УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы).

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: igor-gricenko-95@mail.ru **в течении ТРЕХ дней.**

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)132-63-42

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 13.

Тема: Файловая система, файлы и методы работы с файлами в PHP.

Цель: Изучить работу с файловой системой, файлами с помощью языка PHP.

Создание файла

Функция fopen

Вообще говоря, в PHP не существует функции, предназначенной именно для *создания файлов*. Большинство функций работают с уже существующими файлами в файловой системе сервера. Есть несколько функций, которые позволяют создавать временные файлы, или, что то же самое, файлы с уникальным для текущей директории именем. А вот для того, чтобы создать самый обычный файл, нужно воспользоваться функцией, которая открывает локальный или удаленный файл. Называется эта функция fopen(). Что значит «открывает файл»? Это значит, что *fopen* связывает данный файл с потоком управления программы. Причем связывание бывает различным в зависимости от того, что мы хотим делать с этим файлом: читать его, *записывать* в него данные или делать и то и другое. Синтаксис этой функции такой:

```
resource fopen ( имя_файла, тип_доступа
   [, use_include_path])
```

В результате работы эта функция возвращает указатель (типа ресурс) на открытый ею файл. В качестве параметров этой функции передаются: имя файла, который нужно открыть, *тип доступа к файлу* (определяется тем, что мы собираемся делать с ним) и, возможно, параметр, определяющий, искать ли

указанный файл в include_path. Есть еще один опциональный параметр, но о нем мы говорить не будем, дабы не усложнять изложение. Обсудим подробнее каждый из этих трех параметров.

Параметр имя_файла должен быть строкой, содержащей правильное локальное имя файла или URL-адрес файла в сети. Если имя файла начинается с указания протокола доступа (например, http://... или ftp://...), то интерпретатор считает это имя адресом URL и ищет обработчик указанного в URL протокола. Если обработчик найден, то PHP проверяет, разрешено ли работать с объектами URL как с обычными файлами (директива allow_url_fopen). Если allow_url_fopen=off, то функция *fopen* вызывает ошибку и генерируется предупреждение. Если имя файла не начинается с протокола, то считается, что указано имя локального файла. Чтобы открыть локальный файл, нужно, чтобы PHP имел соответствующие права доступа к этому файлу.

Параметр use_include_path, установленный в значение 1 или TRUE, заставляет интерпретатор искать указанный в fopen() файл в include_path. Напомним, что include_path – это директива из файла настроек PHP, задающая список директорий, в которых могут находиться файлы для включения. Кроме функции fopen() она используется функциями include() и require().

Параметр тип_доступа может принимать одно из следующих значений (см. таб. 13.1).

Итак, чтобы создать файл, нужно, как бы нелепо это ни звучало, открыть несуществующий файл на запись.

```
<?php
$h = fopen(“my_file.html”,”w”);
/* открывает на запись файл my_file.html,
если он существует, или создает пустой
файл с таким именем, если его еще нет */
$h = fopen(“dir/another_file.txt”,”w+”);
/* открывает на запись и чтение или создает
файл another_file.txt в директории dir */
$h = fopen(
  “http://www.server.ru/dir/file.php”,”r”);
/* открывает на чтение файл, находящийся по
указанному адресу*/
?>
```

Пример 13.1. Использование функции fopen() (html, txt)

Создавая файл, нужно учитывать, под какой операционной системой вы работаете, и под какой ОС предположительно этот файл будет читаться. Дело в том, что разные операционные системы по-разному отмечают конец строки. В Unix-подобных ОС конец строки обозначается \n, в системах типа Windows – \r\n. Windows предлагает специальный флаг t для перевода символов конца строки систем типа Unix в свои символы конца строки. В противоположность этому существует флаг b, используемый чаще всего для бинарных файлов, благодаря которому такой трансляции не происходит. Использовать эти флаги можно, просто дописав их после последнего символа выбранного *типа доступа к файлу* . Например, открывая файл на чтение, вместо r следует использовать rt, чтобы перекодировать все символы конца строки в \r\n. Если не использовать флаг b при открытии бинарных файлов, то могут появляться ошибки, связанные с изменением содержимого файла. Из соображений переносимости программы на различные платформы рекомендуется всегда использовать флаг b при открытии файлов с помощью fopen().

Таблица 13.1. Значения принимаемые параметром тип доступа

Тип доступа	**Описание**
r	Открывает файл только для чтения; устанавливает указатель позиции в файле на начало файла.
r+	Открывает файл для чтения и записи; устанавливает указатель файла на его начало.
w	Открывает файл только для записи; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
w+	Открывает файл для записи и для чтения; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
a	Открывает файл только для записи; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
a+	Открывает файл для записи и для чтения; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
x	Создает и открывает файл только для записи; помещает указатель файла на его начало. Если файл уже существует, то fopen() возвращает false и генерируется предупреждение. Если файл не существует, то делается попытка создать его. Этот тип доступа поддерживается начиная с версии PHP 4.3.2 и работает только с локальными файлами.

x+	Создает и открывает файл для записи и для чтения; помещает указатель файла на его начало. Если файл уже существует, то fopen() возвращает false и генерируется предупреждение. Если файл не существует, то делается попытка создать его. Этот тип доступа поддерживается, начиная с версии PHP 4.3.2, и работает только с локальными файлами.

Что происходит, если открыть или создать файл с помощью fopen не удается? В этом случае PHP генерирует предупреждение, а функция fopen возвращает как результат своей работы значение false. Такого рода предупреждения можно «подавить» (запретить) с помощью символа @ .

Например, такая команда не выведет предупреждения, даже если открыть файл не удалось:

```
$h = @fopen("dir/another_file.txt","w+");
```

Таким образом, функция fopen() позволяет создать только лишь пустой файл и сделать его *доступным для записи*. Как же *записать данные* в этот файл? Как прочитать данные из уже существующего файла?

Прежде чем ответить на эти вопросы, рассмотрим, как *закрыть* установленное с помощью fopen() соединение.

Закрытие соединения с файлом

После выполнения необходимых действий с файлом, будь то *чтение* или *запись данных* или что-либо другое, соединение, установленное с этим файлом функцией fopen(), нужно *закрыть* . Для этого используют функцию fclose(). Синтаксис у нее следующий:

```
fclose (указатель на файл)
```

Эта функция возвращает TRUE, если соединение успешно *закрыто*, и FALSE – в противном случае. Параметр этой функции должен указывать на файл, успешно открытый, например, с помощью функции fopen().

```
<?php
$h = fopen("my_file.html","w");
fclose($h);
?>
```

Пример 13.2. Использование функции fclose() (html, txt)

Конечно, если не закрывать соединение с файлом, никаких ошибок выполнения скрипта не произойдет. Но в целом для сервера это может иметь серьезные

последствия. Например, хакер может воспользоваться открытым соединением и записать в файл вирус, не говоря уже о лишней трате ресурсов сервера. Так что советуем всегда закрывать соединение с файлом после выполнения необходимых действий.

Запись данных в файл

Функция fwrite

Для того чтобы *записать данные в файл*, доступ к которому открыт функцией fopen(), можно использовать функцию fwrite(). Синтаксис у нее следующий:

```
int fwrite ( указатель на файл,
        строка [, длина])
```

Эта функция записывает содержимое строки строка в файл, на который указывает указатель на файл. Если указан дополнительный аргумент длина, то запись заканчивается после того, как записано количество символов, равное значению этого аргумента, или когда будет достигнут конец строки.

В результате своей работы функция fwrite() возвращает число записанных байтов или false, в случае ошибки.

Пример 13.3. Пусть в нашей рабочей директории нет файла my_file.html. Создадим его и запишем в него строку текста:

```
<?php
$h = fopen("my_file.html","w");
$text = "Этот текст запишем в файл.";
if (fwrite($h,$text))
  echo "Запись прошла успешно";
else
  echo "Произошла ошибка при записи данных";
fclose($h);
?>
```

Пример 13.3. Использование функции fwrite() (html, txt)

В результате работы этого скрипта в браузере мы увидим сообщение о том, что запись прошла успешно, а в файле my_file.html появится строка “Этот текст запишем в файл.”. Если бы этот *файл существовал* до того, как мы выполнили этот скрипт, все находящиеся в нем данные были бы удалены.

Если же мы напишем такой скрипт:

```
<?php
$h = fopen("my_file.html","a");
```

```
$add_text = "Добавим текст в файл.";
if(fwrite($h,$add_text,7))
  echo "Добавление текста прошло
    успешно<br>";
else echo "Произошла ошибка при
   добавлении данных<br>";
fclose($h);
?>
```

то к строке, уже существующей в файле my_file.html, добавится еще семь символов из строки, содержащейся в переменной $add_text, т.е. слово «Добавим»

Функция fwrite() имеет псевдоним fputs(), используемый таким же образом, что и сама функция.

Далее мы рассмотрим, какие методы *чтения данных из файла* предлагает язык PHP.

Чтение данных из файла

Если мы хотим прочитать данные из существующего файла, одной функции fopen(), как и в случае с *записью данных*, недостаточно. Она лишь возвращает указатель на открытый файл, но не *считывает* ни одной строки из этого файла. Поэтому для того, чтобы прочитать данные из файла, нужно воспользоваться одной из специальных функций: file, *readfile*, *file_get_contents*, *fread*, fgets и т.п.

Функция fread

Эта функция осуществляет *чтение данных из файла*. Ее можно использовать и для *чтения данных* из бинарных файлов, не опасаясь их повреждения. Синтаксис fread() такой:

string fread (указатель на файл, длина)

При вызове этой функции происходит *чтение данных* длины (в байтах), определенной параметром длина, из файла, на который указывает указатель на файл. Параметр указатель на файл должен быть реально существующей переменной типа ресурс, содержащей в себе связь с файлом, открытую, например, с помощью функции fopen(). Чтение данных происходит до тех пор, пока не встретится конец файла или пока не будет прочитано указанное параметром длина число байтов.

В результате работы функция fread() возвращает строку со *считанной* из файла информацией.

Как вы заметили, в этой функции параметр длина – обязательный. Следовательно, если мы хотим *считать* весь файл в строку, нужно знать его длину. PHP может самостоятельно вычислить длину указанного файла. Для этого нужно воспользоваться функцией filesize(имя файла). В случае ошибки эта функция вернет false. К сожалению, ее можно использовать только для получения *размера локальных файлов*.

Пример 13.4. Прочитаем содержимое файла my_file.html

```
<?php
$h = fopen("my_file.html","r+");
// отрываем файл на запись и чтение
$content = fread($h,
    filesize(“my_file.html”));
// считываем содержимое файла в строку
fclose($h); // закрываем соединение с файлом
echo $content;
// выводим содержимое файла
// на экран браузера
?>
```

Пример 13.4. Использование функции fread() (html, txt)

Для того чтобы *считать* содержимое бинарного файла, например изображения, в таких системах, как Windows, рекомендуется открывать файл с помощью флага rb или ему подобных, содержащих символ b в конце.

Функция filesize() кэширует результаты своей работы. Если изменить содержимое файла my_file.html и снова запустить приведенный выше скрипт, то результат его работы не изменится. Более того, если запустить скрипт, считывающий данные из этого файла с помощью другой функции (например, fgetss), то результат может оказаться таким, как если бы файл не изменился. Чтобы этого избежать, нужно очистить статический кэш, добавив в код программы команду clearstatcache();

Функция fgets

С помощью функции fgets() можно *считать из файла строку* текста. Синтаксис этой функции практически такой же, как и у fread(), за исключением того, что длину *считываемой строки* указывать необязательно:

string fgets (указатель на файл [, длина])

В результате работы функция fgets() возвращает строку длиной (длина–1) байт из файла, на который указывает указатель на файл. Чтение заканчивается, если

прочитано (длина–1) символов и встретился символ перевода строки или конец файла. Напомним, что в PHP один символ – это один байт. Если длина *считываемой строки* не указана (данная возможность появилась начиная с PHP 4.2.0), то считывается 1 Кбайт (1024 байт) текста или, что то же самое, 1024 символа. Начиная с версии PHP 4.3, если параметр длина не задан, *считывается строка* целиком. В случае ошибки функция fgets() возвращает false. Для версий PHP начиная с 4.3 эта функция безопасна для двоичных файлов.

```
<?php
$h = fopen("my_file.html","r+");
$content = fgets($h,2);
// считает первый символ из
// первой строки файла my_file.html
fclose($h);
echo $content;
?>
```

Пример 13.5. Использование функции fgets() (html, txt)

Обе функции, fread() и fgets(), прекращают считывание данных из файла, если встречают конец файла. В PHP есть специальная функция, проверяющая, смотрит ли указатель позиции файла на конец файла. Это булева функция feof(), в качестве параметра которой передается указатель на соединение с файлом.

Например, вот так можно *считать* все строки файла my_file.html:

```
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
   $content = fgets($h);
   echo $content,"<br>";
}
fclose($h);
?>
```

Функция fgetss

Существует разновидность функции fgets() – функция fgetss(). Она тоже позволяет *считывать* строку из указанного файла, но при этом удаляет из него все встретившиеся html-теги, за исключением, быть может, некоторых. Синтаксис fgetss() такой:

```
string fgetss(указатель на файл,
  длина [, допустимые теги])
```

Обратите внимание, что здесь аргумент длина обязательный.

Пример 13.6. Пусть у нас имеется файл my_file.html следующего содержания:

```
<h1>Без труда не вынешь и рыбку из пруда.</h1>
<b>Тише едешь – дальше будешь</b>
У семи нянек<i> дитя без глазу</i>.
```

Выведем на экран все строки файла my_file.html, удалив из них все теги, кроме <b> и <i>:

```
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
   $content = fgetss($h,1024,'<b><i>');
  echo $content,"<br>";
}
fclose($h);
?>
```

Пример 13.6. Использование функции fgetss() (html, txt)

В результате работы этого скрипта получим:

Тише едешь – дальше будешь

Без труда не вынешь и рыбку из пруда.

У семи нянек дитя без глазу.

Функция fgetc

Естественно, если можно считывать информацию из файла построчно, то можно *считывать* ее и посимвольно. Для этого предназначена функция fgetc(). Легко догадаться, что синтаксис у нее следующий:

```
string fgetc ( указатель на файл )
```

Эта функция возвращает символ из файла, на который ссылается указатель на файл, и значение, вычисляемое как FALSE, если встречен конец строки.

Вот так, например, можно *считать* файл по одному символу:

```
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
   $content = fgetc($h);
   echo $content,"<br>";
}
fclose($h);
?>
```

На самом деле для того чтобы прочитать содержимое файла, открывать соединение с ним посредством функции fopen() совсем не обязательно. В PHP

есть функции, которые позволяют делать это, используя лишь имя файла. Это функции readfile( ), file( ) и file_get_contents( ). Рассмотрим каждую их них подробнее.

Функция readfile

Синтаксис:

```
int readfile ( имя_файла
   [, use_include_path])
```

Функция readfile() считывает файл, имя которого передано ей в качестве параметра имя_файла, и выводит его содержимое на экран. Если дополнительный аргумент use_include_path имеет значение TRUE, то поиск файла с заданным именем производится и по директориям, входящим в include_path.

В программу эта функция возвращает число *считанных* байтов (символов) файла, а в случае ошибки – FALSE. Сообщения об ошибке в этой функции можно подавить оператором @ .

Пример 13.7. Следующий скрипт выведет на экран содержимое файла my_file1.html и *размер этого файла*, если он существует. В противном случае выведется наше сообщение об ошибке – строка "Error in readfile".

```
<?php
$n = @readfile ("my_file1.html");
/* выводит на экран содержимое файла и
записывает его размер в переменную $n */
if (!$n) echo "Error in readfile";
/* если функция readfile() выполнилась
с ошибкой, то $n=false и выводим
сообщение об ошибке */
else echo $n;
  // если ошибки не было, то выводим число
  // считанных символов
?>
```

Пример 13.7. Использование функции readfile() (html, txt)

С помощью функции readfile() можно читать содержимое удаленных файлов, указывая их URL-адрес в качестве имени файла, если эта опция не отключена в настройках сервера.

Сразу же выводить содержимое файла на экран не всегда удобно. Порой нужно записать информацию из файла в переменную, чтобы в дальнейшем произвести с

ней какие-либо действия. Для этого можно использовать функцию file() или file_get_contents().

Функция file

Функция *file()* предназначена для считывания информации из файла в переменную типа *массив*. Синтаксис у нее такой же, как и у функции readfile(), за исключением того, что в результате работы она возвращает массив:

```
array file ( имя_файла
   [, use_include_path])
```

Что за массив возвращает эта функция? Каждый элемент данного массива является строкой в файле, информацию из которого мы считываем (его имя задано аргументом имя_файла). Символ новой строки тоже включается в каждый из элементов массива. В случае ошибки функция *file()*, как и все уже рассмотренные, возвращает false. Дополнительный аргумент use_include_path опять же определяет, искать или нет данный файл в директориях include_path. Открывать удаленные файлы с помощью этой функции тоже можно, если не запрещено сервером. Начиная с PHP 4.3 работа с бинарными файлами посредством этой функции стала безопасной.

Например, у нас имеется файл my_file.html следующего содержания:

```
<h1>Без труда не вынешь
   и рыбку из пруда.</h1>
<b>Тише едешь – дальше будешь</b>
```

Прочитаем его содержимое с помощью функции file():

```
<?php
$arr = file ("my_file.html");
foreach($arr as $i => $a) echo $i,": ",
   htmlspecialchars($a), "<br>";
?>
```

В результате на экран будет выведено следующее сообщение:

```
0: <h1>Без труда не вынешь
   и рыбку из пруда.</h1>
1: <b>Тише едешь – дальше будешь</b>
```

Функция file_get_contents

В версиях PHP начиная с 4.3 появилась возможность считывать содержимое файла в строку. Делается это с помощью функции file_get_contents(). Как и две предыдущие функции, в качестве параметров она принимает значение имени

файла и, возможно, указание искать его в директориях include_path. Для порядка все равно приведем ее синтаксис:

string file_get_contents (

имя_файла [, use_include_path])

Эта функция абсолютно идентична функции file(), только возвращает она содержимое файла в виде строки. Кроме того, она безопасна для обработки бинарных данных и может считывать информацию из удаленных файлов, если это не запрещено настройками сервера.

Проверка существования файла

Итак, создавать файл мы научились, записывать данные в него – научились, считывать данные из файла – тоже научились. Но вот вопрос: а что если файла, с которым мы пытаемся проделать все эти операции, не существует? Или он недоступен для чтения или записи? Очевидно, что в таком случае ни одна из изученных нами функций работать не будет и PHP выдаст сообщение об ошибке. Чтобы отслеживать такого рода ошибки, можно использовать функции file_exists(), is_writable(), is_readable().

Функция file_exists

Синтаксис:

bool file_exists (имя файла или директории)

Функция file_exist() проверяет, существует ли файл или директория, имя которой передано ей в качестве аргумента. Если директория или файл в файловой системе сервера существует, то функция возвращает TRUE, в противном случае – FALSE. Результат работы этой функции кэшируется. Соответственно очистить кэш можно, как уже отмечалось, с помощью функции clearstatcache(). Для нелокальных файлов использовать функцию file_exists() нельзя.

```
<?php
$filename = 'c:/users/files/my_file.html';
if (file_exists($filename)) {
  print "Файл <b>$filename</b> существует";
} else {
  print "Файл <b>$filename</b>
       НЕ существует";
}
?>
```

Пример 13.8. Использование функции file_exist() (html, txt)

Функция is_writable

Если кроме проверки *существования файла* нужно узнать еще, разрешено ли записывать информацию в этот файл, следует использовать функцию is_writable() или ее псевдоним – функцию is_writeable().

Синтаксис:

bool is_writable (имя файла или директории)

Эта функция возвращает TRUE, если файл (или директория) существует и *доступен для записи*. Доступ к файлу осуществляется под той учетной записью пользователя, под которой работает сервер (чаще всего это пользователь nobody или www). Результаты работы функции is_writable кэшируются.

Функция is_readable

Если кроме проверки *существования файла* нужно узнать еще, разрешено ли читать информацию из него, нужно использовать функцию is_readable().

Синтаксис:

bool is_readable (имя файла)

Эта функция работает подобно функции is_writable().

```
<?php
$filename = 'c:/users/files/my_file.html';
if (is_readable($filename)) {
  print "Файл <b>$filename</b> существует
     и доступен для чтения";
} else {
  print "Файл <b>$filename</b>
     НЕ существует или
     НЕ доступен для чтения";
}
?>
```

Удаление файла

Последнее, что мы хотим изучить из действий над файлами, – это *удаление файлов*. Для того чтобы удалить файл с помощью языка PHP, нужно воспользоваться функцией unlink(). Синтаксис этой функции можно описать следующим образом:

bool unlink (имя_файла)

Данная функция удаляет файл, имеющий имя имя_файла, возвращает TRUE в случае успеха этой операции и FALSE – в случае ошибки. Чтобы удалить файл,

нужно тоже иметь соответствующие права доступа к нему (например, *доступа только на чтение* для *удаления файла* недостаточно).

```
<?php
$filename = 'c:/users/files/my_file.html';
unlink($filename);
  // удаляем файл с именем
  // c:/users/files/my_file.html
?>
```

Пример 13.10. Использование функции unlink() (html, txt)

Загрузка файла на сервер

Теперь решим более сложную и часто возникающую на практике задачу *загрузки файла на сервер*. Первое, что нужно сделать, чтобы загрузить файл на сервер, это создать html-форму. Для того чтобы с помощью этой формы можно было загружать файлы, она должна содержать атрибут enctype в теге form со значением multipart/form-data, а также элемент input типа file.

Пример 13.11.

```
<form enctype="multipart/form-data"
    action="parse.php" method="post">
  <input type="hidden" name="MAX_FILE_SIZE"
    value="30000" />
  Загрузить файл: <input type="file"
    name="myfile" /><br>
  <input type="submit"
    value="Отправить файл" />
</form>
```

Пример 13.11. Форма для загрузки файла на сервер (html, txt)

Заметим, что мы добавили в форме скрытое поле, которое содержит в себе максимальный допустимый *размер* загружаемого файла в байтах. При попытке загрузить файл, *размер* которого больше указанного в этом поле значения, будет зафиксирована ошибка. В браузере созданная нами форма будет выглядеть как строка для ввода текста с дополнительной кнопкой для выбора файла с локального диска (рис 13.1).

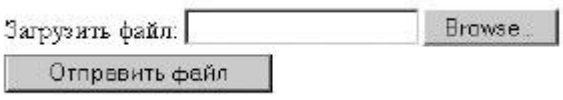

Рис. 13.1. Пример формы для загрузки файла на сервер

Теперь нужно написать скрипт, который будет обрабатывать полученный файл.

Вся информация о загруженном на сервер файле содержится в глобальном массиве $_FILES. Этот массив появился начиная с PHP 4.1.0. Если включена директива register_globals, то значения переданных переменных доступны просто по их именам.

Если мы загрузили с компьютера-клиента файл с именем critics.htm *размером* 15136 байт, то скрипт с единственной командой print_r($_FILES); выведет на экран следующее:

```
Array ( [myfile] =>
  Array ( [name] => critics.htm
    [type] => text/html
    [tmp_name] => C:\WINDOWS\TEMP\php49F.tmp
    [error] => 0
    [size] => 15136
    )
)
```

Вообще говоря, массив $_FILES всегда имеет следующие элементы:

- $_FILES['myfile']['name'] – имя, которое имел файл на машине клиента.
- $_FILES['myfile']['type'] – mime-тип отправленного файла, если браузер предоставил эту информацию. В нашем примере это text/html.
- $_FILES['myfile']['size'] – *размер* загруженного файла в байтах.
- $_FILES['myfile']['tmp_name'] – временное имя файла, под которым он был сохранен на сервере.
- $_FILES['myfile']['error'] – код ошибки, появившейся при загрузке.

Здесь 'myfile' – это имя элемента формы, с помощью которого была произведена *загрузка файла на сервер*. То есть оно может быть другим, если элемент формы назвать иначе. Но вот другие ключи (name, type и т. д.) остаются неизменными для любой формы.

Если register_globals=On, то доступны также дополнительные переменные, такие как $myfile_name, которая эквивалентна $_FILES['myfile']['name'], и т.п.

Ошибок при загрузке в PHP выделяют пять типов и соответственно $_FILES['myfile']['error'] может иметь пять значений:

0 – ошибки не произошло, файл загружен успешно

1 – загружаемый файл превышает *размер*, установленный директивой upload_max_filesize в файле настроек php.ini

2 – загружаемый файл превышает *размер*, установленный элементом MAX_FILE_SIZE формы html

3 – файл был загружен частично

4 – файл загружен не был

По умолчанию загруженные файлы сохраняются во временной директории сервера, если другая директория не указана с помощью опции upload_tmp_dir в файле настроек php.ini. Переместить загруженный файл в нужную директорию можно с помощью функции move_uploaded_file().

Функция move_uploaded_file() имеет следующий синтаксис:

bool move_uploaded_file (временное_имя_файла, место_назначения)

Эта функция проверяет, действительно ли файл, обозначенный строкой временное_имя_файла, был загружен через механизм загрузки HTTP методом POST. Если это так, то файл перемещается в файл, заданный параметром место_назначения (этот параметр содержит как путь к новой директории для хранения, так и новое имя файла).

Если временное_имя_файла задает неправильный загруженный файл, то никаких действий произведено не будет, и move_uploaded_file() вернет FALSE. То же самое произойдет, если файл по каким-то причинам не может быть перемещен. В этом случае интерпретатор выведет соответствующее предупреждение. Если файл, заданный параметром место_назначения, существует, то функция move_uploaded_file() перезапишет его.

```
<?
/* В версиях PHP, более ранних,
чем 4.1.0, вместо массива
$_FILES нужно использовать
массив $HTTP_POST_FILES */

$uploaddir = 'c:/uploads/';
   // будем сохранять загружаемые
   // файлы в эту директорию
$destination = $uploaddir .
    $_FILES['myfile']['name'];
   // имя файла оставим неизменным
print "<pre>";
if (move_uploaded_file(
   $_FILES['myfile']['tmp_name'],
   $destination)) {
/* перемещаем файл из временной папки
в выбранную директорию для хранения */
```

```
    print "Файл успешно загружен <br>";
} else {
  echo "Произошла ошибка при загрузке файла.
    Некоторая отладочная информация:<br>";
    print_r($_FILES);
}
print "</pre>";
?>
```