is invalid, because the letter **a** follows the symbol – in the POSIX locale. To use a hyphen as the starting range point, it must either come first in the bracket expression or be specified as a collating symbol, for example: [][.–.]–0], which matches either a right bracket or any character or collating element that collates between hyphen and 0, inclusive.

If a bracket expression must specify both – and], the] must be placed first (after the ^, if any) and the – last within the bracket expression.

BREs Matching Multiple Characters

The following rules can be used to construct BREs matching multiple characters from BREs matching a single character:

1. The concatenation of BREs matches the concatenation of the strings matched by

each component of the BRE.

2. A *subexpression* can be defined within a BRE by enclosing it between the character pairs \(and \) . Such a subexpression matches whatever it would have matched without the \(and \), except that anchoring within subexpressions is optional behavior; see **BRE Expression Anchoring**, below. Subexpressions can be arbitrarily nested.

3. The *back-reference* expression *n* matches the same (possibly empty) string of characters as was matched by a subexpression enclosed between \(and \) preceding the *n*. The character *n* must be a digit from 1 to 9 inclusive, *n*th subexpression (the one that begins with the *n*th \(and ends with the corresponding paired \)). The expression is invalid if less than *n* subexpressions precede the *n*. For example, the expression ^\(.*\)\1$ matches a line consisting of two adjacent appearances of the same string, and the expression \(a\)*\1 fails to match **a.** The limit of nine back-references to subexpressions in the RE is based on the use of a single digit identifier. This does not imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten subexpressions:

   \(\(\(ab\)*c\)*d\)\(ef\)*\(gh\)\{2\}\(ij\)*\(kl\)*\(mn\)*\(op\)*\(qr\)*

4. When a BRE matching a single character, a subexpression or a back-reference is followed by the special character asterisk (*), together with that asterisk it matches what zero or more consecutive occurrences of the BRE would match. For example, **[ab]* and [ab][ab]** are equivalent when matching the string **ab.**

5. When a BRE matching a single character, a subexpression, or a back-reference is followed by an *interval expression* of the format \{*m*\}, \{*m*,\} or \{*m*,*n*\}, together with that interval expression it matches what repeated consecutive occurrences of the BRE would match. The values of *m* and *n* will be decimal integers in the range $0 \le m \le n \le$ {**RE_DUP_MAX**}, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression \{*m*\} matches exactly *m* occurrences of the preceding BRE, \{*m*,\} matches at least *m* occurrences and \{*m*,*n*\} matches any number of occurrences between *m* and *n*, inclusive.

   For example, in the string **abababccccccd,** the BRE c\{3\} is matched by characters seven to nine, the BRE \(ab\)\{4,\} is not matched at all and the BRE c\{1,3\}d is matched by characters ten to thirteen.

The behavior of multiple adjacent duplication symbols **(** * and intervals) produces undefined results.

BRE Precedence

The order of precedence is as shown in the following table:

BRE Precedence (from high to low)	
collation-related bracket symbols	[= =] [: :] [. .]
escaped characters	\\<*special character*>
bracket expression	[ ]
subexpressions/back-references	\\(\\) *n*
single-character-BRE duplication	* \\{*m*,*n*\\}
concatenation	
anchoring	^ $

BRE Expression Anchoring

A BRE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters will be considered BRE anchors in the following contexts:

1. A circumflex **(^)** is an anchor when used as the first character of an entire BRE. The implementation may treat circumflex as an anchor when used as the first character of a subexpression. The circumflex will anchor the expression to the beginning of a string; only sequences starting at the first character of a string will be matched by the BRE. For example, the BRE ^ab matches **ab** in the string **abcdef,** but fails to match in the string **cdefab.** A portable BRE must escape a leading circumflex in a subexpression to match a literal circumflex.
2. A dollar sign **($)** is an anchor when used as the last character of an entire BRE. The implementation may treat a dollar sign as an anchor when used as the last character of a subexpression. The dollar sign will anchor the expression to the end of the string being matched; the dollar sign can be said to match the end-of-string following the last character.
3. A BRE anchored by both ^ and **$** matches only an entire string. For example, the BRE ^**abcdef$** matches strings consisting only of **abcdef.**
4. ^ and **$** are not special in subexpressions.

EXTENDED REGULAR EXPRESSIONS

The rules specifed for BREs apply to Extended Regular Expressions (EREs) with the following exceptions:

- The characters **|**, **+**, and **?** have special meaning, as defined below
- The subexpression **( )** and duplication **{ }** operators need not be preceded by a backslash (\\)
- The back reference operator is not supported.
- Anchoring (^**$**) is supported in subexpressions.

EREs Matching a Single Character

An ERE ordinary character, a special character preceded by a backslash, or a period matches a single character. A bracket expression matches a single character or a single collating element. An *ERE matching a single character* enclosed in parentheses matches the same as the ERE without parentheses would have matched.

ERE Ordinary Characters

An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in **ERE Special Characters** below. The interpretation of an ordinary character preceded by a backslash (\) is undefined.

ERE Special Characters

An *ERE special character* has special properties in certain contexts. Outside those contexts, or when preceded by a backslash, such a character is an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they have their special meaning are:

. [\ (The period, left-bracket, backslash and left-parenthesis are special except when used in a bracket expression (see **RE Bracket Expression**, above). Outside a bracket expression, a left-parenthesis immediately followed by a right-parenthesis produces undefined results.

) The right-parenthesis is special when matched with a preceding left-parenthesis, both outside a bracket expression.

* + ? { The asterisk, plus-sign, question-mark and left-brace are special except when used in a bracket expression (see **RE Bracket Expression**, above). Any of the following uses produce undefined results:

- if these characters appear first in an ERE, or immediately following a vertical-line, circumflex or left-parenthesis
- if a left-brace is not part of a valid interval expression.

| The vertical-line is special except when used in a bracket expression (see **RE Bracket Expression**, above). A vertical-line appearing first or last in an ERE, or immediately following a vertical-line or a left-parenthesis, or immediately preceding a right-parenthesis, produces undefined results.

^ The circumflex is special when used:

- as an anchor (see **ERE Expression Anchoring**, below).
- as the first character of a bracket expression (see **RE Bracket Expression**, above).

$ The dollar sign is special when used as an anchor.

Periods in EREs

A period (**.**), when used outside a bracket expression, is an ERE that matches any character in the supported character set except NUL.

ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see **RE Bracket Expression**, above).

EREs Matching Multiple Characters

The following rules will be used to construct EREs matching multiple characters from EREs matching a single character:

1. A *concatenation of EREs* matches the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses matches whatever the concatenation without the parentheses matches. For example, both the ERE **cd** and the ERE **(cd)** are matched by the third and

fourth character of the string **abcdefabcdef.**

2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character plus-sign (+), together with that plus-sign it matches what one or more consecutive occurrences of the ERE would match. For example, the ERE **b+(bc)** matches the fourth to seventh characters in the string **acabbbcde; [ab]** + and **[ab][ab]**∗ are equivalent.
3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character asterisk (∗), together with that asterisk it matches what zero or more consecutive occurrences of the ERE would match. For example, the ERE **b**∗**c** matches the first character in the string **cabbbcde,** and the ERE **b**∗**cd** matches the third to seventh characters in the string **cabbbcdebbbbbbcdbc.** And, **[ab]**∗ and **[ab][ab]** are equivalent when matching the string **ab.**
4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character question-mark (**?**), together with that question-mark it matches what zero or one consecutive occurrences of the ERE would match. For example, the ERE **b?c** matches the second character in the string **acabbbcde.**
5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by an *interval expression* of the format {*m*}, {*m*,} or {*m*,*n*}, together with that interval expression it matches what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* will be decimal integers in the range $0 \le m \le n \le$ **{RE_DUP_MAX}**, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression {*m*} matches exactly *m* occurrences of the preceding ERE, {*m*,} matches at least *m* occurrences and {*m*,*n*} matches any number of occurrences between *m* and *n*, inclusive.

For example, in the string **abababccccccd** the ERE c{3} is matched by characters seven to nine and the ERE (ab){2,} is matched by characters one to six.

The behavior of multiple adjacent duplication symbols **(+,** ∗, **?** and intervals) produces undefined results.

ERE Alternation

Two EREs separated by the special character vertical-line (|) match a string that is matched by either. For example, the ERE a((bc) | d) matches the string abc and the string ad. Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, will be treated as an ERE matching a single character.

ERE Precedence

The order of precedence will be as shown in the following table:

ERE Precedence (from high to low)	
collation-related bracket symbols	[= =] [: :] [. .]
escaped characters	\<*special character*>
bracket expression	[ ]
grouping	( )
single-character-ERE duplication	* + ? {*m*,*n*}
concatenation	
anchoring	^ $
alternation	|

For example, the ERE **abba | cde** matches either the string **abba** or the string **cde** (rather than the string **abbade** or **abbcde**, because concatenation has a higher order of precedence than alternation).

ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called *anchoring*. The circumflex and dollar sign special characters are considered ERE anchors when used anywhere outside a bracket expression. This has the following effects:

1. A circumflex (^) outside a bracket expression anchors the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs ^ab and (^ab) match ab in the string abcdef, but fail to match in the string cdefab, and the ERE a^b is valid, but can never match because the **a** prevents the expression ^b from matching starting at the first character.
2. A dollar sign **($)** outside a bracket expression anchors the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs ef$ and (ef$) match ef in the string abcdef, but fail to match in the string cdefab, and the ERE e$f is valid, but can never match because the **f** prevents the expression e$ from matching ending at the last character.

SEE ALSO

localedef(1), **regcomp**(3C), **environ**(5), **locale**(5), **regexp**(5)