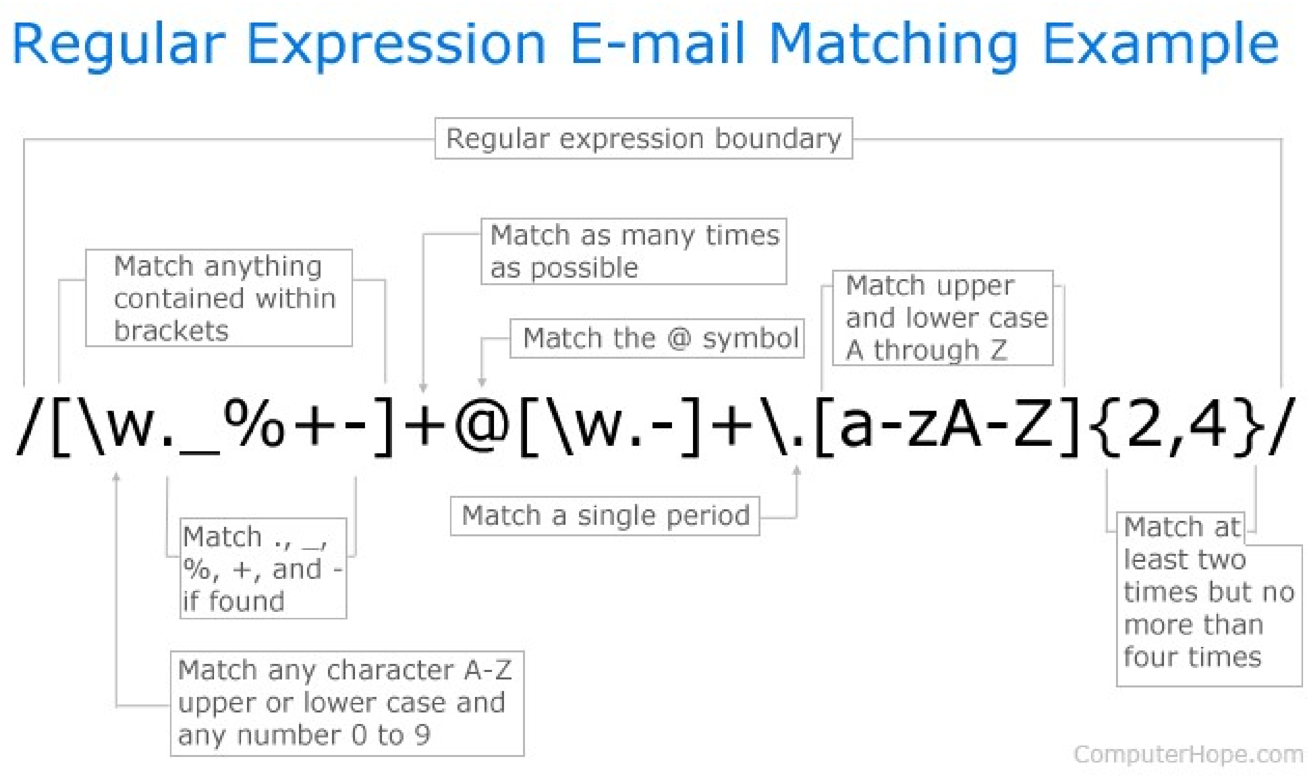
Regex

Updated: 04/26/2017 by Computer Hope

Short for **regular expression**, a **regex** is a string of text that allows you to create patterns that help match, locate, and manage text. Perl is a great example of a programming language that utilizes regular expressions. However, its only one of the many places you can find regular expressions. Regular Expressions can also be used from the command line and in text editors to find text within a file.

When first trying to understand regular expressions it seems as if it is a different language. However, mastering regular expressions can save you thousands of hours if you work with text or need to parse large amounts of data. Below is an example of a regular expression with each of its components labeled. This regular expression is also shown in the Perl programming examples shown later on this page.


Regular Expression E-mail Matching Example
Regular expression boundary
Match anything contained within brackets
Match as many times as possible
Match the @ symbol
Match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
Match ., _, %, +, and - if found
Match a single period
Match at least two times but no more than four times
Match any character A-Z upper or lower case and any number 0 to 9
ComputerHope.com

The basics of regular expressions (cheat sheet)

Looking at the above example may be overwhelming. However, once you understand the basic commands of how regular express commands operate you can read the above example just as if you are reading this sentence. Unfortunately, not all programs, commands, and programming languages use the same regular expressions, but they all share similarities.

Character	What does it do?	Example	Matches
^	Matches beginning of line	^abc	abc, abcdef.., abc123
$	Matches end of line	abc$	my:abc, 123abc, theabc
.	Match any character	a.c	abc, asg, a123c
|	OR operator	abc|xyz	abc or xyz
(...)	Capture anything matched	(a)b(c)	Captures 'a' and 'c'
(?:...)	Non-capturing group	(a)b(?:c)	Captures 'a' but only groups 'c'
[...]	Matches anything contained in brackets	[abc]	a,b, or c
[^...]	Matches anything not contained in brackets	[^abc]	xyz, 123, 1de
[a-z]	Matches any characters between 'a' and 'z'	[b-z]	bc, mind, xyz
{x}	The exact 'x' amount of times to match	(abc){2}	abcabc
{x,}	Match 'x' amount of times or more	(abc){2,}	abcabc, abcabcabc
{x,y}	Match between 'x' and 'y' times.	(a){2,4}	aa, aaa, aaaaa
*****	Greedy match that matches everything in place of the *	ab*c	abc, abbcc, abcdc
+	Matches character before + one or more times	a+c	ac, aac, aaac,
?	Matches the character before the ? zero or one times. Also, used as a non-greedy match	ab?c	ac, abc
\	Escape the character after the backslash or	a\sc	a c

create an escape sequence.

Escape characters (escape sequence)

Note: escape characters are case sensitive.

Character	What does it do?
\	Any character not mentioned below preceded with a \ will be escaped. For example, \. matches a period and does not perform the function mentioned above. Characters that should be escaped are () [] {} ^ $. | * + ? \
\0	Null character
\a	Match a bell or alarm.
\b	Word boundary in most or backspace
\B	Non word boundary
\d	Match any decimal digit (0-9)
\D	Match any non digit
\e	Match an escape
\f	Match a form feed
\n	Match a new line
\Q...\E	Ignores any special meaning in what is being matched.
\r	Match a carriage return
\s	Matches a space character (space, \t, \r, \n)
\S	Matches any non-white space character

\t	Match a tab
\v	Match a vertical tab
\w	Matches any one word character [a-zA-Z_0-9]
\W	Matches any one non word character

Regular expression flags

Outside the regular expression (at the end) flags can be used to help with the pattern matching.

Character	What does it do?
i	Ignore the case (upper and lower case allowed)
m	Multi-line match
s	Match new lines
x	Allow spaces and comments
J	Duplicate group names allowed
U	Ungreedy match

Perl programming language regular expression examples

Below are a few examples of regular expressions and pattern matching in Perl. Many of these examples are similar or the same to other programming languages and programs that support regular expressions.

```
$data =~ s/bad data/good data/i;
```

The above example replaces any "bad data" with "good data" using an case-insensitive match.

So if the $data variable was "Here is bad data" it would become "Here is good data".

```
$data =~ s/a/A/;
```

This example replaces any lowercase **a** with an uppercase **A**. So if $data was "example" it would become "exAmple".

```
$data =~ s/[a-z]/*/;
```

The above example replaces any lowercase letter, a through z, with an asterisk. So if $data was "Example" it would become "E******".

```
$data =~ s/e$/es/;
```

This example uses the $ character, which tells the regular expression to match the text before it at the end of the string. So if $data was "example" it would become "examples".

```
$data =~ s/\./!/;
```

In the above example, we are replacing a period with an exclamation mark. Because the period is a meta-character if you only entered a period without the \ (escape) it is treated as any character. In this example, if $data were "example." it would become "example!", however, if you did not have the escape it would replace every character and become "!!!!!!!!"

```
$data =~ s/^e/E/;
```

Finally, in this above example the carrot (^) tells the regular expression to match anything at the beginning of the line. In this example, this would match any lowercase e at the beginning of the line and replace it with a capital E. Therefore, if $data was "example" it would become "Example".

Tip: If you want to explore regular expressions even more in commands like grep, or regular expressions in programming language's check out the O'Reilly book "Mastering Regular Expressions."