Filesystem

Introduction

No external libraries are needed to build this extension, but if you want PHP to support LFS (large files) on Linux, then you need to have a recent glibc and you need compile PHP with the following compiler flags: *-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64*.

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*.

Filesystem and Streams Configuration Options

Name	Default	Changeable	Changelog
allow_url_fopen	"1"	PHP_INI_ALL	PHP_INI_ALL in PHP <= 4.3.4. PHP_INI_SYSTEM in PHP < 6. Available since PHP 4.0.4.
allow_url_include	"0"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP 5. Available since PHP 5.2.0.
user_agent	NULL	PHP_INI_ALL	Available since PHP 4.3.0.
default_socket_timeout	"60"	PHP_INI_ALL	Available since PHP 4.3.0.
from	""	PHP_INI_ALL	
auto_detect_line_endings	"0"	PHP_INI_ALL	Available since PHP 4.3.0.

Here's a short explanation of the configuration directives.

allow_url_fopen boolean

This option enables the URL-aware fopen wrappers that enable accessing URL object like files. Default wrappers are provided for the access of remote files using the ftp or http protocol, some extensions like zlib may register additional wrappers.

Note
This setting can only be set in php.ini due to security reasons.

Note
This option was introduced immediately after the release of version 4.0.3. For versions up to and including 4.0.3 you can only disable this feature at compile time by using the configuration switch *--disable-url-fopen-wrapper*.

Warning
On Windows versions prior to PHP 4.3.0, the following functions do not support remote file accessing: **include()**, **include_once()**, **require()**, **require_once()** and the imagecreatefromXXX functions in the GD Functions extension.

allow_url_include boolean

This option allows the use of URL-aware fopen wrappers with the following functions: **include()**, **include_once()**, **require()**, **require_once()**.

Note
This setting requires allow_url_fopen to be on.

user_agent string

Define the user agent for PHP to send.

default_socket_timeout integer

Default timeout (in seconds) for socket based streams.

Note
This configuration option was introduced in PHP 4.3.0

from string

Define the anonymous ftp password (your email address).

auto_detect_line_endings boolean

When turned on, PHP will examine the data read by fgets() and file() to see if it is using Unix, MS-Dos or Macintosh line-ending conventions. This enables PHP to interoperate with Macintosh systems, but defaults to Off, as there is a very small performance penalty when detecting the EOL conventions for the first line, and also because people using carriage-returns as item separators under Unix systems would experience non-backwards-compatible behaviour.

Note
This configuration option was introduced in PHP 4.3.0

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

GLOB_BRACE (integer)

GLOB_ONLYDIR (integer)

GLOB_MARK (integer)

GLOB_NOSORT (integer)

GLOB_NOCHECK (integer)

GLOB_NOESCAPE (integer)

PATHINFO_DIRNAME (integer)

PATHINFO_BASENAME (integer)

PATHINFO_EXTENSION (integer)

PATHINFO_FILENAME (integer)
Since PHP 5.2.0.

FILE_USE_INCLUDE_PATH (integer)
Search for *filename* in include_path (since PHP 5).

FILE_APPEND (integer)
Append content to existing file.

FILE_IGNORE_NEW_LINES (integer)
Strip EOL characters (since PHP 5).

FILE_SKIP_EMPTY_LINES (integer)
Skip empty lines (since PHP 5).

FILE_BINARY (integer)
Binary mode (since PHP 6).

FILE_TEXT (integer)
Text mode (since PHP 6).

Filesystem Functions

See Also

For related functions, see also the Directory and Program Execution sections.

For a list and explanation of the various URL wrappers that can be used as remote files, see also List of Supported Protocols/Wrappers.

basename

basename -- Returns filename component of path

Description

string **basename** (string `$path` [, string `$suffix`])

Given a string containing a path to a file, this function will return the base name of the file.

Parameters

`path`
A path. On Windows, both slash (/) and backslash (\ ) are used as directory separator character. In other environments, it is the forward slash (/).

`suffix`
If the filename ends in *`suffix`* this will also be cut off.

Return Values

Returns the base name of the given *`path`*.

ChangeLog

Version	Description
4.1.0	The *`suffix`* parameter was added

Examples

Example #1 - basename() example

```
<?php
$path = "/home/httpd/html/index.php";
$file = basename($path);         // $file is set to "index.php"
$file = basename($path, ".php"); // $file is set to "index"
?>
```

See Also

- dirname()
- pathinfo()

chgrp

chgrp -- Changes file group

Description

bool **chgrp** (string `$filename`, mixed `$group`)

Attempts to change the group of the file *`filename`* to *`group`*.

Only the superuser may change the group of a file arbitrarily; other users may change the group of a file to any group of which that user is a member.

Parameters

`filename`
: Path to the file.

`group`
: A group name or number.

Return Values

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

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
When safe mode is enabled, PHP checks whether the files or directories being operated upon have the same UID (owner) as the script that is being executed.

See Also

- chown()

- chmod()

chmod

chmod -- Changes file mode

Description

bool **chmod** (string `$filename`, int `$mode`)

Attempts to change the mode of the specified file to that given in *`mode`*.

Parameters

`filename`
: Path to the file.

`mode`
: Note that *`mode`* is not automatically assumed to be an octal value, so strings (such as "g+w") will not work properly. To ensure the expected operation, you need to prefix *`mode`* with a zero (0):

```
<?php
chmod("/somedir/somefile", 755);   // decimal; probably incorrect
chmod("/somedir/somefile", "u+rwx,go+rx"); // string; incorrect
chmod("/somedir/somefile", 0755);  // octal; correct value of mode
?>
```

The *`mode`* parameter consists of three octal number components specifying access restrictions for the owner, the user group in which the owner is in, and to everybody else in this order. One component can be computed by adding up the needed permissions for that target user base. Number 1 means that you grant execute rights, number 2 means that you make the file writeable, number 4 means that you make the file readable. Add up these numbers to specify needed rights. You can also read more about modes on Unix systems with 'man 1 chmod' and 'man 2 chmod'.

```
<?php
// Read and write for owner, nothing for everybody else
chmod("/somedir/somefile", 0600);

// Read and write for owner, read for everybody else
chmod("/somedir/somefile", 0644);

// Everything for owner, read and execute for others
chmod("/somedir/somefile", 0755);

// Everything for owner, read and execute for owner's group
chmod("/somedir/somefile", 0750);
?>
```

Return Values

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

Notes

Note
The current user is the user under which PHP runs. It is probably not the same user you use for normal shell or FTP access. The mode can be changed only by user who owns the file on most systems.

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
When safe mode is enabled, PHP checks whether the files or directories you are about to operate on have the same UID (owner) as the script that is being executed. In addition, you cannot set the SUID, SGID and sticky bits.

See Also

- chown()
- chgrp()

chown

chown -- Changes file owner

Description

bool **chown** (string `$filename`, mixed `$user`)

Attempts to change the owner of the file *`filename`* to user *`user`*. Only the superuser may change the owner of a file.

Parameters

`filename`
: Path to the file.

`user`
: A user name or number.

Return Values

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

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
When safe mode is enabled, PHP checks whether the files or directories being operated upon have the same UID (owner) as the script that is being executed.

See Also

- chmod()

clearstatcache

clearstatcache -- Clears file status cache

Description

void **clearstatcache** (void)

When you use stat(), lstat(), or any of the other functions listed in the affected functions list (below), PHP caches the information those functions return in order to provide faster performance. However, in certain cases, you may want to clear the cached information. For instance, if the same file is being checked multiple times within a single script, and that file is in danger of being removed or changed during that script's operation, you may elect to clear the status cache. In these cases, you can use the clearstatcache() function to clear the information that PHP caches about a file.

You should also note that PHP doesn't cache information about non-existent files. So, if you call file_exists() on a file that doesn't exist, it will return **FALSE** until you create the file. If you create the file, it will return **TRUE** even if you then delete the file. However unlink() clears the cache automatically.

Note
This function caches information about specific filenames, so you only need to call clearstatcache() if you are performing multiple operations on the same filename and require the information about that particular file to not be cached.

Affected functions include stat(), lstat(), file_exists(), is_writable(), is_readable(), is_executable(), is_file(), is_dir(), is_link(), filectime(), fileatime(), filemtime(), fileinode(), filegroup(), fileowner(), filesize(), filetype(), and fileperms().

Return Values

No value is returned.

copy

copy -- Copies file

Description

bool **copy** (string `$source`, string `$dest` [, resource `$context`])

Makes a copy of the file *`source`* to *`dest`*.

If you wish to move a file, use the rename() function.

Parameters

`source`
Path to the source file.

`dest`
The destination path. If *`dest`* is a URL, the copy operation may fail if the wrapper does not support overwriting of existing files.

Warning
If the destination file already exists, it will be overwritten.

`context`
A valid context resource created with stream_context_create().

Return Values

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

ChangeLog

Version	Description
5.3.0	Added context support.
4.3.0	Both *`source`* and *`dest`* may now be URLs if the "fopen wrappers" have been enabled. See fopen() for more details.

Examples

Example #2 - copy() example

```
<?php
$file = 'example.txt';
$newfile = 'example.txt.bak';

if (!copy($file, $newfile)) {
   echo "failed to copy $file...\n";
}
?>
```

See Also

- move_uploaded_file()
- rename()
- The section of the manual about handling file uploads

delete

delete -- See unlink() or unset()

Description

void **delete** (void)

This is a dummy manual entry to satisfy those people who are looking for unlink() or unset() in the wrong place.

Return Values

No value is returned.

See Also

- unlink() to delete files
- unset() to delete variables

dirname

dirname -- Returns directory name component of path

Description

string **dirname** (string `$path`)

Given a string containing a path to a file, this function will return the name of the directory.

Parameters

`path`
: A path. On Windows, both slash (/) and backslash (\ ) are used as directory separator character. In other environments, it is the forward slash (/).

Return Values

Returns the name of the directory. If there are no slashes in *`path`*, a dot ('. ') is returned, indicating the current directory. Otherwise, the returned string is *`path`* with any trailing */component* removed.

ChangeLog

Version	Description
5.0.0	dirname() is now binary safe
4.0.3	dirname() was fixed to be POSIX-compliant.

Examples

Example #3 - dirname() example

```
<?php
$path = "/etc/passwd";
$file = dirname($path); // $file is set to "/etc"
?>
```

Notes

Note
Since PHP 4.3.0, you will often get a slash or a dot back from dirname() in situations where the older functionality would have given you the empty string.

Check the following change example:

```
<?php

//before PHP 4.3.0
dirname('c:/'); // returned '.'

//after PHP 4.3.0
dirname('c:/x'); // returns 'c:\'
dirname('c:/Temp/x'); // returns 'c:/Temp'
dirname('/x'); // returns '\'

?>
```

See Also

- basename()
- pathinfo()
- realpath()

disk_free_space

disk_free_space -- Returns available space in directory

Description

float **disk_free_space** (string `$directory`)

Given a string containing a directory, this function will return the number of bytes available on the corresponding filesystem or disk partition.

Parameters

`directory`
A directory of the filesystem or disk partition.

Note
Given a file name instead of a directory, the behaviour of the function is unspecified and may differ between operating systems and PHP versions.

Return Values

Returns the number of available bytes as a float.

Examples

Example #4 - disk_free_space() **example**

```
<?php
// $df contains the number of bytes available on "/"
$df = disk_free_space("/");

// On Windows:
disk_free_space("C:");
disk_free_space("D:");
?>
```

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

See Also

- disk_total_space()

disk_total_space

disk_total_space -- Returns the total size of a directory

Description

float **disk_total_space** (string `$directory`)

Given a string containing a directory, this function will return the total number of bytes on the corresponding filesystem or disk partition.

Parameters

`directory`
A directory of the filesystem or disk partition.

Return Values

Returns the total number of bytes as a float.

Examples

Example #5 - disk_total_space() **example**

```
<?php
// $df contains the total number of bytes available on "/"
$df = disk_total_space("/");

// On Windows:
disk_total_space("C:");
disk_total_space("D:");
?>
```

Notes

Note

This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

See Also

- disk_free_space()

diskfreespace

diskfreespace -- Alias of disk_free_space()

Description

This function is an alias of: disk_free_space().

fclose

fclose -- Closes an open file pointer

Description

bool **fclose** (resource `$handle`)

The file pointed to by *`handle`* is closed.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen().

Return Values

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

Examples

Example #6 - A simple fclose() example

```
<?php

$handle = fopen('somefile.txt', 'r');

fclose($handle);

?>
```

See Also

- fopen()
- fsockopen()

feof

feof -- Tests for end-of-file on a file pointer

Description

bool **feof** (resource `$handle`)

Tests for end-of-file on a file pointer.

Parameters

`handle`
The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

Return Values

Returns **TRUE** if the file pointer is at EOF or an error occurs (including socket timeout); otherwise returns **FALSE**.

Notes

Warning

If a connection opened by fsockopen() wasn't closed by the server, feof() will wait until a timeout has been reached to return **TRUE**. The default timeout value is 60 seconds. You may use stream_set_timeout() to change this value.

Warning

If passed file pointer is not valid you may get an infinite loop, because EOF fails to return TRUE.

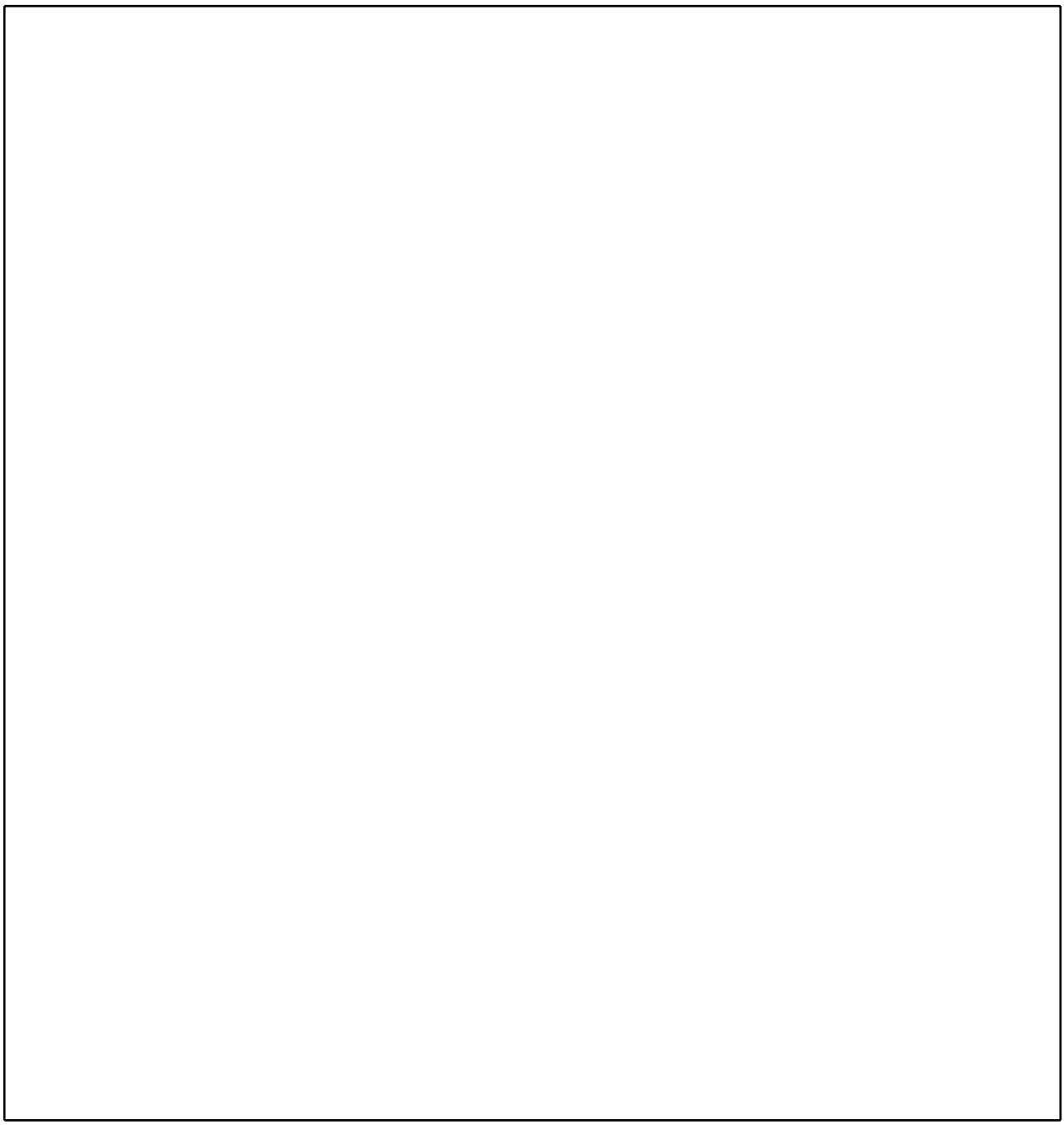
Example #7 - feof() example with an invalid file pointer

```
<?php
// if file can not be read or doesn't exist fopen function returns FALSE
$file = @fopen("no_such_file", "r");

// FALSE from fopen will issue warning and result in infinite loop here
while (!feof($file)) {
}

fclose($file);
```

```
?>
```



fflush

fflush -- Flushes the output to a file

Description

bool **fflush** (resource `$handle`)

This function forces a write of all buffered output to the resource pointed to by the file *`handle`*.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

Return Values

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

fgetc

fgetc -- Gets character from file pointer

Description

string **fgetc** (resource `$handle`)

Gets a character from the given file pointer.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

Return Values

Returns a string containing a single character read from the file pointed to by *`handle`*. Returns **FALSE** on EOF.

Warning

This function may return Boolean **FALSE**, but may also return a non-Boolean value which evaluates to **FALSE**, such as *0* or "". Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Examples

Example #8 - A fgetc() example

```
<?php
$fp = fopen('somefile.txt', 'r');
if (!$fp) {
   echo 'Could not open file somefile.txt';
}
while (false !== ($char = fgetc($fp))) {
   echo "$char\n";
}
?>
```

Notes

Note
This function is binary-safe.

See Also

- fread()
- fopen()
- popen()
- fsockopen()
- fgets()

fgetcsv

fgetcsv -- Gets line from file pointer and parse for CSV fields

Description

array **fgetcsv** (resource `$handle` [, int `$length` [, string `$delimiter` [, string `$enclosure` [, string `$escape`]]]])

Similar to fgets() except that fgetcsv() parses the line it reads for fields in CSV format and returns an array containing the fields read.

Parameters

`handle`
A valid file pointer to a file successfully opened by fopen(), popen(), or fsockopen().

`length`
Must be greater than the longest line (in characters) to be found in the CSV file (allowing for trailing line-end characters). It became optional in PHP 5. Omitting this parameter (or setting it to 0 in PHP 5.0.4 and later) the maximum line length is not limited, which is slightly slower.

`delimiter`
Set the field delimiter (one character only). Defaults as a comma.

`enclosure`
Set the field enclosure character (one character only). Defaults as a double quotation mark.

`escape`
Set the escape character (one character only). Defaults as a backslash (\ )

Return Values

Returns an indexed array containing the fields read.

Note
A blank line in a CSV file will be returned as an array comprising a single null field, and will not be treated as an error.

Note
If PHP is not properly recognizing the line endings when reading files either on or created by a Macintosh computer, enabling the auto_detect_line_endings run-time configuration option may help resolve the problem.

fgetcsv() returns **FALSE** on error, including end of file.

ChangeLog

Version	Description
5.3.0	The *escape* parameter was added
4.3.5	fgetcsv() is now binary safe
4.3.0	The *enclosure* parameter was added

Examples

Example #9 - Read and print the entire contents of a CSV file

```
<?php
$row = 1;
$handle = fopen("test.csv", "r");
while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
   $num = count($data);
   echo "<p> $num fields in line $row: <br /></p>\n";
   $row++;
   for ($c=0; $c < $num; $c++) {
       echo $data[$c] . "<br />\n";
   }
}
fclose($handle);
?>
```

Notes

Note
Locale setting is taken into account by this function. If *LANG* is e.g. *en_US.UTF-8*, files in one-byte encoding are read wrong by this function.

See Also

- str_getcsv()
- explode()
- file()
- pack()
- fputcsv()

fgets

fgets -- Gets line from file pointer

Description

string **fgets** (resource `$handle` [, int `$length`])

Gets a line from file pointer.

Parameters

`handle`
The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

`length`
Reading ends when *`length`* - 1 bytes have been read, on a newline (which is included in the return value), or on EOF (whichever comes first). If no length is specified, it will keep reading from the stream until it reaches the end of the line.

Note
Until PHP 4.3.0, omitting it would assume 1024 as the line length. If the majority of the lines in the file are all larger than 8KB, it is more resource efficient for your script to specify the maximum line length.

Return Values

Returns a string of up to *`length`* - 1 bytes read from the file pointed to by *`handle`*.

If an error occurs, returns **FALSE**.

ChangeLog

Version	Description
4.3.0	fgets() is now binary safe
4.2.0	The *`length`* parameter became optional

Examples

Example #10 - Reading a file line by line

```
<?php
$handle = @fopen("/tmp/inputfile.txt", "r");
if ($handle) {
   while (!feof($handle)) {
       $buffer = fgets($handle, 4096);
       echo $buffer;
   }
   fclose($handle);
}
?>
```

Notes

Note

If PHP is not properly recognizing the line endings when reading files either on or created by a Macintosh computer, enabling the auto_detect_line_endings run-time configuration option may help resolve the problem.

Note

People used to the 'C' semantics of fgets() should note the difference in how *EOF* is returned.

See Also

- fgetss()
- fread()
- fgetc()
- stream_get_line()
- fopen()
- popen()
- fsockopen()
- stream_set_timeout()

fgetss

fgetss -- Gets line from file pointer and strip HTML tags

Description

string **fgetss** (resource `$handle` [, int `$length` [, string `$allowable_tags`]])

Identical to fgets(), except that fgetss() attempts to strip any HTML and PHP tags from the text it reads.

Parameters

`handle`
The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

`length`
Length of the data to be retrieved.

`allowable_tags`
You can use the optional third parameter to specify tags which should not be stripped.

Return Values

Returns a string of up to *`length`* - 1 bytes read from the file pointed to by *`handle`*, with all HTML and PHP code striped.

If an error occurs, returns **FALSE**.

ChangeLog

Version	Description
5.0.0	The *`length`* parameter is optional
3.0.13 and 4.0.0	The *`allowable_tags`* parameter was added,

Notes

Note
If PHP is not properly recognizing the line endings when reading files either on or created by a Macintosh computer, enabling the auto_detect_line_endings run-time configuration option may help resolve the problem.

See Also

- fgets()
- fopen()
- popen()
- fsockopen()
- strip_tags()

file_exists

file_exists -- Checks whether a file or directory exists

Description

bool **file_exists** (string `$filename`)

Checks whether a file or directory exists.

Parameters

`filename`
Path to the file or directory. On windows, use *//computername/share/filename* or *\\computername\share\filename* to check files on network shares.

Return Values

Returns **TRUE** if the file or directory specified by *`filename`* exists; **FALSE** otherwise.

Note
This function will return **FALSE** for symlinks pointing to non-existing files.

Warning
This function returns **FALSE** for files inaccessible due to safe mode restrictions. However these files still can be included if they are located in safe_mode_include_dir.

Note
The check is done using the real UID/GID instead of the effective one.

Examples

Example #11 - Testing whether a file exists

```
<?php
$filename = '/path/to/foo.txt';
```

```
if (file_exists($filename)) {
   echo "The file $filename exists";
} else {
   echo "The file $filename does not exist";
}
?>
```

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_readable()
- is_writable()
- is_file()
- file()

file_get_contents

file_get_contents -- Reads entire file into a string

Description

string **file_get_contents** (string `$filename` [, int `$flags` [, resource `$context` [, int `$offset` [, int `$maxlen`]]]])

This function is similar to file(), except that file_get_contents() returns the file in a string, starting at the specified *`offset`* up to *`maxlen`* bytes. On failure, file_get_contents() will return **FALSE**.

file_get_contents() is the preferred way to read the contents of a file into a string. It will use memory mapping techniques if supported by your OS to enhance performance.

Note
If you're opening a URI with special characters, such as spaces, you need to encode the URI with urlencode().

Parameters

`filename`
Name of the file to read.

`flags`

Warning
For all versions prior to PHP 6, this parameter is called *`use_include_path`* and is a bool. The *`flags`* parameter is only available since PHP 6. If you use an older version and want to search for *`filename`* in the include path, this parameter must be **TRUE**. Since PHP 6, you have to use the **FILE_USE_INCLUDE_PATH** flag instead.

The value of *`flags`* can be any combination of the following flags (with some restrictions), joined with the binary OR (|) operator.

Available flags

Flag	Description
FILE_USE_INCLUDE_PATH	Search for *`filename`* in the include directory. See include_path for more information.

FILE_TEXT	If unicode semantics are enabled, the default encoding of the read data is UTF-8. You can specify a different encoding by creating a custom context or by changing the default using **stream_default_encoding()**. This flag cannot be used with **FILE_BINARY**.
FILE_BINARY	With this flag, the file is read in binary mode. This is the default setting and cannot be used with **FILE_TEXT**.

context
: A valid context resource created with stream_context_create(). If you don't need to use a custom context, you can skip this parameter by **NULL**.

offset
: The offset where the reading starts.

maxlen
: Maximum length of data read.

Return Values

The function returns the read data or **FALSE** on failure.

ChangeLog

Version	Description
5.0.0	Added context support.
5.1.0	Added the *offset* and *maxlen* parameters.
6.0.0	The *use_include_path* parameter was replaced by the *flags* parameter.

Notes

Note
This function is binary-safe.

Tip

A URL can be used as a filename with this function if the fopen wrappers have been enabled. See fopen() for more details on how to specify the filename and List of Supported Protocols/Wrappers for a list of supported URL protocols.

Warning

When using SSL, Microsoft IIS will violate the protocol by closing the connection without sending a *close_notify* indicator. PHP will report this as "SSL: Fatal Protocol Error" when you reach the end of the data. To work around this, the value of error_reporting should be lowered to a level that does not include warnings. PHP 4.3.7 and higher can detect buggy IIS server software when you open the stream using the *https://* wrapper and will suppress the warning. When using fsockopen() to create an *ssl://* socket, the developer is responsible for detecting and suppressing this warning.

See Also

- file()
- fgets()
- fread()
- readfile()
- file_put_contents()
- stream_get_contents()
- stream_context_create()

file_put_contents

file_put_contents -- Write a string to a file

Description

int **file_put_contents** (string `$filename`, mixed `$data` [, int `$flags` [, resource `$context`]])

This function is identical to calling fopen(), fwrite() and fclose() successively to write data to a file.

If *`filename`* does not exist, the file is created. Otherwise, the existing file is overwritten, unless the **FILE_APPEND** flags is set.

Parameters

`filename`
: Path to the file where to write the data.

`data`
: The data to write. Can be either a string, an array or a stream resource (explained above). If *`data`* is a stream resource, the remaining buffer of that stream will be copied to the specified file. This is similar with using stream_copy_to_stream(). You can also specify the *`data`* parameter as a single dimension array. This is equivalent to *file_put_contents($filename, implode('', $array))*.

`flags`
: The value of *`flags`* can be any combination of the following flags (with some restrictions), joined with the binary OR (|) operator.

Available flags

Flag	Description
FILE_USE_INCLUDE_PATH	Search for *`filename`* in the include directory. See include_path for more information.
FILE_APPEND	If file *`filename`* already exists, append the data to the file instead of overwriting it.
LOCK_EX	Acquire an exclusive lock on the file while proceeding to the writing.
FILE_TEXT	*`data`* is written in text mode. If unicode semantics are enabled, the default encoding is UTF-8. You can specify a different encoding by creating a custom context or by

	using the **stream_default_encoding()** to change the default. This flag cannot be used with **FILE_BINARY**. This flag is only available since PHP 6.
FILE_BINARY	*data* will be written in binary mode. This is the default setting and cannot be used with **FILE_TEXT**. This flag is only available since PHP 6.

context
A valid context resource created with stream_context_create().

Return Values

The function returns the number of bytes that were written to the file, or **FALSE** on failure.

ChangeLog

Version	Description
5.0.0	Added context support
5.1.0	Added support for **LOCK_EX** and the ability to pass a stream resource to the *data* parameter
6.0.0	Added support for the **FILE_TEXT** and **FILE_BINARY** flags

Notes

Note
This function is binary-safe.

Tip
A URL can be used as a filename with this function if the fopen wrappers have been enabled. See fopen() for more details on how to specify the filename and List of Supported Protocols/Wrappers for a list of supported URL protocols.

See Also

- fopen()
- fwrite()
- file_get_contents()
- stream_context_create()

file

file -- Reads entire file into an array

Description

array **file** (string `$filename` [, int `$flags` [, resource `$context`]])

Reads an entire file into an array.

Note
You can use file_get_contents() to return the contents of a file as a string.

Parameters

`filename`
Path to the file.

Tip
A URL can be used as a filename with this function if the fopen wrappers have been enabled. See fopen() for more details on how to specify the filename and List of Supported Protocols/Wrappers for a list of supported URL protocols.

`flags`
The optional parameter *`flags`* can be one, or more, of the following constants:
FILE_USE_INCLUDE_PATH
Search for the file in the include_path.

FILE_IGNORE_NEW_LINES
Do not add newline at the end of each array element

FILE_SKIP_EMPTY_LINES
Skip empty lines

FILE_TEXT
The content is returned in UTF-8 encoding. You can specify a different encoding by creating a custom context. This flag cannot be used with **FILE_BINARY**. This flag is only available since PHP 6.

FILE_BINARY
The content is read as binary data. This is the default setting and cannot be used with **FILE_TEXT**. This flag is only available since PHP 6.

context
A context resource created with the stream_context_create() function.

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

Returns the file in an array. Each element of the array corresponds to a line in the file, with the newline still attached. Upon failure, file() returns **FALSE**.

Note
Each line in the resulting array will include the line ending, unless **FILE_IGNORE_NEW_LINES** is used, so you still need to use rtrim() if you do not want the line ending present.

Note
If PHP is not properly recognizing the line endings when reading files either on or created by a Macintosh computer, enabling the auto_detect_line_endings run-time configuration option may help resolve the problem.

ChangeLog

Version	Description
6.0.0	Added support for the **FILE_TEXT** and **FILE_BINARY** flags.
5.0.0	The *context* parameter was added
5.0.0	Prior to PHP 5.0.0 the *flags* parameter only covered include_path and was enabled with 1
4.3.0	file() became binary safe

Examples

Example #12 - file() example

```
<?php
// Get a file into an array.  In this example we'll go through HTTP to get
// the HTML source of a URL.
$lines = file('http://www.example.com/');

// Loop through our array, show HTML source as HTML source; and line numbers
too.
foreach ($lines as $line_num => $line) {
   echo "Line #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br
/>\n";
}

// Another example, let's get a web page into a string.  See also
file_get_contents().
$html = implode('', file('http://www.example.com/'));

// Using the optional flags parameter since PHP 5
$trimmed = file('somefile.txt', FILE_IGNORE_NEW_LINES |
FILE_SKIP_EMPTY_LINES);
?>
```

Notes

Warning

When using SSL, Microsoft IIS will violate the protocol by closing the connection without sending a *close_notify* indicator. PHP will report this as "SSL: Fatal Protocol Error" when you reach the end of the data. To work around this, the value of error_reporting should be lowered to a level that does not include warnings. PHP 4.3.7 and higher can detect buggy IIS server software when you open the stream using the *https://* wrapper and will suppress the warning. When using fsockopen() to create an *ssl://* socket, the developer is responsible for detecting and suppressing this warning.

See Also

- readfile()
- fopen()
- fsockopen()
- popen()
- file_get_contents()
- **include()**
- stream_context_create()

fileatime

fileatime -- Gets last access time of file

Description

int **fileatime** (string `$filename`)

Gets the last access time of the given file.

Parameters

`filename`
Path to the file.

Return Values

Returns the time the file was last accessed, or **FALSE** in case of an error. The time is returned as a Unix timestamp.

Examples

Example #13 - fileatime() example

```
<?php

// outputs e.g.  somefile.txt was last accessed: December 29 2002 22:16:23.

$filename = 'somefile.txt';
if (file_exists($filename)) {
   echo "$filename was last accessed: " . date("F d Y H:i:s.",
fileatime($filename));
}

?>
```

Notes

Note

The atime of a file is supposed to change whenever the data blocks of a file are being read. This can be costly performance-wise when an application regularly accesses a very large number of files or directories.

Some Unix filesystems can be mounted with atime updates disabled to increase the performance of such applications; USENET news spools are a common example. On such filesystems this function will be useless.

Note

The results of this function are cached. See clearstatcache() for more details.

Tip

As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- filemtime()
- fileinode()
- date()

filectime

filectime -- Gets inode change time of file

Description

int **filectime** (string `$filename`)

Gets the inode change time of a file.

Parameters

`filename`
Path to the file.

Return Values

Returns the time the file was last changed, or **FALSE** in case of an error. The time is returned as a Unix timestamp.

Examples

Example #14 - A filectime() example

```
<?php

// outputs e.g.  somefile.txt was last changed: December 29 2002 22:16:23.

$filename = 'somefile.txt';
if (file_exists($filename)) {
   echo "$filename was last changed: " . date("F d Y H:i:s.",
filectime($filename));
}

?>
```

Notes

Note

Note: In most Unix filesystems, a file is considered changed when its inode data is changed; that is, when the permissions, owner, group, or other metadata from the inode is updated. See also filemtime() (which is what you want to use when you want

to create "Last Modified" footers on web pages) and fileatime().

Note

Note also that in some Unix texts the ctime of a file is referred to as being the creation time of the file. This is wrong. There is no creation time for Unix files in most Unix filesystems.

Note

The results of this function are cached. See clearstatcache() for more details.

Tip

As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- filemtime()

filegroup

filegroup -- Gets file group

Description

int **filegroup** (string `$filename`)

Gets the file group. The group ID is returned in numerical format, use posix_getgrgid() to resolve it to a group name.

Parameters

`filename`
Path to the file.

Return Values

Returns the group ID of the file, or **FALSE** in case of an error. The group ID is returned in numerical format, use posix_getgrgid() to resolve it to a group name. Upon failure, **FALSE** is returned.

Errors/Exceptions

Upon failure, an **E_WARNING** is emitted.

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- fileowner()
- safe_mode_gid

fileinode

fileinode -- Gets file inode

Description

int **fileinode** (string `$filename`)

Gets the file inode.

Parameters

`filename`
Path to the file.

Return Values

Returns the inode number of the file, or **FALSE** in case of an error.

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- stat()

filemtime

filemtime -- Gets file modification time

Description

int **filemtime** (string `$filename`)

This function returns the time when the data blocks of a file were being written to, that is, the time when the content of the file was changed.

Parameters

`filename`
Path to the file.

Return Values

Returns the time the file was last modified, or **FALSE** in case of an error. The time is returned as a Unix timestamp, which is suitable for the date() function.

Examples

Example #15 - filemtime() example

```
<?php
// outputs e.g.  somefile.txt was last modified: December 29 2002 22:16:23.

$filename = 'somefile.txt';
if (file_exists($filename)) {
   echo "$filename was last modified: " . date ("F d Y H:i:s.",
filemtime($filename));
}
?>
```

Notes

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- filectime()
- stat()
- touch()
- getlastmod()

fileowner

fileowner -- Gets file owner

Description

int **fileowner** (string `$filename`)

Gets the file owner.

Parameters

`filename`
Path to the file.

Return Values

Returns the user ID of the owner of the file, or **FALSE** in case of an error. The user ID is returned in numerical format, use posix_getpwuid() to resolve it to a username.

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- stat()

fileperms

fileperms -- Gets file permissions

Description

int **fileperms** (string `$filename`)

Gets permissions for the given file.

Parameters

`filename`
Path to the file.

Return Values

Returns the permissions on the file, or **FALSE** in case of an error.

Examples

Example #16 - Display permissions as an octal value

```
<?php
echo substr(sprintf('%o', fileperms('/tmp')), -4);
echo substr(sprintf('%o', fileperms('/etc/passwd')), -4);
?>
```

The above example will output:

```
1777
0644
```

Example #17 - Display full permissions

```
<?php
$perms = fileperms('/etc/passwd');

if (($perms & 0xC000) == 0xC000) {
   // Socket
   $info = 's';
} elseif (($perms & 0xA000) == 0xA000) {
   // Symbolic Link
   $info = 'l';
} elseif (($perms & 0x8000) == 0x8000) {
```

```
   // Regular
   $info = '-';
} elseif (($perms & 0x6000) == 0x6000) {
   // Block special
   $info = 'b';
} elseif (($perms & 0x4000) == 0x4000) {
   // Directory
   $info = 'd';
} elseif (($perms & 0x2000) == 0x2000) {
   // Character special
   $info = 'c';
} elseif (($perms & 0x1000) == 0x1000) {
   // FIFO pipe
   $info = 'p';
} else {
   // Unknown
   $info = 'u';
}

// Owner
$info .= (($perms & 0x0100) ? 'r' : '-');
$info .= (($perms & 0x0080) ? 'w' : '-');
$info .= (($perms & 0x0040) ?
           (($perms & 0x0800) ? 's' : 'x' ) :
           (($perms & 0x0800) ? 'S' : '-'));

// Group
$info .= (($perms & 0x0020) ? 'r' : '-');
$info .= (($perms & 0x0010) ? 'w' : '-');
$info .= (($perms & 0x0008) ?
           (($perms & 0x0400) ? 's' : 'x' ) :
           (($perms & 0x0400) ? 'S' : '-'));

// World
$info .= (($perms & 0x0004) ? 'r' : '-');
$info .= (($perms & 0x0002) ? 'w' : '-');
$info .= (($perms & 0x0001) ?
           (($perms & 0x0200) ? 't' : 'x' ) :
           (($perms & 0x0200) ? 'T' : '-'));

echo $info;
?>
```

The above example will output:

```
-rw-r--r--
```

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_readable()
- stat()

filesize

filesize -- Gets file size

Description

int **filesize** (string `$filename`)

Gets the size for the given file.

Parameters

`filename`
Path to the file.

Return Values

Returns the size of the file in bytes, or **FALSE** (and generates an error of level **E_WARNING**) in case of an error.

Note
Because PHP's integer type is signed and many platforms use 32bit integers, filesize() may return unexpected results for files which are larger than 2GB. For files between 2GB and 4GB in size this can usually be overcome by using *sprintf("%u", filesize($file))* .

Examples

Example #18 - filesize() example

```
<?php

// outputs e.g.  somefile.txt: 1024 bytes

$filename = 'somefile.txt';
echo $filename . ': ' . filesize($filename) . ' bytes';

?>
```

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- file_exists()

filetype

filetype -- Gets file type

Description

string **filetype** (string `$filename`)

Returns the type of the given file.

Parameters

`filename`
Path to the file.

Return Values

Returns the type of the file. Possible values are fifo, char, dir, block, link, file, socket and unknown.

Returns **FALSE** if an error occurs. filetype() will also produce an **E_NOTICE** message if the stat call fails or if the file type is unknown.

Examples

Example #19 - filetype() example

```
<?php

echo filetype('/etc/passwd');  // file
echo filetype('/etc/');        // dir

?>
```

Notes

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_dir()
- is_file()
- is_link()
- file_exists()
- stat()
- mime_content_type()

flock

flock -- Portable advisory file locking

Description

bool **flock** (resource `$handle`, int `$operation` [, int `&$wouldblock`])

flock() allows you to perform a simple reader/writer model which can be used on virtually every platform (including most Unix derivatives and even Windows).

The lock is released also by fclose() (which is also called automatically when script finished).

PHP supports a portable way of locking complete files in an advisory way (which means all accessing programs have to use the same way of locking or it will not work).

Parameters

`handle`
An open file pointer.

`operation`
`operation` is one of the following:

- **LOCK_SH** to acquire a shared lock (reader).
- **LOCK_EX** to acquire an exclusive lock (writer).
- **LOCK_UN** to release a lock (shared or exclusive).
- **LOCK_NB** if you don't want flock() to block while locking. (not supported on Windows)

`wouldblock`
The optional third argument is set to **TRUE** if the lock would block (EWOULDBLOCK errno condition).

Return Values

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

ChangeLog

Version	Description
4.0.1	The *LOCK_XXX* constants were added. Prior to that you must use 1 for **LOCK_SH**, 2 for **LOCK_EX**, 3 for **LOCK_UN** and 4 for **LOCK_NB**

Examples

Example #20 - flock() example

```
<?php

$fp = fopen("/tmp/lock.txt", "w+");

if (flock($fp, LOCK_EX)) { // do an exclusive lock
   fwrite($fp, "Write something here\n");
   flock($fp, LOCK_UN); // release the lock
} else {
   echo "Couldn't lock the file !";
}

fclose($fp);

?>
```

Notes

Note

flock() locks mandatory under Windows.

Note

Because flock() requires a file pointer, you may have to use a special lock file to protect access to a file that you intend to truncate by opening it in write mode (with a "w" or "w+" argument to fopen()).

Warning

flock() will not work on NFS and many other networked file systems. Check your operating system documentation for more details.

On some operating systems flock() is implemented at the process level. When using a multithreaded server API like ISAPI you may not be able to rely on flock() to protect files against other PHP scripts running in parallel threads of the same server instance!

flock() is not supported on antiquated filesystems like *FAT* and its derivates and will therefore always return **FALSE** under this environments (this is especially true for Windows 98 users).

fnmatch

fnmatch -- Match filename against a pattern

Description

bool **fnmatch** (string `$pattern`, string `$string` [, int `$flags`])

fnmatch() checks if the passed *`string`* would match the given shell wildcard *`pattern`*.

Parameters

`pattern`
The shell wildcard pattern.

`string`
The tested string. This function is especially useful for filenames, but may also be used on regular strings. The average user may be used to shell patterns or at least in their simplest form to *'?'* and *'*'* wildcards so using fnmatch() instead of ereg() or preg_match() for frontend search expression input may be way more convenient for non-programming users.

`flags`
See the Unix manpage on *fnmatch(3)* for flag names (as long as they are not documented here).

Return Values

Returns **TRUE** if there is a match, **FALSE** otherwise.

Examples

Example #21 - Checking a color name against a shell wildcard pattern

```
<?php
if (fnmatch("*gr[ae]y", $color)) {
 echo "some form of gray ...";
}
?>
```

Notes

Warning
For now this function is not available on Windows or other non-POSIX compliant systems.

See Also

- glob()
- ereg()
- preg_match()
- sscanf()
- printf()
- sprintf()

fopen

fopen -- Opens file or URL

Description

resource **fopen** (string `$filename`, string `$mode` [, bool `$use_include_path` [, resource `$context`]])

fopen() binds a named resource, specified by *`filename`*, to a stream.

Parameters

`filename`

If *`filename`* is of the form "scheme://...", it is assumed to be a URL and PHP will search for a protocol handler (also known as a wrapper) for that scheme. If no wrappers for that protocol are registered, PHP will emit a notice to help you track potential problems in your script and then continue as though *`filename`* specifies a regular file. If PHP has decided that *`filename`* specifies a local file, then it will try to open a stream on that file. The file must be accessible to PHP, so you need to ensure that the file access permissions allow this access. If you have enabled safe mode, or open_basedir further restrictions may apply. If PHP has decided that *`filename`* specifies a registered protocol, and that protocol is registered as a network URL, PHP will check to make sure that allow_url_fopen is enabled. If it is switched off, PHP will emit a warning and the fopen call will fail.

Note
The list of supported protocols can be found in List of Supported Protocols/Wrappers. Some protocols (also referred to as *wrappers*) support *context* and/or *php.ini* options. Refer to the specific page for the protocol in use for a list of options which can be set. (e.g. *php.ini* value *user_agent* used by the *http* wrapper).

On the Windows platform, be careful to escape any backslashes used in the path to the file, or use forward slashes.

```
<?php
$handle = fopen("c:\\data\\info.txt", "r");
?>
```

`mode`

The *`mode`* parameter specifies the type of access you require to the stream. It may be any of the following:

A list of possible modes for fopen() using *`mode`*

mode	**Description**
'r'	Open for reading only; place the file pointer at the beginning of the file.
'r+'	Open for reading and writing; place the file pointer at the beginning of the file.
'w'	Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'w+'	Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'a'	Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'a+'	Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'x'	Create and open for writing only; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning **FALSE** and generating an error of level **E_WARNING**. If the file does not exist, attempt to create it. This is equivalent to specifying *O_EXCL|O_CREAT* flags for the underlying *open(2)* system call.
'x+'	Create and open for reading and writing; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning **FALSE** and generating an error of level **E_WARNING**. If the file does not exist, attempt to create it. This is equivalent to specifying *O_EXCL|O_CREAT* flags for the underlying *open(2)* system call.

Note
Different operating system families have different line-ending conventions. When you write a text file and want to insert a line break, you need to use the correct line-ending character(s) for your operating system. Unix based systems use *\n* as the line ending character, Windows based systems use *\r\n* as the line ending characters and

Macintosh based systems use *\r* as the line ending character.

If you use the wrong line ending characters when writing your files, you might find that other applications that open those files will "look funny".

Windows offers a text-mode translation flag (*'t'*) which will transparently translate *\n* to *\r\n* when working with the file. In contrast, you can also use *'b'* to force binary mode, which will not translate your data. To use these flags, specify either *'b'* or *'t'* as the last character of the `mode` parameter.

The default translation mode depends on the SAPI and version of PHP that you are using, so you are encouraged to always specify the appropriate flag for portability reasons. You should use the *'t'* mode if you are working with plain-text files and you use *\n* to delimit your line endings in your script, but expect your files to be readable with applications such as notepad. You should use the *'b'* in all other cases.

If you do not specify the 'b' flag when working with binary files, you may experience strange problems with your data, including broken image files and strange problems with *\r\n* characters.

Note

For portability, it is strongly recommended that you always use the 'b' flag when opening files with fopen().

Note

Again, for portability, it is also strongly recommended that you re-write code that uses or relies upon the *'t'* mode so that it uses the correct line endings and *'b'* mode instead.

`use_include_path`

The optional third `use_include_path` parameter can be set to '1' or **TRUE** if you want to search for the file in the include_path, too.

`context`

Note

Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

Returns a file pointer resource on success, or **FALSE** on error.

Errors/Exceptions

If the open fails, the function an error of level **E_WARNING** is generated. You may use @ to suppress this warning.

ChangeLog

Version	Description
4.3.2	As of PHP 4.3.2, the default mode is set to binary for all platforms that distinguish between binary and text mode. If you are having problems with your scripts after upgrading, try using the *'t'* flag as a workaround until you have made your script more portable as mentioned below
4.3.2	The *'x'* and *'x+'* option was added

Examples

Example #22 - fopen() examples

```
<?php
$handle = fopen("/home/rasmus/file.txt", "r");
$handle = fopen("/home/rasmus/file.gif", "wb");
$handle = fopen("http://www.example.com/", "r");
$handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
?>
```

Notes

Warning

When using SSL, Microsoft IIS will violate the protocol by closing the connection without sending a *close_notify* indicator. PHP will report this as "SSL: Fatal Protocol Error" when you reach the end of the data. To work around this, the value of error_reporting should be lowered to a level that does not include warnings. PHP 4.3.7 and higher can detect buggy IIS server software when you open the stream using the *https://* wrapper and will suppress the warning. When using fsockopen() to create an *ssl://* socket, the developer is responsible for detecting and suppressing this warning.

Note
When safe mode is enabled, PHP checks whether the directory in which the script is operating has the same UID (owner) as the script that is being executed.

If you are experiencing problems with reading and writing to files and you're using the server module version of PHP, remember to make sure that the files and directories you're using are accessible to the server process.

See Also

- List of Supported Protocols/Wrappers
- fclose()
- fgets()
- fread()
- fwrite()
- fsockopen()
- file()
- file_exists()
- is_readable()
- stream_set_timeout()
- popen()
- stream_context_create()

fpassthru

fpassthru -- Output all remaining data on a file pointer

Description

int **fpassthru** (resource `$handle`)

Reads to EOF on the given file pointer from the current position and writes the results to the output buffer.

You may need to call rewind() to reset the file pointer to the beginning of the file if you have already written data to the file.

If you just want to dump the contents of a file to the output buffer, without first modifying it or seeking to a particular offset, you may want to use the readfile(), which saves you the fopen() call.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

Return Values

If an error occurs, fpassthru() returns **FALSE**. Otherwise, fpassthru() returns the number of characters read from *`handle`* and passed through to the output.

Examples

Example #23 - Using fpassthru() with binary files

```
<?php

// open the file in a binary mode
$name = './img/ok.png';
$fp = fopen($name, 'rb');

// send the right headers
header("Content-Type: image/png");
header("Content-Length: " . filesize($name));

// dump the picture and stop the script
fpassthru($fp);
exit;
```

```
?>
```

Notes

Note
When using fpassthru() on a binary file on Windows systems, you should make sure to open the file in binary mode by appending a *b* to the mode used in the call to fopen(). You are encouraged to use the *b* flag when dealing with binary files, even if your system does not require it, so that your scripts will be more portable.

See Also

- readfile()
- fopen()
- popen()
- fsockopen()

fputcsv

fputcsv -- Format line as CSV and write to file pointer

Description

int **fputcsv** (resource `$handle`, array `$fields` [, string `$delimiter` [, string `$enclosure`]])

fputcsv() formats a line (passed as a *`fields`* array) as CSV and write it (terminated by a newline) to the specified file *`handle`*.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or fsockopen() (and not yet closed by fclose()).

`fields`
: An array of values.

`delimiter`
: The optional *`delimiter`* parameter sets the field delimiter (one character only). Defaults as a comma:,.

`enclosure`
: The optional *`enclosure`* parameter sets the field enclosure (one character only) and defaults to a double quotation mark: ".

Return Values

Returns the length of the written string, or **FALSE** on failure.

Examples

Example #24 - fputcsv() example

```
<?php

$list = array (
    'aaa,bbb,ccc,dddd',
    '123,456,789',
    '"aaa","bbb"'
);

$fp = fopen('file.csv', 'w');
```

```
foreach ($list as $line) {
   fputcsv($fp, split(',', $line));
}

fclose($fp);
?>
```

Notes

Note
If PHP is not properly recognizing the line endings when reading files either on or created by a Macintosh computer, enabling the auto_detect_line_endings run-time configuration option may help resolve the problem.

See Also

- fgetcsv()

fputs

fputs -- Alias of fwrite()

Description

This function is an alias of: fwrite().

fread

fread -- Binary-safe file read

Description

string **fread** (resource `$handle`, int `$length`)

fread() reads up to *`length`* bytes from the file pointer referenced by *`handle`*. Reading stops as soon as one of the following conditions is met:

- *`length`* bytes have been read
- EOF (end of file) is reached
- a packet becomes available (for network streams)
- 8192 bytes have been read (after opening userspace stream)

Parameters

`handle`
A file system pointer resource that is typically created using fopen().

`length`
Up to *`length`* number of bytes read.

Return Values

Returns the read string or **FALSE** in case of error.

Examples

Example #25 - A simple fread() example

```
<?php
// get contents of a file into a string
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
?>
```

Example #26 - Binary fread() example

Warning

On systems which differentiate between binary and text files (i.e. Windows) the file must be opened with 'b' included in fopen() mode parameter.

```
<?php
$filename = "c:\\files\\somepic.gif";
$handle = fopen($filename, "rb");
$contents = fread($handle, filesize($filename));
fclose($handle);
?>
```

Example #27 - Remote fread() examples

Warning

When reading from anything that is not a regular local file, such as streams returned when reading remote files or from popen() and fsockopen(), reading will stop after a packet is available. This means that you should collect the data together in chunks as shown in the examples below.

```
<?php
// For PHP 5 and up
$handle = fopen("http://www.example.com/", "rb");
$contents = stream_get_contents($handle);
fclose($handle);
?>

<?php
$handle = fopen("http://www.example.com/", "rb");
$contents = '';
while (!feof($handle)) {
 $contents .= fread($handle, 8192);
}
fclose($handle);
?>
```

Notes

Note
If you just want to get the contents of a file into a string, use file_get_contents() as it has much better performance than the code above.

See Also

- fwrite()
- fopen()
- fsockopen()
- popen()
- fgets()
- fgetss()
- fscanf()
- file()
- fpassthru()

fscanf

fscanf -- Parses input from a file according to a format

Description

mixed **fscanf** (resource `$handle`, string `$format` [, mixed `&$...`])

The function fscanf() is similar to sscanf(), but it takes its input from a file associated with *`handle`* and interprets the input according to the specified *`format`*, which is described in the documentation for sprintf().

Any whitespace in the format string matches any whitespace in the input stream. This means that even a tab \t in the format string can match a single space character in the input stream.

Parameters

`handle`
A file system pointer resource that is typically created using fopen().

`format`
The specified format as described in the sprintf() documentation.

`...`
The optional assigned values.

Return Values

If only two parameters were passed to this function, the values parsed will be returned as an array. Otherwise, if optional parameters are passed, the function will return the number of assigned values. The optional parameters must be passed by reference.

ChangeLog

Version	Description
4.3.0	Before this time, the maximum number of characters read from the file was 512 (or up to the first \n, whichever came first). But now, arbitrarily long lines will be read and scanned.

Examples

Example #28 - fscanf() Example

```
<?php
$handle = fopen("users.txt", "r");
while ($userinfo = fscanf($handle, "%s\t%s\t%s\n")) {
   list ($name, $profession, $countrycode) = $userinfo;
   //... do something with the values
}
fclose($handle);
?>
```

Example #29 - Contents of users.txt

```
javier  argonaut        pe
hiroshi sculptor        jp
robert  slacker us
luigi   florist it
```

See Also

- fread()
- fgets()
- fgetss()
- sscanf()
- printf()
- sprintf()

fseek

fseek -- Seeks on a file pointer

Description

int **fseek** (resource `$handle`, int `$offset` [, int `$whence`])

Sets the file position indicator for the file referenced by *`handle`*. The new position, measured in bytes from the beginning of the file, is obtained by adding *`offset`* to the position specified by *`whence`*.

Parameters

`handle`
A file system pointer resource that is typically created using fopen().

`offset`
The offset. To move to a position before the end-of-file, you need to pass a negative value in *`offset`*.

`whence`
`whence` values are:

- **SEEK_SET** - Set position equal to *`offset`* bytes.
- **SEEK_CUR** - Set position to current location plus *`offset`*.
- **SEEK_END** - Set position to end-of-file plus *`offset`*.

If *`whence`* is not specified, it is assumed to be **SEEK_SET**.

Return Values

Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

Examples

Example #30 - fseek() example

```
<?php

$fp = fopen('somefile.txt', 'r');

// read some data
$data = fgets($fp, 4096);

// move back to the beginning of the file
```

```
// same as rewind($fp);
fseek($fp, 0);

?>
```

Notes

Note
If you have opened the file in append ("a" or "a+") mode, any data you write to the file will always be appended, regardless of the file position.

Note
May not be used on file pointers returned by fopen() if they use the "http://" or "ftp://" formats. fseek() gives also undefined results for append-only streams (opened with "a" flag).

See Also

- ftell()
- rewind()

fstat

fstat -- Gets information about a file using an open file pointer

Description

array **fstat** (resource `$handle`)

Gathers the statistics of the file opened by the file pointer *`handle`*. This function is similar to the stat() function except that it operates on an open file pointer instead of a filename.

Parameters

`handle`
A file system pointer resource that is typically created using fopen().

Return Values

Returns an array with the statistics of the file; the format of the array is described in detail on the stat() manual page.

Examples

Example #31 - fstat() example

```
<?php

// open a file
$fp = fopen("/etc/passwd", "r");

// gather statistics
$fstat = fstat($fp);

// close the file
fclose($fp);

// print only the associative part
print_r(array_slice($fstat, 13));

?>
```

The above example will output something similar to:

```
Array
(
   [dev] => 771
   [ino] => 488704
```

```
    [mode] => 33188
    [nlink] => 1
    [uid] => 0
    [gid] => 0
    [rdev] => 0
    [size] => 1114
    [atime] => 1061067181
    [mtime] => 1056136526
    [ctime] => 1056136526
    [blksize] => 4096
    [blocks] => 8
)
```

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

ftell

ftell -- Tells file pointer read/write position

Description

int **ftell** (resource `$handle`)

Tells the file pointer read/write position.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen() or popen(). ftell() gives undefined results for append-only streams (opened with "a" flag).

Return Values

Returns the position of the file pointer referenced by *`handle`*; i.e., its offset into the file stream.

If an error occurs, returns **FALSE**.

Examples

Example #32 - ftell() example

```
<?php

// opens a file and read some data
$fp = fopen("/etc/passwd", "r");
$data = fgets($fp, 12);

// where are we ?
echo ftell($fp); // 11

fclose($fp);

?>
```

See Also

- fopen()
- popen()
- fseek()
- rewind()

ftruncate

ftruncate -- Truncates a file to a given length

Description

bool **ftruncate** (resource `$handle`, int `$size`)

Takes the filepointer, *`handle`*, and truncates the file to length, *`size`*.

Parameters

`handle`
The file pointer.

Note
The *`handle`* must be open for writing.

`size`
The size to truncate to.

Note
If *`size`* is larger than the file it is extended with null bytes. If *`size`* is smaller than the extra data will be lost.

Return Values

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

ChangeLog

Version	Description
PHP 4.3.3	Prior to this release ftruncate() returned an integer value of 1 on success, instead of boolean **TRUE**.

Notes

Note
The file pointer is *not* changed.

See Also

- fopen()
- fseek()

fwrite

fwrite -- Binary-safe file write

Description

int **fwrite** (resource `$handle`, string `$string` [, int `$length`])

fwrite() writes the contents of *`string`* to the file stream pointed to by *`handle`*.

Parameters

`handle`
: A file system pointer resource that is typically created using fopen().

`string`
: The string that is to be written.

`length`
: If the *`length`* argument is given, writing will stop after *`length`* bytes have been written or the end of *`string`* is reached, whichever comes first. Note that if the *`length`* argument is given, then the magic_quotes_runtime configuration option will be ignored and no slashes will be stripped from *`string`*.

Return Values

fwrite() returns the number of bytes written, or **FALSE** on error.

Notes

Note
On systems which differentiate between binary and text files (i.e. Windows) the file must be opened with 'b' included in fopen() mode parameter.

Note
If *`handle`* was fopen() ed in append mode, fwrite() s are atomic (unless the size of *`string`* exceeds the filesystem's block size, on some platforms, and as long as the file is on a local filesystem). That is, there is no need to flock() a resource before calling fwrite(); all of the data will be written without interruption.

Note

If writing twice to the file pointer, then the data will be appended to the end of the file content, meaning that the example below wouldn't work as expected:

```
<?php
$fp = fopen('data.txt', 'w');
fwrite($fp, '1');
fwrite($fp, '23');
fclose($fp);

// the content of 'data.txt' is now 123 and not 23!
?>
```

Examples

Example #33 - A simple fwrite() example

```
<?php
$filename = 'test.txt';
$somecontent = "Add this to the file\n";

// Let's make sure the file exists and is writable first.
if (is_writable($filename)) {

   // In our example we're opening $filename in append mode.
   // The file pointer is at the bottom of the file hence
   // that's where $somecontent will go when we fwrite() it.
   if (!$handle = fopen($filename, 'a')) {
         echo "Cannot open file ($filename)";
         exit;
   }

   // Write $somecontent to our opened file.
   if (fwrite($handle, $somecontent) === FALSE) {
       echo "Cannot write to file ($filename)";
       exit;
   }

   echo "Success, wrote ($somecontent) to file ($filename)";

   fclose($handle);

} else {
   echo "The file $filename is not writable";
}
?>
```

See Also

- fread()
- fopen()
- fsockopen()
- popen()
- file_get_contents()

glob

glob -- Find pathnames matching a pattern

Description

array **glob** (string `$pattern` [, int `$flags`])

The glob() function searches for all the pathnames matching *`pattern`* according to the rules used by the libc glob() function, which is similar to the rules used by common shells.

Parameters

`pattern`
: The pattern. No tilde expansion or parameter substitution is done.

`flags`
: Valid flags:

- **GLOB_MARK** - Adds a slash to each item returned
- **GLOB_NOSORT** - Return files as they appear in the directory (no sorting)
- **GLOB_NOCHECK** - Return the search pattern if no files matching it were found
- **GLOB_NOESCAPE** - Backslashes do not quote metacharacters
- **GLOB_BRACE** - Expands {a,b,c} to match 'a', 'b', or 'c'
- **GLOB_ONLYDIR** - Return only directory entries which match the pattern
- **GLOB_ERR** - Stop on read errors (like unreadable directories), by default errors are ignored.

Return Values

Returns an array containing the matched files/directories, an empty array if no file matched or **FALSE** on error.

Note
On some systems it is impossible to distinguish between empty match and an error.

ChangeLog

Version	Description
5.1.0	**GLOB_ERR** was added
4.3.3	**GLOB_ONLYDIR** became available on Windows and other systems not using the GNU C library

Examples

Example #34 - Convenient way how glob() can replace opendir() and friends.

```
<?php
foreach (glob("*.txt") as $filename) {
   echo "$filename size " . filesize($filename) . "\n";
}
?>
```

The above example will output something similar to:

```
funclist.txt size 44686
funcsummary.txt size 267625
quickref.txt size 137820
```

Notes

Note

This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note

This function isn't available on some systems (e.g. old Sun OS).

Note

The **GLOB_BRACE** flag is not available on some non GNU systems, like Solaris.

See Also

- opendir()
- readdir()
- closedir()
- fnmatch()

is_dir

is_dir -- Tells whether the filename is a directory

Description

bool **is_dir** (string `$filename`)

Tells whether the given filename is a directory.

Parameters

`filename`
: Path to the file. If *`filename`* is a relative filename, it will be checked relative to the current working directory.

Return Values

Returns **TRUE** if the filename exists and is a directory, **FALSE** otherwise.

Examples

Example #35 - is_dir() example

```
<?php
var_dump(is_dir('a_file.txt'));
var_dump(is_dir('bogus_dir/abc'));

var_dump(is_dir('..')); //one dir up
?>
```

The above example will output:

```
bool(false)
bool(false)
bool(true)
```

Notes

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- chdir()
- dir
- opendir()
- is_file()
- is_link()

is_executable

is_executable -- Tells whether the filename is executable

Description

bool **is_executable** (string `$filename`)

Tells whether the filename is executable.

Parameters

`filename`
Path to the file.

Return Values

Returns **TRUE** if the filename exists and is executable, or **FALSE** on error.

ChangeLog

Version	Description
5.0.0	is_executable() became available with Windows

Examples

Example #36 - is_executable() example

```
<?php

$file = '/home/vincent/somefile.sh';

if (is_executable($file)) {
   echo $file.' is executable';
} else {
   echo $file.' is not executable';
}

?>
```

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_file()
- is_link()

is_file

is_file -- Tells whether the filename is a regular file

Description

bool **is_file** (string `$filename`)

Tells whether the given file is a regular file.

Parameters

`filename`
Path to the file.

Return Values

Returns **TRUE** if the filename exists and is a regular file, **FALSE** otherwise.

Examples

Example #37 - is_file() example

```
<?php
var_dump(is_file('a_file.txt')) . "\n";
var_dump(is_file('/usr/bin/')) . "\n";
?>
```

The above example will output:

```
bool(true)
bool(false)
```

Notes

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_dir()
- is_link()

is_link

is_link -- Tells whether the filename is a symbolic link

Description

bool **is_link** (string `$filename`)

Tells whether the given file is a symbolic link.

Parameters

`filename`
Path to the file.

Return Values

Returns **TRUE** if the filename exists and is a symbolic link, **FALSE** otherwise.

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_dir()
- is_file()
- readlink()

is_readable

is_readable -- Tells whether the filename is readable

Description

bool **is_readable** (string `$filename`)

Tells whether the filename is readable.

Parameters

`filename`
Path to the file.

Return Values

Returns **TRUE** if the file or directory specified by *`filename`* exists and is readable, **FALSE** otherwise.

Examples

Example #38 - is_readable() example

```
<?php
$filename = 'test.txt';
if (is_readable($filename)) {
   echo 'The file is readable';
} else {
   echo 'The file is not readable';
}
?>
```

Notes

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account before PHP 5.1.5.

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

Note
The check is done using the real UID/GID instead of the effective one.

See Also

- is_writable()
- file_exists()
- fgets()

is_uploaded_file

is_uploaded_file -- Tells whether the file was uploaded via HTTP POST

Description

bool **is_uploaded_file** (string `$filename`)

Returns **TRUE** if the file named by *`filename`* was uploaded via HTTP POST. This is useful to help ensure that a malicious user hasn't tried to trick the script into working on files upon which it should not be working--for instance, */etc/passwd*.

This sort of check is especially important if there is any chance that anything done with uploaded files could reveal their contents to the user, or even to other users on the same system.

For proper working, the function is_uploaded_file() needs an argument like $_FILES['userfile']['tmp_name'], - the name of the uploaded file on the clients machine $_FILES['userfile']['name'] does not work.

Parameters

`filename`
The filename being checked.

Return Values

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

Examples

Example #39 - is_uploaded_file() example

```
<?php

if (is_uploaded_file($_FILES['userfile']['tmp_name'])) {
  echo "File ". $_FILES['userfile']['name'] ." uploaded successfully.\n";
  echo "Displaying contents\n";
  readfile($_FILES['userfile']['tmp_name']);
} else {
  echo "Possible file upload attack: ";
  echo "filename '". $_FILES['userfile']['tmp_name'] . "'.";
}

?>
```

See Also

- move_uploaded_file()
- See Handling file uploads for a simple usage example.

is_writable

is_writable -- Tells whether the filename is writable

Description

bool **is_writable** (string `$filename`)

Returns **TRUE** if the *`filename`* exists and is writable. The filename argument may be a directory name allowing you to check if a directory is writable.

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

Parameters

`filename`
The filename being checked.

Return Values

Returns **TRUE** if the *`filename`* exists and is writable.

Examples

Example #40 - is_writable() example

```
<?php
$filename = 'test.txt';
if (is_writable($filename)) {
   echo 'The file is writable';
} else {
   echo 'The file is not writable';
}
?>
```

Notes

Note

The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- is_readable()
- file_exists()
- fwrite()

is_writeable

is_writeable -- Alias of is_writable()

Description

This function is an alias of: is_writable().

lchgrp

lchgrp -- Changes group ownership of symlink

Description

bool **lchgrp** (string `$filename`, mixed `$group`)

Attempts to change the group of the symlink *`filename`* to *`group`*.

Only the superuser may change the group of a symlink arbitrarily; other users may change the group of a symlink to any group of which that user is a member.

Parameters

`filename`
Path to the symlink.

`group`
The group specified by name or number.

Return Values

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

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
When safe mode is enabled, PHP checks whether the files or directories being operated upon have the same UID (owner) as the script that is being executed.

Note
This function is not implemented on Windows platforms.

See Also

- chgrp()
- lchown()
- chown()
- chmod()

lchown

lchown -- Changes user ownership of symlink

Description

bool **lchown** (string `$filename`, mixed `$user`)

Attempts to change the owner of the symlink *`filename`* to user *`user`*.

Only the superuser may change the owner of a symlink.

Parameters

`filename`
: Path to the file.

`user`
: User name or number.

Return Values

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

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
When safe mode is enabled, PHP checks whether the files or directories being operated upon have the same UID (owner) as the script that is being executed.

Note
This function is not implemented on Windows platforms.

See Also

- chgrp()
- lchgrp()
- chgrp()
- chmod()

link

link -- Create a hard link

Description

bool **link** (string `$target`, string `$link`)

link() creates a hard link.

Parameters

`target`
: Target of the link.

`link`
: The link name.

Return Values

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

Notes

Note
This function will not work on remote files as the file to be examined must be accessible via the server's filesystem.

Note
This function is not implemented on Windows platforms.

See Also

- symlink()
- readlink()
- linkinfo()

linkinfo

linkinfo -- Gets information about a link

Description

int **linkinfo** (string `$path`)

Gets information about a link.

This function is used to verify if a link (pointed to by `path`) really exists (using the same method as the S_ISLNK macro defined in *stat.h*).

Parameters

`path`
Path to the link.

Return Values

linkinfo() returns the *st_dev* field of the Unix C stat structure returned by the *lstat* system call. Returns 0 or **FALSE** in case of error.

Examples

Example #41 - linkinfo() example

```
<?php

echo linkinfo('/vmlinuz'); // 835

?>
```

Notes

Note

This function is not implemented on Windows platforms.

See Also

- symlink()
- link()
- readlink()

lstat

lstat -- Gives information about a file or symbolic link

Description

array **lstat** (string `$filename`)

Gathers the statistics of the file or symbolic link named by *`filename`*.

Parameters

`filename`
: Path to a file or a symbolic link.

Return Values

See the manual page for stat() for information on the structure of the array that lstat() returns. This function is identical to the stat() function except that if the *`filename`* parameter is a symbolic link, the status of the symbolic link is returned, not the status of the file pointed to by the symbolic link.

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- stat()

mkdir

mkdir -- Makes directory

Description

bool **mkdir** (string `$pathname` [, int `$mode` [, bool `$recursive` [, resource `$context`]]])

Attempts to create the directory specified by pathname.

Parameters

`pathname`
: The directory path.

`mode`
: The mode is 0777 by default, which means the widest possible access. For more information on modes, read the details on the chmod() page.

Note
`mode` is ignored on Windows.

Note that you probably want to specify the mode as an octal number, which means it should have a leading zero. The mode is also modified by the current umask, which you can change using umask().

`recursive`
: Default to **FALSE**.

`context`

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

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

ChangeLog

Version	Description
5.0.0	The *recursive* parameter was added
5.0.0	As of PHP 5.0.0 mkdir() can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support mkdir()
4.2.0	The *mode* parameter became optional.

Examples

Example #42 - mkdir() example

```
<?php
mkdir("/path/to/my/dir", 0700);
?>
```

Notes

Note

When safe mode is enabled, PHP checks whether the directory in which the script is operating has the same UID (owner) as the script that is being executed.

See Also

- rmdir()

move_uploaded_file

move_uploaded_file -- Moves an uploaded file to a new location

Description

bool **move_uploaded_file** (string `$filename`, string `$destination`)

This function checks to ensure that the file designated by *`filename`* is a valid upload file (meaning that it was uploaded via PHP's HTTP POST upload mechanism). If the file is valid, it will be moved to the filename given by *`destination`*.

This sort of check is especially important if there is any chance that anything done with uploaded files could reveal their contents to the user, or even to other users on the same system.

Parameters

`filename`
The filename of the uploaded file.

`destination`
The destination of the moved file.

Return Values

If *`filename`* is not a valid upload file, then no action will occur, and move_uploaded_file() will return **FALSE**.

If *`filename`* is a valid upload file, but cannot be moved for some reason, no action will occur, and move_uploaded_file() will return **FALSE**. Additionally, a warning will be issued.

Notes

Note
move_uploaded_file() is both safe mode and open_basedir aware. However, restrictions are placed only on the *`destination`* path as to allow the moving of uploaded files in which *`filename`* may conflict with such restrictions. move_uploaded_file() ensures the safety of this operation by allowing only those files uploaded through PHP to be moved.

Warning
If the destination file already exists, it will be overwritten.

See Also

- is_uploaded_file()
- See Handling file uploads for a simple usage example

parse_ini_file

parse_ini_file -- Parse a configuration file

Description

array **parse_ini_file** (string `$filename` [, bool `$process_sections`])

parse_ini_file() loads in the ini file specified in *`filename`*, and returns the settings in it in an associative array.

The structure of the ini file is the same as the *php.ini* 's.

Parameters

`filename`
The filename of the ini file being parsed.

`process_sections`
By setting the last *`process_sections`* parameter to **TRUE**, you get a multidimensional array, with the section names and settings included. The default for *`process_sections`* is **FALSE**

Return Values

The settings are returned as an associative array.

ChangeLog

Version	Description
5.2.4	Keys and section names consisting of numbers are now evaluated as PHP integers thus numbers starting by 0 are evaluated as octals and numbers starting by 0x are evaluated as hexadecimals.
5.0.0	Values enclosed in double quotes can contain new lines.
4.2.1	This function is now affected by safe mode and open_basedir.

Examples

Example #43 - Contents of *sample.ini*

```
; This is a sample configuration file
; Comments start with ';', as in php.ini

[first_section]
one = 1
five = 5
animal = BIRD

[second_section]
path = "/usr/local/bin"
URL = "http://www.example.com/~username"
```

Example #44 - parse_ini_file() example

Constants may also be parsed in the ini file so if you define a constant as an ini value before running parse_ini_file(), it will be integrated into the results. Only ini values are evaluated. For example:

```
<?php

define('BIRD', 'Dodo bird');

// Parse without sections
$ini_array = parse_ini_file("sample.ini");
print_r($ini_array);

// Parse with sections
$ini_array = parse_ini_file("sample.ini", true);
print_r($ini_array);

?>
```

The above example will output something similar to:

```
Array
(
   [one] => 1
   [five] => 5
   [animal] => Dodo bird
   [path] => /usr/local/bin
   [URL] => http://www.example.com/~username
)
Array
(
   [first_section] => Array
       (
           [one] => 1
           [five] => 5
           [animal] = Dodo bird
```

```
        )

    [second_section] => Array
        (
            [path] => /usr/local/bin
            [URL] => http://www.example.com/~username
        )

)
```

Notes

Note
This function has nothing to do with the *php.ini* file. It is already processed, the time you run your script. This function can be used to read in your own application's configuration files.

Note
If a value in the ini file contains any non-alphanumeric characters it needs to be enclosed in double-quotes (").

Note
There are reserved words which must not be used as keys for ini files. These include: null, yes, no, true, and false. Values null, no and false results in "", yes and true results in "1". Characters *{}|&~![()"* must not be used anywhere in the key and have a special meaning in the value.

pathinfo

pathinfo -- Returns information about a file path

Description

mixed **pathinfo** (string `$path` [, int `$options`])

pathinfo() returns an associative array containing information about *`path`*.

Parameters

`path`
: The path being checked.

`options`
: You can specify which elements are returned with optional parameter *`options`*. It composes from **PATHINFO_DIRNAME**, **PATHINFO_BASENAME**, **PATHINFO_EXTENSION** and **PATHINFO_FILENAME**. It defaults to return all elements.

Return Values

The following associative array elements are returned: *dirname*, *basename*, *extension* (if any), and *filename*.

If *`options`* is used, this function will return a string if not all elements are requested.

ChangeLog

Version	Description
5.2.0	The **PATHINFO_FILENAME** constant was added.

Examples

Example #45 - pathinfo() Example

```
<?php
```

```
$path_parts = pathinfo('/www/htdocs/index.html');

echo $path_parts['dirname'], "\n";
echo $path_parts['basename'], "\n";
echo $path_parts['extension'], "\n";
echo $path_parts['filename'], "\n"; // since PHP 5.2.0
?>
```

The above example will output:

```
/www/htdocs
index.html
html
index
```

Notes

Note
For information on retrieving the current path info, read the section on predefined reserved variables.

See Also

- dirname()
- basename()
- parse_url()
- realpath()

pclose

pclose -- Closes process file pointer

Description

int **pclose** (resource `$handle`)

Closes a file pointer to a pipe opened by popen().

Parameters

`handle`
: The file pointer must be valid, and must have been returned by a successful call to popen().

Return Values

Returns the termination status of the process that was run.

See Also

- popen()

popen

popen -- Opens process file pointer

Description

resource **popen** (string `$command`, string `$mode`)

Opens a pipe to a process executed by forking the command given by command.

Parameters

`command`
The command

`mode`
The mode

Return Values

Returns a file pointer identical to that returned by fopen(), except that it is unidirectional (may only be used for reading or writing) and must be closed with pclose(). This pointer may be used with fgets(), fgetss(), and fwrite().

If an error occurs, returns **FALSE**.

Examples

Example #46 - popen() example

```
<?php
$handle = popen("/bin/ls", "r");
?>
```

If the command to be executed could not be found, a valid resource is returned. This may seem odd, but makes sense; it allows you to access any error message returned by the shell:

Example #47 - popen() example

```
<?php
error_reporting(E_ALL);

/* Add redirection so we can get stderr. */
```

```
$handle = popen('/path/to/spooge 2>&1', 'r');
echo "'$handle'; " . gettype($handle) . "\n";
$read = fread($handle, 2096);
echo $read;
pclose($handle);
?>
```

Notes

Note

If you're looking for bi-directional support (two-way), use proc_open().

Note

When safe mode is enabled, you can only execute files within the safe_mode_exec_dir. For practical reasons, it is currently not allowed to have.. components in the path to the executable.

Warning

With safe mode enabled, the command string is escaped with escapeshellcmd(). Thus, *echo y | echo x* becomes *echo y \| echo x*.

See Also

- pclose()
- fopen()
- proc_open()

readfile

readfile -- Outputs a file

Description

int **readfile** (string `$filename` [, bool `$use_include_path` [, resource `$context`]])

Reads a file and writes it to the output buffer.

Parameters

`filename`
The filename being read.

`use_include_path`
You can use the optional second parameter and set it to **TRUE**, if you want to search for the file in the include_path, too.

`context`
A context stream resource.

Return Values

Returns the number of bytes read from the file. If an error occurs, **FALSE** is returned and unless the function was called as @ readfile(), an error message is printed.

Notes

Tip
A URL can be used as a filename with this function if the fopen wrappers have been enabled. See fopen() for more details on how to specify the filename and List of Supported Protocols/Wrappers for a list of supported URL protocols.

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

See Also

- fpassthru()
- file()
- fopen()
- **include()**
- **require()**
- virtual()
- file_get_contents()
- List of Supported Protocols/Wrappers

readlink

readlink -- Returns the target of a symbolic link

Description

string **readlink** (string `$path`)

readlink() does the same as the readlink C function.

Parameters

`path`
The symbolic link path.

Return Values

Returns the contents of the symbolic link path or **FALSE** on error.

Examples

Example #48 - readlink() example

```
<?php

// output e.g. /boot/vmlinux-2.4.20-xfs
echo readlink('/vmlinuz');

?>
```

Notes

Note

This function is not implemented on Windows platforms.

See Also

- is_link()

- symlink()
- linkinfo()

realpath

realpath -- Returns canonicalized absolute pathname

Description

string **realpath** (string `$path`)

realpath() expands all symbolic links and resolves references to '/./', '/../' and extra '/' characters in the input *`path`*. and return the canonicalized absolute pathname.

Parameters

`path`
The path being checked.

Return Values

Returns the canonicalized absolute pathname on success. The resulting path will have no symbolic link, '/./' or '/../' components.

realpath() returns **FALSE** on failure, e.g. if the file does not exist. On BSD systems realpath() doesn't fail if only the last *`path`* component doesn't exist, while other systems will return **FALSE**.

Examples

Example #49 - realpath() example

```
<?php
chdir('/var/www/');
echo realpath('./../../etc/passwd');
?>
```

The above example will output:

```
/etc/passwd
```

Example #50 - realpath() on Windows

On windows realpath() will change unix style paths to windows style.

```
<?php
echo realpath('/windows/system32');
```

```
?>
```

The above example will output:

```
C:\WINDOWS\System32
```

See Also

- basename()
- dirname()
- pathinfo()

rename

rename -- Renames a file or directory

Description

bool **rename** (string `$oldname`, string `$newname` [, resource `$context`])

Attempts to rename *`oldname`* to *`newname`*.

Parameters

`oldname`

Note
The old name. The wrapper used in *`oldname`* *must* match the wrapper used in *`newname`*.

`newname`
The new name.

`context`

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

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

ChangeLog

Version	Description
5.0.0	rename() can now also be used with *some* URL wrappers. Refer to List of Supported

	Protocols/Wrappers for a listing of which wrappers support rename().
4.3.3	rename() is now able to rename files across partitions on *nix based systems.

Examples

Example #51 - Example with rename()

```
<?php
rename("/tmp/tmp_file.txt", "/home/user/login/docs/my_file.txt");
?>
```

See Also

- copy()
- unlink()
- move_uploaded_file()

rewind

rewind -- Rewind the position of a file pointer

Description

bool **rewind** (resource `$handle`)

Sets the file position indicator for *`handle`* to the beginning of the file stream.

Note
If you have opened the file in append ("a" or "a+") mode, any data you write to the file will always be appended, regardless of the file position.

Parameters

`handle`
: The file pointer must be valid, and must point to a file successfully opened by fopen().

Return Values

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

See Also

- fseek()
- ftell()

rmdir

rmdir -- Removes directory

Description

bool **rmdir** (string `$dirname` [, resource `$context`])

Attempts to remove the directory named by *`dirname`*. The directory must be empty, and the relevant permissions must permit this.

Parameters

`dirname`
Path to the directory.

`context`

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

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

ChangeLog

Version	Description
5.0.0	As of PHP 5.0.0 rmdir() can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support rmdir().

Notes

Note
When safe mode is enabled, PHP checks whether the directory in which the script is operating has the same UID (owner) as the script that is being executed.

See Also

- mkdir()
- unlink()

set_file_buffer

set_file_buffer -- Alias of stream_set_write_buffer()

Description

This function is an alias of: stream_set_write_buffer().

stat

stat -- Gives information about a file

Description

array **stat** (string `$filename`)

Gathers the statistics of the file named by *`filename`*. If *`filename`* is a symbolic link, statistics are from the file itself, not the symlink.

lstat() is identical to stat() except it would instead be based off the symlinks status.

Parameters

`filename`
Path to the file.

Return Values

stat() and fstat() result format

Numeric	Associative (since PHP 4.0.6)	Description
0	dev	device number
1	ino	inode number
2	mode	inode protection mode
3	nlink	number of links
4	uid	userid of owner
5	gid	groupid of owner
6	rdev	device type, if inode device *
7	size	size in bytes
8	atime	time of last access (Unix timestamp)
9	mtime	time of last modification

		(Unix timestamp)
10	ctime	time of last inode change (Unix timestamp)
11	blksize	blocksize of filesystem IO *
12	blocks	number of blocks allocated *

* Only valid on systems supporting the st_blksize type - other systems (e.g. Windows) return -1.

In case of error, stat() returns **FALSE**.

Errors/Exceptions

Upon failure, an **E_WARNING** is emitted.

ChangeLog

Version	Description
4.0.6	In addition to returning these attributes in a numeric array, they can be accessed with associative indices, as noted next to each parameter

Notes

Note
The results of this function are cached. See clearstatcache() for more details.

Tip
As of PHP 5.0.0, this function can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support stat() family of functionality.

See Also

- lstat()
- fstat()
- filemtime()
- filegroup()

symlink

symlink -- Creates a symbolic link

Description

bool **symlink** (string `$target`, string `$link`)

symlink() creates a symbolic link to the existing *`target`* with the specified name *`link`*.

Parameters

`target`
: Target of the link.

`link`
: The link name.

Return Values

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

Notes

Note
This function is not implemented on Windows platforms.

See Also

- link() to create hard links
- readlink() along with linkinfo()

tempnam

tempnam -- Create file with unique file name

Description

string **tempnam** (string `$dir`, string `$prefix`)

Creates a file with a unique filename, with access permission set to 0600, in the specified directory. If the directory does not exist, tempnam() may generate a file in the system's temporary directory, and return the name of that.

Parameters

`dir`
The directory where the temporary filename will be created.

`prefix`
The prefix of the generated temporary filename.

Return Values

Returns the new temporary filename, or **FALSE** on failure.

ChangeLog

Version	Description
4.0.6	Prior to PHP 4.0.6, the behaviour of the tempnam() function was system dependent. On Windows the TMP environment variable will override the *`dir`* parameter, on Linux the TMPDIR environment variable has precedence, while SVR4 will always use your *`dir`* parameter if the directory it points to exists. Consult your system documentation on the tempnam(3) function if in doubt.
4.0.3	This function's behavior changed in 4.0.3. The temporary file is also created to avoid a race condition where the file might appear in the filesystem between the time the string was generated and before the script gets

	around to creating the file. Note, that you need to remove the file in case you need it no more, it is not done automatically.

Examples

Example #52 - tempnam() example

```
<?php
$tmpfname = tempnam("/tmp", "FOO");

$handle = fopen($tmpfname, "w");
fwrite($handle, "writing to tempfile");
fclose($handle);

// do here something

unlink($tmpfname);
?>
```

Notes

Note

If PHP cannot create a file in the specified *dir* parameter, it falls back on the system default.

See Also

- tmpfile()
- sys_get_temp_dir()
- unlink()

tmpfile

tmpfile -- Creates a temporary file

Description

resource **tmpfile** (void)

Creates a temporary file with a unique name in read-write (w+) mode and returns a file handle .

The file is automatically removed when closed (using fclose()), or when the script ends.

For details, consult your system documentation on the *tmpfile(3)* function, as well as the *stdio.h* header file.

Return Values

Returns a file handle, similar to the one returned by fopen(), for the new file, or **FALSE** on failure.

Examples

Example #53 - tmpfile() example

```
<?php
$temp = tmpfile();
fwrite($temp, "writing to tempfile");
fseek($temp, 0);
echo fread($temp, 1024);
fclose($temp); // this removes the file
?>
```

The above example will output:

```
writing to tempfile
```

See Also

- tempnam()
- sys_get_temp_dir()

touch

touch -- Sets access and modification time of file

Description

bool **touch** (string `$filename` [, int `$time` [, int `$atime`]])

Attempts to set the access and modification times of the file named in the *`filename`* parameter to the value given in *`time`*. Note that the access time is always modified, regardless of the number of parameters.

If the file does not exist, it will be created.

Parameters

`filename`
The name of the file being touched.

`time`
The touch time. If *`time`* is not supplied, the current system time is used.

`atime`
If present, the access time of the given filename is set to the value of *`atime`*

Return Values

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

Examples

Example #54 - touch() example

```
<?php
if (touch($FileName)) {
   echo "$FileName modification time has been changed to present time";
} else {
   echo "Sorry, could not change modification time of $FileName";
}
?>
```

Notes

Warning
It is not currently possible to change the modification time of a directory with this function under Windows.

umask

umask -- Changes the current umask

Description

int **umask** ([int `$mask`])

umask() sets PHP's umask to *`mask`* & 0777 and returns the old umask. When PHP is being used as a server module, the umask is restored when each request is finished.

Parameters

`mask`
The new umask.

Return Values

umask() without arguments simply returns the current umask otherwise the old umask is returned.

Examples

Example #55 - umask() example

```
<?php
$old = umask(0);
chmod("/path/some_dir/some_file.txt", 0755);
umask($old);

// Checking
if ($old != umask()) {
   die('An error occured while changing back the umask');
}
?>
```

Notes

Note

Avoid using this function in multithreaded webservers. It is better to change the file permissions with chmod() after creating the file. Using umask() can lead to unexpected behavior of concurrently running scripts and the webserver itself because they all use

the same umask.

unlink

unlink -- Deletes a file

Description

bool **unlink** (string `$filename` [, resource `$context`])

Deletes *`filename`*. Similar to the Unix C unlink() function.

Parameters

`filename`
Path to the file.

`context`

Note
Context support was added with PHP 5.0.0. For a description of *contexts*, refer to Stream Functions.

Return Values

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

ChangeLog

Version	Description
5.0.0	As of PHP 5.0.0 unlink() can also be used with *some* URL wrappers. Refer to List of Supported Protocols/Wrappers for a listing of which wrappers support unlink().

See Also

- rmdir() for removing directories