Search

Getting Started
- Introduction
- A simple tutorial

Language Reference
- Basic syntax
- Types
- Variables
- Constants
- Expressions
- Operators
- Control Structures
- Functions
- Classes and Objects
- Namespaces
- Enumerations
- Errors
- Exceptions
- Fibers
- Generators
- Attributes
- References Explained
- Predefined Variables
- Predefined Exceptions
- Predefined Interfaces and Classes
- Predefined Attributes
- Context options and parameters
- Supported Protocols and Wrappers

Security
- Introduction
- General considerations
- Installed as CGI binary
- Installed as an Apache module
- Session Security
- Filesystem Security
- Database Security
- Error Reporting
- User Submitted Data
- Hiding PHP
- Keeping Current

Features
- HTTP authentication with PHP
- Cookies
- Sessions
- Dealing with XForms
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- Command line usage

Garbage Collection
DTrace Dynamic Tracing

Function Reference
Affecting PHP's Behaviour
Audio Formats Manipulation
Authentication Services
Command Line Specific Extensions
Compression and Archive Extensions
Cryptography Extensions
Database Extensions
Date and Time Related Extensions
File System Related Extensions
Human Language and Character Encoding Support
Image Processing and Generation
Mail Related Extensions
Mathematical Extensions
Non-Text MIME Output
Process Control Extensions
Other Basic Extensions
Other Services
Search Engine Extensions
Server Specific Extensions
Session Extensions
Text Processing
Variable and Type Related Extensions
Web Services
Windows Only Extensions
XML Manipulation
GUI Extensions

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

Объекты »
« Числовые строки

- Руководство по PHP
- Справочник языка
- Типы

Change language: Russian

Массивы

Массив в PHP — это упорядоченная структура данных, которая связывает *значения* и *ключи*. Этот тип данных оптимизирован для разных целей, поэтому с ним работают как с массивом, списком (вектором), хеш-таблицей (реализацией карты), словарём, коллекцией, стеком, очередью и, возможно, чем-то ещё. Поскольку значениями массива бывают другие массивы, также доступны деревья и многомерные массивы.

Объяснение этих структур данных выходит за рамки этого руководства, но как минимум один пример будет приведён для каждой из них. За дополнительной информацией обращаются к большому объему литературы по этой обширной теме.

Синтаксис

Определение при помощи array()

Массив (array) создают языковой конструкцией array(). В качестве аргументов она принимает любое количество разделённых запятыми пар `ключ => значение`.

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

Запятая после последнего элемента массива необязательна и её можно опустить. Обычно это делается для однострочных массивов, — лучше предпочесть `array(1, 2)` вместо `array(1, 2, )`. Для многострочных массивов, наоборот, обычно указывают висящую запятую, так как упрощает добавление новых элементов в конец массива.

> **Замечание**:
>
> Существует короткий синтаксис массива, который заменяет языковую конструкцию `array()` выражение `[]`.

Пример #1 Простой массив

```
<?php

$array = array(
"foo" => "bar",
"bar" => "foo",
);

// Работа с коротким синтаксисом массива
$array = [
"foo" => "bar",
"bar" => "foo",
];

?>
```

Ключ массива (key) разрешено указывать либо как целочисленное значение (int), либо как строку (string). Значение массива (value) может принадлежать любому типу данных.

Дополнительно произойдут следующие преобразования ключа key:

- Строки (string), содержащие целое число (int) (исключая случаи, когда перед числом указывают знак +), будут преобразованы в целое число (int). Например, ключ со значением «`8`» сохранится со значением `8`. При этом, значение «`08`» не преобразуется, так как оно — не корректное десятичное целое.
- Числа с плавающей точкой (float) также преобразуются в целочисленные значения (int) — дробная часть будет отброшена. Например, ключ со значением `8.7` будет сохранится со значением `8`.
- Логический тип (bool) также преобразовывается в целочисленный тип (int). Например, ключ со значением **`true`** сохранится со значением `1`, а ключ со значением **`false`** сохранится со значением `0`.
- Тип null преобразуется в пустую строку. Например, ключ со значением `null` сохранится со значением `""`.
- Массивы (array) и объекты (object) *нельзя* указывать как ключи. Это сгенерирует предупреждение: `Недопустимый`

```
тип смещения (Illegal offset type).
```

Если нескольким элементам в объявлении массива указан одинаковый ключ, то только последний будет сохранён, а другие будут перезаписаны.

Пример #2 Пример преобразования типов и перезаписи элементов

```php
<?php

$array = array(
1 => "a",
"1" => "b",
1.5 => "c",
true => "d",
);
var_dump($array);

?>
```

Результат выполнения приведённого примера:

```
array(1) {
  [1]=>
  string(1) "d"
}
```

Поскольку все ключи в приведённом примере преобразуются к `1`, значение будет перезаписано на каждый новый элемент и останется только последнее присвоенное значение — «`d`».

PHP разрешает массивам содержать одновременно целочисленные (int) и строковые (string) ключи, поскольку PHP одинаково воспринимает индексные и ассоциативные массивы.

Пример #3 Смешанные целочисленные (int) и строковые (string) ключи

```php
<?php

$array = array(
"foo" => "bar",
"bar" => "foo",
100 => -100,
-100 => 100,
);
var_dump($array);

?>
```

Результат выполнения приведённого примера:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Ключ (key) — необязателен. Если он не указан, PHP инкрементирует предыдущее наибольшее целочисленное (int) значение ключа.

Пример #4 Индексные массивы без ключа

```php
<?php

$array = array("foo", "bar", "hallo", "world");
var_dump($array);
```

```
?>
```

Результат выполнения приведённого примера:

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hallo"
  [3]=>
  string(5) "world"
}
```

Разрешено указывать ключ одним элементам и пропускать для других:

Пример #5 Ключи для некоторых элементов

```
<?php

$array = array(
"a",
"b",
6 => "c",
"d",
);
var_dump($array);

?>
```

Результат выполнения приведённого примера:

```
array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Видно, что последнее значение «d» присвоилось ключу 7. Это произошло потому, что перед этим самым большим значением целочисленного ключа было 6.

Пример #6 Расширенный пример преобразования типов и перезаписи элементов

```
<?php

$array = array(
1 => 'a',
'1' => 'b', // значение «b» перезапишет значение «a»
1.5 => 'c', // значение «c» перезапишет значение «b»
-1 => 'd',
'01' => 'e', // поскольку это не целочисленная строка, она НЕ перезапишет ключ 1
'1.5' => 'f', // поскольку это не целочисленная строка, она НЕ перезапишет ключ 1
true => 'g', // значение «g» перезапишет значение «c»
false => 'h',
'' => 'i',
null => 'j', // значение «j» перезапишет значение «i»
'k', // значение «k» присваивается ключу 2. Потому что самый большой целочисленный ключ до этого был 1
2 => 'l', // значение «l» перезапишет значение «k»
);

var_dump($array);

?>
```

Результат выполнения приведённого примера:

```
array(7) {
  [1]=>
  string(1) "g"
  [-1]=>
  string(1) "d"
  ["01"]=>
  string(1) "e"
  ["1.5"]=>
  string(1) "f"
  [0]=>
  string(1) "h"
  [""]=>
  string(1) "j"
  [2]=>
  string(1) "l"
}
```

Этот пример включает все вариации преобразования ключей и перезаписи элементов

Доступ к элементам массива через синтаксис квадратных скобок

Доступ к элементам массива разрешено получать, используя синтакс `array[key]`.

Пример #7 Доступ к элементам массива

```
<?php

$array = array(
"foo" => "bar",
42 => 24,
"multi" => array(
"dimensional" => array(
"array" => "foo"
)
)
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);

?>
```

Результат выполнения приведённого примера:

```
string(3) "bar"
int(24)
string(3) "foo"
```

> **Замечание**:
>
> До PHP 8.0.0 квадратные и фигурные скобки могли взаимозаменяться при доступе к элементам массива (например, в приведённом примере `$array[42]` и `$array{42}` делали одно и то же). Синтаксис фигурных скобок устарел с PHP 7.4.0 и больше не поддерживается с PHP 8.0.0.

Пример #8 Разыменование массива

```
<?php

function getArray() {
return array(1, 2, 3);
}

$secondElement = getArray()[1];

?>
```

Замечание:

Попытка доступа к неопределённому ключу в массиве — это то же самое, что и попытка доступа к любой другой неопределённой переменной: будет выдана ошибка уровня **`E_WARNING`** (ошибка уровня **`E_NOTICE`** до PHP 8.0.0), и результат будет равен **`null`**.

Замечание:

Попытка разыменовать не массив, а скалярное значение, которое отличается от строки (string), отдаст **`null`**, тогда как разыменовывание строки (string) трактует её как индексный массив. При такой попытке до PHP 7.4.0 не выдавалось сообщение об ошибке. С PHP 7.4.0 выдаётся ошибка уровня **`E_NOTICE`**; с PHP 8.0.0 выдаётся ошибка уровня **`E_WARNING`**.

Создание и модификация с применением синтаксиса квадратных скобок

Разработчик может изменять существующий массив явной установкой значений.

Это делается путём присвоения значений массиву (array) с указанием ключа в квадратных скобках. Кроме того, если опустить ключ, получится пустая пара скобок (`[]`).

```
$arr[key] = value;
$arr[] = value;
// Ключ key может принадлежать типу int или string
// Значение value может быть любым значением любого типа
```

Если массив *$arr* ещё не существует или для него задано значение **`null`** или **`false`**, он будет создан. Таким образом, это ещё один способ определить массив array. Однако такой способ применять не рекомендовано, так как если переменная *$arr* уже содержит значение (например, строку (string) из переменной запроса), то это значение останется на месте и выражение `[]` может означать доступ к символу в строке. Лучше инициализировать переменную явным присваиванием значения.

Замечание: Начиная с PHP 7.1.0 оператор пустого индекса на строке выбросит фатальную ошибку. Раньше строка молча преобразовывалась в массив.

Замечание: С PHP 8.1.0 способ, которым создавали новый массив приведением к нему значения **`false`**, устарел. Способ, которым создают новый массив приведением к нему значения **`null`** и неопределённого значения, по-прежнему доступен.

Для изменения конкретного значения элементу просто присваивают новое значение, указывая его ключ. Если нужно удалить пару ключ/значение, необходимо вызывать конструкцию unset().

```
<?php

$arr = array(5 => 1, 12 => 2);

$arr[] = 56; // В этом месте скрипта это
// то же самое, что и $arr[13] = 56;

$arr["x"] = 42; // Это добавляет в массив новый
// элемент с ключом «x»

unset($arr[5]); // Это удаляет элемент из массива

unset($arr); // Это удаляет весь массив

?>
```

Замечание:

Как было сказано ранее, если разработчик не указал ключ, то будет взят максимальный из существующих целочисленных (int) индексов, и новым ключом будет это максимальное значение (в крайнем случае 0) плюс 1. Если целочисленных (int) индексов ещё нет, то ключом будет `0` (ноль).

Учитывают, что максимальное целое значение ключа *не обязательно существует в массиве в текущий момент*. Оно могло существовать в массиве какое-то время с момента последней переиндексации. Следующий пример это иллюстрирует:

```php
<?php

// Создаём простой массив.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Теперь удаляем каждый элемент, но массив оставляем нетронутым:
foreach ($array as $i => $value) {
unset($array[$i]);
}
print_r($array);

// Добавляем элемент (обратите внимание, что новым ключом будет 5, а не 0).
$array[] = 6;
print_r($array);

// Переиндексация:
$array = array_values($array);
$array[] = 7;
print_r($array);

?>
```

Результат выполнения приведённого примера:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Деструктуризация массива

Массивы разрешено деструктурировать языковыми конструкциями `[]` (начиная с PHP 7.1.0) или list(). Эти языковые конструкции разрешено использовать для деструктуризации массива на отдельные переменные.

```php
<?php

$source_array = ['foo', 'bar', 'baz'];
[$foo, $bar, $baz] = $source_array;
echo $foo; // выведет «foo»
echo $bar; // выведет «bar»
echo $baz; // выведет «baz»

?>
```

Деструктуризацию массива также выполняют в конструкции foreach для деструктуризации многомерного массива во время итерации по массиву.

```php
<?php

$source_array = [
[1, 'John'],
```

```
[2, 'Jane'],
];
foreach ($source_array as [$id, $name]) {
// логика работы с $id и $name
}

?>
```

Элементы массива будут проигнорированы, если переменная не указана. Деструктуризация массива начинается с индекса `0`.

```
<?php

$source_array = ['foo', 'bar', 'baz'];

// Присваивание элемента с индексом 2 переменной $baz
[, , $baz] = $source_array;

echo $baz; // выведет "baz"

?>
```

С PHP 7.1.0 ассоциативные массивы также разрешено деструктурировать. Это упрощает выбор нужного элемента в массивах с числовым индексом, так как индекс может быть указан явно.

```
<?php

$source_array = ['foo' => 1, 'bar' => 2, 'baz' => 3];

// Присваивание элемента с индексом «baz» переменной $three
['baz' => $three] = $source_array;

echo $three; // выведет 3

$source_array = ['foo', 'bar', 'baz'];

// Присваивание элемента с индексом 2 переменной $baz
[2 => $baz] = $source_array;

echo $baz; // выведет «baz»

?>
```

Деструктуризацией массив пользуются, чтобы поменять две переменные местами.

```
<?php

$a = 1;
$b = 2;
[$b, $a] = [$a, $b];
echo $a; // выведет 2
echo $b; // выведет 1

?>
```

> **Замечание**:
>
> Оператор ... не поддерживается в присваиваниях.

> **Замечание**:
>
> Попытка получить доступ к неопределённому ключу массива аналогична обращению к любой другой неопределённой переменной: будет выдано сообщение об ошибке уровня **`E_WARNING`** (ошибки уровня **`E_NOTICE`** до PHP 8.0.0), а результатом будет значение **`null`**.

Полезные функции

Для работы с массивами есть довольного много полезных функций. Подробнее об этом рассказано в разделе «Функции для работы с массивами».

> **Замечание**:
>
> Языковая конструкция unset() умеет удалять ключи массива. Обратите внимание, что массив *НЕ* будет переиндексирован. Если нужно поведение в стиле «удалить и сдвинуть», можно переиндексировать массив функцией array_values().

```
<?php

$a = array(1 => 'один', 2 => 'два', 3 => 'три');
unset($a[2]);
/* даст массив, представленный так:
$a = array(1 => 'один', 3 => 'три');
а НЕ так:
$a = array(1 => 'один', 2 => 'три');
*/

$b = array_values($a);
// Теперь $b это array(0 => 'один', 1 => 'три')
?>
```

Управляющая конструкция foreach существует специально для массивов. Она предлагает простой способ перебора массива.

Что можно и нельзя делать с массивами

Почему выражение `$foo[bar]` неверно?

Рекомендовано заключать в кавычки строковый литерал в индексе ассоциативного массива. Например, нужно писать `$foo['bar']`, а не `$foo[bar]`. Но почему? Часто в старых скриптах можно встретить следующий синтаксис:

```
<?php

$foo[bar] = 'враг';
echo $foo[bar];
// и т. д.

?>
```

Это неверно, хотя и работает. Причина в том, что этот код содержит неопределённую константу (`bar`), а не строку (`'bar'` — обратите внимание на кавычки). Это работает, потому что PHP автоматически преобразовывает *«голую строку»* (не заключённую в кавычки строку, которая не соответствует ни одному из известных символов языка) в строку со значением этой «голой строки». Например, если константа с именем **`bar`** не определена, то PHP заменит bar на строку «`bar`» и будет работать с ней.

Внимание

Резервный вариант для обработки неопределённой константы как пустой строки выдаёт ошибку уровня **`E_NOTICE`**. Начиная с PHP 7.2.0 поведение объявлено устаревшим и выдаёт ошибку уровня **`E_WARNING`**. Начиная с PHP 8.0.0 удалено и выбрасывает исключение Error.

> **Замечание**: Это не значит, что нужно *всегда* заключать ключ в кавычки. Не обязательно заключать в кавычки константы или переменные, поскольку это помешает PHP обрабатывать их.

```
<?php

error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);

// Простой массив:
```

```
$array = array(1, 2);
$count = count($array);

for ($i = 0; $i < $count; $i++) {
echo "\nПроверяем $i: \n";
echo "Плохо: " . $array['$i'] . "\n";
echo "Хорошо: " . $array[$i] . "\n";
echo "Плохо: {$array['$i']}\n";
echo "Хорошо: {$array[$i]}\n";
}

?>
```

Результат выполнения приведённого примера:

```
Проверяем 0:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 1
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 1

Проверяем 1:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Плохо:
Хорошо: 2
Notice: Undefined index:  $i in /path/to/script.html on line 11
Плохо:
Хорошо: 2
```

Дополнительные примеры, которые подтверждают этот факт:

```php
<?php

// Показываем все ошибки
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Верно
print $arr['fruit']; // apple
print $arr['veggie']; // carrot

// Неверно. Это работает, но из-за неопределённой константы с
// именем fruit также выдаёт ошибку PHP уровня E_NOTICE
//
// Notice: Use of undefined constant fruit - assumed 'fruit' in...
print $arr[fruit]; // apple

// Давайте определим константу, чтобы продемонстрировать, что
// происходит. Присвоим константе с именем fruit значение «veggie».
define('fruit', 'veggie');

// Теперь обратите внимание на разницу
print $arr['fruit']; // apple
print $arr[fruit]; // carrot

// Внутри строки это нормально. Внутри строк константы не
// рассматриваются, так что ошибки E_NOTICE здесь не произойдёт
print "Hello $arr[fruit]"; // Hello apple

// С одним исключением: фигурные скобки вокруг массивов внутри
// строк позволяют константам там находиться
print "Hello {$arr[fruit]}"; // Hello carrot
print "Hello {$arr['fruit']}"; // Hello apple
```

```php
// Это не будет работать и вызовет ошибку обработки:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or T_NUM_STRING'
// Это, конечно, также действует и с суперглобальными переменными в строках
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";

// Ещё одна возможность — конкатенация
print "Hello " . $arr['fruit']; // Hello apple

?>
```

Если директива error_reporting настроена на режим отображения ошибок уровня **`E_NOTICE`** (например, **`E_ALL`**), ошибки сразу будут видны. По умолчанию директива error_reporting настроена на то, чтобы не показывать предупреждения.

Как указано в разделе о синтаксисе, внутри квадратных скобок («`[`» и «`]`») должно быть выражение. То есть вот такой код работает:

```php
<?php

echo $arr[somefunc($bar)];

?>
```

Это пример работы с возвращаемым функцией значением в качестве индекса массива. PHP также знает о константах:

```php
<?php

$error_descriptions[E_ERROR] = "Произошла фатальная ошибка";
$error_descriptions[E_WARNING] = "PHP сообщает о предупреждении";
$error_descriptions[E_NOTICE] = "Это лишь неофициальное замечание";

?>
```

Обратите внимание, что **`E_ERROR`** — это такой же допустимый идентификатор, как и `bar` в первом примере. Но последний пример по существу эквивалентен такой записи:

```php
<?php

$error_descriptions[1] = "Произошла фатальная ошибка";
$error_descriptions[2] = "PHP выдал предупреждение";
$error_descriptions[8] = "Это просто уведомление";

?>
```

поскольку значение константы **`E_ERROR`** соответствует значению `1` и т. д.

Так что же в этом плохого?

Когда-нибудь в будущем команда разработчиков PHP, возможно, захочет добавить ещё одну константу или ключевое слово, либо константа из другого кода может вмешаться. Например, неправильно использовать слова `empty` и `default`, поскольку они относятся к зарезервированным ключевым словам.

> **Замечание**: Повторим, внутри строки (string) в двойных кавычках допустимо не окружать индексы массива кавычками, поэтому «`$foo[bar]`» — допустимая запись. В примерах выше объяснено, почему, дополнительная информация дана в разделе об обработке переменных в строках.

Преобразование в массив

Преобразование целого числа (int), числа с плавающей точкой (float), строки (string), логического значения (bool) или ресурса (resource) в массив — создаёт массив с одним элементом с индексом 0 и значением скаляра, который был преобразован. Говоря по-другому, выражение `(array) $scalarValue` аналогично выражению `array($scalarValue)`.

Если объект (object) будет преобразован в массив, элементами массива будут свойства (переменные-члены) этого

объекта. Ключами будут имена переменных-членов, со следующими примечательными исключениями: целочисленные свойства станут недоступны; к закрытым полям класса (private) в начало будет дописано имя класса; к защищённым полям класса (protected) в начало будет добавлен символ '*'. Эти добавленные с обоих сторон значения также получат NUL-байты. Неинициализированные типизированные свойства автоматически отбрасываются.

```
<?php

class A {
private $B;
protected $C;
public $D;
function __construct()
{
$this->{1} = null;
}
}
var_export((array) new A());

?>
```

Результат выполнения приведённого примера:

```
array (
  '' . "\0" . 'A' . "\0" . 'B' => NULL,
  '' . "\0" . '*' . "\0" . 'C' => NULL,
  'D' => NULL,
  1 => NULL,
)
```

Это может вызвать несколько неожиданное поведение:

```
<?php

class A {
private $A; // Это станет '\0A\0A'
}
class B extends A {
private $A; // Это станет '\0B\0A'
public $AA; // Это станет 'AA'
}
var_dump((array) new B());

?>
```

Результат выполнения приведённого примера:

```
array(3) {
  ["BA"]=>
  NULL
  ["AA"]=>
  NULL
  ["AA"]=>
  NULL
}
```

Приведённый код покажет 2 ключа с именем «AA», хотя один из них на самом деле имеет имя «\0A\0A».

Если преобразовать в массив значение **null**, получится пустой массив.

Сравнение

Массивы сравнивают функцией array_diff() и операторами массивов.

Распаковка массива

Массив, перед которым указан оператор ..., будет распакован во время определения массива. Только массивы и объекты, которые реализуют интерфейс Traversable, разрешено распаковывать. Распаковка массива оператором ... доступна начиная с PHP 7.4.0.

Массив разрешено распаковывать несколько раз и добавлять обычные элементы до или после оператора ...:

Пример #9 Простая распаковка массива

```
<?php

// Применение короткого синтаксиса массива.
// Работает также с синтаксисом array().
$arr1 = [1, 2, 3];
$arr2 = [...$arr1]; // [1, 2, 3]
$arr3 = [0, ...$arr1]; // [0, 1, 2, 3]
$arr4 = [...$arr1, ...$arr2, 111]; // [1, 2, 3, 1, 2, 3, 111]
$arr5 = [...$arr1, ...$arr1]; // [1, 2, 3, 1, 2, 3]

function getArr() {
return ['a', 'b'];
}

$arr6 = [...getArr(), 'c' => 'd']; // ['a', 'b', 'c' => 'd']

?>
```

Распаковка массива оператором ... соблюдает семантику функции array_merge(). То есть более поздние строковые ключи перезаписывают более ранние, а целочисленные ключи перенумеровываются:

Пример #10 Распаковка массива с дублирующим ключом

```
<?php

// строковый ключ
$arr1 = ["a" => 1];
$arr2 = ["a" => 2];
$arr3 = ["a" => 0, ...$arr1, ...$arr2];
var_dump($arr3); // ["a" => 2]

// целочисленный ключ
$arr4 = [1, 2, 3];
$arr5 = [4, 5, 6];
$arr6 = [...$arr4, ...$arr5];
var_dump($arr6); // [1, 2, 3, 4, 5, 6]
// Который [0 => 1, 1 => 2, 2 => 3, 3 => 4, 4 => 5, 5 => 6]
// где исходные целочисленные ключи не были сохранены.

?>
```

> **Замечание**:
>
> Ключи, тип которых не принадлежит ни целыми числами, ни строками, выбрасывают исключение TypeError. Такие ключи генерируются только объектом Traversable.

> **Замечание**:
>
> До PHP 8.1 распаковка массива со строковым ключом не поддерживалась:
>
> ```
> <?php
>
> $arr1 = [1, 2, 3];
> $arr2 = ['a' => 4];
> $arr3 = [...$arr1, ...$arr2];
> // Fatal error: Uncaught Error: Cannot unpack array with string keys in example.php:5
> $arr4 = [1, 2, 3];
> $arr5 = [4, 5];
> $arr6 = [...$arr4, ...$arr5]; // работает. [1, 2, 3, 4, 5]
>
> ?>
> ```

Примеры

Массив в PHP — гибкий тип данных. Вот несколько примеров:

```php
<?php

// Этот код:
$a = array( 'color' => 'красный',
'taste' => 'сладкий',
'shape' => 'круг',
'name' => 'яблоко',
4 // ключом будет 0
);

$b = array('a', 'b', 'c');

// ...эквивалентен этому:
$a = array();
$a['color'] = 'красный';
$a['taste'] = 'сладкий';
$a['shape'] = 'круг';
$a['name'] = 'яблоко';
$a[] = 4; // ключом будет 0

$b = array();
$b[] = 'a';
$b[] = 'b';
$b[] = 'c';

// после выполнения приведённого кода, переменная $a будет массивом
// array('color' => 'красный', 'taste' => 'сладкий', 'shape' => 'круг',
// 'name' => 'яблоко', 0 => 4), а переменная $b будет
// array(0 => 'a', 1 => 'b', 2 => 'c'), или просто array('a', 'b', 'c').

?>
```

Пример #11 Вызов языковой конструкции array()

```php
<?php

// Массив как карта (свойств)
$map = array(
'version' => 4,
'OS' => 'Linux',
'lang' => 'english',
'short_tags' => true
);

// строго числовые ключи
$array = array(
7,
8,
0,
156,
-10
);
// это то же самое, что и array(0 => 7, 1 => 8, ...)

$switching = array(
10, // ключ = 0
5 => 6,
3 => 7,
```

```
'a' => 4,
11, // ключ = 6 (максимальным числовым индексом было 5)
'8' => 2, // ключ = 8 (число!)
'02' => 77, // ключ = '02'
0 => 12 // значение 10 будет перезаписано на 12
);

// пустой массив
$empty = array();

?>
```

Пример #12 Коллекция

```
<?php

$colors = array('красный', 'голубой', 'зелёный', 'жёлтый');

foreach ($colors as $color) {
echo "Вам нравится $color?\n";
}

?>
```

Результат выполнения приведённого примера:

```
Вам нравится красный?
Вам нравится голубой?
Вам нравится зелёный?
Вам нравится жёлтый?
```

Непосредственное изменение значений массива допустимо через передачу значений по ссылке.

Пример #13 Изменение элемента в цикле

```
<?php

foreach ($colors as &$color) {
$color = mb_strtoupper($color);
}
unset($color); /* это нужно, чтобы очередные записи в
переменной $color не меняли последний элемент массива */

print_r($colors);

?>
```

Результат выполнения приведённого примера:

```
Array
(
    [0] => КРАСНЫЙ
    [1] => ГОЛУБОЙ
    [2] => ЗЕЛЁНЫЙ
    [3] => ЖЁЛТЫЙ
)
```

Следующий пример создаёт массив, индексация которого начинается с единицы.

Пример #14 Индекс, начинающийся с единицы

```
<?php

$firstquarter = array(1 => 'Январь', 'Февраль', 'Март');
print_r($firstquarter);

?>
```

Результат выполнения приведённого примера:

```
Array
(
    [1] => 'Январь'
    [2] => 'Февраль'
    [3] => 'Март'
)
```

Пример #15 Заполнение массива

```php
<?php

// заполняем массив всеми элементами из директории
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
$files[] = $file;
}
closedir($handle);

?>
```

Массивы упорядочены. Порядок изменяют разными функциями сортировки. Подробнее об этом рассказано в разделе «Функции для работы с массивами». Для подсчёта количества элементов в массиве вызывают функцию count().

Пример #16 Сортировка массива

```php
<?php

sort($files);
print_r($files);

?>
```

Поскольку значению массива разрешено быть любым, значение может быть также другим массивом. Поэтому разрешено создавать рекурсивные и многомерные массивы.

Пример #17 Рекурсивные и многомерные массивы

```php
<?php

$fruits = array ( "fruits" => array ( "a" => "апельсин",
"b" => "банан",
"c" => "яблоко"
),
"numbers" => array ( 1,
2,
3,
4,
5,
6
),
"holes" => array ( "первая",
5 => "вторая",
"третья"
)
);

// Несколько примеров доступа к значениям предыдущего массива
echo $fruits["holes"][5]; // напечатает «вторая»
echo $fruits["fruits"]["a"]; // напечатает «апельсин»
unset($fruits["holes"][0]); // удалит «первая»

// Создаст новый многомерный массив
$juices["apple"]["green"] = "хороший";
```

```
?>
```

Присваивание массива включает копирование значения. Чтобы скопировать массив по ссылке, указывают оператор присваивания по ссылке.

```
<?php

$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // Массив $arr2 изменился,
// Массив $arr1 всё ещё выглядит так: array(2, 3)

$arr3 = &$arr1;
$arr3[] = 4; // Теперь массивы $arr1 и $arr3 одинаковы

?>
```


User Contributed Notes 5 notes

128
mlvljr ¶
12 years ago
please note that when arrays are copied, the "reference status" of their members is preserved (http://www.php.net/manual/en/language.references.whatdo.php).

71
thomas tulinsky ¶
7 years ago
I think your first, main example is needlessly confusing, very confusing to newbies:

```
$array = array(
"foo" => "bar",
"bar" => "foo",
);
```

It should be removed.

For newbies:
An array index can be any string value, even a value that is also a value in the array.
The value of array["foo"] is "bar".
The value of array["bar"] is "foo"

The following expressions are both true:
$array["foo"] == "bar"
$array["bar"] == "foo"

58
ken underscore yap atsign email dot com ¶
16 years ago
"If you convert a NULL value to an array, you get an empty array."

This turns out to be a useful property. Say you have a search function that returns an array of values on success or NULL if nothing found.

```
<?php $values = search(...); ?>
```

Now you want to merge the array with another array. What do we do if $values is NULL? No problem:

```
<?php $combined = array_merge((array)$values, $other); ?>

Voila.
```

up
down
53

jeff splat codedread splot com ¶
18 years ago

```
Beware that if you're using strings as indices in the $_POST array, that periods are transformed into underscores:

<html>
<body>
<?php
printf("POST: "); print_r($_POST); printf("<br/>");
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="hidden" name="Windows3.1" value="Sux">
<input type="submit" value="Click" />
</form>
</body>
</html>

Once you click on the button, the page displays the following:

POST: Array ( [Windows3_1] => Sux )
```

up
down
43

chris at ocportal dot com ¶
10 years ago

```
Note that array value buckets are reference-safe, even through serialization.

<?php
$x='initial';
$test=array('A'=>&$x,'B'=>&$x);
$test=unserialize(serialize($test));
$test['A']='changed';
echo $test['B']; // Outputs "changed"
?>

This can be useful in some cases, for example saving RAM within complex structures.
```

＋add a note

- Типы
  - Введение
  - Система типов
  - NULL
  - Логические значения
  - Целые числа
  - Числа с плавающей точкой
  - Строки
  - Числовые строки
  - Массивы
  - Объекты
  - Перечисления
  - Ресурсы
  - Callable и callback-функции
  - Mixed
  - Void
  - Never
  - Относительные типы классов

- Типы значений
- Итерируемые значения
- Объявления типов
- Манипуляции с типами