



CLion 2021.3 ▼

[Reference](#) / Regular expression syntax reference

Regular expression syntax reference

Last modified: 23 September 2021

This section is a brief summary of [regexp syntax](#) ↗ that can be used for creating [search](#) and [replace](#) as well as issue navigation patterns.

RegEx syntax reference [↗](#)

Character	Description
<code>\</code>	<p>Marks the next character as either a special character or a literal. For example:</p> <ul style="list-style-type: none"> <code>n</code> matches the character <code>n</code>. <code>"\n"</code> matches a newline character. The sequence <code>\\</code> matches <code>\</code> and <code>\(</code> matches <code>(</code>.
<code>^</code>	Matches the beginning of input.
<code>\$</code>	Matches the end of input.
<code>*</code>	Matches the preceding character zero or more times. For example, <code>"zo*"</code> matches either <code>z</code> or <code>zoo</code> .
<code>+</code>	Matches the preceding character one or more times. For example, <code>"zo+"</code> matches <code>zoo</code> but not <code>z</code> .
<code>?</code>	Matches the preceding character zero or one time. For example, <code>a?ve?</code> matches the <code>ve</code> in <code>never</code> .
<code>.</code>	Matches any single character except a newline character.
<code>(subexpression)</code>	<p>Matches <i>subexpression</i> and remembers the match. If a part of a regular expression is enclosed in parentheses, that part of the regular expression is grouped together. Thus a regex operator can be applied to the entire group.</p> <ul style="list-style-type: none"> If you need to use the matched substring within the same regular expression, you can retrieve it using the backreference <code>\num</code>, where <code>num = 1..n</code>. If you need to refer the matched substring somewhere outside the current regular expression (for example, in another regular expression as a replacement string), you can retrieve it using the dollar sign <code>\$num</code>, where <code>num = 1..n</code>. If you need to include the parentheses characters into a <i>subexpression</i>, use <code>\(</code> or <code>\)</code>.

Character	Description
<code>x y</code>	Matches either <i>x</i> or <i>y</i> . For example, <code>z wood</code> matches <i>z</i> or <i>wood</i> . <code>(z w)oo</code> matches <i>zoo</i> or <i>wood</i> .
<code>{ n }</code>	<i>n</i> is a non negative integer. Matches <i>exactly</i> <i>n</i> times. For example, <code>o{2}</code> does not match the <i>o</i> in <i>Bob</i> , but matches the first two <i>o</i> 's in <i>fooooood</i> .
<code>{ n , }</code>	<i>n</i> is a non negative integer. Matches <i>at least</i> <i>n</i> times. For example, <code>o{2,}</code> does not match the <i>o</i> in <i>Bob</i> and matches all the <i>o</i> 's in "fooooood." <code>o{1,}</code> is equivalent to <code>o+</code> . <code>o{0,}</code> is equivalent to <code>o*</code> .
<code>{ n , m }</code>	<i>m</i> and <i>n</i> are nonnegative integers. Matches <i>at least</i> <i>n</i> and <i>at most</i> <i>m</i> times. For example, <code>o{1,3}</code> matches the first three <i>o</i> 's in "fooooood." <code>o{0,1}</code> is equivalent to <code>o?</code> .
<code>[xyz]</code>	A character set. Matches any one of the enclosed characters. For example, <code>[abc]</code> matches the <i>a</i> in <i>plain</i> .
<code>[^ xyz]</code>	A negative character set. Matches any character not enclosed. For example, <code>[^abc]</code> matches the <i>p</i> in <i>plain</i> .
<code>[a-z]</code>	A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range <i>a</i> through <i>z</i> .
<code>[^ m-z]</code>	A negative range characters. Matches any character not in the specified range. For example, <code>[^m-z]</code> matches any character not in the range <i>m</i> through <i>z</i> .
<code>\b</code>	Matches a word boundary, that is, the position between a word and a space. For example, <code>er\b</code> matches the <i>er</i> in <i>never</i> but not the <i>er</i> in <i>verb</i> .
<code>\B</code>	Matches a non-word boundary. <code>ea*r\B</code> matches the <i>ear</i> in <i>never early</i> .
<code>\d</code>	Matches a digit character. Equivalent to <code>[0-9]</code> .


Character	Description
<code>\D</code>	Matches a non-digit character. Equivalent to <code>[^0-9]</code> .
<code>\f</code>	Matches a form-feed character.
<code>\n</code>	Matches a newline character.
<code>\r</code>	Matches a carriage return character.
<code>\s</code>	Matches any white space including space, tab, form-feed, and so on. Equivalent to <code>[\f\n\r\t\v]</code> .
<code>\S</code>	Matches any nonwhite space character. Equivalent to <code>[^\f\n\r\t\v]</code> .
<code>\t</code>	Matches a tab character.
<code>\v</code>	Matches a vertical tab character.
<code>\w</code>	Matches any word character including underscore. Equivalent to <code>[A-Za-z0-9_]</code> . Use it in the <i>search</i> field.
<code>\W</code>	Matches any non-word character. Equivalent to <code>[^A-Za-z0-9_]</code> .
<code>\ num</code>	Matches <i>num</i> , where <i>num</i> is a positive integer, denoting a reference back to remembered matches. For example, <code>(.)\1</code> matches two consecutive identical characters.
<code>\ n</code>	Matches <i>n</i> , where <i>n</i> is an octal escape value. Octal escape values should be 1, 2, or 3 digits long. For example, <code>\11</code> and <code>\011</code> both match a tab character. <code>\0011</code> is the equivalent of <code>\001</code> & 1. Octal escape values should not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.

Character	Description
<code>\x n</code>	<p>Matches <i>n</i>, where <i>n</i> is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long.</p> <p>For example, <code>\x41</code> matches <code>A</code>. <code>\x041</code> is equivalent to <code>\x04 & 1</code>.</p> <p>Allows ASCII codes to be used in regular expressions.</p>
<code>\\$</code>	Finds a <code>\$</code> character.
<code>\\\$</code>	This regex entered in the <i>search</i> field, means that you are trying to find a <code>\</code> character at the end of the line.
<code>\l</code>	Changes the case of the next character to the lower case. Use this type of regex in the <i>replace</i> field.
<code>\u</code>	Changes the case of the next character to the upper case. Use this type of regex in the <i>replace</i> field.
<code>\L</code>	Changes the case of all the subsequent characters up to <code>\E</code> to the lower case. Use this type of regex in the <i>replace</i> field.
<code>\U</code>	Changes the case of all the subsequent characters up to <code>\E</code> to the upper case. Use this type of regex in the <i>replace</i> field.
<code>(?!)</code>	This is a pattern for "negative lookahead". For example, <code>A(?!B)</code> means that CLion will search for <code>A</code> , but only if not followed by <code>B</code> .
<code>(?=)</code>	This is a pattern for "positive lookahead". For example, <code>A(=B)</code> means that CLion will search for <code>A</code> , but match if only followed by <code>B</code> .
<code>(?<=)</code>	This is a pattern for "positive lookbehind". For example, <code>(?<=B)A</code> means that CLion will search for <code>A</code> , but only if there is <code>B</code> before it.
<code>(?<!)</code>	This is a pattern for "negative lookbehind". For example, <code>(?<!B)A</code> means that CLion will search for <code>A</code> , but only if there is no <code>B</code> before it.

Since CLion supports all the standard regular expressions syntax, you can check <https://www.regular-expressions.info> ↗ for more information about the syntax.

Tips and Tricks

CLion provides intention actions to check validity of the regular expressions, and edit regular expressions in a scratchpad. Place the caret at a regular expression, and press **Alt+Enter**. The suggestion list of intention actions, available in this context, appears:

- Choose **Check RegExp**, and press **Enter**. The dialog that pops up, shows the current regular expression in the upper pane. In the lower pane, type the string to which this expression should match. If the regular expression matches the entered string, CLion displays a green check mark against the regex. If the regular expression doesn't match, then  is displayed.
- Choose **Edit RegExp Fragment**, and press **Enter**. The regular expression opens for editing in a separate tab in the editor. Note that this is only a scratchpad and no file is physically created:

As you type in the scratchpad, all changes are synchronized with the original regular expression. To close the scratchpad, press **.**