

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

Grep Regexp Syntax Color codes: <ul style="list-style-type: none"> Standard ERE syntax. Should be available in all grep implementations. Extension. Available in GNU grep and BSD grep. Extension. Only available in BSD grep such as macOS grep. Extension only available on GNU grep. Notes <ul style="list-style-type: none"> POSIX regex engines (like POSIX.2) are fundamentally "greedy," meaning they always match the longest possible string. 	Ref: The Open Group Base Specifications Issue 7, 2018 edition IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)			
	Special characters: \ ^ . [\$ () * + ? {			
	Boundary/anchors:			
	^	Matches the null string at beginning of a line .	\$	Matches the null string at the end of a line .
	\<	Matches the null string at the beginning of a word (a set of alnum characters and underscores).	\>	Matches the null string at the end of a word (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'.
	Shortcuts:			
	\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:_]]
Atoms: (greedy := the longest valid match)	Repetition postfix operators:			
	()	Match the null string.	{n}	n repetitions. • For example, x\{4\} matches the string xxxx and nothing else.
	?	0 or 1 of the previous expression - greedy.		
	*	0 or more of the previous expression - greedy.	{n,m}	Between n and m repetitions: must match at least n times but no more than m times. If m is present it must be > n and <=255. If m is omitted there is no upper limit.
	+	1 or more of the previous expression - greedy.		
Literal sequences:				
\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.	
Escaping: Note that "-quoted strings are first processed by the shell, performing escape processing. This describes what is seen by grep.				
• \ : followed by any of '^ .[\$() *+?{' matches that character taken as an ordinary character.				
• \ : followed by any other character is undefined.				
• \ : is invalid at the end of a regular expression.				
Grouping with: (..)		Back-references !		
Grouping is supported with non-escaped parentheses: ()		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 not reliable and should be avoided .		
Alternate with:		No backslash is required for expressing an alternate with		
		For example: • "syserr syslog" matches either "syserr" or "syslog"		
Bracket expressions: []				
Inside the [] brackets we can place: <ul style="list-style-type: none"> A character range: c1–c2 where c1 is the first character in the range and c2 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 ^ : 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. [] : To include the literal [in the list, make it the first character (following a possible ^).] : To include the literal] in the list, make it the first character (following a possible ^). - : To include the literal - in the list, make it the first or last character (following a possible ^) or the second endpoint of a range. - : 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] 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'. [:C:] : character class C (as defined by the ctype(3) man page), where C can be any of the following (eg. [[:alnum:]]): <ul style="list-style-type: none"> alnum : any letter or digit alpha : any letter blank : horizontal whitespace: a space or tab character ctrl : any ASCII control character digit : any digit character, same as [0-9]. [-[:digit:]] matches any digit as well as '+' and '-'. graph : any graphic character; everything except whitespace, ASCII and non-ASCII control characters, surrogates and code points unassigned by Unicode. lower : lower-case letters. print : matches any printing character, either whitespace, or graphic character matched by [:graph:] punct : any punctuation character. For multibyte character matches anything that has non-word syntax. space : any whitespace character. upper : any upper-case letter. xdigit : the hexadecimal digits: '0' through '9', 'a' through 'f' and 'A' through 'F'. All other special characters, including \, loose their special significance within a bracket expression. 				
Examples				
To identify lines that have an opening [without the closing] in .c files	The following are recursive grep using extended regular expression search in all .c files in the current directory tree.			
	<pre>grep -REn --include *.c "[[]]" grep -Ev "[[]]"</pre> <pre>egrep -Rn --include *.c "[[]]" grep -Ev "[[]]"</pre>			
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>			
	The [-] matches a single hyphen character.			
	If the 2 hypends are not alone they can be placed inside the regexp.			
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>			
	When single quotes are used there's no need to escape the \			
	With double quotes the backslash must be escaped.			
Match lines that end with a dollar character				
<pre>grep -REn --include *.c "[\\$]"</pre>				Use [\$] to represent the dollar sign character.

Dollar character.		
Searching for repeated word using back reference. ⚠	<pre>grep --REn --include *.c '\\$'</pre> <pre>grep --REn --include *.c '\b(level) (for to).+\b\1'</pre> <pre>grep --REn --include *.c "\b(level) (for to).+\b\1"</pre>	<p>Use single quoting because using "\\$\$" fails with invalid backreference number.</p> <p>Back-references (\1) are working with grep, but 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.</p> <ul style="list-style-type: none"> Some use of back reference work, but not all. So they should be avoided. ugrep explicitly detects errors in those.