

# Grep Regular Expressions - IEEE Std 1003.2 (POSIX.2) - Modern/extended RE

Last updated on:	2026-01-30																			
<b>Grep Regexp Syntax</b>	Ref: <a href="#">The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)</a>																			
<b>Color codes:</b>	<b>Special characters:</b> \ ^ . [ \$ ( )   * + ? {  To match any of these as ordinary character, it must be escaped.																			
<b>Notes</b>	<b>Atoms:</b> (greedy := the longest valid match)																			
	<table border="1"> <tr> <td>( )</td><td>Match the null string.</td> <td rowspan="4" style="vertical-align: middle;"></td><td rowspan="4" style="vertical-align: middle;"><b>{n}</b></td><td rowspan="4" style="vertical-align: middle;"><b>n repetitions.</b> • For example, <code>x{4}</code> matches the string xxxx and nothing else.</td></tr> <tr> <td>?</td><td>0 or 1 of the previous expression - greedy.</td><td></td></tr> <tr> <td>*</td><td>0 or more of the previous expression - greedy.</td><td></td></tr> <tr> <td>+</td><td>1 or more of the previous expression - greedy.</td><td></td></tr> </table>				( )	Match the null string.		<b>{n}</b>	<b>n repetitions.</b> • For example, <code>x{4}</code> matches the string xxxx and nothing else.	?	0 or 1 of the previous expression - greedy.		*	0 or more of the previous expression - greedy.		+	1 or more of the previous expression - greedy.			
( )	Match the null string.		<b>{n}</b>	<b>n repetitions.</b> • For example, <code>x{4}</code> matches the string xxxx and nothing else.																
?	0 or 1 of the previous expression - greedy.																			
*	0 or more of the previous expression - greedy.																			
+	1 or more of the previous expression - greedy.																			
	<b>Repetition postfix operators:</b>																			
	<table border="1"> <tr> <td>{n,m}</td><td>Between <b>n</b> and <b>m</b> repetitions: must match at least <b>n</b> times but no more than <b>m</b> times. If <b>m</b> is present it must be &gt;<b>n</b> and &lt;=255. If <b>m</b> is omitted there is no upper limit.</td><td></td></tr> </table>				{n,m}	Between <b>n</b> and <b>m</b> repetitions: must match at least <b>n</b> times but no more than <b>m</b> times. If <b>m</b> is present it must be > <b>n</b> and <=255. If <b>m</b> is omitted there is no upper limit.														
{n,m}	Between <b>n</b> and <b>m</b> repetitions: must match at least <b>n</b> times but no more than <b>m</b> times. If <b>m</b> is present it must be > <b>n</b> and <=255. If <b>m</b> is omitted there is no upper limit.																			
	<b>Boundary/anchors:</b>																			
	<table border="1"> <tr> <td>^</td><td>Matches the null string at <b>beginning of a line</b>.</td> <td>\$</td><td>Matches the null string at the <b>end of a line</b>.</td></tr> <tr> <td>\&lt;</td><td>Matches the null string at the beginning of a <b>word</b> (a set of alnum characters and underscores).</td> <td>\&gt;</td><td>Matches the null string at the end of a <b>word</b> (a set of alnum characters and underscores).</td></tr> <tr> <td>[[:&lt;:]]</td><td></td> <td>[[:&gt;:]]</td><td></td></tr> <tr> <td>\b</td><td>Matches the null string at a word boundary (either the beginning or end of a word).</td> <td>\B</td><td>Matches the null string where there is no word boundary. • This is the opposite of '\b'.</td></tr> </table>				^	Matches the null string at <b>beginning of a line</b> .	\$	Matches the null string at the <b>end of a line</b> .	\<	Matches the null string at the beginning of a <b>word</b> (a set of alnum characters and underscores).	\>	Matches the null string at the end of a <b>word</b> (a set of alnum characters and underscores).	[[:<:]]		[[:>:]]		\b	Matches the null string at a word boundary (either the beginning or end of a word).	\B	Matches the null string where there is no word boundary. • This is the opposite of '\b'.
^	Matches the null string at <b>beginning of a line</b> .	\$	Matches the null string at the <b>end of a line</b> .																	
\<	Matches the null string at the beginning of a <b>word</b> (a set of alnum characters and underscores).	\>	Matches the null string at the end of a <b>word</b> (a set of alnum characters and underscores).																	
[[:<:]]		[[:>:]]																		
\b	Matches the null string at a word boundary (either the beginning or end of a word).	\B	Matches the null string where there is no word boundary. • This is the opposite of '\b'.																	
	<b>Escaping:</b> Note that "-quoted strings are first processed by the shell, performing escape processing. This describes what is seen by grep.																			
	<ul style="list-style-type: none"> <li>Therefore, inside a double-quoted string, backslash characters must be doubled. "\\\(" represents the single open parenthesis.</li> <li>\ : followed by any of '^.[\${}] *+?{' matches that character taken as an ordinary character.</li> <li>\ : followed by any other character is undefined.</li> <li>\ : is invalid at the end of a regular expression.</li> </ul>																			
	<b>Literal sequences:</b>																			
	<table border="1"> <tr> <td>\a</td><td>The "bell" character (ASCII code 7).</td> <td>\n</td><td>The "new-line/line-feed" character (ASCII code 10).</td></tr> <tr> <td>\e</td><td>The "escape" character (ASCII code 27).</td> <td>\r</td><td>The "carriage-return" character (ASCII code 13).</td></tr> <tr> <td>\f</td><td>The "form-feed" character (ASCII code 12).</td> <td>\t</td><td>The "horizontal-tab" character (ASCII code 9).</td></tr> <tr> <td>\xx..</td><td>Arbitrary 8-bit value with zero, one or two hexadecimal digits.</td> <td>\x{x..}</td><td>An arbitrary, up to to 32-bit value. The x.. sequence is using as many hexadecimal digits necessary to represent the value.</td></tr> </table>				\a	The "bell" character (ASCII code 7).	\n	The "new-line/line-feed" character (ASCII code 10).	\e	The "escape" character (ASCII code 27).	\r	The "carriage-return" character (ASCII code 13).	\f	The "form-feed" character (ASCII code 12).	\t	The "horizontal-tab" character (ASCII code 9).	\xx..	Arbitrary 8-bit value with zero, one or two hexadecimal digits.	\x{x..}	An arbitrary, up to to 32-bit value. The x.. sequence is using as many hexadecimal digits necessary to represent the value.
\a	The "bell" character (ASCII code 7).	\n	The "new-line/line-feed" character (ASCII code 10).																	
\e	The "escape" character (ASCII code 27).	\r	The "carriage-return" character (ASCII code 13).																	
\f	The "form-feed" character (ASCII code 12).	\t	The "horizontal-tab" character (ASCII code 9).																	
\xx..	Arbitrary 8-bit value with zero, one or two hexadecimal digits.	\x{x..}	An arbitrary, up to to 32-bit value. The x.. sequence is using as many hexadecimal digits necessary to represent the value.																	
	<b>Shortcuts:</b>																			
	<table border="1"> <tr> <td>\d</td><td>Matches a digit character. Equivalent to [[:digit:]]</td> <td>\D</td><td>Matches a non-digit character. Equivalent to [^[:digit:]]</td></tr> <tr> <td>\s</td><td>Matches a space character. Equivalent to [[:space:]]</td> <td>\S</td><td>Matches a non-space character. Equivalent to [^[:space:]]</td></tr> <tr> <td>\w</td><td>Matches a word character. Equivalent to [[:alnum:_]]</td> <td>\W</td><td>Matches a non-word character. Equivalent to [^[:alnum:_]]</td></tr> </table>				\d	Matches a digit character. Equivalent to [[:digit:]]	\D	Matches a non-digit character. Equivalent to [^[:digit:]]	\s	Matches a space character. Equivalent to [[:space:]]	\S	Matches a non-space character. Equivalent to [^[:space:]]	\w	Matches a word character. Equivalent to [[:alnum:_]]	\W	Matches a non-word character. Equivalent to [^[:alnum:_]]				
\d	Matches a digit character. Equivalent to [[:digit:]]	\D	Matches a non-digit character. Equivalent to [^[:digit:]]																	
\s	Matches a space character. Equivalent to [[:space:]]	\S	Matches a non-space character. Equivalent to [^[:space:]]																	
\w	Matches a word character. Equivalent to [[:alnum:_]]	\W	Matches a non-word character. Equivalent to [^[:alnum:_]]																	
	<b>Grouping with:</b> (...)																			
	Grouping is supported with non-escaped parentheses: ( )																			
	Back-references																			
	Back-references to previously defined groups, starting with \1 for the first group. As described in re-format(7) Man page, the implementation in grep is <b>not reliable and should be avoided</b> .																			
	<b>Alternate with:</b>																			
	For example: No backslash is required for expressing an alternate with																			
	• "syserr syslog" matches either "syserr" or "syslog"																			
	<b>Bracket expressions:</b> []																			
	<ul style="list-style-type: none"> <li><b>Inside the [] brackets we can place:</b> <ul style="list-style-type: none"> <li>A character range: c<sup>1</sup>-c<sup>2</sup> where c<sup>1</sup> is the first character in the range and c<sup>2</sup> is the last, inclusive one. Example: [a-z] matches all lowercase characters (on case sensitive search). Ranges are very collating-sequence-dependent and therefore not portable. Avoid them in programs</li> <li>^ : complements the set (ie: means that we want to match anything but what is in the set. For example: [!alpha:] fact everything except any letter.</li> <li>[ : To include the literal [ in the list, make it the first character (following a possible ^).</li> <li>] : To include the literal ] in the list, make it the first character (following a possible ^).</li> <li>- : To include the literal - in the list, make it the first or last character (following a possible ^) or the second endpoint of a range.</li> <li>- : To include the literal - in the list, and using it as the first endpoint of a range, include it in a [. .] range: as in: [.-.]-0]</li> <li>Collating element: one or a sequence of characters inside the [. .] brackets define a collating element: character or symbol to be treated as a single unit, rather than a range operator. Example [[.ch.]] can be used in a Czech locale where "ch" is treated as a single letter that sorts between 'h' and 'i'.</li> <li>[:C:] : character class C (as defined by the ctype(3) or wctype(3) man page), where C can be any of the following (eg. [[:alnum:]]): <ul style="list-style-type: none"> <li>alnum : any letter or digit</li> <li>alpha : any letter</li> <li>blank : horizontal whitespace: a space or tab character</li> <li>cntrl : any ASCII control character</li> <li>digit : any digit character, same as [0-9]. [-[:digit:]] matches any digit as well as '+' and '-'.</li> <li>graph : any graphic character; everything except whitespace, ASCII and non-ASCII control characters, surrogates and code points unassigned by Unicode.</li> <li>lower : lower-case letters.</li> <li>print : matches any printing character, either whitespace, or graphic character matched by [:graph:].</li> <li>punct : any punctuation character. For multibyte character matches anything that has non-word syntax.</li> <li>space : any whitespace character.</li> <li>upper : any upper-case letter.</li> <li>xdigit : the hexadecimal digits: '0' through '9', 'a' through 'f' and 'A' through 'F'.</li> </ul> </li> <li>All other special characters, including \, loose their special significance within a bracket expression.</li> </ul> </li> </ul>																			
<b>Examples</b>	The following are recursive grep using extended regular expression search in all .c files in the current directory tree.																			
To identify lines that have an opening [ without the closing ] in .c files	<pre>grep -REn --include \*.c "[[]]"   grep -Ev "[]]"</pre> <pre>egrep -Rn --include \*.c "[[]]"   grep -Ev "[]]"</pre>																			
	<ul style="list-style-type: none"> <li>Extended regular expressions are used by egrep and by grep when the -E option is specified.</li> <li>To perform a logic AND, each portion must be done by a seepage search; the result of the first search is filtered by the second search.</li> </ul>																			
To match lines that end with at least two consecutive hyphen-minus (0x2d) characters ('-').	<pre>grep -REn --include \*.c "^.*[-][-]+\$"</pre> <pre>grep -REn --include \*.c "^.*--+\$"</pre> <pre>grep -REn --include \*.c "[-][-]"</pre>																			
	<p>The [-] matches a single hyphen character.</p> <p>If the 2 hypends are not alone they can be placed inside the regexp.</p> <p>When only searching for 2 consecutive hyphen character, the special syntax must be used because otherwise it could be identified by the shell as the separator that identifies the end of options.</p>																			
Match lines that end with trailing spaces	<pre>grep -REn --include \*.c "^[[:blank:]]+[:blank:]]+\$"</pre>																			
	Match lines that have nn whitespace followed by whitespace characters. It does not match lines with only whitespaces.																			
Identify files that have hard tabs	<pre>grep -REn --include \*.c '\t'</pre> <pre>grep -REn --include \*.c "^\*\*\t\*"</pre> <pre>grep -REn "^\*\*\t\*"</pre>																			
	<p>When single quotes are used there's no need to escape the \</p> <p>With double quotes the backslash must be escaped.</p> <p>This example searches for all files in the directory tree.</p>																			

Last updated on:	2026-01-30	
Match lines that end with a dollar character.	<pre>grep --REn --include \*.c "[\\$]\$"</pre> <pre>grep --REn --include \*.c '\\$\\$'</pre>	Use [\$] to represent the dollar sign character.
Searching for repeated word using back reference.  ⚠	<pre>grep --REn --include \*.c '\b(level) (for to).+\b\1'</pre> <pre>grep --REn --include \*.c "\b(level) (for to).+\b\1"</pre>	Use single quoting because using "\\$\$" fails with invalid backreference number. Back-references (\1) are working with grep, <b>but</b> they are not well documented, and identified as a botched/buggy and un-reliable implementation in grep Man page, stating that they should be avoided. <ul style="list-style-type: none"> <li>Some use of back reference work, but not all. So they should be avoided.</li> <li><b>ugrep</b> explicitly detects errors in those.</li> </ul>