The Dot Matches (Almost) Any Character

In regular expressions, the dot or period is one of the most commonly used metacharacters. Unfortunately, it is also the most commonly misused metacharacter.

The dot matches a single character, without caring what that character is. The only exception are line break characters. In all regex flavors discussed in this tutorial, the dot does *not* match line breaks by default.

This exception exists mostly because of historic reasons. The first tools that used regular expressions were line-based. They would read a file line by line, and apply the regular expression separately to each line. The effect is that with these tools, the string could never contain line breaks, so the dot could never match them.

Modern tools and languages can apply regular expressions to very large strings or even entire files. Except for JavaScript and VBScript, all regex flavors discussed here have an option to make the dot match all characters, including line breaks.

In PowerGREP, tick the checkbox labeled “dot matches line breaks” to make the dot match all characters. In EditPad Pro, turn on the “Dot” or “Dot matches newline” search option.

In Perl, the mode where the dot also matches line breaks is called “single-line mode”. This is a bit unfortunate, because it is easy to mix up this term with “multi-line mode”. Multi-line mode only affects anchors, and single-line mode only affects the dot. You can activate single-line mode by adding an `s` after the regex code, like this: `m/^regex$/s;`.

Other languages and regex libraries have adopted Perl’s terminology. When using the regex classes of the .NET framework, you activate this mode by specifying `RegexOptions.Singleline`, such as in `Regex.Match("string", "regex", RegexOptions.Singleline)`.

JavaScript and VBScript do not have an option to make the dot match line break characters. In those languages, you can use a character class such as `[\s\S]` to match any character. This character matches a character that is either a whitespace character (including line break characters), or a character that is not a whitespace character. Since all characters are either whitespace or non-whitespace, this character class matches any character.

In all of Boost’s regex grammars the dot matches line breaks by default. Boost’s ECMAScript grammar allows you to turn this off with `regex_constants::no_mod_m`.

Line Break Characters

While support for the dot is universal among regex flavors, there are significant differences in which characters they treat as line break characters. All flavors treat the newline `\n` as a line break. UNIX text files terminate lines with a single newline. All the scripting languages discussed in this tutorial do not treat any other characters as line breaks. This isn’t a problem even on Windows where text files normally break lines with a `\r\n` pair. That’s because these scripting languages read and write files in *text mode* by default. When running on Windows, `\r\n` pairs are automatically converted into `\n` when a file is read, and `\n` is automatically written to file as `\r\n`.

std::regex, XML Schema and XPath also treat the carriage return `\r` as a line break character. JavaScript adds the Unicode line separator `\u2028` and paragraph separator `\u2029` on top of that. Java includes these plus the Latin-1 next line control character `\u0085`. Boost adds the form feed `\f` to the list. Only Delphi and the JGsoft flavor supports all Unicode line breaks, completing the mix with the vertical tab.

.NET is notably absent from the list of flavors that treat characters other than `\n` as line breaks. Unlike scripting languages that have their roots in the UNIX world, .NET is a Windows development framework that does not automatically strip carriage return characters from text files that it reads. If you read a Windows text file as a whole

into a string, it will contain carriage returns. If you use the regex `abc.*` on that string, without setting RegexOptions.SingleLine, then it will match `abc` plus all characters that follow on the same line, plus the carriage return at the end of the line, but without the newline after that.

Some flavors allow you to control which characters should be treated as line breaks. Java has the UNIX_LINES option which makes it treat only `\n` as a line break. PCRE has options that allow you to choose between `\n` only, `\r` only, `\r\n`, or all Unicode line breaks.

On POSIX systems, the POSIX locale determines which characters are line breaks. The C locale treats only the newline `\n` as a line break. Unicode locales support all Unicode line breaks.

\N Never Matches Line Breaks

Perl 5.12 and PCRE 8.10 introduced `\N` which matches any single character that is not a line break, just like the dot does. Unlike the dot, `\N` is not affected by "single-line mode". `(?s)\N.` turns on single-line mode and then matches any character that is not a line break followed by any character regardless of whether it is a line break.

PCRE's options that control which characters are treated as line breaks affect `\N` in exactly the same way as they affect the dot.

PHP 5.3.4 and R 2.14.0 also support `\N` as their regex support is based on PCRE 8.10 or later. JGsoft V2 also supports `\N`.

Use The Dot Sparingly

The dot is a very powerful regex metacharacter. It allows you to be lazy. Put in a dot, and everything matches just fine when you test the regex on valid data. The problem is that the regex also matches in cases where it should not match. If you are new to regular expressions, some of these cases may not be so obvious at first.

Let's illustrate this with a simple example. Say we want to match a date in mm/dd/yy format, but we want to leave the user the choice of date separators. The quick solution is `\d\d.\d\d.\d\d`. Seems fine at first. It matches a date like `02/12/03` just fine. Trouble is: `02512703` is also considered a valid date by this regular expression. In this match, the first dot matched `5`, and the second matched `7`. Obviously not what we intended.

`\d\d[- /.]\d\d[- /.]\d\d` is a better solution. This regex allows a dash, space, dot and forward slash as date separators. Remember that the dot is not a metacharacter inside a character class, so we do not need to escape it with a backslash.

This regex is still far from perfect. It matches `99/99/99` as a valid date. `[01]\d[- /.][0-3]\d[- /.]\d\d` is a step ahead, though it still matches `19/39/99`. How perfect you want your regex to be depends on what you want to do with it. If you are validating user input, it has to be perfect. If you are parsing data files from a known source that generates its files in the same way every time, our last attempt is probably more than sufficient to parse the data without errors. You can find a better regex to match dates in the example section.

Use Negated Character Classes Instead of the Dot

A negated character class is often more appropriate than the dot. The tutorial section that explains the repeat operators star and plus covers this in more detail. But the warning is important enough to mention it here as well. Again let's illustrate with an example.

Suppose you want to match a double-quoted string. Sounds easy. We can have any number of any character between the double quotes, so `".*"` seems to do the trick just fine. The dot matches any character, and the star

allows the dot to be repeated any number of times, including zero. If you test this regex on `Put a "string" between double quotes`, it matches `"string"` just fine. Now go ahead and test it on `Houston, we have a problem with "string one" and "string two". Please respond.`

Ouch. The regex matches `"string one" and "string two"`. Definitely not what we intended. The reason for this is that the star is *greedy*.

In the date-matching example, we improved our regex by replacing the dot with a character class. Here, we do the same with a negated character class. Our original definition of a double-quoted string was faulty. We do not want any number of *any character* between the quotes. We want any number of characters that are not double quotes or newlines between the quotes. So the proper regex is `"[^"\r\n]*"`.

Make a Donation

Did this website just save you a trip to the bookstore? Please make a donation to support this site, and you'll get a **lifetime of advertisement-free access** to this site!

Page URL: https://www.regular-expressions.info/dot.html
Page last updated: 30 January 2020
Site last updated: 08 April 2020