Regular expressions are a powerful tool for examining and modifying text. Regular expressions themselves, w
pattern notation almost like a mini programming language, allow you to describe and parse text. They enable
for patterns within a string, extracting matches flexibly and precisely. However, you should note that because
expressions are more powerful, they are also slower than the more basic string functions. You should only us
expressions if you have a particular need.

This tutorial gives a brief overview of basic regular expression syntax and then considers the functions that P
for working with regular expressions.

PHP supports two different types of regular expressions: POSIX-extended and Perl-Compatible Regular Exp
(PCRE). The PCRE functions are more powerful than the POSIX ones, and faster too, so we will concentrate

The Basics

In a regular expression, most characters match only themselves. For instance, if you search for the regular ex
in the string "John plays football," you get a match because "foo" occurs in that string. Some characters have
meanings in regular expressions. For instance, a dollar sign ($) is used to match strings that end with the give
Similarly, a caret (^) character at the beginning of a regular expression indicates that it must match the begin
string. The characters that match themselves are called literals. The characters that have special meanings are
metacharacters.

The dot (.) metacharacter matches any single character except newline (\). So, the pattern h.t matches hat, ho
etc. The vertical pipe (|) metacharacter is used for alternatives in a regular expression. It behaves much like a
operator and you should use it if you want to construct a pattern that matches more than one set of characters
the pattern Utah|Idaho|Nevada matches strings that contain "Utah" or "Idaho" or "Nevada". Parentheses give
group sequences. For example, (Nant|b)ucket matches "Nantucket" or "bucket". Using parentheses to group t
characters for alternation is called grouping.

If you want to match a literal metacharacter in a pattern, you have to escape it with a backslash.

To specify a set of acceptable characters in your pattern, you can either build a character class yourself or use
one. A character class lets you represent a bunch of characters as a single item in a regular expression. You c
own character class by enclosing the acceptable characters in square brackets. A character class matches any
characters in the class. For example a character class [abc] matches a, b or c. To define a range of characters,
first and last characters in, separated by hyphen. For example, to match all alphanumeric characters: [a-zA-Z
also create a negated character class, which matches any character that is not in the class. To create a negated
class, begin the character class with ^: [^0-9].

The metacharacters +, *, ?, and {} affect the number of times a pattern should be matched. + means "Match o
the preceding expression", * means "Match zero or more of the preceding expression", and ? means "Match z
the preceding expression". Curly braces {} can be used differently. With a single integer, {n} means "match

occurrences of the preceding expression", with one integer and a comma, {n,} means "match n or more occur
preceding expression", and with two comma-separated integers {n,m} means "match the previous character i
least n times, but no more than m times".

Now, have a look at the examples:

Regular Expression	Will match...
foo	The string "foo"
^foo	"foo" at the start of a string
foo$	"foo" at the end of a string
^foo$	"foo" when it is alone on a string
[abc]	a, b, or c
[a-z]	Any lowercase letter
[^A-Z]	Any character that is not a uppercase letter
(gif|jpg)	Matches either "gif" or "jpeg"
[a-z]+	One or more lowercase letters
[0-9\.\-]	Any number, dot, or minus sign
^[a-zA-Z0-9_]{1,}$	Any word of at least one letter, number or _
([wx])([yz])	wy, wz, xy, or xz
[^A-Za-z0-9]	Any symbol (not a number or a letter)
([A-Z]{3}|[0-9]{4})	Matches three letters or four numbers

Perl-Compatible Regular Expressions emulate the Perl syntax for patterns, which means that each pattern mu
in a pair of delimiters. Usually, the slash (/) character is used. For instance, /pattern/.

The PCRE functions can be divided in several classes: matching, replacing, splitting and filtering.

Back to top

Matching Patterns

The *preg_match()* function performs Perl-style pattern matching on a string. *preg_match()* takes two basic ar
optional parameters. These parameters are, in order, a regular expression string, a source string, an array vari
stores matches, a flag argument and an offset parameter that can be used to specify the alternate place from w
the search:

```
preg_match ( pattern, subject [, matches [, flags [, offset]]])
```

The *preg_match()* function returns 1 if a match is found and 0 otherwise. Let's search the string "Hello World
letters "ll":

```
<?php
if (preg_match("/ell/", "Hello World!", $matches)) {
  echo "Match was found <br />";
  echo $matches[0];
}
?>
```

The letters "ll" exist in "Hello", so *preg_match()* returns 1 and the first element of the $matches variable is fi
the string that matched the pattern. The regular expression in the next example is looking for the letters "ell",
for them with following characters:

```
<?php
if (preg_match("/ll.*/", "The History of Halloween", $matches)) {
  echo "Match was found <br />";
  echo $matches[0];
}
?>
```

Now let's consider more complicated example. The most popular use of regular expressions is validation. Th
below checks if the password is "strong", i.e. the password must be at least 8 characters and must contain at l
case letter, one upper case letter and one digit:

```
<?php
$password = "Fyfjk34sdfjfsjq7";

if (preg_match("/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/", $password)) {
    echo "Your passwords is strong.";
} else {
    echo "Your password is weak.";
}
?>
```

The ^ and $ are looking for something at the start and the end of the string. The ".*" combination is used at b
and the end. As mentioned above, the .(dot) metacharacter means any alphanumeric character, and * metacha
"zero or more". Between are groupings in parentheses. The "?=" combination means "the next text must be li
construct doesn't capture the text. In this example, instead of specifying the order that things should appear, i
it must appear but we're not worried about the order.

The first grouping is (?=.*{8,}). This checks if there are at least 8 characters in the string. The next grouping
means "any alphanumeric character can happen zero or more times, then any digit can happen". So this check
least one number in the string. But since the string isn't captured, that one digit can appear anywhere in the st
groupings (?=.*[a-z]) and (?=.*[A-Z]) are looking for the lower case and upper case letter accordingly anywh
string.

Finally, we will consider regular expression that validates an email address:

```
<?php
$email = firstname.lastname@aaa.bbb.com;
$regexp = "/^[^0-9][A-z0-9_]+([.][A-z0-9_]+)*[@][A-z0-9_]+([.][A-z0-9_]+)*[.][A-z]{2,4}$/";
```

```
if (preg_match($regexp, $email)) {
   echo "Email address is valid.";
} else {
   echo "Email address is <u>not</u> valid.";
}
?>
```

This regular expression checks for the number at the beginning and also checks for multiple periods in the us
domain name in the email address. Let's try to investigate this regular expression yourself.

For the speed reasons, the *preg_match()* function matches only the first pattern it finds in a string. This mean
quick to check whether a pattern exists in a string. An alternative function, *preg_match_all()*, matches a patte
string as many times as the pattern allows, and returns the number of times it matched.

Back to top

Replacing Patterns

In the above examples, we have searched for patterns in a string, leaving the search string untouched. The *pr*
function looks for substrings that match a pattern and then replaces them with new text. *preg_replace()* takes
parameters and an additional one. These parameters are, in order, a regular expression, the text with which to
found pattern, the string to modify, and the last optional argument which specifies how many matches will be

preg_replace(pattern, replacement, subject [, limit])

The function returns the changed string if a match was found or an unchanged copy of the original string othe
following example we search for the copyright phrase and replace the year with the current.

```
<?php
echo preg_replace("/([Cc]opyright) 200(3|4|5|6)/", "$1 2007", "Copyright 2005");
?>
```

In the above example we use back references in the replacement string. Back references make it possible for
of a matched pattern in the replacement string. To use this feature, you should use parentheses to wrap any el
regular expression that you might want to use. You can refer to the text matched by subpattern with a dollar s
number of the subpattern. For instance, if you are using subpatterns, $0 is set to the whole match, then $1, $2
set to the individual matches for each subpattern.

In the following example we will change the date format from "yyyy-mm-dd" to "mm/dd/yyy":

```
<?php
echo preg_replace("/(\d+)-(\d+)-(\d+)/", "$2/$3/$1", "2007-01-25");
?>
```

We also can pass an array of strings as *subject* to make the substitution on all of them. To perform multiple s

the same string or array of strings with one call to *preg_replace()*, we should pass arrays of patterns and repla
a look at the example:

```
<?php
$search = array ( "/(\w{6}\s\(w{2})\s(\w+)/e",
                "/(\d{4})-(\d{2})-(\d{2})\s(\d{2}:\d{2}:\d{2})/");

$replace = array ('"$1 ".strtoupper("$2")',
                "$3/$2/$1 $4");

$string = "Posted by John | 2007-02-15 02:43:41";

echo preg_replace($search, $replace, $string);?>
```

In the above example we use the other interesting functionality - you can say to PHP that the match text shou
as PHP code once the replacement has taken place. Since we have appended an "e" to the end of the regular e
PHP will execute the replacement it makes. That is, it will take strtoupper(name) and replace it with the resul
strtoupper() function, which is NAME.

Back to top

Array Processing

PHP's *preg_split()* function enables you to break a string apart basing on something more complicated than a
sequence of characters. When it's necessary to split a string with a dynamic expression rather than a fixed one
comes to the rescue. The basic idea is the same as *preg_match_all()* except that, instead of returning matched
subject string, it returns an array of pieces that didn't match the specified pattern. The following example uses
expression to split the string by any number of commas or space characters:

```
<?php
$keywords = preg_split("/[\s,]+/", "php, regular expressions");
print_r( $keywords );
?>
```

Another useful PHP function is the *preg_grep()* function which returns those elements of an array that match
pattern. This function traverses the input array, testing all elements against the supplied pattern. If a match is
matching element is returned as part of the array containing all matches. The following example searches thr
and all the names starting with letters A-J:

```
<?php
$names = array('Andrew','John','Peter','Nastin','Bill');
$output = preg_grep('/^[a-m]/i', $names);
print_r( $output );
?>
```

Back to top

Tags: PHP REGULAR EXPRESSION MATCH REPLACE PATTERN SPLIT FILTER PERL

Add To: dzone | digg | del.icio.us | stumbleupon

- Comments (22)
- Blog It

Hide Comments | Add New

Subscription

Your Name *	
E-Mail Address	
Comments *	

It has been dugg:
http://digg.com/programming/Awesome_PHP_regular_expression_cheatsheet
# Posted by Frank | 2 Apr 2007 08:42:17
The REGEX (gif|jpg) match with gif or jpg, not gif or jpeg.
# Posted by Fábio | 15 Apr 2007 23:09:06
This article is not a light intro to regexp in PHP, nor an advanced one but your code lines turned out to be straight useful for me. Thanks a lot from Strasbourg, France.
# Posted by Joetke | 3 Aug 2007 23:15:00
very nice explanation .....
# Posted by Archana | 10 Aug 2007 03:09:49
Very helpful note......
# Posted by jijo | 16 Aug 2007 03:11:25
This is the BEST explanation of regular expression I have ever seen. I never understood it until now.
# Posted by Yacahuma | 21 Aug 2007 08:47:53
Please healp me.

I want to replace height=200 and width=300 from following string
-->
<object width="425" height="366"><param name="movie"

value="http://www.youtube.com/v/qgAYasUw6vk"></param><param name="wmode" value="transparent"></param><embed src="http://www.youtube.com/v/qgAYasUw6vk" type="application/x-shockwave-flash" wmode="transparent" width="425" height="366"></embed></object>
<--
Using regular repression but can't able to found soution. Please provide me php script for this.

Best Regards,
Shailendra
# Posted by shailendra | 4 Oct 2007 03:25:23
It is really a good article to start with and I got helped from this article.
# Posted by Zia | 30 Oct 2007 01:52:46
Extreamly useful info and a great reference when dealing with regular expressions!
# Posted by Justin | 1 Nov 2007 03:43:05
([A-Z]{3}|[0-9]{4}) matches three UPERCASE letters or 4 digits.

Usefull exlanation using regexps. Thanks.

Stefan
# Posted by Stefan | 3 Nov 2007 14:17:52
dear Shailendra,

try using str_replace, manual on php.net...

# Posted by Jeroen | 18 Nov 2007 18:28:52
really a good one, explained the regex concepts clearly..

Ram Prasath.R
# Posted by ram | 22 Nov 2007 04:01:44
hi how do i do this

KarenFred123 to Karen Fred
MichaelFrance488 to Michael France

i did this

```
$str = 'AasdasdasGasasd';
$chars = preg_split('/[A-Z]/', $str, -1, PREG_SPLIT_NO_EMPTY);
print_r($chars);
```

but it outputs

```
Array ( [0] => asdasdas [1] => asasd )
```

thanks very much
# Posted by shane | 23 Nov 2007 05:28:15

First of all, this regex tut is awesome. Really simple explanation on a really complex topic. So, thanks a ton.

Now, the only part that I didn't understand is the following regex. So, can you explain how does it work? I mean I got the grouping logic but where the strtoupper() function is getting applied?

```
<?php
$search = array ( "/(w{6}s(w{2})s(w+)/e",
"/(d{4})-(d{2})-(d{2})s(d{2}:d{2}:d{2})/");

$replace = array ('"$1 ".strtoupper("$2")',
"$3/$2/$1 $4");

$string = "Posted by John | 2007-02-15 02:43:41";

echo preg_replace($search, $replace, $string);?>
```

Thanks in advance!
# Posted by Chillyroll | 11 Jan 2008 23:15:58
Great,
I'm remembering the day when i've searched for the regx. Its really awsome. Great. Very helpful site. So i'm posting two comments in same day.

good and great job. no ads here how do you survive....
# Posted by Sumitra Acharya | 17 Jan 2008 06:20:46
Really fantastic step-by-step explanation...
Very useful for tyro in Reg exp...
Thanks...
# Posted by Muthukumaran | 5 Feb 2008 22:17:43
this topic "must seen" thing .

its the best php topic, better than official php site, i swear the god.

I decided to stop learning php b4 I read this explanation.

its kind of tricky stuff i belive.

god bless you,

and thank you very much.

ADDED 2 favorite...OK!

you'll c me here all the day.
<- so excited.

# Posted by rania | 29 Feb 2008 10:14:13
very nice site
# Posted by narendra ojha | 23 Mar 2008 00:45:51
Hey pretty straight and useful!!! Thanks a ton.
# Posted by ronair | 27 Mar 2008 02:58:06
This is a realy good explanation of regular expressions.
I have been using PHP for quite a while but I have never known how to use regular expressions.

Chaim Chaikin
http://chaimchaikin.za.net
# Posted by Chaim Chaikin | 13 Apr 2008 02:00:32
Great tutorial. Made regular expressions easy for me. Thank you so very much from Pune, India.
# Posted by Arun | 27 May 2008 13:44:28
Very nice article, very helpful
# Posted by Yogesh Agrawal | 28 May 2008 08:04:03

Java=====www.moko.ru/doc/Java_Script/index.html

Regular=http://www.codeproject.com/KB/dotnet/regextutorial.aspx

Java=www.htmlgoodies.com/beyond/javascript/article.php/3697716

Reg===www.regular-expressions.info/reference.html