Часть 2. Практическая работа №6

Работа с файлами

В этой Практической работе вы:

* узнаете о приятных особенностях работы с файлами в PHP
* узнаете о двух режимах работы с файлом
* узнаете обо всех необходимых функциях для работы с файлами
* реализуете журнал посещений сайта
* научитесь писать скрипты, загружающие файлы на сервер
* узнаете о функциях для работы с каталогами
* научитесь получать список файлов и подпапок в каталоге

1. Особенности работы с файлами в PHP

PHP снова вас порадует! При работе с файлами можно забыть о различии прямых и обратных слешей для Unix систем и Windows. Вы можете писать их как угодно, интерпретатор самостоятельно преобразует путь к нужному виду. Мы рекомендуем всегда использовать прямые слеши, вот такие:

images/logo.png

И еще одна хорошая новость: вы можете работать с файлами на удаленных серверах Web в точности так же, как и со своими собственными. Если вы предваряете имя файла строкой http:// или ftp://, то PHP понимает, что нужно на самом деле установить сетевое соединение и работать именно с ним, а не с файлом. При этом в программе такой файл ничем не отличается от обычного.

1. Два режима работы с файлом

Можно выделить два режима работы с файлом:

1. текстовый
2. бинарный

Первый режим предполагает, что файл состоит из печатных символов, то есть букв, цифр и тех знаков, которые есть на клавиатуре. С текстовыми файлами обычно работают построчно, то есть записывают и читают по одной строке.

Второй режим служит для работы с любыми файлами. В этом случае читают и записывают не строки текста, а байты, то есть двоичную информацию.

В PHP в обоих случаях при чтении и записи используется строковый тип данных, так как типа «массив байт» здесь просто нет. Но отличие все же есть, и заключается оно в трансляции символа переноса строки.

В системах семейства Unix для перевода строки используется символ «\п», в то время как в Windows - последовательность из двух символов «\г\п». Трансляция символа перевода строки нужна для того, чтобы программист не зависел от операционной системы. Таким образом, можно считать, что для новой строки следует всегда использовать «\п», интерпретатор сам откорректирует чтение и запись.

1. Функции для работы с файлами

int fopen(string $filename, string $mode, bool $use\_include\_path=false)

Открывает файл с именем $filename в режиме $mode и возвращает дескриптор открытого файла. Если операция "провалилась", то, как это принято, fopen() возвращает false. Необязательный параметр $use\_include\_path говорит PHP о том, что, если задано относительное имя файла, его следует искать также и в списке путей, используемом инструкциями include и require. Обычно этот параметр не используют.

Параметр $mode может принимать следующие значения:

|  |  |
| --- | --- |
| r | Файл открывается только для чтения. Если файла не существует, вызов регистрирует ошибку. После удачного открытия указатель файла устанавливается на его первый байт, т. е. на начало. |
| r+ | Файл открывается одновременно на чтение и запись. Указатель текущей позиции устанавливается на его первый байт. Как и для режима г, если файла не существует, возвращается false. |
| w | Создает новый пустой файл. Если на момент вызова уже был файл с таким именем, то он предварительно уничтожается. |
| w+ | Аналогичен r+, но если файла изначально не существовало, создает его. После этого с файлом можно работать как в режиме чтения, так и записи. Если файл существовал до момента вызова, его содержимое удаляется. |
| a | Открывает существующий файл в режиме записи, и при этом сдвигает указатель текущей позиции за последний байт файла. Этот режим полезен, если требуется что-то дописать в конец уже имеющегося файла. Вызов неуспешен в случае отсутствия файла. |
| a+ | Открывает файл в режиме чтения и записи, указатель файла устанавливается на конец файла, при этом содержимое файла не уничтожается. Отличается от a тем, что если файла изначально не существовало, то он создается. Этот режим полезен, если вам нужно что-то дописать в файл (например, в журнал), но вы не знаете, создан ли уже такой файл. |

В конце любой из строк r, w, a, r+, w+ и a+ может находиться еще один необязательный символ — b или t. Если указан b (или не указан вообще никакой), то файл открывается в режиме бинарного чтения/записи. Если же это t, то для файла устанавливается режим трансляции символа перевода строки, т. е. он воспринимается как текстовый.

int tmpfile**O**

Создает новый файл с уникальным именем и открывает его на чтение и запись. В дальнейшем вся работа должна вестись с возвращенным файловым дескриптором, потому что имя файла недоступно. Функцию следует использовать, если нужен временный файл, который при завершении программы хотелось бы удалить. Пространство, занимаемое временным файлом, автоматически освобождается при его закрытии и при завершении работы программы.

int fclose(int $fp)

Закрывает файл. Возвращает истину при успехе, false при «провале». PHP автоматически закрывает открытые файлы при завершении выполнения сценария. Однако хорошим тоном считается самостоятельно закрывать дескриптор сразу после завершения работы с файлом.

string fread(int $f, int $numbytes)

Читает из файла $f $numbytes символов и возвращает строку этих символов. После чтения указатель файла продвигается к следующим после прочитанного блока позициям (это происходит и для всех остальных функций чтения и записи). Если $numbytes больше, чем можно прочитать из файла (например, раньше достигается конец файла), возвращается то, что удалось считать.

int fwrite(int $f, string $st)

Записывает в файл $f все содержимое строки $st. Эта функция составляет пару для fread(), действуя «в обратном направлении». При работе с текстовыми файлами (то есть когда указан символ t в режиме открытия файла) все «\п» автоматически преобразуются в тот разделитель строк, который принят в вашей операционной системе.

string fgets(int $f, int $length)

Читает из файла одну строку, заканчивающуюся символом новой строки «\п». Этот символ также считывается и включается в результат. Если строка в файле занимает больше $length-1 байтов, то возвращаются только ее $length-1 символов. Функция полезна, если вы открыли файл и хотите "пройтись" по всем его строкам. Однако даже в этом случае лучше (и быстрее) будет воспользоваться функцией File(), которая рассматривается ниже.

int fputs(int $f, string $st)

Эта функция — синоним fwrite(). Отличие в том, что пару fread/fwrite принято использовать при работе с бинарными файлами, а пару fputs/fgets - с текстовыми (когда запись и чтение осуществляются построчно).

int feof(int $f)

Возвращает true, если достигнут конец файла.

int fseek(int $f, in $offset, int $whence = SEEK\_SET)

Устанавливает указатель файла на байт со смещением $offset (от начала файла, от его конца или от текущей позиции, в зависимости от параметра $whence).

Параметр $whence, как уже упоминалось, задает, с какого места отсчитывается смещение $offset. В PHP для этого существуют три константы, равные, соответственно, 0, 1 и 2:

* SEEK\_SET — отсчитывает позицию относительно начала файла;
* SEEK\_CUR — отсчитывает позицию относительно текущей позиции;
* SEEK\_END — отсчитывает позицию относительно конца файла.

В случае использования последних двух констант параметр $offset вполне может быть отрицательным (а при применении SEEK\_END он будет отрицательным наверняка).

Как это ни странно, но в случае успешного завершения эта функция возвращает 0, а в случае неудачи -1. Должно быть, так сделано по аналогии с Си-эквивалентом функции.

int fteU(int $f)

Возвращает положение указателя файла. Вот так можно определить длину файла:

$f = fopen('x.txt', 'r'); fseek($f, 0, SEEK\_END);

$size = ftell($f);

echo "Размер файла: $size байт";

fclose($f);

int filesize(string $filename)

Более простой способ определения размера файла.

bool file\_exists(string $filename)

Возвращает true, если файл с именем $filename существует.

bool is\_file(string $filename)

Возвращает true, если $filename — обычный файл.bool is\_dir(string $filename)

Возвращает true, если $filename — каталог.

string basename(string $path)

Выделяет основное имя файла из пути $path.

string dirname(string $path)

Возвращает имя каталога, выделенное из пути $path.

bool copy(string $src, string $dst)

Копирует файл с именем $src в файл с именем $dst. При этом, если файл $dst на момент вызова существовал, осуществляется его перезапись. Функция возвращает true, если копирование прошло успешно, а в случае провала — false.

bool rename(string $oldname, string $newname)

Переименовывает (или перемещает, что одно и то же) файл с именем $oldname в файл с именем $newname. Если файл $newname уже существует, регистрируется ошибка, и функция возвращает false.

bool unlink(string $filename)

Удаляет файл с именем $filename. В случае неудачи возвращает false, иначе — true.

array file(string $filename)

Считывает файл с именем $filename целиком и возвращает массив, каждый элемент которого соответствует строке в прочитанном файле. Функция работает очень быстро — гораздо быстрее, чем если бы мы использовали fopen() и читали файл по одной строке. Неудобство этой функции состоит в том, что символы конца строки не вырезаются из строк файла, а также не транслируются, как это делается для текстовых файлов. Так что, каждый элемент массива иногда имеет смысл преобразовать с помощью функции rtrim().

Это не полный перечень функций для работы с файлами, однако, на первое время их более чем достаточно. Если вам потребовалось что-то сделать с файлом и вы не нашли в приведенном выше списке нужной функции, обратитесь к справочнику! Наверняка такая функция есть в PHP.

1. Журнал посещений сайта

Приступим к работе с файлами. Первый пример - журнал посещений сайта.

Когда посетитель попадает на главную страницу, будем фиксировать время захода, IP адрес и страницу, откуда он к нам пришел. Просматривать эту информацию мы будем с помощью страницы журнала посещений.

Вот так должна выглядеть главная страница:



<- С Ф http://1 о с а I h о st/te st/i n d ex. p h p

Это главная страница сайта

**Мы следим за ее посещаемостью!**

На нас ссылаются два сайта: щз= два.

А вот так страница журнала посещений: 

<- С Ф http://1 о с а I h о st/te st/vi s its. р h р

Журнал посещений

На главную

|  |  |  |
| --- | --- | --- |
| Время | IP-адрес | Откуда |
| 2010-03-13 00:07:42 | 127.0.0.1 |  |
| 2010-03-13 00:07:44 | 127.0.0.1 | httpiWocalbost/test^site 1 .htm |
| 2010-03-13 00:07:48 | 127.0.0.1 | Ьйр/'Ъсайю st''test''srte2 .htm |
| 2010-03-13 00:07:55 | 127.0.0.1 | ЬйрУЛосаШо st'''test'4isits.php |
| 2010-03-13 00:07:58 | 127.0.0.1 | ЬйрУЛосаПю st/test/site 1 .htm |
| 2010-03-13 00:08:00 | 127.0.0.1 |  |
| 2010-03-13 00:08:03 | 127.0.0.1 | http/fecalbo st/test/site 1 .htm |

Помимо этого добавим две статичные странички, которые просто ссылаются на главную страницу нашего сайта. Они нужны для того, чтобы продемонстрировать возможность определять, откуда пользователь попал на сайт.



Информацию о посещениях мы будем записывать в файл visits.txt: 2010-03-13 00:07:42

1. 0.1

2010-03-13 00:07:44

1. 0.1

<http://localhost/test/site1.htm>

2010-03-13 00:07:48

1. 0.1

<http://localhost/test/site2.htm>

2010-03-13 00:07:55

1. 0.1

<http://localhost/test/visits.php>

2010-03-13 00:07:58

1. 0.1

<http://localhost/test/site1.htm>

2010-03-13 00:08:00

1. 0.1

2010-03-13 00:08:03

1. 0.1

<http://localhost/test/site1.htm>

Файл состоит из троек:

* дата и время
* IP адрес
* откуда пришел

Поле «откуда пришел» может быть пустым, если это прямой заход на сайт (то есть когда адрес набирается вручную).

Теперь рассмотрим исходные тексты скриптов.

Файл index.php: <?php

// Запись в файл информации о посещении страницы.

$f = fopen('visits.txt', 'a+'); fwrite($f, date('Y-m-d H:i:s') . "\n");

fwrite($f, $\_SERVER['REMOTE\_ADDR'] . "\n");

fwrite($f, $\_SERVER['HTTP\_REFERER'] . "\n"); fclose($f);

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<**html**>

<**head**>

<meta http-equiv="content-type" content="text/html; charset=windows- 1251">

<title>Наш сайт</^^е>

</**head**>

<**body**>

<h1>Это главная страница сайта</^>

Мы <a href="visits.php">следим</a> за ее посещаемостью!

<br/><br/>

На нас ссылаются два сайта: <а href="site1.htm">раз</a>, <а href="site2.htm">два</a>.

</**body**>

</ **html** >

Мы открываем файл режимом «а+», что означает следующее:

* файл будет открыт для работы в текстовом режиме
* данные будут записываться в конец файла
* если файл не существует, он будет создан

См. таблицу режимов открытия файла выше.

При открытии файла мы получаем его целочисленный дескриптор, который будем использовать при дальнейшей работе.

Далее мы записываем в файл три строки. Обратите внимание, что для перехода на новую строку используется последовательность «\п». Ее следует писать в двойных кавычках, тогда она заменится на спецсимвол переноса строки. Если же ‘\n’ написать в апострофах, то это будет просто символ «\», за которым следует символ «п».

Наконец, когда работа с файлом завершена, его следует закрыть. Для этого служит функция fclose().

Теперь рассмотрим код журнала посещений:

<?php

// Чтение из файла всей информации о посещениях.

$lines = file('visits.txt');

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

* **html** >
* **head**>

<meta http-equiv="content-type" content="text/html; charset=windows- 1251">

<^^е>Наш сайт</^^е>

</ head>

<**body**>

<^>Журнал посещений</^>

<a href="index.php">На главную</а>

<br/>

<br/>

<table border=”1”>

<tr>

<td>Время</td> <td>IP-адрес</td> <td>Откуда</td> </tr>

<?php

$n = count($lines);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ($i = | 0; $i < | $n; $i += | 3) |  |  |
| echo | '<tr>'; |  |  |  |  |
| echo | '<td>' . | $lines[$i | + | 0] . | '</td> |
| echo | '<td>' . | $lines[$i | + | 1] . | '</td> |
| echo | '<td>' . | $lines[$i | + | 2] . | '</td> |
| echo | '</tr>'; |  |  |  |  |

**}**

?>

</**table**> </**body**> </**html**>

Мы легко читаем все строки файла в массив с помощью функции file(). При использовании этой функции не нужно открывать и закрывать файл - она все делает сама.

Исходный код статичных страничек site1.htm и site2.htm ничем не примечателен. Они просто содержат ссылку на главную страницу сайта.

site1.htm: <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

* **html** >
* **head**>

<meta http-equiv="content-type" content="text/html; charset=windows- 1251">

<title>Сайт #1</title>

</ head>

<**body**>

<h1>Это чей-то сайт #1</h1>

Он <a href="index.php">ссылается</а> на наш сайт.

</**body**>

</ **html** > site2.htm: <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<**html**>

<**head**>

<meta http-equiv="content-type" content="text/html; charset=windows-

1251">

^^^>Сайт #2</title>

</ head>

<**body**>

<h1>Это чей-то сайт #2</h1>

Он тоже <a href="index.php">ссылается</a> на наш сайт.

</**body**>

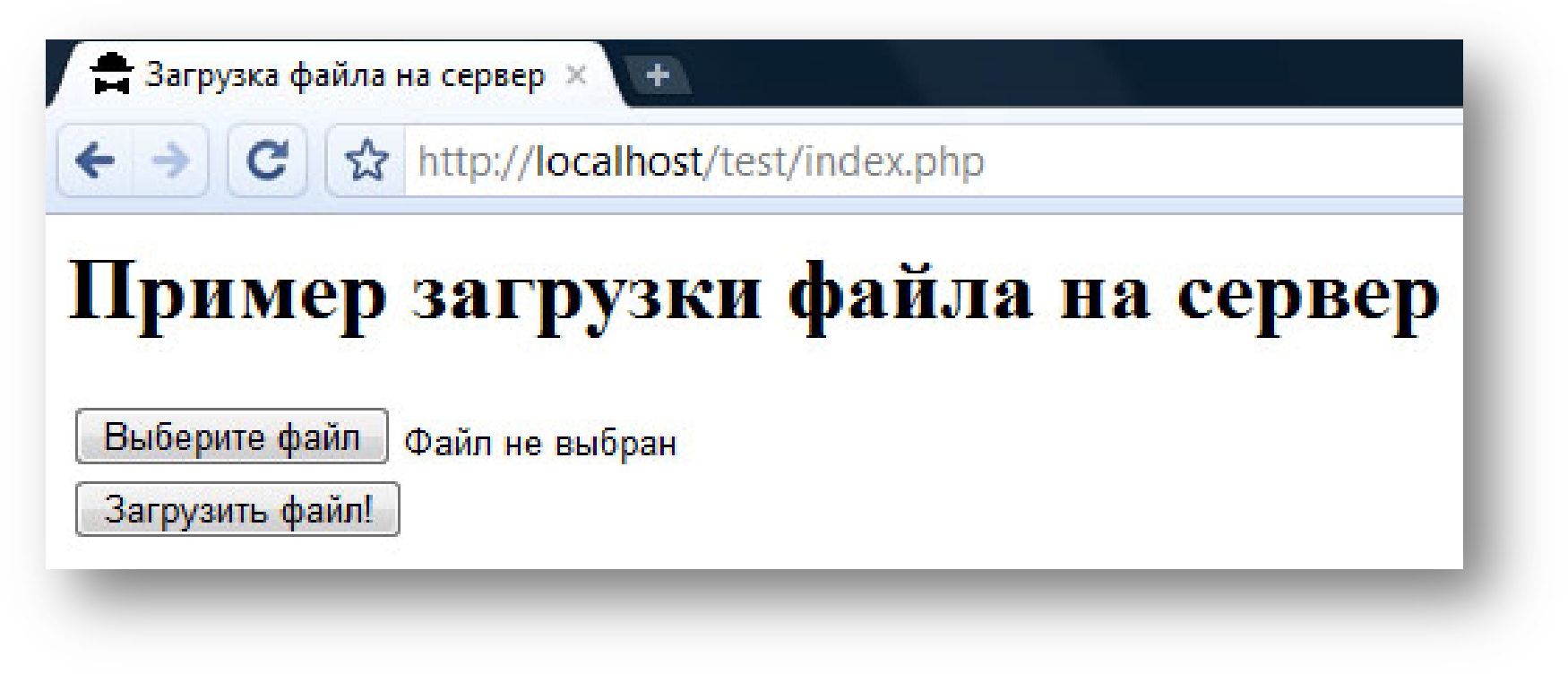
</**html**>

1. Загрузка файлов на сервер

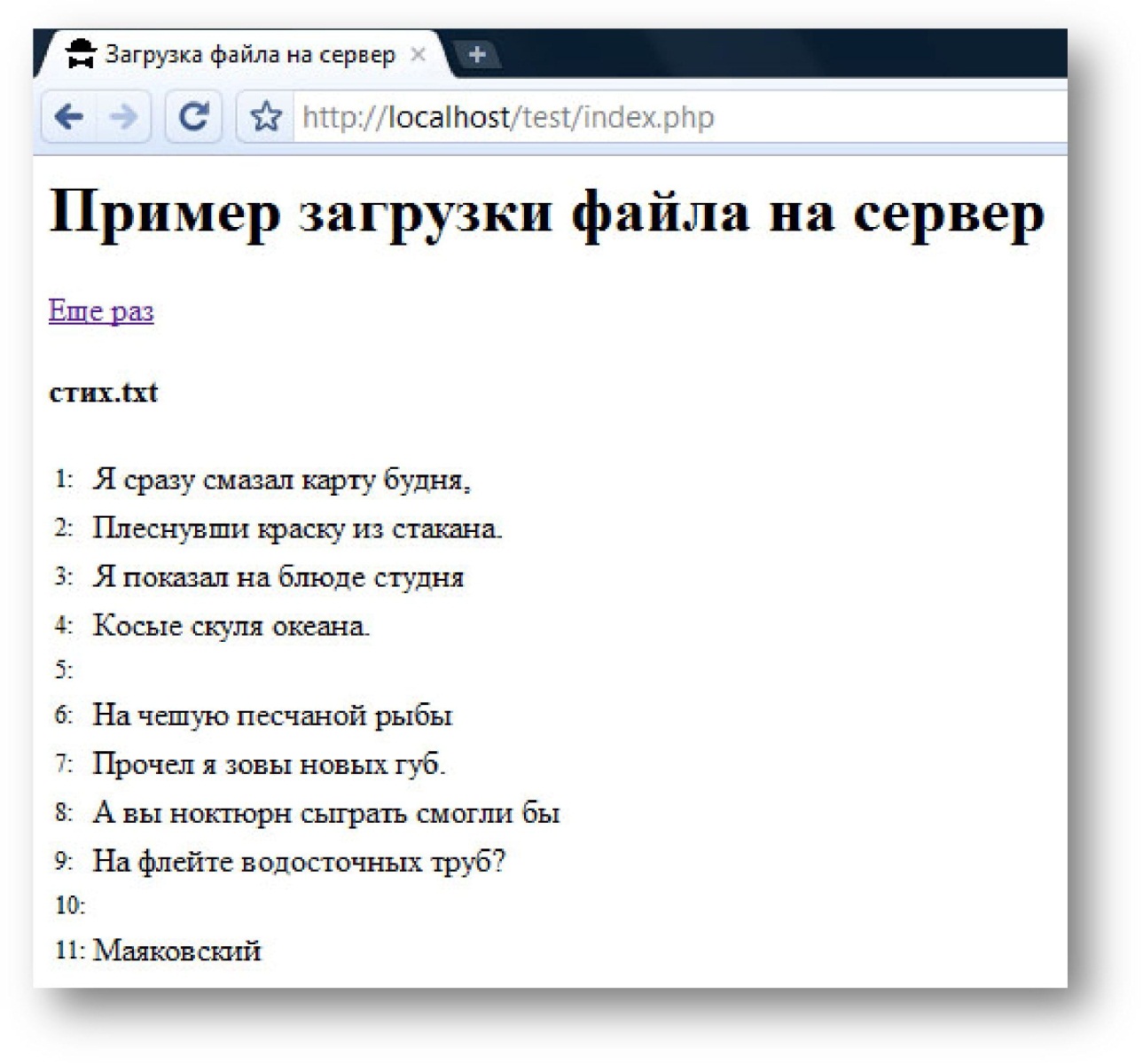
Существует определенный механизм загрузки файлов на сервер, отработаем его на следующем примере.

Пускай пользователь может выбрать текстовый файл и отправить его на сервер. Далее на сайте отобразится текст этого файла, причем строки будут пронумерованы. Для чего это можно было бы использовать? Трудно сказать. Но как пример загрузки файла подойдет!

Форма загрузки будет выглядеть так:



А вывод файла так:



Загружать следует только текстовые файлы, ведь проверки на тип файл в скрипте нет

Теперь рассмотрим код скрипта.

<?php

// Функция вывода формы отправки файла. function print form()

{ \_

echo '<form method="post" enctype="multipart/form-data">'; echo '<input type="file" name="text" />';

echo '<br/>';

echo '<input type="submit" value="Загрузить файл!" />'; echo '</form>';

}

// Функция вывода содержимого файла. function print file($file)

{ \_

echo '<a href="index.php">Еще раз<^>'; echo ' <br/><br/>';

if ($file['name'] == '')

{

echo 'Файл не выбран! ' ;

**return**;

$lines = file($file['tmp name']);

$i = 1;

echo "<b>" . $ file['name'] . "</b>";

echo "<br/><br/>"; echo "<table>";

foreach ($lines as $s)

{

echo "<tr><td><small>$i:</small></td><td>$s</td></tr>";

$i++;

}

echo "</table>";

}

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<**html**>

<**head**>

<meta http-equiv="content-type" content="text/html; charset=windows- 1251">

^^^>Загрузка файла на сервер</title>

</**head**>

<**body**>

<h1>Пример загрузки файла на сервер</^>

<?php

if (isset($ FILES['text']))

{ \_

print file($ FILES['text']);

} \_ \_

**else**

{

print form();

} \_

?>

</**body**>

</**html**>

Для загрузки файла используется элемент HTML input с типом «файл»:

<input type="file" name="text" />

При этом важно указать у формы атрибут enctype:

<form method="post" enctype="multipart/form-data">

Если это не сделать, файл не будет загружен на сервер.

При обработке отправки формы информацию о загруженных файлах можно найти в системной переменной (словаре) $\_FILES.

Информация о файле включает:

* name: имя файла (как он называется у пользователя)
* tmp\_name: путь к временному файлу (на сервере)
* size: размер файла (в байтах)
* type: тип выбранного файла (например, «image/jpg»)
* • error: код ошибки (возникает в том случае, если попытка загрузки была неудачной)

Когда браузер отправляет файл на сервер, PHP сохраняет его во временной директории. Добраться до файла помогает свойство tmp\_name, там указан его путь. Вот так мы читаем содержимое файла в массив строк:

$lines = file($file['tmp name']);

1. Функции для работы с каталогами

bool mkdir(string Sname, int Sperms)

Создает каталог с именем $name и правами доступа $perms. Права доступа для каталогов указываются точно так же, как и для файлов. Чаще всего значение $perms устанавливают равным 0770 (предваряющий ноль обязателен — он указывает PHP на то, что это — восьмеричная константа, а не десятичное число).

Атрибут доступа 0770 означает "доступен для чтения, записи и исполнения для владельца и его группы". Атрибут исполнения, установленный для каталога, показывает, что пользователь сможет просмотреть содержимое каталога. Но это специфично для операционных систем семейства Unix.

В случае успеха функция возвращает true, иначе — false.

bool rmdir(string Sname)

Удаляет каталог с именем $name. В случае успеха возвращает true, иначе — false.

bool chdir(string Spath)

Сменяет текущий каталог на указанный. Если такого каталога не существует, возвращает false. Параметр $path может определять и относительный путь, задающийся от текущего каталога. Вот несколько примеров:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| chdir("/tmp/data" ) ; | // | переходим | по | абсолютному | пути |
| chdir("./somathing" | ; // | переходим | в | подкаталог текущего каталога | |
| chdir("something" ) ; | // | то же самое | |  |  |
| chdir("..; | // | переходим | в | родительский | каталог |

string getcwd()

Возвращает полный путь к текущему каталогу, начиная от "корня" (/). Если такой путь не может быть отслежен (это иногда бывает в Unix из-за того, что права на чтение для родительских каталогов могут быть сняты), вызов "проваливается" и возвращает false.

int opendir(string Spath)

Открывает каталог $path для дальнейшего считывания из него информации о файлах и подкаталогах и возвращает его идентификатор. Дальнейшие вызовы readdir() с идентификатором в параметрах будут обращены именно к этому каталогу. Функция возвращает false, если произошла ошибка.

string readdir(int $handle)

Считывает очередное имя файла или подкаталога из открытого ранее каталога с идентификатором $handle и возвращает его в виде строки. Вместе с именами подкаталогов и файлов будут также получены два специальных элемента: это «.» (ссылка на текущий каталог) и «..» (ссылка на родительский каталог). В подавляющем большинстве случаев нам нужно их игнорировать. Когда считывать больше нечего функция возвращает false.

void closedir(int $handle)

Закрывает ранее открытый каталог с идентификатором $handle. Не возвращает ничего.

void rewinddir(int $handle)

"Перематывает" внутренний указатель открытого каталога на начало. После этого можно воспользоваться readdir(), чтобы заново начать считывать содержимое каталога.

1. Получение списка файлов и подпапок в каталоге

Рассмотрим простой пример: выведем содержимое некоторого каталога.

<?php

$handle = opendir('/path/to/files') ;

if ($handle != false)

{

echo "Дескриптор каталога: $handle<br/>"; echo "Файлы:<Ьг/>";

while (false !== ($file = readdir($handle))) echo "$file<br/>";

closedir($handle);

}

?>

Обратите внимание на строгую проверку (включающую проверку типов):

false !== ($file = readdir($handle))

Иначе файл с именем «0» обработался бы неверно.

Резюме

Теперь вы знаете все необходимые функции для работы с файловой системой. Это позволит сохранять информацию на сайте, делать ее доступной многим пользователям. Однако для хранения текстовых и числовых данных, как правило, использую базу данных. Она позволяет осуществлять поиск записей, связывать данные разных типов, решает проблемы синхронизации. Часто информацию хранят совместно: частично в базе данных (текстовые и числовые данные), частично в виде файлов (картинки, документы, архивы).

Работа с базой данных будет рассмотрена в следующем Практическая работа.

Задания:

Создайте галерею фотографий.

Требования:

1. возможность загрузки фотографии на сервер (при загрузке должен проверяться тип файла)
2. при загрузке должна автоматически создаваться ее уменьшенная копия (не более 200 пикселей по наибольшей стороне)
3. возможность просмотра уменьшенных фотографий (все на одной странице)
4. возможность увеличения фотографии при клике по ней (переход на отдельную страницу)

Комментарии:

* для хранения файлов создайте два каталога - один для маленьких изображений, другой - для больших;
* ответ, как получить уменьшенную копию картинки, самостоятельно найдите в Интернете;
* для получения списка фотографий галереи воспользуйтесь механизмом обхода каталога.