Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

var_dump »
« unserialize

- Руководство по PHP
- Справочник функций
- Модули, относящиеся к переменным и типам
- Обработка переменных
- Функции для работы с переменными

unset

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

unset — Удаляет переменную

Описание

unset(mixed `$var`, mixed `...$vars`): void

Языковая конструкция **unset()** удаляет перечисленные переменные.

Поведение языковой конструкции **unset()** внутри функции может меняться и зависит от типа удаляемой переменной.

Если глобальная переменная удаляется внутри функции, то будет удалена только локальная переменная. Переменная в области видимости вызова функции сохранит то же значение, что и до вызова **unset()**.

```
<?php
function destroy_foo()
{
global $foo;
unset($foo);
}

$foo = 'bar';
destroy_foo();
echo $foo;
?>
```

Результат выполнения приведённого примера:

```
bar
```

Чтобы удалить глобальную переменную внутри функции, используют массив *$GLOBALS*:

```
<?php
function foo()
{
unset($GLOBALS['bar']);
}

$bar = "something";
foo();
?>
```

Если переменная, которая передаётся ПО ССЫЛКЕ, удаляется внутри функции, то будет удалена только локальная переменная. Переменная в области видимости вызова функции сохранит то же значение, что и до вызова **unset()**.

```
<?php
function foo(&$bar)
{
unset($bar);
$bar = "blah";
}

$bar = 'something';
echo "$bar\n";

foo($bar);
echo "$bar\n";
```

```
?>
```

Результат выполнения приведённого примера:

```
something
something
```

Если статическая переменная удаляется внутри функции, конструкция **unset()** удалит переменную только в контексте оставшейся части функции. При следующем вызове предыдущее значение переменной будет восстановлено.

```
<?php
function foo()
{
static $bar;
$bar++;
echo "До удаления: $bar, ";
unset($bar);
$bar = 23;
echo "После удаления: $bar\n";
}

foo();
foo();
foo();
?>
```

Результат выполнения приведённого примера:

```
До удаления: 1, После удаления: 23
До удаления: 2, После удаления: 23
До удаления: 3, После удаления: 23
```

Список параметров

`var`

Удаляемая переменная.

`vars`

Следующие переменные.

Возвращаемые значения

Функция не возвращает значения после выполнения.

Примеры

Пример #1 Пример использования unset()

```
<?php
// удаляем одну переменную
unset($foo);

// удаляем один элемент массива
unset($bar['quux']);

// удаляем несколько переменных
unset($foo1, $foo2, $foo3);
?>
```

Пример #2 Использование приведения типа `(unset)`

Приведение типа `(unset)` часто путают с конструкцией **unset()**. Приведение типа `(unset)` приводит только к значению `NULL` и было добавлено только для полноты реализации. Оно не меняет значение приводимой переменной. С PHP 7.2.0 преобразование типа (unset) объявлено устаревшим и удалено в PHP 8.0.0.

```php
<?php
$name = 'Felipe';

var_dump((unset) $name);
var_dump($name);
?>
```

Результат выполнения приведённого примера:

```
NULL
string(6) "Felipe"
```

Примечания

> **Замечание**: Поскольку это языковая конструкция, а не функция, её нельзя вызывать как переменную функцию или передавать как именованный аргумент.

> **Замечание**:
>
> Можно удалить свойства объекта, видимые в текущем контексте.
>
> Если определены (в классе), метод __get() будет вызван при попытке получить неустановленное свойство, а метод __set() будет вызван при попытке установить неопределённое свойство.

> **Замечание**:
>
> Невозможно удалить псевдопеременную текущего контекста `$this` внутри метода объекта.

> **Замечание**:
>
> При вызове конструкции **unset()** на недоступных или необъявленных свойствах объекта будет вызван метод перегрузки __unset(), если он определён.

Смотрите также

- isset() - Определяет, была ли установлена переменная значением, отличным от null
- empty() - Проверяет, пуста ли переменная
- __unset()
- array_splice() - Удаляет часть массива и заменяет её чем-нибудь ещё
- Приведение типа (unset)

User Contributed Notes 32 notes

629
Phantasma_9 at yahoo dot com ¶
7 years ago
This is probably trivial but there is no error for unsetting a non-existing variable.

4
tecdoc at ukr dot net ¶
1 year ago
Be careful!, unset() element in array advances the internal array pointer one place forward

186
Hayley Watson ¶
10 years ago
You don't need to check that a variable is set before you unset it.

```
<?php
unset($a);
?>
```

is harmless.

```
<?php
if(isset($a)) {
unset($a);
}
?>
```

is pointless complication.

This doesn't apply to properties of objects that have __isset() methods that visibly change object state or __unset() methods that don't properly check their arguments or have extra side effects.

The latter case means that __unset shouldn't do more than what it says on the tin, and also has the responsibility for checking (possibly using __isset()) that what it's being asked to do makes sense.

The former case is just plain bad design.

up
down
50

nox at oreigon dot de ¶

15 years ago

if you try to unset an object, please be careful about references.

Objects will only free their resources and trigger their __destruct method when *all* references are unsetted.
Even when they are *in* the object... sigh!

```
<?php

class A {
function __destruct() {
echo "cYa later!!\n";
}
}

$a = new A();
$a -> a = $a;
#unset($a); # Just uncomment, and you'll see

echo "No Message ... hm, what now?\n";
unset($a -> a);
unset($a);

echo "Finally that thing is gone\n";

?>
```

Of course the object completely dies at the end of the script.

up
down
44

Kai Kunstmann ¶

15 years ago

Since unset() is a language construct, it cannot be passed anything other than a variable. It's sole purpose is to "unset" this variable, ie. to remove it from the current scope and destroy it's associated data. This is true especially for reference variables, where not the actual value is destroyed but the reference to that value. This is why you can't wrap 'unset()' in a user defined function: You would either unset a copy of the data if the parameter is passed by value, or you would just unset the reference variable within the functions scope if the parameter is passed by reference. There is no workaround for that, as you cannot pass 'scope' to a function in PHP. Such a function can only work for variables that exist in a common or global scope (compare 'unset($_GLOBALS[variable])').

I don't know how PHP handles garbage collection internally, but I guess this behavior can result in a huge memory leak: if a value variable goes out of scope with a second variable still holding a reference to the in-memory value, then unsetting that reference would still hold the value in memory but potentially unset the last reference to that in-memory data,

hence: occupied memory that is rendered useless as you cannot reference it anymore.

up

down

37

j dot vd dot merwe at enovision dot net ¶

13 years ago

A sample how to unset array elements from an array result coming from a mysql request. In this sample it is checking if a file exists and removes the row from the array if it not exists.

```
<?php
$db->set_query("select * from documents where document_in_user = 0"); //1
$documents = $db->result_to_array($db->get_result()); //1

foreach ($documents as $key => $row) { //2

$file = "uploads/".rawurldecode($row['document_name']);

if ( file_exists ( $file ) == FALSE ) {
unset($documents[$key]); //3
}
}

$documents = array_values($documents); // reindex the array (4)
?>

variables:
mysql table = documents,
array = $documents
array key (index) = $key
array row (record sort of speak) = $row

explanation:

1.
it gets the array from the table (mysql)

2.
foreach goes through the array $documents

3.
unset if record does not exist

4.
the array_values($documents) reindexes the $documents array, for otherwise you might end up in trouble when your process will start expecting an array starting with key ($key) 0 (zero).
```

up

down

34

franckraynal at free dot fr ¶

18 years ago

Here is another way to make 'unset' work with session variables from within a function :

```
<?php
function unsetSessionVariable ($sessionVariableName) {
unset($GLOBALS[_SESSION][$sessionVariableName]);
}
?>

May it work with others than me...
F.
```

up

down

33

andre at twg dot com dot au ¶

19 years ago

Only This works with register_globals being 'ON'.

```
unset( $_SESSION['variable'] );
```

The above will not work with register_globals turned on (will only work outside of a function).

```
$variable = $_SESSION['variable'];
unset( $_SESSION['variable'], $variable );
```

The above will work with register_globals on & inside a function

up

down

26

anon at no spam dot no address dot com ¶

19 years ago

Adding on to what bond at noellebond dot com said, if you want to remove an index from the end of the array, if you use unset, the next index value will still be what it would have been.

Eg you have

```
<?php
$x = array(1, 2);

for ($i = 0; $i < 5; $i++)
{
unset($x[(count($x)-1)]); //remove last set key in the array

$x[] = $i;
}
?>
```

You would expect:

```
Array([0] => 1, [1] => 4)
```

as you want it to remove the last set key....

but you actually get

```
Array ( [0] => 1 [4] => 2 [5] => 3 [6] => 4 )
```

This is since even though the last key is removed, the auto indexing still keeps its previous value.

The only time where this would not seem right is when you remove a value off the end. I guess different people would want it different ways.

The way around this is to use array_pop() instead of unset() as array_pop() refreshes the autoindexing thing for the array.

```
<?php
$x = array(1, 2);

for ($i = 0; $i < 5; $i++)
{
array_pop($x); // removes the last item in the array

$x[] = $i;
}
?>
```

This returns the expected value of x = Array([0] => 1, [1] => 4);

Hope this helps someone who may need this for some odd reason, I did.

up

down
25
hessodreamy at gmail dot com ¶
17 years ago
To clarify what hugo dot dworak at gmail dot com said about unsetting things that aren't already set:

unsetting a non-existent key within an array does NOT throw an error.

```
<?
$array = array();

unset($array[2]);
//this does not throw an error

unset($array[$undefinedVar]);
//Throws an error because of the undefined variable, not because of a non-existent key.
?>
```

up
down
21
pfreilly at umd dot umich dot edu ¶
12 years ago
Despite much searching, I have not yet found an explanation as to how one can manually free resources from variables, not so much objects, in PHP. I have also seen many comments regarding the merits and demerits of unset() versus setting a variable to null. Thus, here are the results of some benchmarks performed comparing unset() of numerous variables to setting them to null (with regards to memory usage and processing time):

```
10 variables:
Unset:
Memory Usage: 296
Time Elapsed: 1.0013580322266E-5

Null set:
Memory Usage: 1736
Time Elapsed: 5.9604644775391E-6

50 variables:
Unset:
Memory Usage: 296
Time Elapsed: 3.6001205444336E-5

Null set:
Memory Usage: 8328
Time Elapsed: 3.2901763916016E-5

100 variables:
Unset:
Memory Usage: 296
Time Elapsed: 5.6982040405273E-5

Null set:
Memory Usage: 15928
Time Elapsed: 5.8174133300781E-5

1000 variables:
Unset:
Memory Usage: 296
Time Elapsed: 0.00041294097900391

Null set:
Memory Usage: 168096
Time Elapsed: 0.00067591667175293
```

```
10000 variables:
Unset:
Memory Usage: 296
Time Elapsed: 0.0042569637298584

Null set:
Memory Usage: 1650848
Time Elapsed: 0.0076930522918701

100000 variables:
Unset:
Memory Usage: 296
Time Elapsed: 0.042603969573975

Null set:
Memory Usage: 16249080
Time Elapsed: 0.087724924087524

300000 variables:
Unset:
Memory Usage: 296
Time Elapsed: 0.13177299499512

Null set:
Memory Usage: 49796320
Time Elapsed: 0.28617882728577
```

Perhaps my test code for the null set was flawed, but despite that possibility it is simple to see that unset() has minimal processing time impact, and no apparent memory usage impact (unless the values returned by memory_get_usage() are flawed). If you truly care about the ~4 microseconds saved over <50 variables, more power to you. Otherwise, use unset() to minimize script impact on your system.
Note: Tested on PHP 5.3.8 installed via RPM on Fedora 14

up
down
18

clark at everettsconsulting dot com ¶

18 years ago

In PHP 5.0.4, at least, one CAN unset array elements inside functions from arrays passed by reference to the function. As implied by the manual, however, one can't unset the entire array by passing it by reference.

```
<?php
function remove_variable (&$variable) // pass variable by reference
{
unset($variable);
}

function remove_element (&$array, $key) // pass array by reference
{
unset($array[$key]);
}

$scalar = 'Hello, there';
echo 'Value of $scalar is: ';
print_r ($scalar); echo '<br />';
// Value of $scalar is: Hello, there

remove_variable($scalar); // try to unset the variable
echo 'Value of $scalar is: ';
print_r ($scalar); echo '<br />';
// Value of $scalar is: Hello, there

$array = array('one' => 1, 'two' => 2, 'three' => 3);
```

```
echo 'Value of $array is: ';
print_r ($array); echo '<br />';
// Value of $array is: Array ( [one] => 1 [two] => 2 [three] => 3 )

remove_variable($array); // try to unset the array
echo 'Value of $array is: ';
print_r ($array); echo '<br />';
// Value of $array is: Array ( [one] => 1 [two] => 2 [three] => 3 )

remove_element($array, 'two'); // successfully remove an element from the array
echo 'Value of $array is: ';
print_r ($array); echo '<br />';
// Value of $array is: Array ( [one] => 1 [three] => 3 )

?>
```

up
down
13

phpmanual at kennel17 dot co dot uk ¶

13 years ago

Note that PHP 4 will generate a warning if you try to unset an array index that doesn't exist and whose parent doesn't exist.

```
Example:

<?php

$foo = array();

unset($foo['Bar']['Baz']);

?>

RESULT: "Notice: Undefined index: Bar"

On PHP5 no error is raised, which seems to me like the correct behaviour.

Note that using unset($foo['Bar']) in the above example does not generate a warning in either version.

(Tested on 4.4.9 and 5.2.4)
```

up
down
11

Anonymous ¶

17 years ago

Just to confirm, USING UNSET CAN DESTROY AN ENTIRE ARRAY. I couldn't find reference to this anywhere so I decided to write this.

The difference between using unset and using $myarray=array(); to unset is that obviously the array will just be overwritten and will still exist.

```
<?php

$myarray=array("Hello","World");

echo $myarray[0].$myarray[1];

unset($myarray);
//$myarray=array();

echo $myarray[0].$myarray[1];
```

```
echo $myarray;
?>

Output with unset is:
<?
HelloWorld

Notice: Undefined offset: 0 in C:\webpages\dainsider\myarray.php on line 10

Notice: Undefined offset: 1 in C:\webpages\dainsider\myarray.php on line 10

Output with $myarray=array(); is:
?>

<?
HelloWorld

Notice: Undefined offset: 0 in C:\webpages\dainsider\myarray.php on line 10

Notice: Undefined offset: 1 in C:\webpages\dainsider\myarray.php on line 10

Array
?>
```

up
down
12

dan AT --nospam-- cubeland DOT co DOT uk ¶

19 years ago

dh at argosign dot de -
it is possible to unset globals from within functions thanks to the $GLOBALS array:

```
<?php
$x = 10;

function test() {
// don't need to do ' global $x; '
unset ($GLOBALS['x']);
echo 'x: ' . $GLOBALS['x'] . '<br />';
}

test();
echo "x: $x<br />";

// will result in
/*
x:
x:
*/
?>
```

up
down
9

pauljamescampbell at gmail dot com ¶

15 years ago

Here's my variation on the slightly dull unset method. It throws in a bit of 80's Stallone action spice into the mix.
Enjoy!

```
<?php
/**
* function rambo (first blood)
*
* Completely and utterly destroys everything, returning the kill count of victims
```

```
*
* @param It don't matter, it’s Rambo baby
* @return Integer Body count (but any less than 500 and it's not really worth mentioning)
*/
function rambo() {

// Get the victims and initiate that body count status
$victims = func_get_args();
$body_count = 0;

// Kill those damn punks
foreach($victims as $victim) {
if($death_and_suffering = @unset($victim)) {
$body_count++;
}
}

// How many kills did Rambo tally up on this mission?
return($body_count);
}
?>
```

up
down
9
lion_cat at mail ru ¶
15 years ago

```
about unset for arrays

if you unset the last array member
$ar[0]==2
$ar[1]==7
$ar[2]==9

unset ($ar[2])

after addition a new member by $ar[]=7,

you will get
$ar[0]==2
$ar[1]==7
$ar[3]==7,

So, unset has no effect to internal array counter!!!
```

up
down
11
thorry at thorry dot net ¶
19 years ago

```
The documentation is not entirely clear when it comes to static variables. It says:

If a static variable is unset() inside of a function, unset() destroys the variable and all its references.

<?php
function foo()
{
static $a;
$a++;
echo "$a\n";
unset($a);
}

foo();
```

```
foo();
foo();
?>
```

The above example would output:

```
1
2
3
```

And it does! But the variable is NOT deleted, that's why the value keeps on increasing, otherwise the output would be:

```
1
1
1
```

The references are destroyed within the function, this handeling is the same as with global variables, the difference is a static variable is a local variable.

Be carefull using unset and static values as the output may not be what you expect it to be. It appears to be impossible to destroy a static variable. You can only destroy the references within the current executing function, a successive static statement will restore the references.

The documentation would be better if it would say:
"If a static variable is unset() inside of a function, unset() destroys all references to the variable. "

Example: (tested PHP 4.3.7)

```
<?php
function foo()
{
static $a;
$a++;
echo "$a\n";
unset($a);
echo "$a\n";
static $a;
echo "$a\n";
}

foo();
foo();
foo();
?>
```

Would output:

```
1

1
2

2
3

3
```

up
down
4

Anonymous ¶

13 years ago

further I realized that an object, when getting detroyed, does care about destroying variable in object space visibility but not those in local visibility, be aware of the found pattern:

```
<?php
class release_test{
private $buffer;
private $other_object;
public function __construct(){
$this->other_object=new other_object_class();
}
public function __destruct(){
//note that you always have to unset class objects, in order to get the resources released
unset($this->other_object);
}
public allocate_mem_A(){
$this->buffer=file("/tmp/bigfile");
}
public allocate_mem_B(){
$buffer=file("/tmp/bigfile");
}
public allocate_mem_C(){
$buffer=file("/tmp/bigfile");
unset($buffer);
}
public allocate_mem_D(){
$this->other_buffer=file("/tmp/bigfile");
}
}

//this does not lead to a resource problem
$A=new release_test();
$A->allocate_mem_A();
$A->__destruct();
unset($A);

//this DOES lead to a resource problem
$B=new release_test();
$B->allocate_mem_B();
$B->__destruct();
unset($B);

//this does not lead to a resource problem
$C=new release_test();
$C->allocate_mem_C();
$C->__destruct();
unset($C);

//this does not lead to a resource problem
$D=new release_test();
$D->allocate_mem_D();
$D->__destruct();
unset($D);
?>
```

up
down
3

edouard dot berge at gmail dot com ¶

13 years ago

```
Warning!

When unset from an array, if you unset all elements, the array is always set

$tab=array('A'=>1,'B'=>2);
unset($tab['A']);
```

```
unset($tab['B']);
echo isset($tab)." ".count($tab);

output: 1 0
```

up
down
6

dibakar dot datta at gmail dot com ¶

17 years ago

Instead of using the unset function for unregistering your session or other array values you can also do this samll feature and get this task done with just 1 line code.

Suppose, if you like to unregister your session store values.
You can use:

```
$_SESSION = array();
```

Well this syntax saves lot's of time instead of unsetting each values.

up
down
7

warhog at warhog dot net ¶

20 years ago

you may wan't to unset all variables which are defined, here's one way:

```
<?php

function unset_all_vars($a)
{ foreach($a as $key => $val)
{ unset($GLOBALS[$key]); }
return serialize($a); }

unset_all_vars(get_defined_vars());

?>
```

you can also save than a serialized var of the "memory" and perhaps store this in a temporary file.. very usefull if you work with text files and/or file uploads when you've got very large variables.

greetz

up
down
3

Hayley Watson ¶

16 years ago

In regard to some confusion earlier in these notes about what causes unset() to trigger notices when unsetting variables that don't exist....

Unsetting variables that don't exist, as in

```
<?php
unset($undefinedVariable);
?>
```

does not trigger an "Undefined variable" notice. But

```
<?php
unset($undefinedArray[$undefinedKey]);
?>
```

triggers two notices, because this code is for unsetting an element of an array; neither $undefinedArray nor $undefinedKey are themselves being unset, they're merely being used to locate what should be unset. After all, if they did exist, you'd still expect them to both be around afterwards. You would NOT want your entire array to disappear just because you unset() one of its elements!

up
down

2

stacionari at gmail dot com ¶

15 years ago

Sometimes you need to assigne values to an array index in some loop (if, while, foreach etc.) but you wish to set starting index key to some number greater then zero (lets say 5). One idea how to do this is:

```
<?php
$values = array(5, 10, 15, 100); //array of values that we wish to add to our new array

$myArray = array(4=>0); //sets starting key to be 4 and assigns some value (lets say 0)
unset($myArray[4]); //delete this index key, but preserves further enumeration

foreach($values as $value){
$myArray[] = $value; //asign values to our array
}

print_r($myArray);

/* Output:

Array ( [5] => 5 [6] => 10 [7] => 15 [8] => 100 )

*/

?>
```

up

down

-1

Andreas ¶

13 years ago

You can not unset a numeric key of an array, if key is a string. See this example:

```
// Create a simple array with 3 different key types
$test[1] = array(
10 => array('apples'),
"20" => array('bananas'),
'30' => array('peaches')
);
$test[2] = (array) json_decode(json_encode($test[1]));
$test[3] = (array) (object) $test[1];
// array form a stdClass object
$testClass = new stdClass();
$testClass->{10} = array('apples');
$testClass->{"20"} = array('bananas');
$test[4] = (array) $testClass[6];

echo "<pre>";
foreach($test as $testNum => $arr) {

echo "\nTest: " . $testNum . " \n";
var_dump($arr);

foreach($arr as $key => $fruit) {
echo "key: " . $key . "\n";
echo "key exists: ";
var_dump(array_key_exists(strval($key), $arr));
echo "typeof key is: " . gettype($key) . "\n";

unset($arr[$key]);
}
var_dump($arr);
echo "\n" . str_repeat("-", 80);
```

```
}
echo "</pre>";
```

And here is the output:

```
Test: 1
array(3) {
[10]=>
array(1) {
[0]=>
string(6) "apples"
}
[20]=>
array(1) {
[0]=>
string(7) "bananas"
}
[30]=>
array(1) {
[0]=>
string(7) "peaches"
}
}
key: 10
key exists: bool(true)
typeof key is: integer
key: 20
key exists: bool(true)
typeof key is: integer
key: 30
key exists: bool(true)
typeof key is: integer
array(0) {
}

----------------------------------------------------------------
Test: 2
array(3) {
["10"]=>
array(1) {
[0]=>
string(6) "apples"
}
["20"]=>
array(1) {
[0]=>
string(7) "bananas"
}
["30"]=>
array(1) {
[0]=>
string(7) "peaches"
}
}
key: 10
key exists: bool(false)
typeof key is: string
key: 20
key exists: bool(false)
typeof key is: string
key: 30
key exists: bool(false)
```

```
typeof key is: string
array(3) {
["10"]=>
array(1) {
[0]=>
string(6) "apples"
}
["20"]=>
array(1) {
[0]=>
string(7) "bananas"
}
["30"]=>
array(1) {
[0]=>
string(7) "peaches"
}
}

---------------------------------------------------------------
Test: 3
array(3) {
[10]=>
array(1) {
[0]=>
string(6) "apples"
}
[20]=>
array(1) {
[0]=>
string(7) "bananas"
}
[30]=>
array(1) {
[0]=>
string(7) "peaches"
}
}
key: 10
key exists: bool(true)
typeof key is: integer
key: 20
key exists: bool(true)
typeof key is: integer
key: 30
key exists: bool(true)
typeof key is: integer
array(0) {
}

---------------------------------------------------------------
Test: 4
array(2) {
["10"]=>
array(1) {
[0]=>
string(6) "apples"
}
["20"]=>
array(1) {
[0]=>
string(7) "bananas"
```

```
}
}
key: 10
key exists: bool(false)
typeof key is: string
key: 20
key exists: bool(false)
typeof key is: string
array(2) {
["10"]=>
array(1) {
[0]=>
string(6) "apples"
}
["20"]=>
array(1) {
[0]=>
string(7) "bananas"
}
}

----------------------------------------------------------------

Fix the problem with a rebuild of the array:
$oldArray = $array();
$array = array();
foreach($oldArray as $key => $item) {
$array[intval($key)] = $item;
}
```

up
down
-1

macnimble at gmail dot com ¶
14 years ago

```
Two ways of unsetting values within an array:

<?php
# remove by key:
function array_remove_key ()
{
$args = func_get_args();
return array_diff_key($args[0],array_flip(array_slice($args,1)));
}
# remove by value:
function array_remove_value ()
{
$args = func_get_args();
return array_diff($args[0],array_slice($args,1));
}

$fruit_inventory = array(
'apples' => 52,
'bananas' => 78,
'peaches' => 'out of season',
'pears' => 'out of season',
'oranges' => 'no longer sold',
'carrots' => 15,
'beets' => 15,
);

echo "<pre>Original Array:\n",
print_r($fruit_inventory,TRUE),
```

```
'</pre>';

# For example, beets and carrots are not fruits...
$fruit_inventory = array_remove_key($fruit_inventory,
"beets",
"carrots");
echo "<pre>Array after key removal:\n",
print_r($fruit_inventory,TRUE),
'</pre>';

# Let's also remove 'out of season' and 'no longer sold' fruit...
$fruit_inventory = array_remove_value($fruit_inventory,
"out of season",
"no longer sold");
echo "<pre>Array after value removal:\n",
print_r($fruit_inventory,TRUE),
'</pre>';
?>
```

up
down
-2

chad 0x40 herballure 0x2e com ¶

16 years ago

It is observed on PHP 5.1.6 that <?php unset($this); ?> inside of a method will remove the reference to $this in that method. $this isn't considered "special" as far as unset() is concerned.

up
down
-12

tigercat at aol dot com ¶

11 years ago

The combination of "global" and "unset" in functions can lead to some unexpected results. This is because the "global" function creates a reference to a variable at the time it's executed, so a variable can be deleted out from under a "global $my_variable" declaration in a function. Accessing data that's been deleted with an obsolete reference is usually a bad thing; in some languages it can generate a machine address fault.

```
<?php
$my_global_var = "old data";
f1();

function f1() // example of invalid variable reference use
{
global $my_global_var; // creates reference to global variable
f2(); // recreates global variable, so reference is now invalid

// bad...
echo $my_global_var; // outputs "old data" (from invalid memory???)

// good...
global $my_global_var; // reestablish reference to new global variable
echo $my_global_var; // outputs "new data" as expected
}

function f2() // recreate global variable
{
unset($GLOBALS['my_global_var']); // this syntax works with all variable types including arrays
global $my_global_var; // must do this after unset to access new global variable
$my_global_var = "new data";
}
?>
```

up
down
-9

muhamad_zakaria at yahoo dot com ¶

18 years ago

We have experienced when we applied 'unset' to the overloaded properties (PHP5), consider the code below:

```
<?php
class TheObj {
public $RealVar1, $RealVar2, $RealVar3, $RealVar4;
public $Var = array();

function __set($var, $val) {
$this->Var[$var] = $val;
}
function __get($var) {
if(isset($this->Var[$var])) return $this->Var[$var];
else return -1;
}
}

$SomeObj = new TheObj;

// here we set for real variables
$SomeObj->RealVar1 = 'somevalue';
$SomeObj->{'RealVar2'} = 'othervalue';
$SomeObj->{'RealVar'.(3)} = 'othervaluetoo';
$SomeObj->{'RealVar'.'4'} = 'anothervalue';

// and here we set for virtual variables
$SomeObj->Virtual1 = 'somevalue';
$SomeObj->{'Virtual2'} = 'othervalue';
$SomeObj->{'Virtual'.(3)} = 'othervaluetoo';
$SomeObj->{'Virtual'.'4'} = 'anothervalue';

// now we will try to unset these variables
unset($SomeObj->RealVar1);
unset($SomeObj->{'RealVar'.(3)});

//the lines below will catch by '__get' magic method since these variables are unavailable anymore
print $SomeObj->RealVar1."\n";
print $SomeObj->{'RealVar'.(3)}."\n";

// now we will try to unset these variables
unset($SomeObj->Virtual1);
unset($SomeObj->{'Virtual'.(3)});

//but, these variables are still available??? eventhough they're "unset"-ed
print $SomeObj->Virtual1."\n";
print $SomeObj->{'Virtual'.(3)}."\n";
?>
```

Please note that PHP doesn't have magic callback to unset overloaded properties. This is the reason why unset($SomeObj->Virtual1) doesn't work.

But it does work when we set 'null' value such as the following code:

```
<?php
// now we will set 'null' value instead of using unset statement
$SomeObj->Virtual1 = null;
$SomeObj->{'Virtual'.(3)} = null;

// and now these variables are no longer available
print $SomeObj->Virtual1."\n";
print $SomeObj->{'Virtual'.(3)}."\n";
?>
```

Sound ugly, yeah?

This applied to the "virtual" array variable too, see more at http://bugs.php.net/bug.php?id=33513 (at feedback) about it.
PS: we used PHP version 5.1.0-dev from the CVS snapshot when we wrote the above codes.

up
down
-7

magnesium dot oxide dot play+php at gmail dot com ¶

9 years ago

You can unset superglobals like $GLOBALS, $_GET etc., but causing an unusual behavior (as of PHP 5.3.3).

1) unsetting of superglobals is done globally, i.e. unsetting inside the function affects GLOBALLY.

2) Recreation of unset'ed superglobals can be done (recreated valiables are superglobals), but original functionality (in $GLOBALS, $_SESSION ...) has lost.

```
<?php

function foo(){
unset($GLOBALS);
}

function bar(){
var_dump($GLOBALS);
}

foo();
bar(); //issues E_NOTICE ($GLOBALS not defined)
$GLOBALS=3;
bar(); //displays int(3)

?>
```

up
down
-3

ronan at shopping-feed dot com ¶

4 years ago

```
<?php

$list = ['a', 'b', 'c'];
next($list);
unset($list[1]);

echo current($list); // result : "c"
```

up
down
-8

ray.paseur sometimes uses gmail ¶

1 year ago

Perhaps because it is a language construct and not a function, if you try to use the error suppression operator ...

```
@unset($var);
```

... you get this: Parse error: syntax error, unexpected 'unset' (T_UNSET) ...

＋add a note

- Функции для работы с переменными
  - boolval
  - debug_zval_dump
  - doubleval
  - empty
  - floatval
  - get_debug_type