


[strtolower »](#)[« strstr](#)

- [Manual de PHP](#)
- [Referencia de funciones](#)
- [Procesamiento de texto](#)
- [Strings](#)
- [Funciones de strings](#)

Change language: Spanish [Submit a Pull Request](#) [Report a Bug](#)

strtok

(PHP 4, PHP 5, PHP 7, PHP 8)

strtok — Tokeniza string

Descripción ¶

strtok(string \$str, string \$token): string
strtok(string \$token): string

strtok() divide un string (*str*) en strings más pequeños (tokens), con cada token delimitado por cualquier caracter de token. Es decir, si se tiene un string como "Este es un string de ejemplo", se puede tokenizar en sus palabras individuales utilizando el caracter de espacio como el token.

Nótese que sólo la primera llamada a strtok utiliza el argumento string. Cada llamada subsecuente a strtok sólo necesita el token a utilizar, ya que realiza un seguimiento del lugar en que se encuentra en el string actual. Para volver a empezar, o para dividir una cadena nueva, simplemente se llama a strtok de nuevo con el argumento string para inicializarlo. Tener en cuenta que se pueden poner tokens múltiples como parámetro. El string será tokenizado cuando cualquiera de los caracteres en el argumento sea encontrado.

Parámetros ¶

str

El string a ser dividido en strings más pequeños (tokens).

token

El delimitador usado cuando se divide *str*.

Valores devueltos ¶

Un token de string.

Ejemplos ¶

Ejemplo #1 Ejemplo de strtok()

```
<?php
$string = "This is\tan example\nstring";
/* Utiliza tabulador y nueva línea como caracteres de tokenización, así */
$tok = strtok($string, " \n\t");
```

```
while ($tok !== false) {
    echo "Word=$tok<br />";
    $tok = strtok(" \n\t");
}
?>
```

El comportamiento cuando se encuentra una parte vacía, cambió con PHP 4.1.0. El comportamiento anterior devolvía una cadena vacía, mientras que el comportamiento nuevo y correcto simplemente se salta esa parte del string:

Ejemplo #2 Comportamiento anterior de strtok()

```
<?php
$first_token = strtok('/something', '/');
$second_token = strtok('/');
var_dump($first_token, $second_token);
?>
```

El resultado del ejemplo sería:

```
string(0) ""
string(9) "something"
```

Ejemplo #3 Comportamiento nuevo de strtok()

```
<?php
$first_token = strtok('/something', '/');
$second_token = strtok('/');
var_dump($first_token, $second_token);
?>
```

El resultado del ejemplo sería:

```
string(9) "something"
bool(false)
```

Notas ¶

Advertencia

Esta función puede devolver el valor booleano **false**, pero también puede devolver un valor no booleano que se evalúa como **false**. Por favor lea la sección sobre [Booleanos](#) para más información. Use [el operador ===](#) para comprobar el valor devuelto por esta función.

Ver también ¶

- [split\(\)](#)
- [explode\(\)](#) - Divide un string en varios string

[+ add a note](#)

User Contributed Notes 19 notes

[up](#)
[down](#)

91

[eep2004 at ukr dot net ¶](#)

9 years ago

```
<?php
// strtok example
$str = 'Hello to all of Ukraine';
echo strtok($str, ' ').' '.strtok(' ').' '.strtok(' ');
?>
```

Result:

Hello to all

[up](#)

[down](#)

18

[manicdepressive at mindless dot com ¶](#)

18 years ago

```
<pre><?php
/** get leading, trailing, and embedded separator tokens that were 'skipped'
if for some ungodly reason you are using php to implement a simple parser that
needs to detect nested clauses as it builds a parse tree */
```

```
$str = "(((alpha(beta))(gamma)))";
```

```
$seps = '()';
```

```
$tok = strtok( $str,$seps ); // return false on empty string or null
```

```
$cur = 0;
```

```
$dumbDone = FALSE;
```

```
$done = (FALSE=== $tok);
```

```
while (!$done) {
```

```
    // process skipped tokens (if any at first iteration) (special for last)
```

```
    $posTok = $dumbDone ? strlen($str) : strpos($str, $tok, $cur );
```

```
    $skippedMany = substr( $str, $cur, $posTok-$cur ); // false when 0 width
```

```
    $lenSkipped = strlen($skippedMany); // 0 when false
```

```
    if (0!=$lenSkipped) {
```

```
        $last = strlen($skippedMany) -1;
```

```
        for($i=0; $i<=$last; $i++){
```

```
            $skipped = $skippedMany[$i];
```

```
            $cur += strlen($skipped);
```

```
            echo "skipped: $skipped\n";
```

```
        }
```

```
    }
```

```
    if ($dumbDone) break; // this is the only place the loop is terminated
```

```
    // process current tok
```

```
    echo "curr tok: ".$tok."\n";
```

```
    // update cursor
```

```
    $cur += strlen($tok);
```

```
    // get any next tok
```

```
    if (!$dumbDone){
```

```
        $tok = strtok($seps);
```

```
        $dumbDone = (FALSE=== $tok);
```

```
        // you're not really done till you check for trailing skipped
```

```
    }
```

```
};
```

```
?></pre>
```

[up](#)

[down](#)

12

[eep2004 at ukr dot net ¶](#)

7 years ago

Remove GET variables from the URL

```
<?php
echo strtok('http://example.com/index.php?foo=1&bar=2', '?');
?>
```

Result:

<http://example.com/index.php>

[up](#)

[down](#)

10

[elartlang at gmail dot com ¶](#)

11 years ago

If you have memory-usage critical solution, you should keep in mind, that strtok function holds input string parameter (or reference to it?) in memory after usage.

```
<?php
function tokenize($str, $token_symbols) {
    $word = strtok($str, $token_symbols);
    while (false !== $word) {
        // do something here...
        $word = strtok($token_symbols);
    }
}
```

Test-cases with handling ~10MB plain-text file:

Case #1 - unset \$str variable

```
<?php
$token_symbols = " \t\n";
$str = file_get_contents('10MB.txt'); // mem usage 9.75383758545 MB (memory_get_usage() / 1024 / 1024));
tokenize($str, $token_symbols); // mem usage 9.75400161743 MB
unset($str); // 9.75395584106 MB
?>
```

Case #1 result: memory is still used

Case #2 - call strtok again

```
<?php
$token_symbols = " \t\n";
$str = file_get_contents('10MB.txt'); // 9.75401306152 MB
tokenize($str, $token_symbols); // 9.75417709351
strtok('', ''); // 9.75421524048
?>
```

Case #2 result: memory is still used

Case #3 - call strtok again AND unset \$str variable

```
<?php
$token_symbols = " \t\n";
$str = file_get_contents('10MB.txt'); // 9.75410079956 MB
tokenize($str, $token_symbols); // 9.75426483154 MB
unset($str);
strtok('', ''); // 0.0543975830078 MB
?>
```

Case #3 result: memory is free

So, better solution for tokenize function:

```
<?php
function tokenize($str, $token_symbols, $token_reset = true) {
    $word = strtok($str, $token_symbols);
    while (false !== $word) {
```

```

        // do something here...
        $word = strtok($token_symbols);
    }

    if($token_reset)
        strtok('', '');
}
?>

```

[up](#)
[down](#)

5

[benighted at gmail dot com](#)

13 years ago

Simple way to tokenize search parameters, including double or single quoted keys. If only one quote is found, the rest of the string is assumed to be part of that token.

```

<?php
    $token = strtok($keywords, ' ');
    while ($token) {
        // find double quoted tokens
        if ($token{0}=='"') { $token .= ' '.strtok('"').'"; }
        // find single quoted tokens
        if ($token{0}=='"') { $token .= ' '.strtok("'").'"; }

        $tokens[] = $token;
        $token = strtok(' ');
    }
?>

```

Use substr(1,strlen(\$token)) and remove the part that adds the trailing quotes if you want your output without quotes.

[up](#)
[down](#)

5

[Logikos](#)

13 years ago

This looks very simple, but it took me a long time to figure out so I thought I'd share it incase someone else was wanting the same thing:

this should work similar to substr() but with tokens instead!

```

<?php
/* subtok(string,chr,pos,len)
*
* chr = chr used to seperate tokens
* pos = starting postion
* len = length, if negative count back from right
*
* subtok('a.b.c.d.e','.',0)      = 'a.b.c.d.e'
* subtok('a.b.c.d.e','.',0,2)    = 'a.b'
* subtok('a.b.c.d.e','.',2,1)    = 'c'
* subtok('a.b.c.d.e','.',2,-1)   = 'c.d'
* subtok('a.b.c.d.e','.',-4)     = 'b.c.d.e'
* subtok('a.b.c.d.e','.',-4,2)   = 'b.c'
* subtok('a.b.c.d.e','.',-4,-1)  = 'b.c.d'
*/
function subtok($string,$chr,$pos,$len = NULL) {
    return implode($chr,array_slice(explode($chr,$string),$pos,$len));
}

```

```
}
?>
```

explode breaks the tokens up into an array, array slice allows you to pick then tokens you want, and then implode converts it back to a string

although its far from a clone, this was inspired by mIRC's gettok() function

[up](#)
[down](#)

4

[KrazyBox ¶](#)

13 years ago

As of the change in strtok()'s handling of empty strings, it is now useless for scripts that rely on empty data to function.

Take for instance, a standard header. (with UNIX newlines)

```
http/1.0 200 OK\n
Content-Type: text/html\n
\n
--HTML BODY HERE---
```

When parsing this with strtok, one would wait until it found an empty string to signal the end of the header. However, because strtok now skips empty segments, it is impossible to know when the header has ended.

This should not be called 'correct' behavior, it certainly is not. It has rendered strtok incapable of (properly) processing a very simple standard.

This new functionality, however, does not affect Windows style headers. You would search for a line that only contains "\r"

This, however, is not a justification for the change.

[up](#)
[down](#)

2

[Axeia ¶](#)

8 years ago

Might be pointing out the obvious but if you'd rather use a for loop rather than a while (to keep the token strings on the same line for readability for example), it can be done. Added bonus, it doesn't put a \$tok variable outside the loop itself either.

Downside however is that you're not able to manually free up the memory used using the technique mentioned by elarlang.

```
<?php
for($tok = strtok($str, ' _-.'); $tok!==false; $tok = strtok(' _-.'))
{
    echo "$tok </br>";
}
?>
```

[up](#)
[down](#)

3

[info at maisuma dot jp ¶](#)

8 years ago

If you want to tokenize by only one letter, explode() is much faster compared to strtok().

```
<?php
$str=str_repeat('foo ',10000);
```

```
//explode()
$time=microtime(TRUE);
$arr=explode($str, ' ');
$time=microtime(TRUE)-$time;
echo "explode():$time sec.".PHP_EOL;

//strtok()
$time=microtime(TRUE);
$ret=strtok(' ', $str);
while($ret!==FALSE){
    $ret=strtok(' ');
}
$time=microtime(TRUE)-$time;
echo "strtok():$time sec.".PHP_EOL;

?>
```

The result is : (PHP 5.3.3 on CentOS)

```
explode():0.001317024230957 sec.
strtok():0.0058917999267578 sec.
```

explode() is about five times fast in short strings, too.

[up](#)
[down](#)

1

[gilthans at NOSPAM dot gmail dot com ¶](#)

10 years ago

Note that strtok may receive different tokens each time. Therefore, if, for example, you wish to extract several words and then the rest of the sentence:

```
<?php
$text = "13 202 5 This is a long message explaining the error codes.";
$error1 = strtok($text, " "); //13
$error2 = strtok(" "); //202
$error3 = strtok(" "); //5
$error_message = strtok(""); //Notice the different token parameter
echo $error_message; //This is a long message explaining the error codes.
?>
```

[up](#)
[down](#)

1

[azeem ¶](#)

13 years ago

Here is a java like StringTokenizer class using strtok function:

```
<?php

/**
 * The string tokenizer class allows an application to break a string into tokens.
 *
 * @example The following is one example of the use of the tokenizer. The code:
 * <code>
 * <?php
 *     $str = 'this is:@\t\n a test!';
 *     $delim = ' !@:\t\n'; // remove these chars
 *     $st = new StringTokenizer($str, $delim);
 *     while ($st->hasMoreTokens()) {
```

```

*      echo $st->nextToken() . "\n";
*  }
*  prints the following output:
*      this
*      is
*      a
*      test
* ?>
* </code>
*/
class StringTokenizer {

    /**
     * @var string
     */
    private $token;

    /**
     * @var string
     */
    private $delim;

    /**
     * Constructs a string tokenizer for the specified string
     * @param string $str String to tokenize
     * @param string $delim The set of delimiters (the characters that separate tokens)
     * specified at creation time, default to ' '
     */
    public function __construct(/*string*/ $str, /*string*/ $delim = ' ') {
        $this->token = strtok($str, $delim);
        $this->delim = $delim;
    }

    public function __destruct() {
        unset($this);
    }

    /**
     * Tests if there are more tokens available from this tokenizer's string. It
     * does not move the internal pointer in any way. To move the internal pointer
     * to the next element call nextToken()
     * @return boolean - true if has more tokens, false otherwise
     */
    public function hasMoreTokens() {
        return ($this->token !== false);
    }

    /**
     * Returns the next token from this string tokenizer and advances the internal
     * pointer by one.
     * @return string - next element in the tokenized string
     */
    public function nextToken() {
        $current = $this->token;
        $this->token = strtok($this->delim);
        return $current;
    }
}
?>

```


[up](#)
[down](#)

1

[pradador at me dot com ¶](#)

11 years ago

Here's a simple class that allows you to iterate through string tokens using a foreach loop.

```
<?php
/**
 * The TokenIterator class allows you to iterate through string tokens using
 * the familiar foreach control structure.
 *
 * Example:
 * <code>
 * <?php
 * $string = 'This is a test.';
 * $delimiters = ' ';
 * $ti = new TokenIterator($string, $delimiters);
 *
 * foreach ($ti as $count => $token) {
 *     echo sprintf("%d, %s\n", $count, $token);
 * }
 *
 * // Prints the following output:
 * // 0. This
 * // 1. is
 * // 2. a
 * // 3. test.
 * </code>
 */
class TokenIterator implements Iterator
{
    /**
     * The string to tokenize.
     * @var string
     */
    protected $_string;

    /**
     * The token delimiters.
     * @var string
     */
    protected $_delims;

    /**
     * Stores the current token.
     * @var mixed
     */
    protected $_token;

    /**
     * Internal token counter.
     * @var int
     */
    protected $_counter = 0;

    /**
     * Constructor.

```

```

*
* @param string $string The string to tokenize.
* @param string $delims The token delimiters.
*/
public function __construct($string, $delims)
{
    $this->_string = $string;
    $this->_delims = $delims;
    $this->_token = strtok($string, $delims);
}

/**
 * @see Iterator::current()
 */
public function current()
{
    return $this->_token;
}

/**
 * @see Iterator::key()
 */
public function key()
{
    return $this->_counter;
}

/**
 * @see Iterator::next()
 */
public function next()
{
    $this->_token = strtok($this->_delims);

    if ($this->valid()) {
        ++$this->_counter;
    }
}

/**
 * @see Iterator::rewind()
 */
public function rewind()
{
    $this->_counter = 0;
    $this->_token = strtok($this->_string, $this->_delims);
}

/**
 * @see Iterator::valid()
 */
public function valid()
{
    return $this->_token !== FALSE;
}
}
?>

```

[up](#)

[down](#)

0

[heiangus at hotmail dot com ¶](#)**3 years ago**

I found this useful for parsing user entered links in text fields.

e.g. This is a link <<http://example.com>>

```
function parselink($link) {
    $bit1 = trim(strtok($link, '<'));
    $bit2 = trim(strtok('>'));
    $html = '<a href="'. $bit2. '">'. $bit1. '</a>';
    return $html; // <a href="http://example.com">This is a link</a>
}
```

[up](#)[down](#)

-1

[David Spector ¶](#)**1 year ago**

After obtaining zero or more tokens with calls to strtok, you can obtain the remainder of the input string by calling strtok with an empty string as the delimiter.

[up](#)[down](#)

0

[charlie dot ded at orange dot fr ¶](#)**6 years ago**

@maisuma you invert paramaters of explode() and strtok() functions, your code does not do what you expect.

You expect to read the input string token after token so equivalent code for strtok() is `arra_filter(explode())` because `explode()` return lines of empty string when several delimiters are contiguous in the read string, for example 2 whitespaces between.

In fact `strtok()` is much faster (x2 at least) than `arra_filter(explode())` if the read string contains several contiguous delimiters ,
it is slower if the read string contains one and only one delimiter between tokens.

<?php

```
$repeat = 10;
$delimiter = ':';
$str=str_repeat('foo:', $repeat);

$timeStrtok=microtime(TRUE);
$token = strtok($str, $delimiter);
while($token!==FALSE){
    //echo $token . ',';
    $token=strtok($delimiter);
}
$timeStrtok -=microtime(TRUE);

$timeExplo=microtime(TRUE);
$arr = explode($delimiter, $str);
//$arr = array_filter($arr);
$timeExplo -=microtime(TRUE);

//print_r($arr);

$X = 1000000; $unit = 'microsec';
```

```
echo PHP_EOL . ' explode() : ' . -$timeExplo . ' ' . $unit . PHP_EOL . ' strtok() : ' .
-$timeStrtok . ' ' . $unit . PHP_EOL;
```

```
$timeExplo=round(-$timeExplo*$X);
$timeStrtok=round(-$timeStrtok*$X);
```

```
echo PHP_EOL . ' explode() : ' . $timeExplo . ' ' . $unit . PHP_EOL . ' strtok() : ' . $timeStrtok
. ' ' . $unit . PHP_EOL;
echo ' ratio explode / strtok : ' . round($timeExplo / $timeStrtok,1) . PHP_EOL;
```

```
?>
```

[up](#)

[down](#)

```
-1
```

[eep2004 at ukr dot net ¶](#)

8 years ago

Remove GET variables from the URL

```
<?php
```

```
$url = strtok('http://php.net/manual/en/ref.strings.php?foo=1&bar=2', '?');
```

```
// $url = 'http://php.net/manual/en/ref.strings.php'
```

[up](#)

[down](#)

```
-1
```

[mac.com@nemo ¶](#)

16 years ago

This function takes a string and returns an array with words (delimited by spaces), also taking into account quotes, doublequotes, backticks and backslashes (for escaping stuff).

So

```
$string = "cp 'my file' to `Judy's file`";
var_dump(parse_cli($string));
```

would yield:

```
array(4) {
    [0]=>
        string(2) "cp"
    [1]=>
        string(7) "my file"
    [2]=>
        string(5) "to"
    [3]=>
        string(11) "Judy's file"
}
```

Way it works, runs through the string character by character, for each character looking up the action to take, based on that character and its current \$state.

Actions can be (one or more of) adding the character/string to the current word, adding the word to the output array, and changing or (re)storing the state.

For example a space will become part of the current 'word' (or 'token') if \$state is 'doublequoted', but it will start a new token if \$state was 'unquoted'.

I was later told it's a "tokeniser using a finite state automaton". Who knew :-)

```
<?php
```

```
# _____
# parse_cli($string) /
```

```

function parse_cli($string) {
    $state = 'space';
    $previous = '';    // stores current state when encountering a backslash (which changes
                        // $state to 'escaped', but has to fall back into the previous $state afterwards)
    $out = array();    // the return value
    $word = '';
    $type = '';        // type of character
    // array[states][chartypes] => actions
    $chart = array(
        'space'          => array('space'=>' ', 'quote'=>'q', 'doublequote'=>'d',
        'backtick'=>'b', 'backslash'=>'ue', 'other'=>'ua'),
        'unquoted'      => array('space'=>'w ', 'quote'=>'a', 'doublequote'=>'a',
        'backtick'=>'a', 'backslash'=>'e', 'other'=>'a'),
        'quoted'         => array('space'=>'a', 'quote'=>'w ', 'doublequote'=>'a',
        'backtick'=>'a', 'backslash'=>'e', 'other'=>'a'),
        'doublequoted'   => array('space'=>'a', 'quote'=>'a', 'doublequote'=>'w ',
        'backtick'=>'a', 'backslash'=>'e', 'other'=>'a'),
        'backticked'     => array('space'=>'a', 'quote'=>'a', 'doublequote'=>'a', 'backtick'=>'w
        ', 'backslash'=>'e', 'other'=>'a'),
        'escaped'        => array('space'=>'ap', 'quote'=>'ap', 'doublequote'=>'ap',
        'backtick'=>'ap', 'backslash'=>'ap', 'other'=>'ap'));
    for ($i=0; $i<=strlen($string); $i++) {
        $char = substr($string, $i, 1);
        $type = array_search($char, array('space'=>' ', 'quote'=>'\\', 'doublequote'=>'"',
        'backtick'=>'`', 'backslash'=>'\\'));
        if (! $type) $type = 'other';
        if ($type == 'other') {
            // grabs all characters that are also 'other' following the current one in one go
            preg_match("/[ \\\"'\\`\\\\]/", $string, $matches, PREG_OFFSET_CAPTURE, $i);
            if ($matches) {
                $matches = $matches[0];
                $char = substr($string, $i, $matches[1]-$i); // yep, $char length can be > 1
                $i = $matches[1] - 1;
            }else{
                // no more match on special characters, that must mean this is the last word!
                // the .= hereunder is because we *might* be in the middle of a word that just
                contained special chars
                $word .= substr($string, $i);
                break; // jumps out of the for() loop
            }
        }
        $actions = $chart[$state][$type];
        for($j=0; $j<strlen($actions); $j++) {
            $act = substr($actions, $j, 1);
            if ($act == ' ') $state = 'space';
            if ($act == 'u') $state = 'unquoted';
            if ($act == 'q') $state = 'quoted';
            if ($act == 'd') $state = 'doublequoted';
            if ($act == 'b') $state = 'backticked';
            if ($act == 'e') { $previous = $state; $state = 'escaped'; }
            if ($act == 'a') $word .= $char;
            if ($act == 'w') { $out[] = $word; $word = ''; }
            if ($act == 'p') $state = $previous;
        }
    }
    if (strlen($word)) $out[] = $word;
    return $out;
}

```

?>

[up](#)

[down](#)

-1

[fabiolimasouto at gmail dot com ¶](#)

11 years ago

this example will hopefully help you understand how this function works:

```
<?php
$selector = 'div.class#id';
$tagname = strtok($selector, '.#');
echo $tagname.'<br/>';
```

```
while($tok = strtok('.#'))
{
    echo $tok.'<br/>';
}
```

?>

Outputs:

div

class

id

[up](#)

[down](#)

-5

[yanick dot rochon at gmail dot com ¶](#)

13 years ago

Here is a small function I wrote as I needed to extract some named tokens from a string (a la Google). For example, I needed to format a string like "extension:gif size:64M animated:true author:'John Bash'" into

```
array(
    'extension' => 'gif',
    'size' => '64M',
    'animated' => true,
    'author' => 'John Bash'
)
```

So, here's the code:

```
<?php
```

```
header('Content-type: text/plain; charset=utf-8');
```

```
/**
```

```
* NOTE : use mbstring.func_overload for multi-byte support with this function
```

```
*
```

```
* @param string $string          the string to tokenize
```

```
* @param int $offset             the starting offset
```

```
* @param string $defaultTokenName the default token name if none specified
```

```
* @param string $groupDelimiters the characters to delimit token groups
```

```
* @param string $groupNameDelimiter the character(s) to delimit token group names
```

```
* @return array
```

```
*/
```

```
function getTokens(
```

```

$string,
$offset = 0,
$defaultTokenName = null,
$groupDelimiters = '\'',
$groupNameDelimiter = ':' )
{

if ($offset >= strlen($string)) {
    //echo "offset out of range";
    return false;
}

$spaces = " \t\n\r"; // space characters

// add group delimiters to spaces...
$groupSpaces = $spaces . $groupNameDelimiter;
$delimiters = $groupSpaces . $groupDelimiters;

//var_dump($groupSpaces);

$string = ltrim(substr($string, $offset), $groupSpaces);
$token_strings = array();

//echo "String is : " . $string . "\n";

// 1. split all tokens...
while ($offset < strlen($string)) {
    $lastOffset = $offset;
    $escaped = false;

    if (false !== strpos($groupDelimiters, $char = $string[$offset])) {
        $groupChar = $char;
    } else {
        $groupChar = null;
    }

    if (null !== $groupChar) {
        while (($offset < strlen($string)) && (($groupChar !== ($char = $string[++$offset]))
|| $escaped)) {
            //$offset++;
            $escaped = ('\\"' === $char);
        }
        $offset++;
        //echo "**** Grouped : " . substr($string, $lastOffset, $offset - $lastOffset) . "\n";
    } else {
        while (($offset < strlen($string)) && ((false === strpos($delimiters, $char =
$string[$offset])) || $escaped)) {
            $offset++;
            $escaped = ('\\"' === $char);
        }
        //echo "**** Non-group : " . substr($string, $lastOffset, $offset - $lastOffset) .
"\n";
    }
    //skip spaces...
    while (($offset < strlen($string)) && ((false !== strpos($groupSpaces, $char =
$string[$offset])) || $escaped)) {
        $offset++;
        $escaped = ('\\"' === $char);
    }
}

```

```

    }

    $token_strings[] = substr($string, $lastOffset, $offset - $lastOffset);
    //echo "Next token = '" . end($token_strings) . "'\n";
}

$tokens = array();
$tokenName = null;
foreach ($token_strings as $token_str) {
    // clean $token_str
    $token_str = trim(stripslashes($token_str), $spaces);
    $str_value = trim($token_str, $delimiters);
    switch (strtolower($str_value)) {
        case 'true': $str_value = true; break;
        case 'false': $str_value = false; break;
        default: break;
    }

    // is it a token name?
    if (':' === substr($token_str, -1, 1)) {
        if (!empty($tokenName)) {
            $tokens[$tokenName] = '';
        }
        $tokenName = trim($token_str, $delimiters);
    } else {
        if (!empty($tokenName)) {
            if (isset($tokens[$tokenName])) {
                $tokens[$tokenName] = array(
                    $tokens[$tokenName],
                    $str_value
                );
            } else {
                $tokens[$tokenName] = $str_value;
            }
            $tokenName = null;
        } elseif (empty($defaultTokenName)) {
            $tokens[] = trim($token_str, $delimiters);
        } else {
            if (isset($tokens[$defaultTokenName])) {
                $tokens[$defaultTokenName] = array(
                    $tokens[$defaultTokenName],
                    $str_value
                );
            } else {
                $tokens[$defaultTokenName] = $str_value;
            }
        }
    }
}

if (!empty($tokenName)) {
    $tokens[$tokenName] = '';
}

return $tokens;
}

$str = "check1: test "
    . "check2:'hello world' "

```



```
. 'check3: "foo" '
. "check4: \\\"try this\\\""
. '"buz" '
. 'check1:true';
```

?>

[+ add a note](#)

- [Funciones de strings](#)
 - [addslashes](#)
 - [addslashes](#)
 - [bin2hex](#)
 - [chop](#)
 - [chr](#)
 - [chunk_split](#)
 - [convert_uuencode](#)
 - [convert_uuencode](#)
 - [count_chars](#)
 - [crc32](#)
 - [crypt](#)
 - [echo](#)
 - [explode](#)
 - [fprintf](#)
 - [get_html_translation_table](#)
 - [hebrew](#)
 - [hex2bin](#)
 - [html_entity_decode](#)
 - [htmlentities](#)
 - [htmlspecialchars_decode](#)
 - [htmlspecialchars](#)
 - [implode](#)
 - [join](#)
 - [lcfirst](#)
 - [levenshtein](#)
 - [localeconv](#)
 - [ltrim](#)
 - [md5_file](#)
 - [md5](#)
 - [metaphone](#)
 - [money_format](#)
 - [nl_langinfo](#)
 - [nl2br](#)
 - [number_format](#)
 - [ord](#)
 - [parse_str](#)
 - [print](#)
 - [printf](#)
 - [quoted_printable_decode](#)
 - [quoted_printable_encode](#)
 - [quotemeta](#)
 - [rtrim](#)
 - [setlocale](#)
 - [sha1_file](#)
 - [sha1](#)
 - [similar_text](#)
 - [soundex](#)
 - [sprintf](#)
 - [sscanf](#)

- [str_contains](#)
- [str_ends_with](#)
- [str_getcsv](#)
- [str_replace](#)
- [str_pad](#)
- [str_repeat](#)
- [str_replace](#)
- [str_rot13](#)
- [str_shuffle](#)
- [str_split](#)
- [str_starts_with](#)
- [str_word_count](#)
- [strcasecmp](#)
- [strchr](#)
- [strcmp](#)
- [strcoll](#)
- [strcspn](#)
- [strip_tags](#)
- [stripslashes](#)
- [stripos](#)
- [stripslashes](#)
- [stristr](#)
- [strlen](#)
- [strnatcasecmp](#)
- [strnatcmp](#)
- [strncasecmp](#)
- [strncmp](#)
- [strpbrk](#)
- [strpos](#)
- [strrchr](#)
- [strrev](#)
- [strripos](#)
- [strrpos](#)
- [strspn](#)
- [strstr](#)
- [strtok](#)
- [strtolower](#)
- [strtoupper](#)
- [strtr](#)
- [substr_compare](#)
- [substr_count](#)
- [substr_replace](#)
- [substr](#)
- [trim](#)
- [ucfirst](#)
- [ucwords](#)
- [utf8_decode](#)
- [utf8_encode](#)
- [vfprintf](#)
- [vprintf](#)
- [vsprintf](#)
- [wordwrap](#)
- Deprecated
 - [convert_cyr_string](#)
 - [hebrevc](#)
- [Copyright © 2001-2022 The PHP Group](#)
- [My PHP.net](#)

- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)
- [View Source](#)

