Тема 3.10
«Функции PHP по работе с файлами»

Вступление

PHP, как и любой уважающий себя язык программирования, имеет все необходимые функции для работы с объектами файловой системы, а также для чтения и записи из/в файлы.

Рассмотрим…

Проверка существования

bool file_exists (string filename) – возвращает TRUE, если **объект файловой системы** с именем, указанным в параметре filename, существует, и FALSE в противном случае.

Эта функция НЕ РАЗЛИЧАЕТ файлы, каталоги, символические ссылки и метки томов.

Пример:

```
$filename = '/path/to/test.txt';
if (file_exists($filename)) echo "Что-то есть!";
```

Проверка существования

bool is_file (string filename) – возвращает TRUE, если объект файловой системы существует и является файлом.

bool is_dir (string filename) – возвращает TRUE, если объект файловой системы существует и является каталогом.

bool is_link (string filename) – возвращает TRUE, если объект файловой системы существует и является символической ссылкой (только для POSIX-систем).

mixed readlink (string path) – возвращает содержимое пути символической ссылки или FALSE в случае ошибки (только для POSIX-систем).

Пути

Функции file_exists(), is_file(), is_dir(), is_link() действуют только на локальных файлах (т.е. с их помощью нельзя проверить существование URLa).

Вот некоторые правила описания пути к файлу:

просто указание имени файла означает, что он находится в текущем каталоге;

./files/filename – файл находится в папке files, находящейся в текущем каталоге;

../filename – файл находится в родительском каталоге текущего каталога, каждый знак ../ расценивается как возврат в родительский каталог на уровень выше;

../files/filename – файл находится в папке files, которая находится в родительском каталоге текущего каталога.

Внимание: следует чётко понимать, что путь в URL и путь в файловой системе сервера могут не иметь ничего общего. Так, например, путь в URL http://www.domain.com/ может в файловой системе сервера обозначать /users/domain/http_pub/ или C:/www_pub/

О текущем каталоге, его определении и изменении поговорим чуть-чуть позже.

Размер, дата и время

mixed filesize (string filename) – возвращает размер указанного файла в байтах или FALSE в случае возникновения ошибки.

Поскольку PHP использует знаковое представления для чисел целого типа, а многие платформы используют 32-битные целые числа, функция filesize() может возвращать неожиданные значения для файлов, чей размер превосходит 2 Гб.

Если размер файла находится в пределах 2 - 4 Гб, корректное значение можно получить, используя конструкцию sprintf("%u", filesize($file)).

Пример:

```
$filename = 'somefile.txt';
echo 'Размер файла ' . $filename . ': ' . filesize($filename) . ' байт';
```

Вопрос: как получить размер файла в килобайтах, мегабайтах и т.д.?

Поделить размер в байтах на 1024 и т.д. с последующим округлением.

Размер, дата и время

Функция filesize() НЕ умеет считать «весь размер каталога с подкаталогами, файлами и т.д.». Если её аргументом является имя каталога, она возвращает 0.

Для определения размера каталога нужно организовывать рекурсивный обход ветки каталогов и суммировать размеры всех обнаруженных файлов.

Размер, дата и время

mixed filectime (string filename) – возвращает время последнего изменения файла в формате Unixtime или FALSE в случае ошибки.

Примечание: на большинстве платформ POSIX, файл считается изменённым, если изменены данные его i-узла, что включает информацию о правах на файл, о его владельце и группе и любую другую информацию, содержащуюся в i-узле.

mixed filemtime (string filename) – возвращает время последнего изменения указанного файла в Unixtime или FALSE в случае возникновения ошибки. Данная функция возвращает время последней записи блоков файла, иначе говоря, изменения содержания файла.

int fileatime (string filename) – возвращает в Unixtime время, когда в последний раз был осуществлён доступ к указанному файлу, или FALSE в случае ошибки. Предполагается, что время последнего доступа изменяется во время чтения блоков файла. Это может потребовать значительного количества системных ресурсов, особенно когда приложение обращается к большому числу файлов или каталогов. С целью увеличения производительности, некоторые файловые системы на платформах POSIX могут быть примонтированы с отключённой возможностью обновления времени последнего доступа к файлам. В подобных случаях использование данной функции бессмысленно.

Размер, дата и время

Пример 1:

```
$filename = 'somefile.txt';
if (file_exists($filename))
 {
   echo "в последний раз файл $filename был изменён: " . date ("F d Y H:i:s.", filemtime($filename));
 }
```

Пример 2:

```
$filename = 'somefile.txt';
if (file_exists($filename))
 {
  echo "В последний раз обращение к файлу $filename было произведено: " . date("F d Y H:i:s.", fileatime($filename));
 }
```

Размер, дата и время

bool touch (string filename [, int time [, int atime]]) – устанавливает время доступа и модификации файла с именем filename в значение time.

Если аргумент time не указан, используется текущее время.

Если третий аргумент atime передан, время **доступа** указанного файла будет установлено в значение atime.

Обратите внимание, что время доступа изменяется всегда, независимо от количества аргументов.

Если файл не существует, он будет создан. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Пример:

```
if (touch($FileName)) echo "Время модификации файла $FileName было изменено на текущее";
```

Чтение и запись из/в файлы

array file (string filename [, int use_include_path [, resource context]]) – возвращает содержимое прочитанного **текстового** файла в виде массива. Каждый элемент возвращённого массива содержит соответствующую строку с символами конца строки. В случае ошибки, функция file() возвращает FALSE.

Можно указать необязательный параметр use_include_path, равный 1, если нужно, чтобы поиск файла также производился в каталогах, указанных директивой include_path в php.ini.

Параметр context (аналогично – в других функциях, где он присутствует) будет рассмотрен в конце данной темы.

Чтение и запись из/в файлы

Примеры чтения текстового файла.

Пример 1 (чтение в массив):

```
$arr=file('file.txt');
```

Пример 2 (чтение в строковую переменную в виде непрерывной последовательности данных):

```
$a=implode(file('file.txt'));
```

Внимание! НЕ используйте эту функцию для чтения т.н. «бинарных файлов» (картинок и т.п.), т.к. она пытается определить в них переносы строк и, т.о. результат чтения может отличаться от реального содержимого таких файлов. Несмотря на то, что в руководстве PHP на это нет явного указания, в реальной жизни проблемы всё равно есть.

Чтение и запись из/в файлы

Для чтения всего содержимого файла за один раз в строковую переменную или записи всего содержимого за один раз из переменной в файл удобно использовать следующие функции:

string file_get_contents (string filename [, bool use_include_path [, resource context [, int offset [, int maxlen]]]]) – возвращает всё содержимое файла в виде строки (последовательности байт), начиная с указанного смещения offset и до maxlen байтов.

В случае ошибки чтения возвращает FALSE.

Использование функции file_get_contents() удобно в случае необходимости получить содержимое файла целиком, поскольку для улучшения производительности функция использует алгоритм 'memory mapping' (если поддерживается операционной системой).

Пример будет чуть позже…

Чтение и запись из/в файлы

int file_put_contents (string filename, mixed data [, int flags [, resource context]]) – идентична последовательному вызову функций fopen(), fwrite() и fclose(). Возвращает количество записанных в файл байт.

Параметр flags может принимать значение FILE_APPEND для добавления информации в конец существующего файла.

В качестве параметра data можно передать одномерный массив. Это будет эквивалентно вызову

file_put_contents($filename, join('', $array));

Эта функция (равно как и file_get_contents) безопасна для обработки данных в двоичной форме.

Чтение и запись из/в файлы

Пример 1:

```
$a=file_get_contents(“1.txt”);
```

Пример 2:

```
file_put_contents(“2.txt”,$a);
```

Чтение и запись из/в файлы

Также для работы с файлами можно использовать классические (ещё со времён языка C) функции fopen(), fread(), fwrite().

resource fopen (string filename, string mode [, bool use_include_path [, resource zcontext]]) – закрепляет именованный ресурс (чаще всего – файл), указанный в аргументе filename, за потоком и возвращает дескриптор («файловый указатель»).

Если открыть ресурс не удалось, возвращает FALSE.

Поддержка контекста была добавлена в PHP 5.0.0.

Начиная с версии PHP 4.3.2, бинарный режим работы с файлами является режимом по умолчанию для всех платформ, которые различают бинарный и текстовый, режимы.

Параметр mode указывает тип доступа, который вы запрашиваете у потока. Он может быть одним из следующих…

Чтение и запись из/в файлы

mode	Описание
'r'	Открывает файл только для чтения; помещает указатель в начало файла.
'r+'	Открывает файл для чтения и записи; помещает указатель в начало файла.
'w'	Открывает файл только для записи; помещает указатель в начало файла и обрезает файл до нулевой длинны. Если файл не существует - пробует его создать.
'w+'	Открывает файл для чтения и записи; помещает указатель в начало файла и обрезает файл до нулевой длинны. Если файл не существует - пробует его создать.
'a'	Открывает файл только для записи; помещает указатель в конец файла. Если файл не существует - пытается его создать.
'a+'	Открывает файл для чтения и записи; помещает указатель в конец файла. Если файл не существует - пытается его создать.
'x'	Создаёт и открывает файл только для записи; помещает указатель в начало файла. Если файл уже существует, вызов **fopen()** закончится неудачей. Если файл не существует, попытается его создать.
'x+'	Создаёт и открывает файл для чтения и записи; помещает указатель в начало файла. Если файл уже существует, вызов **fopen()** закончится неудачей. Если файл не существует, попытается его создать.

Чтение и запись из/в файлы

Замечание о тексте: разные семейства операционных систем имеют разные соглашения относительно окончаний строк.

Системы семейства Unix (POSIX-совместимые системы) используют \n в качестве символа конца строки, системы семейства Windows используют \r\n в качестве символов окончания строки и системы семейства Macintosh используют \r в качестве символа конца строки.

Если вы используете неверный символ конца строки при редактировании файлов, вы можете обнаружить, что при открытии эти файлы «превратились» в одну длинную строку или, наоборот, после каждой строки появилась пустая строка.

Чтобы явно указать, работать с файлом как с бинарными данными или текстовыми, можно использовать флаги b и t соответственно, добавив их после буквы, указывающей режим доступа (mode), например: rb+, rt+, at.

Необязательный третий параметр use_include_path может быть установлен в 1 или TRUE, если вы также хотите провести поиск файла в include_path.

Чтение и запись из/в файлы

string fread (resource handle, int length) – читает length байт из файлового указателя handle. Чтение останавливается при достижении length байтов, EOF (конца файла) или (для сетевых потоков) завершения передачи пакета.

int fwrite (resource handle, string string [, int length]) – записывает содержимое string в файловый поток handle. Если передан аргумент length, запись остановится после того, как length байт будут записаны (если length больше фактического объёма данных, запись остановится после того, как будут записаны переданные данные).

bool fclose (resource handle) – закрывает файл, на который указывает handle.

Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Дескриптор handle должен указывать на файл, открытый ранее с помощью функции fopen() или fsockopen() (будет рассмотрена позже в теме о сетевых функциях PHP).

bool feof (resource handle) – возвращает TRUE, когда дескриптор указывает на достижение конца файла, или когда произошла ошибка чтения (включая таймаут сокета), иначе возвращает FALSE.

Чтение и запись из/в файлы

Пример использования fopen(), fread(), fwrite(), feof(), fclose():

```
$f1=fopen(“file.dat”,”rb”);
$f2=fopen(“file.dat”,”wb”);
while (!feof($f1))
{
$x=fread($f1,2048);
fwrite($f2,$x);
}
fclose($f1);
fclose($f2);
```

Чтение и запись из/в файлы

bool is_writable (string filename) – возвращает TRUE, если файл filename существует и доступен для записи.

Аргумент filename может быть именем каталога, что позволяет проверять каталоги на доступность для записи.

bool is_readable (string filename) – возвращает TRUE, если файл (или каталог) существует и доступен для чтения.

Чтение и запись из/в файлы

int fpassthru (resource handle) – читает из файла (потока) с указателем (дескриптором) handle информацию с текущей позиции до конца и записывает результат в буфер вывода.

Если происходит ошибка, fpassthru() возвращает FALSE. В ином случае, fpassthru() возвращает количество символов, прочитанных из handle и переданных на вывод.

Чтобы узнать или изменить текущую позицию в файле, следует использовать функции: ftell(), fseek(), rewind().

Чтение и запись из/в файлы

Во время использования fpassthru() на бинарном файле в Windows, следует убедиться в том, что файл открыт в бинарном режиме (при помощи добавления b к режиму открытия файла, использованному в fopen()).

Рекомендуется использовать флаг b при работе с бинарными файлам, даже если ваша система этого не требует, чтобы обеспечить более высокую портируемость скриптов.

Пример:

```
// открываем файл в бинарном режиме
$name = ".\public\dev\img\ok.png";
$fp = fopen($name, 'rb');
// отправляем нужные заголовки
header("Content-Type: image/png");
header("Content-Length: " . filesize($name));
// отправляем картинку
fpassthru($fp);
```

Чтение и запись из/в файлы

int ftell (resource handle) – возвращает текущую позицию (смещение относительно начала) для файла с дескриптором handle.

При возникновении ошибки, возвращает FALSE.

Пример:

```
// открываем файл и читаем немного данных
$fp = fopen("/etc/passwd", "r");
$data = fgets($fp, 12);

// где мы сейчас ?
echo ftell($fp); // 11
fclose($fp);
```

Чтение и запись из/в файлы

int fseek (resource handle, int offset [, int whence]) -- устанавливает позицию (смещение) в файле, на который ссылается дескриптор handle. Новое смещение, измеряемое в байтах от начала файла, получается путём прибавления параметра offset к позиции, указанной в параметре whence, значения которого определяются следующим образом:

SEEK_SET – устанавливает смещение в offset байт от начала файла (по умолчанию);

SEEK_CUR – устанавливает смещение в виде «текущее смещение плюс offset»;

SEEK_END – устанавливает смещение в виде «размер файла плюс offset» (чтобы перейти к смещению перед концом файла, нужно передать отрицательное значение в параметр offset)

В случае успешного выполнения возвращает 0, в противном случае возвращает -1. Обратите внимание, что переход к смещению за концом файла не считается ошибкой.

Чтение и запись из/в файлы

Пример:

```
$fp = fopen('somefile.txt');

// читаем немного данных
$data = fgets($fp, 4096);

// перемещаемся назад к началу файла
// то же самое, что и rewind($fp);
fseek($fp, 0);
```

Возвращает неопределённый результат для файлов, открытых в режиме a (добавление информации в конец файла).

Чтение и запись из/в файлы

bool rewind (resource handle) – устанавливает смещение на начало файла.

Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Замечание: если вы открыли файл в режиме a (добавление информации в конец файла), любые данные, которые вы записываете, будут дописаны в конец файла, независимо от текущего смещения.

Чтение и запись из/в файлы

resource tmpfile (void) – создаёт временный файл с уникальным именем, открывая его в режиме чтения и записи w+, и возвращает файловый указатель таким же образом, как это делает fopen().

Этот файл автоматически удаляется после закрытия (использования fclose()) или после завершения работы скрипта.

Пример:

```
$temp = tmpfile();
fwrite($temp, "записываем во временный файл");
fseek($temp, 0);
echo fread($temp, 1024);
fclose($temp); // происходит удаление файла
```

Копирование, перемещение, удаление

bool copy (string source, string dest) – создаёт копию файла, чьё имя передано в параметре source, в файл с именем dest. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки. Если файл dest существует, он будет перезаписан. **Работает ТОЛЬКО с файлами, т.е. каталог скопировать НЕЛЬЗЯ.**

bool rename (string oldname, string newname [, resource context]) – переименовывает (перемещает) файл с именем oldname в файл с именем newname. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки. Если файл newname существует, он будет перезаписан. **Работает с каталогами.**

bool unlink (string filename [, resource context]) – удаляет файл filename. Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки. **Работает ТОЛЬКО с файлами, т.е. каталог удалить НЕЛЬЗЯ.**

Копирование, перемещение, удаление

Примеры:

copy(“c:/1.txt”,”d:/2.txt”);

rename(“c:/11.txt”,”d:/22.txt”);

unlink(“/home/dir/file.ext”);

Информация о файле

array stat (string filename) – собирает информацию о файле filename.

Если filename является символической ссылкой, информация собирается о самом файле, а не ссылке.

В случае ошибки, stat() вернёт FALSE.

Возвращает массив информации о файле…

Информация о файле

Числовой индекс	Ассоциативный индекс (начиная с PHP 4.0.6)	Описание
0	dev	номер устройства
1	ino	номер inode
2	mode	режим защиты inode
3	nlink	количество ссылок
4	uid	userid владельца
5	gid	groupid владельца
6	rdev	тип устройства, если устройство inode (доступен только на системах, поддерживающих тип st_blksize -- другие системы (например Windows) вернут -1)
7	size	размер в байтах
8	atime	время последнего доступа (Unix timestamp)
9	mtime	время последней модификации (Unix timestamp)
10	ctime	время последнего изменения inode (Unix timestamp)
11	blksize	размер блока ввода-вывода файловой системы (доступен только на системах, поддерживающих тип st_blksize -- другие системы (например Windows) вернут -1)
12	blocks	количество используемых блоков

Информация о файле

array lstat (string filename) – собирает информацию о файле или символической ссылке с именем filename. Эта функция идентична функции stat(), за исключением того, что если filename является символической ссылкой, возвращается статус символической ссылки, а не того файла, на который она указывает.

array fstat (resource handle) – собирает информацию об открытом файле по файловому указателю handle. Эта функция идентична stat(), за исключением того, что она работает с открытым файловым указателем, а не именем файла.

Рассмотрим пример…

Информация о файле

Пример:

```
// открываем файл
$fp = fopen("/etc/passwd", "r");
// собираем информацию
$fstat = fstat($fp);
// закрываем файл
fclose($fp);
// отображаем только ассоциативную часть
print_r(array_slice($fstat, 13));
```

Результатом выполнения данного примера будет нечто подобное:

```
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 )
```

Работа с каталогами

mixed scandir (string directory [, integer sorting_order]) – возвращает array, содержащий имена файлов и каталогов, расположенных по пути, переданном в параметре directory.

Если directory не является таковым, функция возвращает FALSE.

Имена объектов внутри сканируемого каталога по умолчанию сортируются по алфавиту по возрастанию. Если указан необязательный параметр sorting_order равный 1, сортировка производится в алфавитном порядке по убыванию.

```
$d1 = scandir(“/home/”);
$d2 = scandir(“c:/www_pub/”, 1);
```

Работа с каталогами

Для «классической» работы с каталогами применяются функции opendir(), readdir(), rewinddir(), closedir().

resource opendir (string path) – возвращает дескриптор каталога для последующего использования с функциями closedir(), readdir() и rewinddir(). В случае возникновения ошибки возвращает FALSE.

string readdir (resource dir_handle) – возвращает имя следующего по порядку элемента каталога. Имена элементов возвращаются в порядке, зависящем от файловой системы.

void rewinddir (resource dir_handle) – «сбрасывает» поток каталога, переданный в параметре dir_handle таким образом, чтобы тот указывал на начало каталога.

void closedir (resource dir_handle) – закрывает поток, связанный с каталогом и переданный в качестве параметра dir_handle. Перед использованием данной функции, поток должен быть открыт с помощью функции opendir().

Работа с каталогами

Пример (вывести список всех файлов в каталоге):

```
// Правильный способ
if ($handle = opendir('/path/to/files'))
 {
  while (false !== ($file = readdir($handle))) echo $file.”<br />";
 }
closedir($handle);

 // Неправильный способ
 $handle = opendir('/path/to/files')
 while ($file = readdir($handle)) echo $file.”<br />";
 closedir($handle);
```

Работа с каталогами

array glob (string pattern [, int flags]) – ищет все пути, совпадающие с шаблоном pattern согласно правилам, используемым в функции glob() библиотеки libc, которые похожи на правила, используемые большинством распространённых файловых менеджеров.

Возвращает массив, который содержит совпадающие с шаблоном файлы/каталоги или FALSE в случае ошибки.

Допустимые флаги:

GLOB_MARK – добавляет слеш к каждому возвращаемому значению;

GLOB_NOSORT – возвращает значения без сортировки;

GLOB_NOCHECK – возвращает шаблон поиска, если с его помощью не был найден ни один объект;

GLOB_NOESCAPE – обратные слеши не экранируют метасимволы;

GLOB_BRACE – раскрывает {a,b,c} для совпадения с 'a', 'b' или 'c';

GLOB_ONLYDIR -- возвращает только каталоги;

Работа с каталогами

Пример:

```
foreach (glob("*.txt") as $filename)
 {
  echo $filename.“ размером в ".filesize($filename).“байт <br />";
 }
```

Результат будет примерно таким:

file1.txt размером в 44686 байт
file2.txt размером в 267625 байт
file3.txt размером в 137820 байт

Работа с каталогами

string getcwd (void) – возвращает имя текущего рабочего каталога.

bool chdir (string directory) – изменяет текущий рабочий каталог на указанный в качестве параметра directory.

Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Рабочий каталог (working directory) (также «текущий каталог», «текущий путь») – каталог, который используется для нахождения файлов, указанных только по имени либо по относительному пути. При создании нового процесса, он наследует рабочий каталог родительского процесса.

В операционных системах, использующих буквы дисков (OS/2, Windows, DOS), текущих путей может быть несколько (по количеству дисков). При этом активным может быть только один, остальные являются неактивными. Активный путь выбирается исходя из активного (выбранного) диска.

Работа с каталогами

bool mkdir (string pathname [, int mode [, bool recursive [, resource context]]]) – создаёт каталог, указанный в параметре pathname.

Аргумент mode определяет права доступа (рассмотрим чуть-чуть позже), и его необходимо задавать в виде восьмеричного числа (первой цифрой должен быть ноль).

Аргумент mode игнорируется в Windows и стал необязательным начиная с версии PHP 4.2.0. По умолчанию mode равен 0777, что является самыми широкими правами доступа.

Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Пример:

```
mkdir("/path/to/my/dir", 0700);
```

Работа с каталогами

bool rmdir (string dirname [, resource context]) – удаляет каталог с именем dirname.

Каталог должен быть пустым и должны иметься необходимые для его удаления права.

Возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Пример:

```
rmdir(“c:/abcde”);
```

Работа с каталогами

string dirname (string path) – возвращает имя каталога, содержащегося в параметре path.

На платформах Windows в качестве разделителей имён каталогов используются оба слеша (прямой / и обратный \). В других операционных системах разделителем служит прямой слеш (/).

Пример:

```
$path = "/etc/passwd";
$dir_only = dirname($path); //  /etc
```

Замечание: начиная с PHP версии 4.0.3, функция dirname() стала совместима со стандартом POSIX. Это означает, что, если в path отсутствуют слеши, функция вернет точку (.), обозначающую текущий каталог. Иначе результатом выполнения функции будет являться значение параметра path с отброшенным завершающим /компонентом.

Работа с каталогами

string basename (string path [, string suffix]) – возвращает имя файла, чей путь был передан в качестве параметра. Если имя файла оканчивается на suffix, он также будет отброшен.

Пример:

```
$path = "/home/httpd/html/index.php";
$filename = basename($path);             // index.php
$filename = basename($path, ".php");  // index
```

string realpath (string path) – раскрывает все символические ссылки, переходы типа '/./', '/../' и лишние символы '/' в пути path, возвращая канонизированный абсолютный путь к файлу.

Возвращает FALSE при неудаче, например если файл не существует.

Пример:

```
$real_path = realpath("c:/dir1/dir2/../../index.php");
// c:/index.php
```

Работа с каталогами

array pathinfo (string path [, int options]) – возвращает ассоциативный массив, который содержит информацию о пути path.

Возвращаемый массив состоит из следующих элементов: dirname, basename и extension.

При помощи необязательного параметра options можно указать, какие элементы возвратить.

Его значения:

PATHINFO_DIRNAME – имя каталога;

PATHINFO_BASENAME – имя файла;

PATHINFO_EXTENSION – расширение файла.

Пример:

```
$path_parts = pathinfo('/www/htdocs/index.html');
echo $path_parts['dirname'];     // /www/htdocs
echo $path_parts['basename']; // index.html
echo $path_parts['extension'];  // html
```

Права доступа

Теперь мы поговорим о правах доступа к файлам и каталогам в POSIX-системах.

В POSIX-системах пользователи по отношению к объектам файловой системы делятся на следующие категории:

1) Владелец (owner). К этой категории относится пользователь, создавший файл или каталог. По умолчанию имеет самые широкие права.

2) Группа (group). Группа, к которой относится владелец.

3) Остальные, мир (others, world). Обычные пользователи, не относящиеся к первой и второй группе.

Права пользователя также подразделяются на три типа:

r (read) – чтение. Соответствует цифра 4.

w (write) – запись. Соответствует цифра 2.

x (execute) – выполнение (разрешает запустить файл, если он является программой). Соответствует цифра 1.

Существует три категории пользователей и столько же режимов доступа. Следовательно, в символьном варианте назначение прав выглядит следующим образом: rwx rwx rwx т.е. каждая тройка определяет какие права будет иметь та или иная категория пользователей.

Права доступа

Пример:

rwx r-x r-

Данный пример показывает, что владелец имеет полные права, группа, к которой относится владелец имеет права на чтение и исполнение и все остальные пользователи – только на чтение.

Как образуется запись прав в цифровом виде?

Возьмём тот же самый пример:

rwx r-x r-

в цифровом виде это будет

0754

Ноль означает, что это – запись в восьмеричной системе счисления (обычно его опускают в инструкциях, но его нужно передавать в функцию PHP, которая устанавливает права).

Цифры 7, 5 и 4 являются «суммами прав» для «владельца», «группы» и «остальных».

Права доступа

Рассмотрим примеры прав выставляемые на файлы и каталоги.

0777 (rwx rwx rwx) – даёт всем группам пользователей все права. Выставлять такие права на что бы то ни было не рекомендуется по соображениям безопасности.

Если файл не является исполняемым, права

0666 (rw- rw- rw-)

являются для него полным аналогом 0777 (выполнить такой файл всё равно нельзя, а потому право на исполнение можно не выставлять).

Права

0755 (rwx r-x r-x)

позволяют всем пользователям читать и выполнять файл, а запись в файл доступна только владельцу. Это основная запись для исполняемых файлов.

Если необходимо осуществить сохранность конфиденциальных данных в файле и запретить доступ к нему посторонним пользователям, то устанавливаются права

0600 (rw- --- ---)

Права доступа

Про каталоги скажем несколько слов подробнее:

r – право на чтение **содержимого** каталога. Оно никак не влияет на способность пользователя просматривать какой-либо файл в каталоге, т.к. для этого файла есть своя установка права на чтение. Если это право убрать, то пользователь всего лишь не сможет просмотреть, что находится в каталоге. Однако если он знает имя файла в этом каталоге, то он может обратиться к файлу по имени.

w – право на запись **в каталог**. Если его убрать, то пользователь не сможет создавать новые файлы в каталоге и удалять существующие. К редактированию содержимого существующего файла отношения не имеет, для него тоже есть соответствующее право на запись.

x – для файлов это право на запуск, а для каталогов – право на «прохождение через каталог». Если каталог лишить этого права, получится два эффекта:

- не получится что-то сделать (создать, редактировать, удалить и т.д.) с тем, что находится в каталоге, независимо от того, какие права выставлены на объект с которым пользователь попытается что-то сделать. Не получится перейти в такой каталог. Если для каталога выставлено право на чтение, можно просмотреть его содержимое, однако это не относится к подкаталогам.
- даже если у подкаталогов с правами всё в порядке, то с ними и их содержимым также ничего не получится сделать.

Права доступа

Теперь о функциях в PHP, управляющих правами:

bool chmod (string filename, int mode) – изменяет режим доступа файла или каталога, переданного в параметре filename на режим, переданный в параметре mode.

Обратите внимание, что значение параметра mode следует передавать в восьмеричной системе счисления (например, 0777).

Пример:

```
chmod("/somedir/somefile", 0755);
```

Эта функция возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Замечание: данная функция выполняется PHP, т.е. от имени пользователя, «запустившего PHP». Если этот пользователь не имеет права менять права для того или иного объекта файловой системы (как правило – не является их владельцем) – права изменены не будут.

Права доступа

int umask ([int mask]) – устанавливает umask (см. ниже) в значение mask & 0777 и возвращает старую umask.

Что такое «umask». «Umask» (user file creation mode mask, маска режима создания пользовательских файлов) – функция среды POSIX, изменяющая права доступа, которые присваиваются новым файлам и каталогам по умолчанию.

Права доступа файлов, созданных при конкретном значении umask, вычисляются при помощи следующих побитовых операций: побитовое И между унарным дополнением аргумента (используя побитовое НЕ) и режимом полного доступа.

Режим полного доступа для файлов – 666, для каталогов – 777. Заметьте, что функция umask() всё равно использует 0777 для установки «значения umask».

Пример:

```
umask(174);
// каждый новый файл будет иметь права доступа 603:
// т.к. 777(8) И НЕ(174(8)) = 603(8)
```

Права доступа

bool chown (string filename, mixed user) – изменяет владельца объекта файловой системы с именем filename на владельца, чьё имя передано в параметре user (в виде числа или строки).

Функция возвращает TRUE в случае успешного завершения операции или FALSE в случае возникновения ошибки.

bool chgrp (string filename, mixed group) – изменяет группу владельцев объекта файловой системы с именем filename на группу, указанную в параметре group (в виде имени или числа).

Функция возвращает TRUE в случае успешного завершения операции или FALSE в случае возникновения ошибки.

Только «суперпользователь» (как правило – root) может изменять владельца и группу владельцев объекта файловой системы, а поскольку PHP никакой здравомыслящий администратор не станет запускать от своего имени, использование этих двух функций почти никогда не даёт нужного эффекта.

Потоки данных и контекст ресурса

Потоками данных считаются любые «ресурсы», общий стиль работы с которыми схож со стилем работы с файлами: в поток можно писать, из потока можно читать, некоторые потоки (как правило – файлы) допускают позиционирование (установку смещения).

К потокам в PHP относятся:

- файлы;
- данные, передаваемые через сокеты;
- данные, передаваемые по протоколам FTP/HTTP;

и множество других «потоковых сущностей», список и особенности поведения которых слишком велик для подробного рассмотрения в нашем курсе, а поэтому – см. руководство по PHP.

Подавляющее большинство функций по работа с этими сущностями могут получать необязательный параметр context, который определяет некоторые «свойства окружающей среды» при работе с соответствующим потоком.

Рассмотрим на примере – передадим в виде контекста данные, которые мог бы передать браузер…

Потоки данных и контекст ресурса

Пример:

```
$opts = array(
  'http'=>array(
    'method'=>"GET",
    'header'=>"Accept-language: en\r\n" .
              "Cookie: foo=bar\r\n"
  )
);
$context = stream_context_create($opts);

$fp = fopen('http://www.example.com', 'r', false, $context);
fpassthru($fp);
fclose($fp);
```

Здесь мы видим новую функцию stream_context_create().

Потоки данных и контекст ресурса

resource stream_context_create ([array options [, array params]]) – создаёт контекст потока/ресурса на основе списка option и некоторых дополнительных параметров params (они специфичны и редко используются, потому здесь мы их рассматривать не будем).

Параметр options должен быть ассоциативным массивом в формате

$arr['wrapper']['option'] = $value

где wrapper – вид «обёртки» контекста (т.е., фактически, указание на то, «контекст чего создаётся», например, «http»; полный список см. в руководстве по PHP).

Некоторая запутанность описания этой функции, как правило, исчезает после рассмотрения примера…

Потоки данных и контекст ресурса

Пример:

```
$postdata = http_build_query(array(‘name' => ‘Vasya', ‘city' => ‘Minsk‘ ));
$opts = array('http' =>
    array(
        'method'  => 'POST',
        'header'  => 'Content-type: application/x-www-form-urlencoded',
        'content' => $postdata
    )
);
$context  = stream_context_create($opts);
$result = file_get_contents('http://example.com/submit.php', false, $context);
```

Здесь появляется функция http_build_query().

Потоки данных и контекст ресурса

string http_build_query (array formdata [, string numeric_prefix [, string arg_separator]]) – подготавливает данные для отправки в виде «данных из формы»

Массив formdata может быть как одномерным, так и сколь угодно сложным.

Если в массиве form использованы числовые индексы, нужно указать параметр numeric_prefix – строку, которая будет добавлена перед числами (чтобы превратить числовые индексы массива в корректные имена переменных).

Если указан параметр arg_separator, он используется для разделения элементов массива в запросе.

Рассмотрим примеры…

Потоки данных и контекст ресурса

Пример 1 (простой случай):

```
$data = array(‘A'=>‘ABC',
              ‘city'=>‘Minsk',
              ‘lang'=>‘any',
              'php'=>'hypertext processor');
echo http_build_query($data);
// A=ABC&city=Minsk&lang=any&php=hypertext+processor

echo http_build_query($data, '', '&amp;');
// A=ABC&amp;city=Minsk&amp;lang=any&amp;php=hypertext+processor
```

Потоки данных и контекст ресурса

Пример 2 (числовые индексы):

```
$data = array(‘A', ‘B', ‘C');
echo http_build_query($data);
// 0=A&1=B&2=C

echo http_build_query($data, 'var_');
// var_0=A&var_1=B&var_2=C
```

Потоки данных и контекст ресурса

Пример 3 (сложный массив):

```
$data = array('user'=>array('name'=>'Bob Smith',
                            'age'=>47,
                            'sex'=>'M',
                            'dob'=>'5/12/1956'),
              'pastimes'=>array('golf', 'opera', 'poker', ‘rock'),
              'children'=>array('bobby'=>array('age'=>12,
                                               'sex'=>'M'),
                                'sally'=>array('age'=>8,
                                               'sex'=>'F')),
              'CEO');

echo http_build_query($data, 'flags_');
```

Потоки данных и контекст ресурса

Результат (строки перенесены для лучшей читаемости):

user[name]=Bob+Smith&user[age]=47&user[sex]=M&user[dob]=5%2F12%2F1956&

pastimes[0]=golf&pastimes[1]=opera&pastimes[2]=poker&pastimes[3]=rap&

children[bobby][age]=12&children[bobby][sex]=M&children[sally][age]=8&

children[sally][sex]=F&flags_0=CEO

Потоки данных и контекст ресурса

Более подробную информацию об «обёртках», отличных от http (равно как и самой http) см. в руководстве по PHP

http://docs.php.net/manual/ru/context.php

Несмотря на то, что в URL'е присутствует ru, эта часть документации пока не русифицирована.

Получение данных с некоторого URL

Как вы уже могли заметить в некоторых ранее рассмотренных примерах, в качестве «имени файла» во многие функции, работающие с файлами, можно передавать URL.

Эта возможность существует, если директива allow_url_fopen в php.ini имеет значение On.

В таком случае, если PHP пытается автоматически определить протокол (или «обёртку») на основе имени файла и выполняет операции, используя этот протокол.

Рассмотрим, какие протоколы поддерживает PHP.

Получение данных с некоторого URL

Первым и самым простым «протоколом» является файловая система. Для неё допустимы такие имена файлов:

- /path/to/file.ext
- relative/path/to/file.ext
- fileInCwd.ext
- C:/path/to/winfile.ext
- C:\path\to\winfile.ext
- \\smbserver\share\path\to\winfile.ext
- file:///path/to/file.ext

Сокеты. Для работы с ними существует отдельный набор функций, которые мы рассмотрим в нашем курсе позже.

Получение данных с некоторого URL

Протоколы HTTP и HTTPS. Для них допустимы такие «имена файлов»:

- http://example.com
- http://example.com/file.php?var1=val1&var2=val2
- http://user:password@example.com
- https://example.com
- https://example.com/file.php?var1=val1&var2=val2
- https://user:password@example.com

Получение данных с некоторого URL

Протоколы FTP и FTPS. Для них допустимы такие «имена файлов»:

- ftp://example.com/pub/file.txt
- ftp://user:password@example.com/pub/file.txt
- ftps://example.com/pub/file.txt
- ftps://user:password@example.com/pub/file.txt

Получение данных с некоторого URL

Стандартные входные и выходные потоки PHP. Для них допустимы такие «имена файлов»:

- php://stdin
- php://stdout
- php://stderr
- php://output
- php://input
- php://filter (начиная с PHP 5.0.0)
- php://memory (начиная с PHP 5.1.0)
- php://temp (начиная с PHP 5.1.0)

Получение данных с некоторого URL

Сжатые потоки данных. Для них допустимы такие «имена файлов»:

- zlib:
- compress.zlib://
- compress.bzip2://

Получение данных с некоторого URL

«Данные» (определяются через myme-тип, поддерживаются с PHP 5.2.0).

Пример 1:

```
// выводит «I love PHP»
echo file_get_contents('data://text/plain;base64,SSBsb3ZlIFBIUAo=');
```

Пример 2:

```
$fp   = fopen('data://text/plain;base64,', 'r');
$meta = stream_get_meta_data($fp);
// выводит «text/plain»
echo $meta['mediatype'];
```

Получение данных с некоторого URL

Протокол SSH (secure shell). Для него допустимы такие «имена файлов»:

- ssh2.shell://user:pass@example.com:22/xterm
- ssh2.exec://user:pass@example.com:22/usr/local/bin/somecmd
- ssh2.tunnel://user:pass@example.com:22/192.168.0.1:14
- ssh2.sftp://user:pass@example.com:22/path/to/filename

Для использования этой возможности нужно установить специальное расширение.

Получение данных с некоторого URL

Аудиопотоки. MP3 пока не поддерживается. Для них допустимы такие «имена файлов»:

- ogg://soundfile.ogg
- ogg:///path/to/soundfile.ogg
- ogg://http://www.example.com/path/to/soundstream.ogg

Для использования этой возможности нужно установить специальное расширение.

Получение данных с некоторого URL

Межпроцессное взаимодействие. Для него допустимо такое «имя файла»:

- expect://command

Для использования этой возможности нужно установить специальное расширение.