# PHP Options and Information

# Introduction

This functions enable you to get a lot of information about PHP itself, e.g. runtime configuration, loaded extensions, version and much more. You'll also find functions to set options for your running PHP. The probably best known function of PHP - phpinfo() - can be found here.

# Installing/Configuring

## Requirements

No external libraries are needed to build this extension.

## Installation

There is no installation needed to use these functions; they are part of the PHP core.

## Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

### PHP Options/Inf Configuration Options

| Name | Default | Changeable | Changelog |
|---|---|---|---|
| assert.active | "1" | PHP_INI_ALL | |
| assert.bail | "0" | PHP_INI_ALL | |
| assert.warning | "1" | PHP_INI_ALL | |
| assert.callback | NULL | PHP_INI_ALL | |
| assert.quiet_eval | "0" | PHP_INI_ALL | |
| enable_dl | "1" | PHP_INI_SYSTEM | Removed in PHP 6.0.0. |
| max_execution_time | "30" | PHP_INI_ALL | |
| max_input_time | "-1" | PHP_INI_PERDIR | Available since PHP 4.3.0. |
| max_input_nesting_level | "64" | PHP_INI_PERDIR | Available since PHP 4.4.8. Removed in PHP 5.0.0. |
| magic_quotes_gpc | "1" | PHP_INI_PERDIR | PHP_INI_ALL in PHP <= 4.2.3. Removed in PHP 6.0.0. |

| | | | |
|---|---|---|---|
| magic_quotes_runtime | "0" | PHP_INI_ALL | Removed in PHP 6.0.0. |

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

*assert.active* boolean
Enable assert() evaluation.

*assert.bail* boolean
Terminate script execution on failed assertions.

*assert.warning* boolean
Issue a PHP warning for each failed assertion.

*assert.callback* string
user function to call on failed assertions

*assert.quiet_eval* boolean
Use the current setting of error_reporting() during assertion expression evaluation. If enabled, no errors are shown (implicit error_reporting(0)) while evaluation. If disabled, errors are shown according to the settings of error_reporting()

*enable_dl* boolean
This directive is really only useful in the Apache module version of PHP. You can turn dynamic loading of PHP extensions with dl() on and off per virtual server or per directory. The main reason for turning dynamic loading off is security. With dynamic loading, it's possible to ignore all open_basedir restrictions. The default is to allow dynamic loading, except when using safe mode. In safe mode, it's always impossible to use dl().

*max_execution_time* integer
This sets the maximum time in seconds a script is allowed to run before it is terminated by the parser. This helps prevent poorly written scripts from tying up the server. The default setting is *30*. The maximum execution time is not affected by system calls, stream operations etc. Please see the set_time_limit() function for more details. You can not change this setting with ini_set() when running in safe mode. The only workaround is to turn off safe mode or by changing the time limit in the *php.ini*. Your web server can have other timeouts. E.g. Apache has *Timeout* directive, IIS has CGI timeout function, both default to 300 seconds. See the web server documentation for meaning of it.

*max_input_time* integer
This sets the maximum time in seconds a script is allowed to parse input data, like POST, GET and file uploads.

*max_input_nesting_level* integer
Sets the max nesting depth of input variables (i.e. *$_GET*, *$_POST*..)

*magic_quotes_gpc* boolean

**Warning**

This feature has been *DEPRECATED* and *REMOVED* as of PHP 6.0.0. Relying on this feature is highly discouraged.

Sets the magic_quotes state for GPC (Get/Post/Cookie) operations. When magic_quotes are on, all ' (single-quote), " (double quote), \ (backslash) and NUL's are escaped with a backslash automatically.

**Note**

In PHP 4, also *$_ENV* variables are escaped.

**Note**

If the magic_quotes_sybase directive is also ON it will completely override magic_quotes_gpc. Having both directives enabled means only single quotes are escaped as ''. Double quotes, backslashes and NUL's will remain untouched and unescaped.

See also get_magic_quotes_gpc()

`magic_quotes_runtime` boolean

**Warning**

This feature has been *DEPRECATED* and *REMOVED* as of PHP 6.0.0. Relying on this feature is highly discouraged.

If `magic_quotes_runtime` is enabled, most functions that return data from any sort of external source including databases and text files will have quotes escaped with a backslash. If magic_quotes_sybase is also on, a single-quote is escaped with a single-quote instead of a backslash.

## Resource Types

This extension has no resource types defined.

# Predefined Constants

The constants below are always available as part of the PHP core.

**Pre-defined phpcredits() constants**

| Constant | Value | Description |
|---|---|---|
| CREDITS_GROUP | 1 | A list of the core developers |
| CREDITS_GENERAL | 2 | General credits: Language design and concept, PHP authors and SAPI module. |
| CREDITS_SAPI | 4 | A list of the server API modules for PHP, and their authors. |
| CREDITS_MODULES | 8 | A list of the extension modules for PHP, and their authors. |
| CREDITS_DOCS | 16 | The credits for the documentation team. |
| CREDITS_FULLPAGE | 32 | Usually used in combination with the other flags. Indicates that a complete stand-alone HTML page needs to be printed including the information indicated by the other flags. |
| CREDITS_QA | 64 | The credits for the quality assurance team. |
| CREDITS_ALL | -1 | All the credits, equivalent to using: CREDITS_DOCS + CREDITS_GENERAL + CREDITS_GROUP + CREDITS_MODULES + CREDITS_QA CREDITS_FULLPAGE. It generates a complete stand-alone HTML page with the appropriate tags. This is the default value. |

phpinfo() **constants**

| Constant | Value | Description |
|---|---|---|
| INFO_GENERAL | 1 | The configuration line, *php.ini* location, build date, Web Server, System and more. |
| INFO_CREDITS | 2 | PHP Credits. See also phpcredits(). |
| INFO_CONFIGURATION | 4 | Current Local and Master values for PHP directives. See also ini_get(). |
| INFO_MODULES | 8 | Loaded modules and their respective settings. |
| INFO_ENVIRONMENT | 16 | Environment Variable information that's also available in *$_ENV*. |
| INFO_VARIABLES | 32 | Shows all predefined variables from EGPCS (Environment, GET, POST, Cookie, Server). |
| INFO_LICENSE | 64 | PHP License information. See also the » license faq. |
| INFO_ALL | -1 | Shows all of the above. This is the default value. |

**ASSERT_ACTIVE** ( integer )

**ASSERT_CALLBACK** ( integer )

**ASSERT_BAIL** ( integer )

**ASSERT_WARNING** ( integer )

**ASSERT_QUIET_EVAL** ( integer )

# PHP Options/Info Functions

# assert_options

assert_options -- Set/get the various assert flags

## Description

mixed **assert_options** ( int `$what` [, mixed `$value` ] )

Set the various assert() control options or just query their current settings.

## Parameters

*`what`*

**Assert Options**

| option | ini-parameter | default | description |
|---|---|---|---|
| ASSERT_ACTIVE | assert.active | 1 | enable assert() evaluation |
| ASSERT_WARNING | assert.warning | 1 | issue a PHP warning for each failed assertion |
| ASSERT_BAIL | assert.bail | 0 | terminate execution on failed assertions |
| ASSERT_QUIET_EVAL | assert.quiet_eval | 0 | disable error_reporting during assertion expression evaluation |
| ASSERT_CALLBACK | assert.callback | ( **NULL** ) | user function to call on failed assertions |

*`value`*
An optional new value for the option.

## Return Values

Returns the original setting of any option or **FALSE** on errors.

# assert

assert -- Checks if assertion is **FALSE**

## Description

bool **assert** ( mixed `$assertion` )

assert() will check the given *`assertion`* and take appropriate action if its result is **FALSE**.

If the *`assertion`* is given as a string it will be evaluated as PHP code by assert(). The advantages of a string *`assertion`* are less overhead when assertion checking is off and messages containing the *`assertion`* expression when an assertion fails. This means that if you pass a boolean condition as *`assertion`* this condition will not show up as parameter to the assertion function which you may have defined with the assert_options() function, the condition is converted to a string before calling that handler function, and the boolean **FALSE** is converted as the empty string.

Assertions should be used as a debugging feature only. You may use them for sanity-checks that test for conditions that should always be **TRUE** and that indicate some programming errors if not or to check for the presence of certain features like extension functions or certain system limits and features.

Assertions should not be used for normal runtime operations like input parameter checks. As a rule of thumb your code should always be able to work correctly if assertion checking is not activated.

The behavior of assert() may be configured by assert_options() or by .ini-settings described in that functions manual page.

The assert_options() function and/or ASSERT_CALLBACK configuration directive allow a callback function to be set to handle failed assertions.

assert() callbacks are particularly useful for building automated test suites because they allow you to easily capture the code passed to the assertion, along with information on where the assertion was made. While this information can be captured via other methods, using assertions makes it much faster and easier!

The callback function should accept three arguments. The first argument will contain the file the assertion failed in. The second argument will contain the line the assertion failed on and the third argument will contain the expression that failed (if any - literal values such as 1 or "two" will not be passed via this argument)

## Parameters

*`assertion`*
The assertion.

## Return Values

**FALSE** if the assertion is false, **TRUE** otherwise.

## Examples

**Example #1 - Handle a failed assertion with a custom handler**

```
<?php
// Active assert and make it quiet
assert_options(ASSERT_ACTIVE, 1);
assert_options(ASSERT_WARNING, 0);
assert_options(ASSERT_QUIET_EVAL, 1);

// Create a handler function
function my_assert_handler($file, $line, $code)
{
   echo "<hr>Assertion Failed:
       File '$file'<br />
       Line '$line'<br />
       Code '$code'<br /><hr />";
}

// Set up the callback
assert_options(ASSERT_CALLBACK, 'my_assert_handler');

// Make an assertion that should fail
assert('mysql_query("")');
?>
```

# dl

dl -- Loads a PHP extension at runtime

## Description

int **dl** ( string `$library` )

Loads the PHP extension given by the parameter `library`.

Use extension_loaded() to test whether a given extension is already available or not. This works on both built-in extensions and dynamically loaded ones (either through *php.ini* or dl() ).

## Parameters

`library`
: This parameter is *only* the filename of the extension to load which also depends on your platform. For example, the sockets extension (if compiled as a shared module, not the default!) would be called *sockets.so* on Unix platforms whereas it is called *php_sockets.dll* on the Windows platform. The directory where the extension is loaded from depends on your platform: Windows - If not explicitly set in the *php.ini*, the extension is loaded from *c:\php4\extensions* by default. Unix - If not explicitly set in the *php.ini*, the default extension directory depends on

  - whether PHP has been built with *--enable-debug* or not
  - whether PHP has been built with (experimental) ZTS (Zend Thread Safety) support or not
  - the current internal *ZEND_MODULE_API_NO* (Zend internal module API number, which is basically the date on which a major module API change happened, e.g. *20010901* )

  Taking into account the above, the directory then defaults to *<install-dir>/lib/php/extensions/ <debug-or-not>-<zts-or-not>-ZEND_MODULE_API_NO*, e.g. */usr/local/php/lib/php/extensions/debug-non-zts-20010901* or */usr/local/php/lib/php/extensions/no-debug-zts-20010901*.

## Return Values

Returns **TRUE** on success or **FALSE** on failure. If the functionality of loading modules is not available (see Note) or has been disabled (either by turning it off *enable_dl* or by enabling safe mode in *php.ini* ) an **E_ERROR** is emitted and execution is stopped. If dl() fails because the specified library couldn't be loaded, in addition to **FALSE** an **E_WARNING** message is emitted.

## Examples

**Example #2 - dl() examples**

```
<?php
// Example loading an extension based on OS
if (!extension_loaded('sqlite')) {
   if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
       dl('php_sqlite.dll');
   } else {
       dl('sqlite.so');
   }
}

// Or, the PHP_SHLIB_SUFFIX constant is available as of PHP 4.3.0
if (!extension_loaded('sqlite')) {
   $prefix = (PHP_SHLIB_SUFFIX === 'dll') ? 'php_' : '';
   dl($prefix . 'sqlite.' . PHP_SHLIB_SUFFIX);
}
?>
```

## Notes

**Note**

dl() is *not* supported in multithreaded Web servers. Use the *extensions* statement in your *php.ini* when operating under such an environment. However, the *CGI* and *CLI* build are *not* affected !

**Note**

As of PHP 5, the dl() function is deprecated in every SAPI *except* CLI. Use Extension Loading Directives method instead.

**Note**

Since PHP 6 this function is disabled in all SAPIs, except CLI, CGI and embed.

**Note**

dl() is case sensitive on Unix platforms.

| Note |
| --- |
| This function is disabled when PHP is running in safe mode. |

## See Also

- Extension Loading Directives
- extension_loaded()

# extension_loaded

extension_loaded -- Find out whether an extension is loaded

## Description

bool **extension_loaded** ( string `$name` )

Finds out whether the extension is loaded.

## Parameters

*name*
: The extension name. You can see the names of various extensions by using phpinfo() or if you're using the *CGI* or *CLI* version of PHP you can use the *-m* switch to list all available extensions:

```
$ php -m
[PHP Modules]
xml
tokenizer
standard
sockets
session
posix
pcre
overload
mysql
mbstring
ctype

[Zend Modules]
```

## Return Values

Returns **TRUE** if the extension identified by *name* is loaded, **FALSE** otherwise.

## Examples

**Example #3 - extension_loaded() example**

```
<?php
if (!extension_loaded('gd')) {
   if (!dl('gd.so')) {
       exit;
   }
}
?>
```

## Notes

| Note |
| --- |
| extension_loaded() uses the internal extension name to test whether a certain extension is available or not. Most internal extension names are written in lower case but there may be extension available which also use uppercase letters. Be warned that this function compares *case sensitive* ! |

## See Also

- get_loaded_extensions()
- get_extension_funcs()
- phpinfo()
- dl()

# get_cfg_var

get_cfg_var -- Gets the value of a PHP configuration option

## Description

string **get_cfg_var** ( string `$option` )

Gets the value of a PHP configuration *`option`*.

This function will not return configuration information set when the PHP was compiled, or read from an Apache configuration file.

To check whether the system is using a configuration file, try retrieving the value of the cfg_file_path configuration setting. If this is available, a configuration file is being used.

## Parameters

*`option`*
The configuration option name.

## Return Values

Returns the current value of the PHP configuration variable specified by *`varname`*, or **FALSE** if an error occurs.

## See Also

- ini_get()
- ini_get_all()

# get_current_user

get_current_user -- Gets the name of the owner of the current PHP script

## Description

string **get_current_user** ( void )

Returns the name of the owner of the current PHP script.

## Return Values

Returns the username as a string.

## See Also

- getmyuid()
- getmygid()
- getmypid()
- getmyinode()
- getlastmod()

# get_defined_constants

get_defined_constants -- Returns an associative array with the names of all the constants and their values

## Description

array **get_defined_constants** ( [ mixed `$categorize` ] )

Returns the names and values of all the constants currently defined. This includes those created by extensions as well as those created with the define() function.

## Parameters

*`categorize`*
May be passed, causing this function to return a multi-dimensional array with categories in the keys of the first dimension and constants and their values in the second dimension.

```
<?php
define("MY_CONSTANT", 1);
print_r(get_defined_constants(true));
?>
```

The above example will output something similar to:

```
Array
(
   [internal] => Array
       (
           [E_ERROR] => 1
           [E_WARNING] => 2
           [E_PARSE] => 4
           [E_NOTICE] => 8
           [E_CORE_ERROR] => 16
           [E_CORE_WARNING] => 32
           [E_COMPILE_ERROR] => 64
           [E_COMPILE_WARNING] => 128
           [E_USER_ERROR] => 256
           [E_USER_WARNING] => 512
           [E_USER_NOTICE] => 1024
           [E_ALL] => 2047
           [TRUE] => 1
       )

   [pcre] => Array
       (
           [PREG_PATTERN_ORDER] => 1
           [PREG_SET_ORDER] => 2
           [PREG_OFFSET_CAPTURE] => 256
           [PREG_SPLIT_NO_EMPTY] => 1
           [PREG_SPLIT_DELIM_CAPTURE] => 2
           [PREG_SPLIT_OFFSET_CAPTURE] => 4
           [PREG_GREP_INVERT] => 1
```

```
        )

    [user] => Array
        (
            [MY_CONSTANT] => 1
        )

)
```

| Note |
| --- |
| The value of the *categorize* parameter is irrelevant, only its presence is considered. |

## Return Values

## ChangeLog

| Version | Description |
| --- | --- |
| 5.0.0 | The *categorize* parameter was added. |

## Examples

**Example #4 - get_defined_constants() Example**

```
<?php
print_r(get_defined_constants());
?>
```

The above example will output something similar to:

```
Array
(
    [E_ERROR] => 1
    [E_WARNING] => 2
    [E_PARSE] => 4
    [E_NOTICE] => 8
    [E_CORE_ERROR] => 16
    [E_CORE_WARNING] => 32
    [E_COMPILE_ERROR] => 64
    [E_COMPILE_WARNING] => 128
```

```
    [E_USER_ERROR] => 256
    [E_USER_WARNING] => 512
    [E_USER_NOTICE] => 1024
    [E_ALL] => 2047
    [TRUE] => 1
)
```

## See Also

- defined()
- get_loaded_extensions()
- get_defined_functions()
- get_defined_vars()

# get_extension_funcs

get_extension_funcs -- Returns an array with the names of the functions of a module

## Description

array **get_extension_funcs** ( string `$module_name` )

This function returns the names of all the functions defined in the module indicated by *`module_name`*.

## Parameters

*`module_name`*
The module name.

| **Note** |
| --- |
| This parameter must be in *lowercase*. |

## Return Values

Returns an array with all the functions, or **FALSE** if *`module_name`* is not a valid extension.

## Examples

**Example #5 - Prints the XML functions**

```
<?php
print_r(get_extension_funcs("xml"));
?>
```

The above example will output something similar to:

```
Array
(
   [0] => xml_parser_create
   [1] => xml_parser_create_ns
   [2] => xml_set_object
   [3] => xml_set_element_handler
   [4] => xml_set_character_data_handler
   [5] => xml_set_processing_instruction_handler
   [6] => xml_set_default_handler
   [7] => xml_set_unparsed_entity_decl_handler
   [8] => xml_set_notation_decl_handler
```

```
    [9] => xml_set_external_entity_ref_handler
    [10] => xml_set_start_namespace_decl_handler
    [11] => xml_set_end_namespace_decl_handler
    [12] => xml_parse
    [13] => xml_parse_into_struct
    [14] => xml_get_error_code
    [15] => xml_error_string
    [16] => xml_get_current_line_number
    [17] => xml_get_current_column_number
    [18] => xml_get_current_byte_index
    [19] => xml_parser_free
    [20] => xml_parser_set_option
    [21] => xml_parser_get_option
    [22] => utf8_encode
    [23] => utf8_decode
)
```

## See Also

- get_loaded_extensions()

# get_include_path

get_include_path -- Gets the current include_path configuration option

## Description

string **get_include_path** ( void )

Gets the current include_path configuration option value.

## Return Values

Returns the path, as a string.

## Examples

**Example #6 - get_include_path() example**

```
<?php
// Works as of PHP 4.3.0
echo get_include_path();

// Works in all PHP versions
echo ini_get('include_path');
?>
```

## See Also

- ini_get()
- restore_include_path()
- set_include_path()
- **include()**

# get_included_files

get_included_files -- Returns an array with the names of included or required files

## Description

array **get_included_files** ( void )

Gets the names of all files that have been included using **include()**, **include_once()**, **require()** or **require_once()**.

## Return Values

Returns an array of the names of all files.

The script originally called is considered an "included file," so it will be listed together with the files referenced by **include()** and family.

Files that are included or required multiple times only show up once in the returned array.

## ChangeLog

| Version | Description |
| --- | --- |
| 4.0.1 | In PHP 4.0.1 and previous versions this function assumed that the required files ended in the extension.*php*; other extensions would not be returned. The array returned by get_included_files() was an associative array and only listed files included by **include()** and **include_once()**. |

## Examples

**Example #7 - get_included_files() example**

```
<?php
// This file is abc.php

include 'test1.php';
include_once 'test2.php';
require 'test3.php';
require_once 'test4.php';
```

```
$included_files = get_included_files();

foreach ($included_files as $filename) {
   echo "$filename\n";
}

?>
```

The above example will output:

```
abc.php
test1.php
test2.php
test3.php
test4.php
```

## Notes

| **Note** |
| --- |
| Files included using the *auto_prepend_file* configuration directive are not included in the returned array. |

## See Also

- **include()**
- **include_once()**
- **require()**
- **require_once()**
- get_required_files()

# get_loaded_extensions

get_loaded_extensions -- Returns an array with the names of all modules compiled and loaded

## Description

array **get_loaded_extensions** ( [ bool `$zend_extensions` = FALSE ] )

This function returns the names of all the modules compiled and loaded in the PHP interpreter.

## Parameters

*`zend_extensions`*
Return zend_extensions or not, defaults to **FALSE** (do not list zend_extensions).

## Return Values

Returns an indexed array of all the modules names.

## ChangeLog

| Version | Description |
| --- | --- |
| 5.2.4 | The optional *`zend_extensions`* parameter was added |

## Examples

**Example #8 - get_loaded_extensions() Example**

```
<?php
print_r(get_loaded_extensions());
?>
```

The above example will output something similar to:

```
Array
(
  [0] => xml
```

```
    [1] => wddx
    [2] => standard
    [3] => session
    [4] => posix
    [5] => pgsql
    [6] => pcre
    [7] => gd
    [8] => ftp
    [9] => db
    [10] => calendar
    [11] => bcmath
)
```

## See Also

- get_extension_funcs()
- extension_loaded()
- dl()
- phpinfo()

# get_magic_quotes_gpc

get_magic_quotes_gpc -- Gets the current configuration setting of magic quotes gpc

## Description

int **get_magic_quotes_gpc** ( void )

Returns the current configuration setting of magic_quotes_gpc

Keep in mind that the setting magic_quotes_gpc will not work at runtime.

For more information about magic_quotes, see this security section.

## Return Values

Returns 0 if magic quotes gpc are off, 1 otherwise.

## Examples

**Example #9 - get_magic_quotes_gpc() example**

```
<?php
echo get_magic_quotes_gpc();         // 1
echo $_POST['lastname'];             // O\'reilly
echo addslashes($_POST['lastname']); // O\\\'reilly

if (!get_magic_quotes_gpc()) {
   $lastname = addslashes($_POST['lastname']);
} else {
   $lastname = $_POST['lastname'];
}

echo $lastname; // O\'reilly
$sql = "INSERT INTO lastnames (lastname) VALUES ('$lastname')";
?>
```

## Notes

**Note**

If the directive magic_quotes_sybase is ON it will completely override magic_quotes_gpc. So even when get_magic_quotes_gpc() returns **TRUE** neither double quotes, backslashes or NUL's will be escaped. Only single quotes will be escaped. In this case they'll look like: ''

## See Also

- addslashes()
- stripslashes()
- get_magic_quotes_runtime()
- ini_get()

# get_magic_quotes_runtime

get_magic_quotes_runtime -- Gets the current active configuration setting of magic_quotes_runtime

## Description

int **get_magic_quotes_runtime** ( void )

Returns the current active configuration setting of magic_quotes_runtime.

## Return Values

Returns 0 if magic quotes runtime is off, 1 otherwise.

## See Also

- get_magic_quotes_gpc()
- set_magic_quotes_runtime()

# get_required_files

get_required_files -- Alias of get_included_files()

## Description

This function is an alias of: get_included_files().

# getenv

getenv -- Gets the value of an environment variable

## Description

string **getenv** ( string `$varname` )

Gets the value of an environment variable.

You can see a list of all the environmental variables by using phpinfo(). You can find out what many of them mean by taking a look at the » CGI specification, specifically the » page on environmental variables.

## Parameters

*`varname`*
The variable name.

## Return Values

Returns the value of the environment variable *`varname`*, or **FALSE** on an error.

## Examples

**Example #10 - getenv() Example**

```
<?php
// Example use of getenv()
$ip = getenv('REMOTE_ADDR');

// Or simply use a Superglobal ($_SERVER or $_ENV)
$ip = $_SERVER['REMOTE_ADDR'];
?>
```

## See Also

- putenv()
- apache_getenv()
- Superglobals

# getlastmod

getlastmod -- Gets time of last page modification

## Description

int **getlastmod** ( void )

Gets the time of the last modification of the current page.

If you're interested in getting the last modification time of a different file, consider using filemtime().

## Return Values

Returns the time of the last modification of the current page. The value returned is a Unix timestamp, suitable for feeding to date(). Returns **FALSE** on error.

## Examples

**Example #11 - getlastmod() example**

```
<?php
// outputs e.g. 'Last modified: March 04 1998 20:43:59.'
echo "Last modified: " . date ("F d Y H:i:s.", getlastmod());
?>
```

## See Also

- date()
- getmyuid()
- getmygid()
- get_current_user()
- getmyinode()
- getmypid()
- filemtime()

# getmygid

getmygid -- Get PHP script owner's GID

## Description

int **getmygid** ( void )

Gets the group ID of the current script.

## Return Values

Returns the group ID of the current script, or **FALSE** on error.

## See Also

- getmyuid()
- getmypid()
- get_current_user()
- getmyinode()
- getlastmod()

# getmyinode

getmyinode -- Gets the inode of the current script

## Description

int **getmyinode** ( void )

Gets the inode of the current script.

## Return Values

Returns the current script's inode as an integer, or **FALSE** on error.

## See Also

- getmygid()
- getmyuid()
- getmypid()
- get_current_user()
- getlastmod()

# getmypid

getmypid -- Gets PHP's process ID

## Description

int **getmypid** ( void )

Gets the current PHP process ID.

## Return Values

Returns the current PHP process ID, or **FALSE** on error.

## Notes

| **Warning** |
| --- |
| Process IDs are not unique, thus they are a weak entropy source. We recommend against relying on pids in security-dependent contexts. |

## See Also

- getmygid()
- getmyuid()
- get_current_user()
- getmyinode()
- getlastmod()

# getmyuid

getmyuid -- Gets PHP script owner's UID

## Description

int **getmyuid** ( void )

Gets the user ID of the current script.

## Return Values

Returns the user ID of the current script, or **FALSE** on error.

## See Also

- getmygid()
- getmypid()
- get_current_user()
- getmyinode()
- getlastmod()

# getopt

getopt -- Gets options from the command line argument list

## Description

array **getopt** ( string `$options` [, array `$longopts` ] )

Parses options passed to the script.

## Parameters

*`options`*
: Each character in this string will be used as option characters and matched against options passed to the script starting with a single hyphen ( - ). For example, an option string *"x"* recognizes an option *-x*.

*`longopts`*
: An array of options. Each element in this array will be used as option strings and matched against options passed to the script starting with two hyphens ( -- ). For example, an longopts element *"opt"* recognizes an option *--opt*.

| **Note** |
| --- |
| Prior to PHP5.3.0 this parameter was only available on few systems |

The *`options`* parameter may contain the following elements:

- Individual characters (do not accept values)
- Characters followed by a colon (parameter requires value)
- Characters followed by two colons (optional value)

Option values are the first argument after the string. It does not matter if a value has leading white space or not.

| **Note** |
| --- |
| Optional values do not accept " " (space) as a seperator. |

| **Note** |
| --- |
| The format for the *`options`* and *`longopts`* is almost the same, the only difference is that *`longopts`* takes an array of options (where each element is the option) where as |

*options* takes a string (where each character is the option).

## Return Values

This function will return an array of option / argument pairs or **FALSE** on failure.

## ChangeLog

| Version | Description |
|---|---|
| 5.3.0 | Added support for "=" as argument/value separator. |
| 5.3.0 | Added support for optional values (specified with "::"). |
| 5.3.0 | This function is no longer system dependent and works on Windows too. |

## Examples

**Example #12 - getopt() Example**

```
<?php
$options = getopt("f:hp:");
var_dump($options);
?>
```

Running the above script with *php script.php -fvalue -h* will output:

```
array(2) {
 ["f"]=>
 string(5) "value"
 ["h"]=>
 bool(false)
}
```

**Example #13 - getopt() Example#2**

```
<?php
$shortopts  = "";
$shortopts .= "f:";  // Required value
```

```
$shortopts .= "v::"; // Optional value
$shortopts .= "abc"; // These options do not accept values

$longopts  = array(
   "required:",     // Required value
   "optional::",    // Optional value
   "option",        // No value
   "opt",           // No value
);
$options = getopt($shortopts, $longopts);
var_dump($options);
?>
```

Running the above script with *php script.php -f "value for f" -v -a --required value --optional="optional value" --option* will output:

```
array(6) {
 ["f"]=>
 string(11) "value for f"
 ["v"]=>
 bool(false)
 ["a"]=>
 bool(false)
 ["required"]=>
 string(5) "value"
 ["optional"]=>
 string(14) "optional value"
 ["option"]=>
 bool(false)
}
```

**Example #14 -** getopt() **Example#3**

Passing multiple options as one

```
<?php
$options = getopt("abc");
var_dump($options);
?>
```

Running the above script with *php script.php -aaac* will output:

```
array(2) {
 ["a"]=>
 array(3) {
   [0]=>
   bool(false)
   [1]=>
   bool(false)
   [2]=>
   bool(false)
 }
 ["c"]=>
 bool(false)
```

```
}
```

## Notes

| Note |
| --- |
| The register_argc_argv option must be enabled for this function to work |

# getrusage

getrusage -- Gets the current resource usages

## Description

array **getrusage** ( [ int `$who` ] )

This is an interface to getrusage(2). It gets data returned from the system call.

## Parameters

*who*
: If who is 1, getrusage will be called with RUSAGE_CHILDREN.

## Return Values

Returns an associative array containing the data returned from the system call. All entries are accessible by using their documented field names.

## Examples

**Example #15 - getrusage() example**

```
<?php
$dat = getrusage();
echo $dat["ru_nswap"];         // number of swaps
echo $dat["ru_majflt"];        // number of page faults
echo $dat["ru_utime.tv_sec"];  // user time used (seconds)
echo $dat["ru_utime.tv_usec"]; // user time used (microseconds)
?>
```

## Notes

**Note**

This function is not implemented on Windows platforms.

## See Also

- Your system's man page on getrusage(2)

# ini_alter

ini_alter -- Alias of ini_set()

## Description

This function is an alias of: ini_set().

# ini_get_all

ini_get_all -- Gets all configuration options

## Description

array **ini_get_all** ( [ string `$extension` [, bool `$details` ] ] )

Returns all the registered configuration options.

## Parameters

*`extension`*
: An optional extension name. If set, the function return only options specific for that extension.

*`details`*
: Retrieve details settings or only the current value for each setting. Default is **TRUE** (retrieve details).

## Return Values

Returns an associative array with directive name as the array key.

When *`details`* is **TRUE** (default) the array will contain *global_value* (set in *php.ini* ), *local_value* (perhaps set with ini_set() or *.htaccess* ), and *access* (the access level).

When *`details`* is **FALSE** the value will be the current value of the option.

See the manual section for information on what access levels mean.

| **Note** |
| --- |
| It's possible for a directive to have multiple access levels, which is why *access* shows the appropriate bitmask values. |

## ChangeLog

| **Version** | **Description** |
| --- | --- |
| 5.3.0 | Added *`details`*. |

## Examples

**Example #16 - ini_get_all() examples**

```
<?php
print_r(ini_get_all("pcre"));
print_r(ini_get_all());
?>
```

The above example will output something similar to:

```
Array
(
   [pcre.backtrack_limit] => Array
       (
           [global_value] => 100000
           [local_value] => 100000
           [access] => 7
       )

   [pcre.recursion_limit] => Array
       (
           [global_value] => 100000
           [local_value] => 100000
           [access] => 7
       )

)
Array
(
   [allow_call_time_pass_reference] => Array
       (
           [global_value] => 0
           [local_value] => 0
           [access] => 6
       )

   [allow_url_fopen] => Array
       (
           [global_value] => 1
           [local_value] => 1
           [access] => 4
       )

   ...

)
```

**Example #17 - Disabling *details***

```
<?php
print_r(ini_get_all("pcre", false)); // Added in PHP 5.3.0
print_r(ini_get_all(null, false)); // Added in PHP 5.3.0
?>
```

The above example will output something similar to:

```
Array
(
   [pcre.backtrack_limit] => 100000
   [pcre.recursion_limit] => 100000
)
Array
(
   [allow_call_time_pass_reference] => 0
   [allow_url_fopen] => 1
   ...
)
```

## See Also

- How to change configuration settings
- ini_get()
- ini_restore()
- ini_set()
- get_loaded_extensions()
- phpinfo()

# ini_get

ini_get -- Gets the value of a configuration option

## Description

string **ini_get** ( string `$varname` )

Returns the value of the configuration option on success.

## Parameters

*`varname`*
The configuration option name.

## Return Values

Returns the value of the configuration option as a string on success, or an empty string on failure or for *null* values.

## Examples

**Example #18 - A few ini_get() examples**

```
<?php
/*
Our php.ini contains the following settings:

display_errors = On
register_globals = Off
post_max_size = 8M
*/

echo 'display_errors = ' . ini_get('display_errors') . "\n";
echo 'register_globals = ' . ini_get('register_globals') . "\n";
echo 'post_max_size = ' . ini_get('post_max_size') . "\n";
echo 'post_max_size+1 = ' . (ini_get('post_max_size')+1) . "\n";
echo 'post_max_size in bytes = ' . return_bytes(ini_get('post_max_size'));

function return_bytes($val) {
   $val = trim($val);
   $last = strtolower($val[strlen($val)-1]);
   switch($last) {
       // The 'G' modifier is available since PHP 5.1.0
       case 'g':
           $val *= 1024;
       case 'm':
           $val *= 1024;
```

```
        case 'k':
            $val *= 1024;
    }

    return $val;
}

?>
```

The above example will output something similar to:

```
display_errors = 1
register_globals = 0
post_max_size = 8M
post_max_size+1 = 9
post_max_size in bytes = 8388608
```

## Notes

**Note**

**When querying boolean values**

A boolean ini value of *off* will be returned as an empty string or "0" while a boolean ini value of *on* will be returned as "1". The function can also return the literal string of INI value.

**Note**

**When querying memory size values**

Many ini memory size values, such as upload_max_filesize, are stored in the *php.ini* file in shorthand notation. ini_get() will return the exact string stored in the *php.ini* file and *NOT* its integer equivalent. Attempting normal arithmetic functions on these values will not have otherwise expected results. The example above shows one way to convert shorthand notation into bytes, much like how the PHP source does it.

## See Also

- get_cfg_var()
- ini_get_all()
- ini_restore()
- ini_set()

# ini_restore

ini_restore -- Restores the value of a configuration option

## Description

void **ini_restore** ( string `$varname` )

Restores a given configuration option to its original value.

## Parameters

*`varname`*
The configuration option name.

## Return Values

No value is returned.

## See Also

- ini_get()
- ini_get_all()
- ini_set()

# ini_set

ini_set -- Sets the value of a configuration option

## Description

string **ini_set** ( string `$varname`, string `$newvalue` )

Sets the value of the given configuration option. The configuration option will keep this new value during the script's execution, and will be restored at the script's ending.

## Parameters

*`varname`*
: Not all the available options can be changed using ini_set(). There is a list of all available options in the appendix.

*`newvalue`*
: The new value for the option.

## Return Values

Returns the old value on success, **FALSE** on failure.

## See Also

- get_cfg_var()
- ini_get()
- ini_get_all()
- ini_restore()
- How to change configuration settings

# main

main -- Dummy for main()

## Description

There is no function named main() except in the PHP source. In PHP 4.3.0, a new type of error handling in the PHP source (php_error_docref) was introduced. One feature is to provide links to a manual page in PHP error messages when the PHP directives html_errors (on by default) and docref_root (on by default until PHP 4.3.2) are set.

Sometimes error messages refer to a manual page for the function main() which is why this page exists. Please add a user comment below that mentions what PHP function caused the error that linked to main() and it will be fixed and properly documented.

### Known errors that point to main()

| Function name | No longer points here as of |
|---|---|
| **include()** | 5.1.0 |
| **include_once()** | 5.1.0 |
| **require()** | 5.1.0 |
| **require_once()** | 5.1.0 |

## See Also

- html_errors
- display_errors

# memory_get_peak_usage

memory_get_peak_usage -- Returns the peak of memory allocated by PHP

## Description

int **memory_get_peak_usage** ( [ bool `$real_usage` ] )

Returns the peak of memory, in bytes, that's been allocated to your PHP script.

## Parameters

*`real_usage`*
: Set this to **TRUE** to get the real size of memory allocated from system. If not set or **FALSE** only the memory used by *emalloc()* is reported.

## Return Values

Returns the memory peak in bytes.

## ChangeLog

| Version | Description |
|---|---|
| 5.2.1 | Compiling with --enable-memory-limit is no longer required for this function to exist. |
| 5.2.0 | *`real_usage`* was added. |

## See Also

- memory_get_usage()
- memory_limit

# memory_get_usage

memory_get_usage -- Returns the amount of memory allocated to PHP

## Description

int **memory_get_usage** ( [ bool `$real_usage` ] )

Returns the amount of memory, in bytes, that's currently being allocated to your PHP script.

## Parameters

*`real_usage`*
: Set this to **TRUE** to get the real size of memory allocated from system. If not set or **FALSE** only the memory used by *emalloc()* is reported.

## Return Values

Returns the memory amount in bytes.

## ChangeLog

| Version | Description |
|---|---|
| 5.2.1 | Compiling with --enable-memory-limit is no longer required for this function to exist. |
| 5.2.0 | *`real_usage`* was added. |

## Examples

**Example #19 - A memory_get_usage() example**

```
<?php
// This is only an example, the numbers below will
// differ depending on your system

echo memory_get_usage() . "\n"; // 36640

$a = str_repeat("Hello", 4242);
```

```
echo memory_get_usage() . "\n"; // 57960

unset($a);

echo memory_get_usage() . "\n"; // 36744

?>
```

## See Also

- memory_get_peak_usage()
- memory_limit

# php_ini_loaded_file

php_ini_loaded_file -- Retrieve a path to the loaded php.ini file

## Description

string **php_ini_loaded_file** ( void )

Check if a *php.ini* file is loaded, and retrieve its path.

## Parameters

This function has no parameters.

## Return Values

The loaded *php.ini* path, or **FALSE** if one is not loaded.

## Examples

**Example #20 - php_ini_loaded_file() example**

```
<?php
$inipath = php_ini_loaded_file();

if ($inipath) {
   echo 'Loaded php.ini: ' . $inipath;
} else {
   echo 'A php.ini file is not loaded';
}
?>
```

The above example will output something similar to:

```
Loaded php.ini: /usr/local/php/php.ini
```

## See Also

- php_ini_scanned_files()
- phpinfo()
- The configuration file

# php_ini_scanned_files

php_ini_scanned_files -- Return a list of .ini files parsed from the additional ini dir

## Description

string **php_ini_scanned_files** ( void )

php_ini_scanned_files() returns a comma-separated list of configuration files parsed after *php.ini*. These files are found in a directory defined by the *--with-config-file-scan-dir* option which is set during compilation.

The returned configuration files also include the path as declared in the *--with-config-file-scan-dir* option.

## Return Values

Returns a comma-separated string of .ini files on success. Each comma is followed by a newline. If the directive *--with-config-file-scan-dir* wasn't set, **FALSE** is returned. If it was set and the directory was empty, an empty string is returned. If a file is unrecognizable, the file will still make it into the returned string but a PHP error will also result. This PHP error will be seen both at compile time and while using php_ini_scanned_files().

## Examples

**Example #21 - A simple example to list the returned ini files**

```
<?php
if ($filelist = php_ini_scanned_files()) {
   if (strlen($filelist) > 0) {
       $files = explode(',', $filelist);

       foreach ($files as $file) {
           echo "<li>" . trim($file) . "</li>\n";
       }
   }
}
?>
```

## See Also

- ini_set()
- phpinfo()
- php_ini_loaded_file()

# php_logo_guid

php_logo_guid -- Gets the logo guid

## Description

string **php_logo_guid** ( void )

This function returns the ID which can be used to display the PHP logo using the built-in image. Logo is displayed only if expose_php is On.

## Return Values

Returns *PHPE9568F34-D428-11d2-A769-00AA001ACF42*.

## Examples

**Example #22 - php_logo_guid() example**

```
<?php

echo '<img src="' . $_SERVER['PHP_SELF'] .
    '?=' . php_logo_guid() . '" alt="PHP Logo !" />';

?>
```

## See Also

- phpinfo()
- phpversion()
- phpcredits()
- zend_logo_guid()

# php_sapi_name

php_sapi_name -- Returns the type of interface between web server and PHP

## Description

string **php_sapi_name** ( void )

Returns a lowercase string which describes the type of interface between web server and PHP (Server API, SAPI). In CGI PHP, this string is "cgi", in *mod_php* for Apache, this string is "apache" and so on.

## Return Values

Returns the interface type, as a lowercase string.

## Examples

**Example #23 -** php_sapi_name() **example**

```
<?php
$sapi_type = php_sapi_name();
if (substr($sapi_type, 0, 3) == 'cgi') {
   echo "You are using CGI PHP\n";
} else {
   echo "You are not using CGI PHP\n";
}
?>
```

## See Also

- PHP_SAPI

# php_uname

php_uname -- Returns information about the operating system PHP is running on

## Description

string **php_uname** ( [ string `$mode` ] )

php_uname() returns a description of the operating system PHP is running on. For the name of just the operating system, consider using the **PHP_OS** constant, but be reminded this constant will contain the operating system PHP was *built* on.

On Unix, the output reverts to displaying the operating system information PHP was built on if it cannot determine the currently running OS.

## Parameters

*`mode`*

`mode` is a single character that defines what information is returned:

- *'a'*: This is the default. Contains all modes in the sequence *"s n r v m"*.
- *'s'*: Operating system name. eg. *FreeBSD*.
- *'n'*: Host name. eg. *localhost.example.com*.
- *'r'*: Release name. eg. *5.1.2-RELEASE*.
- *'v'*: Version information. Varies a lot between operating systems.
- *'m'*: Machine type. eg. *i386*.

## Return Values

Returns the description, as a string.

## Examples

**Example #24 - Some php_uname() examples**

```
<?php
echo php_uname();
echo PHP_OS;

/* Some possible outputs:
Linux localhost 2.4.21-0.13mdk #1 Fri Mar 14 15:08:06 EST 2003 i686
```

```
Linux

FreeBSD localhost 3.2-RELEASE #15: Mon Dec 17 08:46:02 GMT 2001
FreeBSD

Windows NT XN1 5.1 build 2600
WINNT
*/

if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
   echo 'This is a server using Windows!';
} else {
   echo 'This is a server not using Windows!';
}

?>
```

There are also some related Predefined PHP constants that may come in handy, for example:

**Example #25 - A few OS related constant examples**

```
<?php
// *nix
echo DIRECTORY_SEPARATOR; // /
echo PHP_SHLIB_SUFFIX;    // so
echo PATH_SEPARATOR;      // :

// Win*
echo DIRECTORY_SEPARATOR; // \
echo PHP_SHLIB_SUFFIX;    // dll
echo PATH_SEPARATOR;      // ;
?>
```

## See Also

- phpversion()
- php_sapi_name()
- phpinfo()

# phpcredits

phpcredits -- Prints out the credits for PHP

## Description

bool **phpcredits** ( [ int `$flag` ] )

This function prints out the credits listing the PHP developers, modules, etc. It generates the appropriate HTML codes to insert the information in a page.

## Parameters

*`flag`*

To generate a custom credits page, you may want to use the *`flag`* parameter. *`flag`* is optional, and it defaults to **CREDITS_ALL**.

**Pre-defined** phpcredits() **flags**

| **name** | **description** |
|---|---|
| CREDITS_ALL | All the credits, equivalent to using: CREDITS_DOCS + CREDITS_GENERAL + CREDITS_GROUP + CREDITS_MODULES + CREDITS_FULLPAGE. It generates a complete stand-alone HTML page with the appropriate tags. |
| CREDITS_DOCS | The credits for the documentation team |
| CREDITS_FULLPAGE | Usually used in combination with the other flags. Indicates that a complete stand-alone HTML page needs to be printed including the information indicated by the other flags. |
| CREDITS_GENERAL | General credits: Language design and concept, PHP 4.0 authors and SAPI module. |
| CREDITS_GROUP | A list of the core developers |
| CREDITS_MODULES | A list of the extension modules for PHP, and their authors |
| CREDITS_SAPI | A list of the server API modules for PHP, and their authors |

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #26 - Prints the general credits**

```
<?php
phpcredits(CREDITS_GENERAL);
?>
```

**Example #27 - Prints the core developers and the documentation group**

```
<?php
phpcredits(CREDITS_GROUP + CREDITS_DOCS + CREDITS_FULLPAGE);
?>
```

**Example #28 - Printing all the credits**

```
<html>
<head>
 <title>My credits page</title>
</head>
<body>
<?php
// some code of your own
phpcredits(CREDITS_ALL - CREDITS_FULLPAGE);
// some more code
?>
</body>
</html>
```

## See Also

- phpversion()
- php_logo_guid()
- phpinfo()

# phpinfo

phpinfo -- Outputs lots of PHP information

## Description

bool **phpinfo** ( [ int `$what` ] )

Outputs a large amount of information about the current state of PHP. This includes information about PHP compilation options and extensions, the PHP version, server information and environment (if compiled as a module), the PHP environment, OS version information, paths, master and local values of configuration options, HTTP headers, and the PHP License.

Because every system is setup differently, phpinfo() is commonly used to check configuration settings and for available predefined variables on a given system.

phpinfo() is also a valuable debugging tool as it contains all EGPCS (Environment, GET, POST, Cookie, Server) data.

## Parameters

*`what`*

The output may be customized by passing one or more of the following *constants* bitwise values summed together in the optional *`what`* parameter. One can also combine the respective constants or bitwise values together with the or operator.

phpinfo() **options**

| Name (constant) | Value | Description |
|---|---|---|
| INFO_GENERAL | 1 | The configuration line, *php.ini* location, build date, Web Server, System and more. |
| INFO_CREDITS | 2 | PHP Credits. See also phpcredits(). |
| INFO_CONFIGURATION | 4 | Current Local and Master values for PHP directives. See also ini_get(). |
| INFO_MODULES | 8 | Loaded modules and their respective settings. See also get_loaded_extensions(). |
| INFO_ENVIRONMENT | 16 | Environment Variable |

| | | |
|---|---|---|
| | | information that's also available in *$_ENV*. |
| INFO_VARIABLES | 32 | Shows all predefined variables from EGPCS (Environment, GET, POST, Cookie, Server). |
| INFO_LICENSE | 64 | PHP License information. See also the » license FAQ. |
| INFO_ALL | -1 | Shows all of the above. This is the default value. |

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## ChangeLog

| Version | Description |
|---|---|
| 5.2.2 | The "Loaded Configuration File" information was added, when before only "Configuration File (php.ini) Path" existed. |

## Examples

**Example #29 - phpinfo() Example**

```
<?php

// Show all information, defaults to INFO_ALL
phpinfo();

// Show just the module information.
// phpinfo(8) yields identical results.
phpinfo(INFO_MODULES);

?>
```

## Notes

| Note |
| --- |
| Parts of the information displayed are disabled when the expose_php configuration setting is set to *off*. This includes the PHP and Zend logos, and the credits. |

| Note |
| --- |
| phpinfo() outputs plain text instead of HTML when using the CLI mode. |

## See Also

- phpversion()
- phpcredits()
- php_logo_guid()
- ini_get()
- ini_set()
- get_loaded_extensions()
- Predefined Variables

# phpversion

phpversion -- Gets the current PHP version

## Description

string **phpversion** ( [ string `$extension` ] )

Returns a string containing the version of the currently running PHP parser or extension.

## Parameters

*`extension`*
An optional extension name.

## Return Values

If the optional *`extension`* parameter is specified, phpversion() returns the version of that extension, or **FALSE** if there is no version information associated or the extension isn't enabled.

## Examples

**Example #30 - phpversion() example**

```
<?php
// prints e.g. 'Current PHP version: 4.1.1'
echo 'Current PHP version: ' . phpversion();

// prints e.g. '2.0' or nothing if the extension isn't enabled
echo phpversion('tidy');
?>
```

## Notes

**Note**

This information is also available in the predefined constant **PHP_VERSION**.

## See Also

- version_compare()
- phpinfo()
- phpcredits()
- php_logo_guid()
- zend_version()

# putenv

putenv -- Sets the value of an environment variable

## Description

bool **putenv** ( string `$setting` )

Adds *`setting`* to the server environment. The environment variable will only exist for the duration of the current request. At the end of the request the environment is restored to its original state.

Setting certain environment variables may be a potential security breach. The *safe_mode_allowed_env_vars* directive contains a comma-delimited list of prefixes. In Safe Mode, the user may only alter environment variables whose names begin with the prefixes supplied by this directive. By default, users will only be able to set environment variables that begin with *PHP_* (e.g. *PHP_FOO=BAR* ). Note: if this directive is empty, PHP will let the user modify ANY environment variable!

The *safe_mode_protected_env_vars* directive contains a comma-delimited list of environment variables, that the end user won't be able to change using putenv(). These variables will be protected even if *safe_mode_allowed_env_vars* is set to allow to change them.

## Parameters

*`setting`*
The setting, like *"FOO=BAR"*

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## Examples

**Example #31 - Setting an environment variable**

```
<?php
putenv("UNIQID=$uniqid");
?>
```

## Notes

| Warning |
| --- |
| These directives have only effect when safe-mode itself is enabled! |

## See Also

- getenv()

# restore_include_path

restore_include_path -- Restores the value of the include_path configuration option

## Description

void **restore_include_path** ( void )

Restores the include_path configuration option back to its original master value as set in *php.ini*

## Return Values

No value is returned.

## Examples

**Example #32 - restore_include_path() example**

```
<?php

echo get_include_path();  // .:/usr/local/lib/php

set_include_path('/inc');

echo get_include_path();  // /inc

// Works as of PHP 4.3.0
restore_include_path();

// Works in all PHP versions
ini_restore('include_path');

echo get_include_path();  // .:/usr/local/lib/php

?>
```

## See Also

- ini_restore()
- get_include_path()
- set_include_path()
- **include()**

# set_include_path

set_include_path -- Sets the include_path configuration option

## Description

string **set_include_path** ( string `$new_include_path` )

Sets the include_path configuration option for the duration of the script.

## Parameters

*`new_include_path`*
The new value for the include_path

## Return Values

Returns the old include_path on success or **FALSE** on failure.

## Examples

**Example #33 - set_include_path() example**

```
<?php
// Works as of PHP 4.3.0
set_include_path('/inc');

// Works in all PHP versions
ini_set('include_path', '/inc');
?>
```

**Example #34 - Adding to the include path**

Making use of the **PATH_SEPARATOR** constant, it is possible to extend the include path regardless of the operating system.

In this example we add */usr/lib/pear* to the end of the existing *include_path*.

```
<?php
$path = '/usr/lib/pear';
set_include_path(get_include_path() . PATH_SEPARATOR . $path);
?>
```

## See Also

- ini_set()
- get_include_path()
- restore_include_path()
- **include()**

# set_magic_quotes_runtime

set_magic_quotes_runtime -- Sets the current active configuration setting of magic_quotes_runtime

## Description

bool **set_magic_quotes_runtime** ( int `$new_setting` )

Set the current active configuration setting of magic_quotes_runtime.

## Parameters

*`new_setting`*
: 0 for off, 1 for on.

## Return Values

Returns **TRUE** on success or **FALSE** on failure.

## See Also

- get_magic_quotes_gpc()
- get_magic_quotes_runtime()

# set_time_limit

set_time_limit -- Limits the maximum execution time

## Description

void **set_time_limit** ( int `$seconds` )

Set the number of seconds a script is allowed to run. If this is reached, the script returns a fatal error. The default limit is 30 seconds or, if it exists, the *max_execution_time* value defined in the *php.ini*.

When called, set_time_limit() restarts the timeout counter from zero. In other words, if the timeout is the default 30 seconds, and 25 seconds into script execution a call such as *set_time_limit(20)* is made, the script will run for a total of 45 seconds before timing out.

## Parameters

*seconds*
The maximum execution time, in seconds. If set to zero, no time limit is imposed.

## Return Values

No value is returned.

## Notes

| **Warning** |
| --- |
| This function has no effect when PHP is running in safe mode. There is no workaround other than turning off safe mode or changing the time limit in the *php.ini*. |

| **Note** |
| --- |
| The set_time_limit() function and the configuration directive max_execution_time only affect the execution time of the script itself. Any time spent on activity that happens outside the execution of the script such as system calls using system(), stream operations, database queries, etc. is not included when determining the maximum time that the script has been running. |

## See Also

- max_execution_time
- max_input_time

# sys_get_temp_dir

sys_get_temp_dir -- Returns directory path used for temporary files

## Description

string **sys_get_temp_dir** ( void )

Returns the path of the directory PHP stores temporary files in by default.

## Return Values

Returns the path of the temporary directory.

## See Also

- tmpfile()
- tempnam()

# version_compare

version_compare -- Compares two "PHP-standardized" version number strings

## Description

mixed **version_compare** ( string `$version1`, string `$version2` [, string `$operator` ] )

version_compare() compares two "PHP-standardized" version number strings. This is useful if you would like to write programs working only on some versions of PHP.

The function first replaces _, - and + with a dot. in the version strings and also inserts dots. before and after any non number so that for example '4.3.2RC1' becomes '4.3.2.RC.1'. Then it splits the results like if you were using explode('.', $ver). Then it compares the parts starting from left to right. If a part contains special version strings these are handled in the following order: *dev* < *alpha* = *a* < *beta* = *b* < *RC* < *pl*. This way not only versions with different levels like '4.1' and '4.1.2' can be compared but also any PHP specific version containing development state.

## Parameters

*`version1`*
First version number.

*`version2`*
Second version number.

*`operator`*
If you specify the third optional *`operator`* argument, you can test for a particular relationship. The possible operators are: <, *lt*, <=, *le*, >, *gt*, >=, *ge*, ==, =, *eq*, *!=*, <>, *ne* respectively. This parameter is case-sensitive, so values should be lowercase.

## Return Values

By default, version_compare() returns *-1* if the first version is lower than the second, *0* if they are equal, and *1* if the second is lower.

When using the optional *`operator`* argument, the function will return **TRUE** if the relationship is the one specified by the operator, **FALSE** otherwise.

## Examples

The examples below use the **PHP_VERSION** constant, because it contains the value of the PHP version that is executing the code.

**Example #35 - version_compare() examples**

```
<?php
if (version_compare(PHP_VERSION, '6.0.0') === 1) {
   echo 'I am at least PHP version 6.0.0, my version: ' . PHP_VERSION .
"\n";
}

if (version_compare(PHP_VERSION, '5.3.0') === 1) {
   echo 'I am at least PHP version 5.3.0, my version: ' . PHP_VERSION .
"\n";
}

if (version_compare(PHP_VERSION, '5.0.0', '>')) {
   echo 'I am using PHP 5, my version: ' . PHP_VERSION . "\n";
}

if (version_compare(PHP_VERSION, '5.0.0', '<')) {
   echo 'I am using PHP 4, my version: ' . PHP_VERSION . "\n";
}
?>
```

## Notes

**Note**

The **PHP_VERSION** constant holds current PHP version.

**Note**

Note that pre-release versions, such as 5.3.0-dev, are considered lower than their final release counterparts (like 5.3.0).

## See Also

- phpversion()
- php_uname()
- function_exists()

# zend_logo_guid

zend_logo_guid -- Gets the Zend guid

## Description

string **zend_logo_guid** ( void )

This function returns the ID which can be used to display the Zend logo using the built-in image.

## Return Values

Returns *PHPE9568F35-D428-11d2-A769-00AA001ACF42*.

## Examples

**Example #36 - zend_logo_guid() example**

```
<?php

echo '<img src="' . $_SERVER['PHP_SELF'] .
    '?=' . zend_logo_guid() . '" alt="Zend Logo !" />';

?>
```

## See Also

- php_logo_guid()

# zend_thread_id

zend_thread_id -- Returns a unique identifier for the current thread

## Description

int **zend_thread_id** ( void )

This function returns an unique identifier for the current thread.

## Return Values

Returns the thread id as an integer.

## Notes

| **Note** |
| --- |
| This function is only available if PHP has been built with ZTS (Zend Thread Safety) support. |

# zend_version

zend_version -- Gets the version of the current Zend engine

## Description

string **zend_version** ( void )

Returns a string containing the version of the currently running Zend Engine.

## Return Values

Returns the Zend Engine version number, as a string.

## Examples

**Example #37 -** zend_version() **example**

```
<?php
echo "Zend engine version: " . zend_version();
?>
```

The above example will output something similar to:

```
Zend engine version: 2.2.0
```

## See Also

- phpinfo()
- phpcredits()
- php_logo_guid()
- phpversion()