

Ларри Ульман

Основы программирования на PHP

Самоучитель

Larry Ullman

PHP for the World Wide Web



Peachpit Press

Ларри Ульман

Основы программирования на PHP

Самоучитель



Москва, 2001

УДК 004.438PHP
ББК 32.973.26-018.2
У51

У51 Ульман Л.

Основы программирования на PHP: Пер. с англ. -М.: ДМК Пресс, 2001. - 288 с.: ил. (Самоучитель).

ISBN 5-94074-124-X

Представленная книга посвящена PHP - серверному межплатформенно-му встроенному в HTML языку написания сценариев. Рассматриваются следующие вопросы: синтаксис языка, строки и управляющие структуры, массивы и регулярные выражения, функции; описываются приемы отладки ваших сценариев. Особое внимание уделяется получению введенной в форму информации, работе с файловой **системой**, базами данных, cookie и др.

Включенные в состав книги приложения содержат информацию об установке и настройке Web-сервера, инсталляции языка PHP. Здесь же обсуждаются вопросы безопасности скриптов, даются ссылки на Web-ресурсы, посвященные PHP.

Книга будет полезна как начинающим Web-мастерам, которые только собираются создавать динамические сайты, так и профессиональным дизайнерам, желающим внести **элементы** динамики в проектируемые ими ресурсы.

Authorized translation from the English language edition, entitled "PHP for the World Wide Web **Visual Quickstart Guide**", published by Peachpit Press, Copyright©2001.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by DMK Press. Copyright©2001

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими **быто ни** было средствами без письменного разрешения владельца авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность **технических** ошибок все равно остается, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В **связи** с этим издательство не несет ответственности за **возможный** ущерб любого вида, связанный с применением содержащихся здесь **сведений**.

Все торговые знаки, упомянутые в настоящем **издании**, зарегистрированы. **Случайное** неправильное использование или пропуск торгового знака или названия его законного **владельца** не должно рассматриваться как нарушение прав собственности.

ISBN 0-201-72787-0 (англ.)
ISBN 5-94074-124-X (рус.)

© 2001 by Peachpit Press
© Перевод на русский язык,
оформление ДМК Пресс, 2001

Содержание

Введение	9
Глава 1 т	
Первые шаги с PHP	19
Основы синтаксиса.....	19
Передача данных в браузер.....	20
Тестирование сценария.....	22
Передача простого текста в браузер.....	23
Передача страницы HTML в браузер.....	25
Использование пробельных символов в PHP и HTML	26
Добавление комментариев в сценарий.....	28
Глава 2 т	
Переменные	32
Что такое переменная.....	32
Синтаксис переменных.....	33
Типы переменных.....	34
Присвоение значений переменным.....	36
Предопределенные переменные.....	36
Глава 3 т	
HTML-формы и PHP	38
Создание простой формы.....	38
Использование методов Get и Post.....	41
Получение данных из формы в PHP.....	43
Ввод данных вручную.....	45
Глава 4 т	
Использование чисел	50
Сложение, вычитание, умножение и деление.....	50
Форматирование чисел	53
Инкремент и декремент.....	55
Совместное использование различных операторов	56
Использование встроенных математических функций.....	58

Глава 5 ▼

Использование строк	61
Удаление концевых пробелов.....	61
Соединение строк (сцепление, конкатенация).....	65
Кодирование и декодирование строк.....*	67
Шифрование и дешифрование строк.....	71
Извлечение части строки.....	74

Глава 6 т

Управляющие структуры	79
Условный оператор if.....	79
Другие операторы.....	83
Использование оператора if-else.....	89
Использование конструкции if-elseif.....	91
Условная конструкция switch.....	94
Цикл while.....	101
Цикл for.....	106

Глава 7 т

Массивы	109
Создание массива.....	110
Добавление элементов в массив.....	112
Доступ к элементам массива.....	115
Сортировка массивов.....	118
Преобразование строк и массивов.....	121
Создание массива в экранной форме.....	125
Создание многомерных массивов.....	128

Глава 8 т

Регулярные выражения	130
Что такое регулярные выражения.....	130
Создание простого шаблона.....	131
Сопоставление с шаблонами.....	133
Создание более сложных шаблонов.....	137
Сопоставление с шаблоном и его замена.....	140

Глава 9 ▼

Создание функций	144
Создание и использование простых функций.....	144
Создание и вызов функций, принимающих аргументы.....	148
Создание и использование функций, возвращающих значение.....	152
Переменные и функции.....	157

Глава 10 ▼

Файлы и каталоги	165
Права доступа к файлам.....	165
Запись данных в файл.....	167
Чтение файла.....	173
Каталоги.....	180
Загрузка файла на удаленный компьютер.....	185
Переименование и удаление файлов и каталогов.....	188

Глава 11 ▼

Базы данных	195
Соединение с сервером и создание базы данных.....	197
Создание таблицы.....	200
Отправка данных.....	204
Извлечение данных.....	207

Глава 12 ▼

Использование cookie	211
Создание и чтение cookie.....	212
Добавление параметров в cookie.....	217
Удаление cookie.....	220

Глава 13 ▼

Создание Web-приложений	224
Использование функций include и require.....	224
Определение даты и времени.....	228
Использование HTTP-заголовков.....	236
Отправка электронной почты.....	240

Глава 14 т

Отладка сценариев	245
Распространенные ошибки.....	245
Сообщения о возможных ошибках и их протоколирование.....	248
Отслеживание ошибок	252
Использование инструкции die.....	256

Приложение А т

Установка и конфигурация	260
Установка на сервер Linux.....	260
Установка на сервер Windows.....	265
Конфигурация.....	267

Приложение В ▼**Безопасность.....269**

Криптография и SSL269

Написание безопасного PHP-кода.....270

Ресурсы по вопросам безопасности.....272

Приложение С т**Ресурсы PHP.....273**

Руководство по PHP.....273

Web-сайты и сетевые конференции.....274

Ресурсы по базам данных.....277

Сложные темы.....278

Таблицы.....279

Предметный указатель.....283

Введение

Всемирная паутина удивительна и загадочна. Сталкиваясь с аббревиатурами, обозначающими системы, которые чересчур часто меняются, пользователь может потерять терпение.

Одним из примеров нового направления информационных технологий может служить бурное развитие *программ с открытым кодом* (OSS - Open Source Software), свободно доступных как для **распространения**, так и для модификации всеми желающими. Наиболее известной в этом смысле является операционная система Unix, в частности ядро Linux. Однако, хотя с помощью программ с открытыми исходниками создаются стабильные и очень полезные продукты, освоить их порой затруднительно. Отсутствие учебников для начинающих и удобных описаний не позволяет мощным технологиям стать настолько популярными, насколько они того заслуживают. PHP, весьма доступный язык написания сценариев для Web, представляет собой еще один прекрасный инструмент, который, хоть и обладает уникальными возможностями и легок в использовании, отпугивал многих до сегодняшнего дня.

Представленная книга не только поможет вам изучить PHP, но и подскажет, где искать дополнительную информацию по этому языку. Хотя издание и не является всеохватывающим руководством по программированию, вы получите знания, необходимые для создания динамических Web-сайтов и приложений с помощью PHP.

Что такое PHP

Изначально аббревиатура PHP расшифровывалась как Personal Home Page (личная домашняя страничка). Этот язык был создан в 1994 году Расмусом Лердорфом (Rasmus Lerdorf), чтобы отслеживать пользователей, просматривавших его домашнюю страничку с резюме. Позже, когда функциональность PHP значительно расширилась и профессионалы начали использовать этот язык для создания сложных сайтов, сокращение стали **расшифровывать** как «гипертекстовый препроцессор» (PHP: Hypertext Preprocessor). Определение означает, что данные в этом языке обрабатываются до того, как становятся HTML-страницей (HTML - язык гипертекстовой разметки).

Согласно официальному сайту PHP (www.php.net, рис. 1), PHP является серверным межплатформенным встроенным в HTML языком написания сценариев. Это может показаться довольно сложным определением, но оно станет простым и понятным, если рассмотреть его по частям.

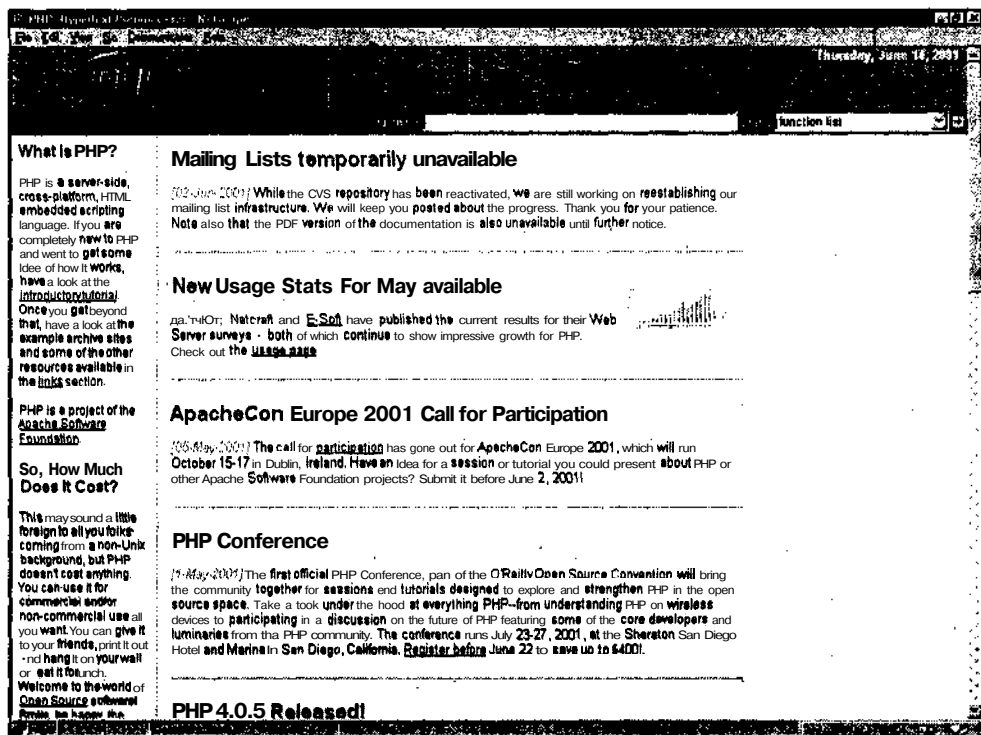


Рис. 1.1 Так выглядел официальный Web-сайт PHP, когда книга **готовилась** к печати. Разумеется, с вопросами, связанными с этим языком, следует обращаться по указанному выше адресу. На сайте имеется последняя версия руководства по PHP с комментариями **пользователей**, там же вы найдете ссылки на другие полезные ресурсы

Во-первых, слово «серверный» означает, что операции PHP выполняются на стороне сервера (в противоположность клиенту, компьютеру пользователя, с которого последний просматривает Web-сайт). **Сервер** – это специальный компьютер. На нем хранятся страницы, которые вы видите, когда заходите по указанному в браузере адресу, например Navigator или Internet Explorer. Этот процесс будет описан подробнее чуть позже.

Определение «межплатформенный» означает, что язык PHP может использоваться под Unix, Windows NT, Macintosh, OS/2 и другими серверными, но не клиентскими операционными системами. При этом вы можете переносить свою работу на другую платформу почти или вообще без **изменений**. Разумеется, PHP-сценарии **также** можно разрабатывать в любой операционной системе, как и в случае HTML-страниц.

Словосочетание «встроенный в HTML» означает, что PHP может быть вложен в код HTML, с помощью которого строятся все Web-страницы. Поэтому реальное программирование на PHP может быть лишь немного сложнее, чем создание кода вручную на HTML.

Наконец, PHP является языком написания сценариев, а это свойство немногих языков программирования. Значит, операции в языке начинают выполняться

только после того, как событие произошло, например когда пользователь передает форму или обращается на URL (Uniform Resource Locator - технический термин, означающий Web-адрес). Языки программирования, такие как Java, С или Perl, можно использовать для написания автономных приложений, которые могут не иметь никакого отношения к Internet. Наиболее известный пример языка написания сценариев – JavaScript. На этом языке можно обрабатывать события, происходящие в Web-браузере. В этом смысле JavaScript похож на PHP, хотя и выполняется исключительно на стороне клиента. Другими способами определения типа языка могут служить термины «интерпретируемый» и «транслируемый». Так, PHP и JavaScript не работают без программы-интерпретатора, примеры транслируемых языков программирования – С и Java.

В настоящее время разработана четвертая версия PHP, но, так как она появилась недавно, на многих серверах по-прежнему используется версия 3.x. Данная книга посвящена именно PHP 4.0, хотя, если говорить о программировании в широком смысле слова, различия между двумя версиями незначительны. Основное преимущество новой разработки – ее улучшенная производительность. Дополнительную информацию по PHP 4.0 можно найти на сайте разработчиков этой версии по адресу www.zend.com (рис. 2).

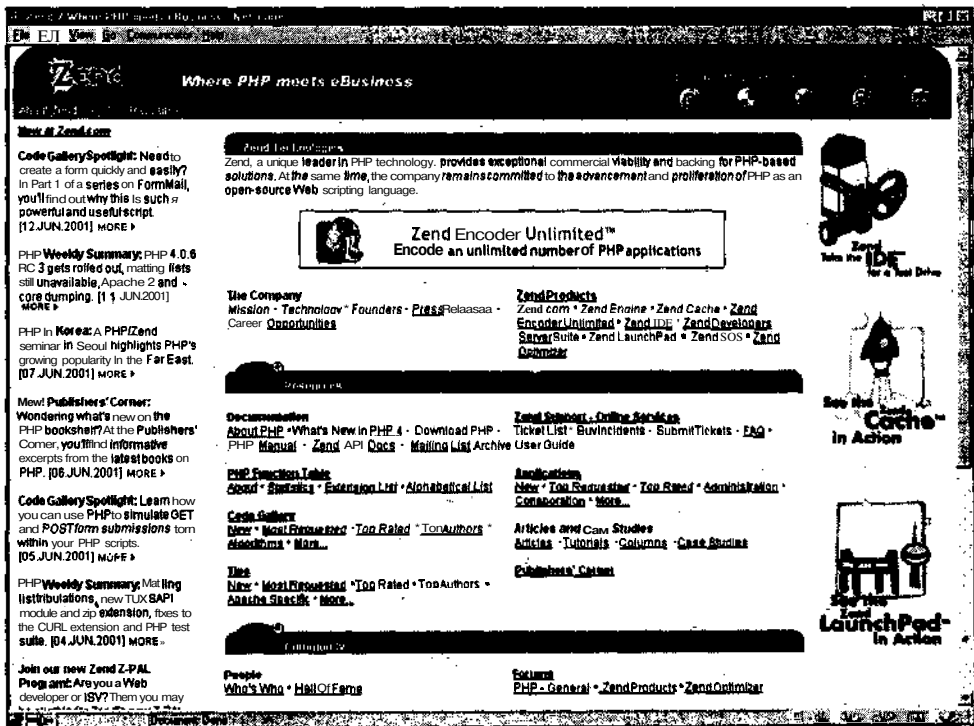


Рис. 2 Это заглавная страница сайта Zend - ресурса создателей кода, лежащего в основе PHP 4.0. Здесь можно найти подробную информацию по последней версии PHP

Преимущества PHP

Попросту говоря, PHP быстрее, лучше и проще, чем другие языки. При разработке **Web-сайтов** основными альтернативами PHP являются: базовый HTML, **CGI-сценарии** (Common Gateway Interface, обычно (но не обязательно) написанные на языке Perl), ASP (**A**ctive **S**erver **P**ages) и JSP (**J**ava **S**erver **P**ages). Язык JavaScript не является альтернативой PHP, поскольку это клиентская технология, которая не может быть использована для создания HTML-страниц таким же образом, как собственно PHP или CGI.

Преимущество PHP перед базовым HTML состоит в том, что последний представляет собой систему с ограниченными возможностями, не обладающую гибкостью или динамичностью. Посетители сайтов видят обычные статические HTML-страницы, без каких либо персональных настроек. С помощью же PHP можно создавать привлекательные оригинальные Web-страницы на основе любых задаваемых вами критериев (например, времени суток или операционной системы **пользователя**). В отличие от HTML язык PHP также может взаимодействовать с базами данных и файлами, с его помощью обрабатывается электронная почта и выполняются многие другие операции.

Web-мастера давно поняли, что невозможно создать действительно привлекательные и универсальные Web-сайты только с помощью HTML. Поэтому серверные технологии, такие как CGI-сценарии, получили широкую популярность. Подобные системы позволяют дизайнерам создавать динамично генерируемые Web-приложения, где во внимание принимаются любые мелочи, которые захотелось учесть программисту. При настроенном взаимодействии с базой данных наиболее продвинутые Web-сайты позволяют обновлять и сопровождать страницы быстрее и легче, чем это возможно с базовым HTML.

Очевидно, что вопрос нужно ставить **таким** образом: почему Web-дизайнер должен использовать язык PHP вместо CGI, ASP или JSP для создания динамического Web-сайта? Во-первых, программировать на PHP проще и быстрее, чем на CGI, к тому же и написанные сценарии выполняются с более высокой скоростью.

Я не буду подробно описывать детали запуска программ на сервере и тем более открывать дискуссию на эту тему. Достаточно упомянуть тот факт, что освоить и использовать PHP намного **легче**, чем универсальные языки программирования. Любой **человек**, в том числе не получивший специальной подготовки по программированию, сможет легко писать сценарии PHP после прочтения данной книги. Сравните: ASP и CGI являются достаточно полными языками и, следовательно, **более** сложны в усвоении, при этом первый требует понимания VBScript, а CGI - Perl (или C).

- **Во-вторых**, в отличие от Perl, VBScript и Java язык PHP был разработан специально для создания динамических Web-страниц, что подразумевает выполнение им именно этих задач быстрее и легче, чем альтернативными языками. Хочу, однако, подчеркнуть, что, хотя для определенных целей PHP подходит лучше, чем

CGI или ASP, он не является оптимальным языком программирования. Например, на PHP удастся делать не все, что возможно на языках Java или Perl.

На PHP можно взглянуть и с другой стороны – как на необходимое и естественное расширение возможностей языка разметки HTML. Необычайная популярность последнего и взрыв интереса к Internet показали ограниченность возможностей этого языка. Несколько дополнений к стандарту HTML повысили изобразительные возможности языка, добавили способность выполнять программы на JavaScript в браузере. Но без PHP HTML так и не стал полноценным языком программирования. В нем нет знакомых любому разработчику операторов организации циклов, условных переходов, функций, структур данных и прочего. Одинаково правильны оба утверждения: у языка PHP встроены все синтаксические конструкции HTML; у HTML, при подключении на сервере модуля PHP, появляются возможности настоящего языка программирования.

PHP уже используется на более чем трех миллионах Web-сайтов, и его популярность продолжает расти, а это ли не последний аргумент в пользу изучения данного языка?

От скрипта до изображения на экране

PHP – серверный язык. Это значит, что написанный вами код постоянно находится на стороне сервера, который посылает Web-страницы в браузер.

Когда пользователь собирается зайти на Web-сайт, например по адресу www.DMCinsights.com, провайдер направляет этот запрос на сервер, на котором хранится информация.

На сервере код PHP читается и выполняется в соответствии с прописанными в нем командами. В нашем примере сервер должен отправить соответствующие командам Web-страницы в браузер посетителя в виде HTML (рис. 3). Образно говоря, PHP создает HTML-страницу «на лету», как это запрограммировано в сценарии, и в этом случае на сервере вообще нет статических HTML-страниц.

На сайте со статичными HTML-страницами все происходит по-другому. На запрос клиента сервер посылает в Web-браузер только данные HTML, при этом на серверной стороне не происходит никакой интерпретации данных (рис. 4). Следовательно, для браузера конечного пользователя может не существовать никакой разницы между тем, как выглядят страницы `home.html` и `home.php`, но путь, по которому пришел ответ, различен. С помощью PHP вы можете «заставить» сервер динамически генерировать HTML-код. Различная информация может быть представлена пользователю при посещении сайта в понедельник и во вторник, при первом и последующем обращениях к этому ресурсу. Создание динамических Web-страниц – вот что отличает менее привлекательные статические сайты от более интересных и, следовательно, более посещаемых ресурсов.

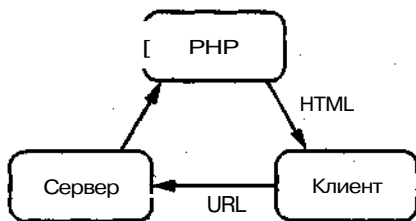


Рис. 3 Данная схема демонстрирует процесс взаимодействия клиента, сервера и модуля PHP (приложение, установленное на сервер для расширения его функциональности) при отправке **HTML-страницы** в браузер. Во всех серверных технологиях, в том числе в ASP, для обработки данных, которые отправляются клиенту, используются подобные дополнительные модули

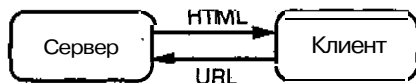


Рис. 4 Сравните прямое непосредственное взаимодействие **Web-сервера** и браузера с тем, что приведено на рис. 3. Простые HTML-страницы можно просматривать в браузере на вашем компьютере без участия сервера. К динамически генерируемым страницам доступ осуществляется только через специально настроенный Web-сервер, которым и проводится необходимая обработка

Основное различие между языком PHP и базовым HTML уже описывалось, однако еще раз остановимся на этом моменте. Все операции PHP выполняются на сервере, и затем последний посылает соответствующую информацию в браузер. В данной книге описано, как использовать PHP для того, чтобы отправить необходимые данные в браузер.

Платформа для Web-приложений

Поскольку PHP является серверным языком написания сценариев, то самое первое требование для программирования - наличие доступа к серверу, позволяющему работать с PHP. Принимая во внимание популярность PHP, вполне вероятно, что ваш провайдер (ISP - Internet Service Provider), предоставляющий Web-хостинг, имеет эту опцию на своих серверах. На всякий случай свяжитесь с представителями фирмы-провайдера и узнайте, какие технологии они поддерживают. На момент написания книги язык PHP поддерживали более тысячи поставщиков Web-хостинга (рис. 5).

Другой вариант - установить PHP на свой **собственный** компьютер (обычно это компьютер, который работает под управлением операционных систем Windows NT или Linux), на котором также должен быть установлен Web-сервер. Можно использовать доступный Apache для операционных систем Unix и NT или Personal Web Sharing - для Windows. Краткая информация по установке PHP содержится в приложении А, «Установка и **конфигурация**». Если вы собираетесь использовать свой собственный сервер, PHP можно бесплатно загрузить с сайта www.php.net. Установка не вызовет затруднений.

Второе требование для работы с PHP - наличие на вашем компьютере любого текстового редактора. Программ NotePad, WordPad, SimpleText и подобных им будет вполне достаточно, хотя BBEdit, WordPerfect, Word и другие коммерческие приложения предоставляют большую функциональность. Если вы привыкли работать в редакторе, в котором используется графический интерфейс (WYSIWYG -

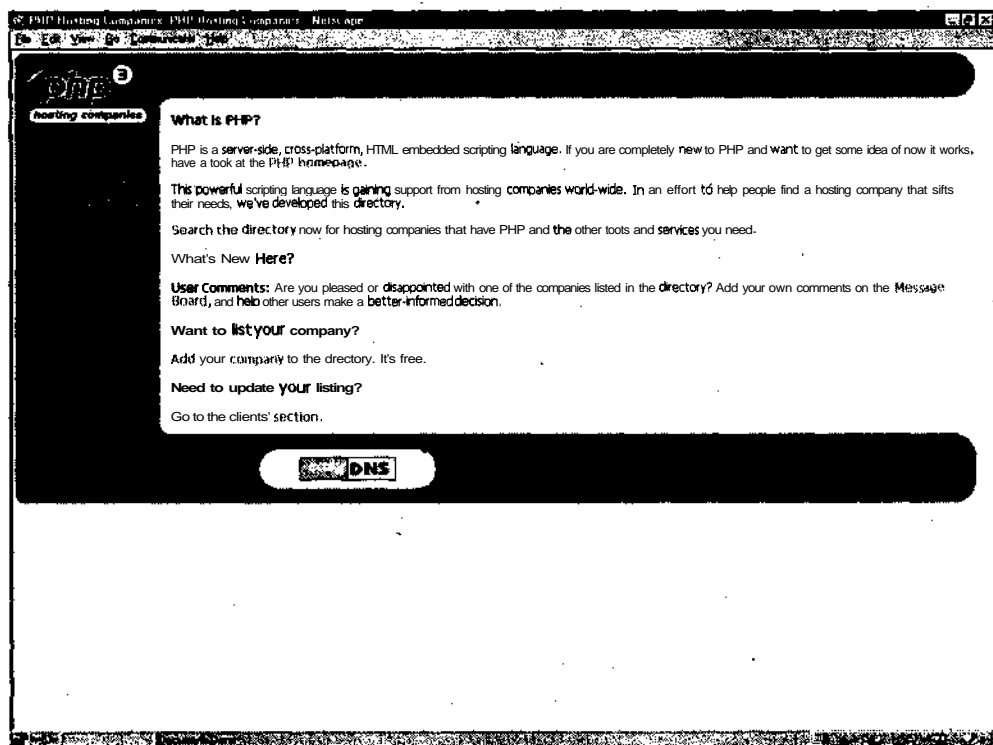


Рис. 5 ▼ На сайте hosts.php.net имеется список провайдеров, предоставляющих возможность выполнения PHP-скриптов на своих серверах

What You See Is What You Get) типа Dreamweaver или GoLive, посмотрите в руководстве этих приложений, как сохранять свои скрипты в чистом текстовом виде.

В-третьих, вам нужен способ передачи написанных сценариев с вашего компьютера на сервер. Если вы работаете на собственном сервере, вы просто сохраняете файлы в определенном каталоге. В противном случае потребуется **FTP-клиент** (File Transfer Protocol) для пересылки файлов на сервер. Другой вариант - использовать сессию Telnet на удаленном сервере и с помощью редакторов типа Vi или Pico писать сценарии прямо на сервере.

Данная книга подразумевает наличие у читателя знания основ HTML. Чем лучше вы знаете, как работать с исходным кодом HTML без помощи таких приложений, как Dreamweaver, GoLive, FrontPage или PageMill, тем легче вам будет освоить язык PHP. Во время изучения PHP каждый программист независимо от своих знаний заглянет в руководство по HTML. Поэтому всегда держите под рукой хорошую книгу по данной теме. Одна из них - «HTML для Всемирной паутины» - написана Элизабет Кастро (Elizabeth Castro) и выпущена издательством Peachpit Press.

При изучении PHP опыт программирования не требуется. Однако, если вы его имеете, это может ускорить усвоение материала, так как вы обнаружите значительное сходство между такими языками, как Perl и PHP или JavaScript и PHP.

Об этой книге

В книге я попытался изложить не только основы программирования на PHP, но и рассказать о **более продвинутых** функциях, которые могут быть вам полезны. Для этого использовался ряд соглашений.

Пошаговые инструкции **покажут**, какой код необходимо добавить к вашим сценариям и где он будет располагаться. Код выделяется другим шрифтом:

```
<?php print ("Hello, World! "); ?>
```

PHP-сценарии будут напечатаны в виде листингов, каждая строка в листинге пронумерована (листинг 1). Не стоит нумеровать строки в сценарии самому, так как в этом случае он станет нерабочим. Как было сказано выше, рекомендуется использовать текстовый редактор, автоматически показывающий номера строк. Это поможет при отладке программы (см. главу 14).

Листинг 1 В хорошем текстовом редакторе номера строк проставляются автоматически.

```
1 <?php  
2 print ("Hello, World!");  
3 ?>
```

Более жирным шрифтом в сценариях будут помечены моменты, которые обсуждаются в тексте или на которые необходимо обратить особое внимание. Таким же образом оформлены фрагменты в окне браузера (рис. 6).

Будет показан также исходный текст HTML (рис. 7), доступный в браузере Netscape Navigator через меню **View** ► **Page Source**, а в Internet Explorer – через меню **Вид** ► **В виде HTML**. Разница между вышеупомянутыми рисунками незначительна, но следует понимать, что на рис. 7 показан текст, который получает браузер, а на рис. 6 представлено, как этот текст интерпретируется. С помощью языка PHP мы будем создавать текст, посылаемый в браузер.

Ширина страницы в обычном текстовом редакторе может быть очень большой, а у книги определенный формат. Поэтому иногда PHP-код придется

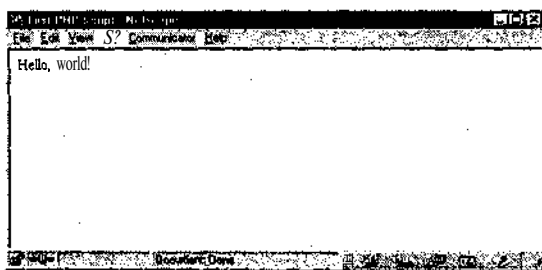


Рис. 6. Такую надпись вы увидите в окне браузера. Для примеров из данной книги не имеет никакого значения, используете ли вы Netscape Navigator, Internet Explorer и т.д. под Macintosh, Windows, Linux или любую другую операционную систему

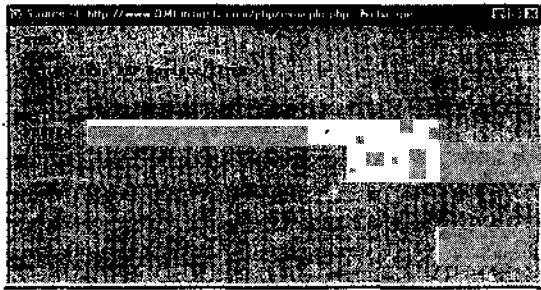


Рис. 7 т Выбрав команду Page Source в меню View в программе Netscape или В виде HTML в меню Вид в Internet Explorer, можно увидеть исходный текст HTML, полученный браузером. В данном случае был получен только текст «Hello,World!»

разбивать на несколько строк. В редакторе же этого делать не надо. Перенос будет обозначен маленькой стрелкой, например:

```
<HTML><HEAD><TITLE>First PHP Script
-></TITLE></HEAD><BODY>
```

В вашем сценарии вы должны писать все в одну строку, иначе при выполнении программы появятся ошибки. (Стрелка не используется в сценариях с пронумерованными строками.)

Демонстрируя новые возможности PHP, я постараюсь объяснить, почему и как они применяются. Надеюсь, что перед использованием конкретной функции вы будете хорошо понимать ее. Если все же что-то останется неясным, обратите внимание на дополнительные источники информации, ссылки на которые даны в конце книги. В указанных изданиях можно найти ответы на интересующие вас вопросы (см. приложение С).

Перед тем как пользователь видит запрошенную Web-страницу, она трижды изменяется. Это связано со спецификой PHP. Сначала код PHP генерирует страницу, затем код посылается в браузер (в основном HTML), и только после этого конечный пользователь видит нужный ресурс. В книге уделено место каждому из этих трех этапов, хотя приоритет и отдается собственно функционированию PHP.



Таким образом в книге оформлены советы, которые призваны облегчить освоение языка PHP и работу с ним.

Сопутствующий Web-сайт

Сайт, посвященный книге, находится по адресу www.DMCinsights.com/php. Там вы найдете все описываемые сценарии и сможете загрузить их на свой компьютер. Однако я бы настоятельно посоветовал вам самостоятельно писать сценарии. Это позволит лучше усвоить структуру и синтаксис языка. На сайте

также имеется большое количество ссылок на Web-страницы, посвященные PHP. Кроме **того**, вы найдете список опечаток, вкравшихся в английское издание книги.

Вопросы, комментарии, предложения

Если у вас имеются конкретные **вопросы** по PHP, можно обратиться с ними на разные сайты, посвященные этому языку. Более подробная информация представлена в приложении С. Вы также можете направить автору свои вопросы, комментарии и предложения по адресу php@DMCinsights.com.

Глава

Первые шаги с PHP

Изучение любого языка программирования всегда должно начинаться с понимания синтаксиса, **ведь** нарушение правил синтаксиса является распространенной причиной возникновения ошибок в коде. В связи с этим главное **внимание** в данной главе уделено основам языка, также сюда включены рекомендации, которые помогут избежать ошибок в будущем.

К концу **главы** мы успешно напишем и выполним наши первые сценарии на языке PHP.

Основы синтаксиса

Разработаем нашу первую страницу на языке PHP точно так же, как начали бы с нуля документ HTML.

Между стандартными **HTML**- и PHP-документами есть два основных различия. Во-первых, файлы PHP-сценария сохраняются с расширением **.php** (например, index.php). А во-вторых, PHP-код заключается в *тэт* **<?PHP** и **?>** для отделения кода PHP от HTML.

Тэги PHP и код HTML в первом сценарии

1. Откройте текстовый редактор SimpleText, WordPad или любой другой.
2. Выберите команду **File** ► **New** для создания нового пустого документа.
3. Напечатайте такую строку:

```
<HTML><HEAD><TITLE>First PHP Script</TITLE></HEAD><BODY>
```

Для большей наглядности можно расположить каждый элемент или группу элементов на отдельной строке.

4. На новой строке **наберите** **<?PHP**.

5. Нажмите клавишу Enter для создания новой строки и наберите символы `>>`.
6. Напечатайте `</BODY></HTML>`.
7. Выберите команду File ► Save As. В появившемся диалоговом окне выберите формат Text Only (или ASCII).
8. Определите место для сохранения сценария.
9. Сохраните сценарий как `first.php` (листинг 1.1).

Листинг 1.1 ▼ Основная структура HTML-документа с тэгами PHP. Все PHP-сценарии должны быть выделены специальными тэгами. Тогда сервер сможет обрабатывать то, что нужно, как PHP-код. Внутри PHP-тэгов все интерпретируется как сценарий PHP, а прочая информация посылается в браузер как стандартный код HTML.

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. ?>
8. </BODY>
9. </HTML>
```

Узнайте у вашего провайдера, какие расширения можно использовать для PHP-документов. Мы применяем расширение `.php`, хотя вместо этого вы можете использовать `.phtml`. На серверах с третьей версией PHP по умолчанию используется расширение `.php3`. Расширение файла дает серверу указание, как интерпретировать файл: `file.php` обрабатывается модулем PHP, `file.asp` будет обработан как ASP, а `file.html` является статическим HTML-документом.

Следует также узнать у провайдера, можно ли использовать короткие тэги (`<? и ?>` вместо `<?PHP и ?>`) или ASP-тэги (`<% и %>`). Такие программы, как Macromedia Dreamweaver, лучше работают с PHP-страницами, если используются тэги ASP.

Передача данных в браузер

Теперь, когда вы создали свой первый PHP-сценарий, самое время попробовать с ним что-нибудь сделать. Как упоминалось в предисловии, PHP «говорит» серверу, какие данные посылать в браузер. Для начала мы используем функцию `phpinfo()` для печати служебной информации. При вызове данная функция pošлет в Web-браузер таблицу с полным перечнем характеристик самого сервера и установленного на этом сервере модуля PHP.

Добавление функции `phpinfo`

1. Откройте в текстовом редакторе сценарий `first.php`.
2. Установите курсор между PHP-тэгами (`<?PHP и ?>`) и создайте новую строку, нажав клавишу Enter.

3. На новой строке напечатайте `phpinfo()` ;.
4. Сделайте иным название страницы, заменив **First** на **Test** в третьей строке HTML (листинг 1.2).
5. Сохраните сценарий как `test.php`.

Листинг 1.2 ▼ Так как этот файл сохраняется отдельно, мы изменили титульную строку HTML при добавлении функции `phpinfo()`.

```
1. <HTML>
2. <HEAD> .
3. <TITLE>Test PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. phpinfo();
8. ?>
9. </BODY>
10. </HTML>
```

Каждая инструкция PHP-кода должна заканчиваться знаком «точка с запятой» (;). Пропуск этого символа - самая распространенная ошибка. Вы можете размещать несколько инструкций на одной строке, отделяя их друг от друга этим знаком. Однако для ясности программ я бы не рекомендовал этого делать.

Инструкция в PHP - это исполняемая строка кода, такая как `print()` или `phpinfo()`. Точка с запятой в конце строк означает указание выполнить команду. И наоборот, строки комментариев, **PHP-тэги**, управляющие структуры (условные операторы, циклы и т.п.) и некоторые другие конструктивные элементы, обсуждаемые далее, не требуют использования данного знака.

Каждый из компонентов нужен, чтобы указывать обстоятельства выполнения инструкций. Тэг PHP указывает только то, что начинается PHP-код; символы комментариев поясняют текст в программе и т.п. Таким образом, точка с запятой завершает конкретное действие и не требуется для конструктивных элементов, которые создают условия.



Хорошо это или нет, PHP достаточно либерален в отношении использования разных регистров во встроенных функциях, таких как `PHPINFO()`. Конечный результат функций `PHPinfo()` и `PHPINFO()` будет одним и тем же. Во второй главе приведены такие примеры, в которых регистр играет важную роль. Кстати, в языке HTML регистр букв не имеет никакого значения.



`Phpinfo()` - пример стандартной встроенной функции PHP. Более подробная информация о функциях и их создании содержится в главе 9.



Очень удобно всегда держать под рукой копию файла `test.php`. Его можно использовать для проверки возможностей PHP на новом сервере или для того, чтобы узнать, какая дополнительная функциональность (базы данных, работа с GIF-изображениями и т.д.) поддерживается. Файл `test.php` можно использовать и для

экспериментов с различными расширениями файлов. Проведя несколько таких опытов, вы узнаете, какие файлы сервер будет обрабатывать правильно, а какие - нет.

Тестирование сценария

В противоположность Коду HTML, который можно протестировать на своем компьютере с помощью Web-браузера, результаты PHP-сценария удастся посмотреть только после сохранения сценария на сервере, поддерживающем технологию PHP. Если вы работаете прямо на сервере, надо лишь позаботиться о сохранении сценариев в нужном каталоге. Если же вы создаете сценарий в текстовом редакторе на домашнем компьютере, для пересылки его на сервер вам потребуется клиент FTP (протокол передачи файлов). Провайдер Web-хостинга также должен **обеспечить** вам доступ к **FTP-серверу**. Потребуется установить на компьютере клиентское приложение FTP, такое как Fetch для Macintosh или WS_FTP для Windows.

Загрузка сценария на сервер с помощью FTP

1. Запустите программу **FTP-клиента**.
2. Установите соединение с сервером, введя его адрес, имя пользователя и пароль, предоставленные вам провайдером (рис. 1.1).
3. Найдите каталог для **HTML-страниц** (обычно это `www/` или `htdocs/`).
4. Сохраните сценарий (`test.php`) на сервере. (Как правило, большинство FTP-приложений сохраняют переданные на сервер страницы под тем именем, которое вы использовали на своем **компьютере**). Если вы пользуетесь программой, позволяющей **указывать** имя файла, назовите его `test.php`.

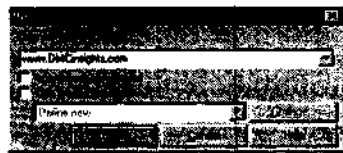


Рис. 1.1 ▼ Введите имя пользователя и пароль вашего провайдера. Если вам известен каталог, в котором следует сохранить файлы, его название также можно ввести в этом окне

Тестирование сценария в браузере

1. Откройте браузер.
2. Введите адрес сайта, на котором вы сохранили сценарий. (В моем случае это <http://www.DMCinsights.com/php>.)
3. Добавьте к адресу запись `/test.php`.
4. Нажмите клавишу **Enter**. Страница должна загрузиться в окне браузера (рис. 1.2).

Функция `phpinfo()` выводит на экран системную информацию модуля PHP, установленного на сервере. Полезно использовать эту функцию после

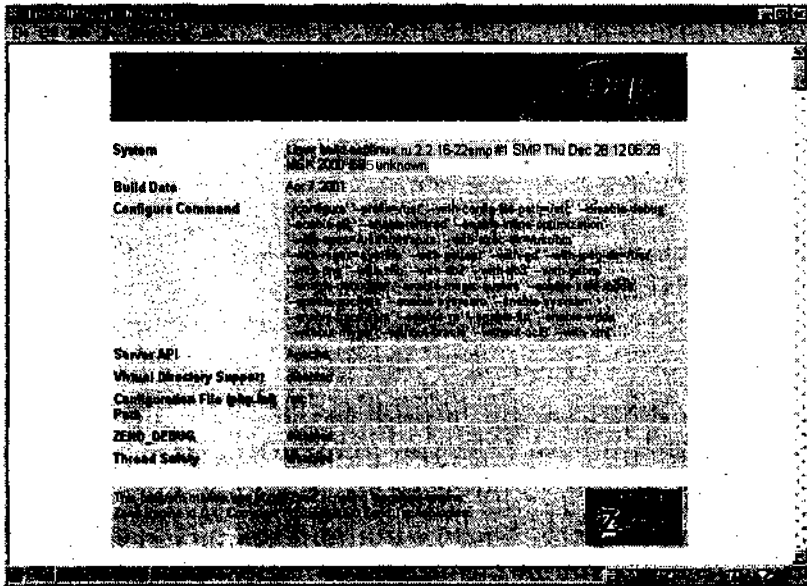


Рис. 1.2 Если вы увидите текст `phpinfo()`, то либо модуль PHP установлен некорректно, либо используемое расширение (в данном случае `.php`) не интерпретируется как PHP-файл

установки новой версии PHP, чтобы определить, какие расширения можно использовать и какие функции PHP поддерживаются.

Некоторые текстовые редакторы, такие как BBEdit, имеют встроенную функциональность FTP, позволяющую сохранять сценарии прямо на сервере.

Передача простого текста в браузер

Если бы PHP использовался только для просмотра конфигурации PHP на сервере, от него было бы мало толку. В основном этот язык применяется для **отправки** информации в браузер в виде обычного текста и **HTML-тэгов**. Для этого используется функция `print()`.

Печать простого сообщения

1. Откройте файл `first.php` в текстовом редакторе.
2. Установите курсор между **PHP-тэгами** и создайте новую строку, нажав клавишу **Enter**.
3. Наберите `print ("Hello, world!");` (листинг 1.3).
4. Сохраните сценарий.
5. Загрузите сценарий на сервер и проверьте результат в браузере (рис. 1.3).

Листинг 1.3 ▼ Вставив инструкцию `print` между **PHP-тегами**, мы даем команду серверу послать приветствие «Hello, world!» в браузер. Это аналогично тому, что мы ввели данный текст в HTML-код.

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. print ("Hello, world! ");
8. ?>
9. </BODY>
10. </HTML>
```

Печать фразы «Hello, world!» - первый шаг, которому учат во многих учебниках по программированию. Банально использовать для этого PHP, но я подчиняюсь традиции в демонстрационных целях.

Для отправки **текста** в браузер, включая `echo()` и `printf()`. Функция `echo` фактически является синонимом `print`, поэтому мы не будем рассматривать ее более подробно. О функции `printf()` говорится в главе 13.

Скобки в описании некоторых функций можно опускать, но кавычки необходимы всегда, например `print "Hello, world!";`. Хотя в книге для выделения аргументов функций мы используем скобки, многие программисты не делают этого. Я бы посоветовал вам определиться с этим вопросом и в дальнейшем придерживаться принятого решения.

Пропуск открывающих Или закрывающих **кавычек**, или скобок, или точки с запятой после каждой инструкции - распространенная причина возникновения ошибок при использовании функции `print()`. Если при исполнении сценария возникают ошибки, прежде всего проверьте эти знаки.

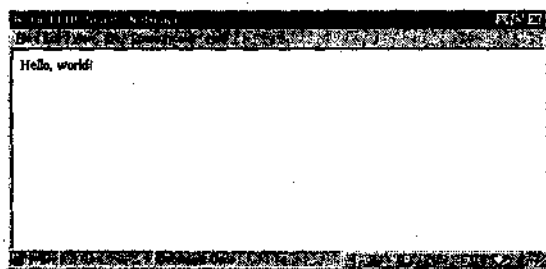


Рис. 1.3 ▼ Так будет выглядеть окно браузера, если сценарий выполнен правильно (не захватывающе, но работает)

Передача страницы HTML в браузер

Те, кто начинает изучать HTML, быстро понимают, что вид простого текста в Web-браузере оставляет желать лучшего. Действительно, язык HTML был создан для разметки простого текста. Так как HTML работает на основе добавления тэгов к тексту, мы можем использовать PHP для отправки HTML-тэгов в браузер вместе с другими нашими данными.

Передача страницы HTML в браузер с помощью PHP

1. Откройте сценарий `first.php` в текстовом редакторе.
2. Отредактируйте текст «Hello, world!» в строке 7, добавив тэги для выделения текста полужирным шрифтом и выровняв текст по центру

```
print ("<B><CENTER>Hello, world!</CENTER></B>");
```
3. Загрузите сценарий (листинг 1.4) на сервер, перезагрузите страницу в браузере (рис. 1.4).

Листинг 1.4 т С помощью функции `print` HTML-тэги можно вместе с текстом послать в браузер, где и произойдет форматирование.

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. print ("<B><CENTER>Hello, world!</CENTER></B>");
8. ?>
9. </BODY>
10. </HTML>
```



Рис. 1.4 т Более привлекательная версия нашего сценария. Любой тэг HTML может быть отправлен в браузер из PHP - просто не забывайте про условные обозначения HTML (закрывающий тэг, например)



HTML-тэги, требующие **кавычек** (например, ``), могут вызвать проблемы при печати из PHP, поскольку функция `print()` также использует кавычки. В таких случаях необходимо использовать обратный слеш (`\`). В нашем примере инструкция будет выглядеть следующим образом: `print "";`. После этого PHP напечатает кавычки, не интерпретируя их как начало или конец самой строки. В книге дано множество примеров подобного экранирования, представлены некоторые другие специальные символы.

Использование пробельных символов в PHP и HTML

Программисты, создающие HTML-код вручную, хорошо понимают, что использование промежутков (пробелов, пустых строк, табуляции и прочих пробельных символов) в коде помогает избежать ненужного загромождения при написании программы, но не влияет на то, что видит пользователь в окне браузера. Вставляя пустые строки между фрагментами кода, отделяя вложенные элементы табуляцией и пробелами, мы делаем сценарий более читаемым. Это облегчает программирование и последующую отладку сценария. Таким образом, разумное использование пробельных символов всячески одобряется и может применяться как в PHP, так и в полученном HTML-коде.

Вы помните, что в книге рассматриваются все три этапа, которые проходит каждое Web-приложение. Вначале это **PHP-сценарий**. Затем данные, посылаемые после выполнения PHP-инструкций в браузер (в основном **HTML**). Наконец, интерпретация и изображение этих данных в браузере клиента. Коротко остановимся на значении пробельных символов на каждом этапе.

При написании PHP-кода необходимо понимать, что промежутки обычно (но не всегда) игнорируются. Любая пустая строка или несколько таких строк подряд абсолютно не влияют на конечный результат. Табуляция и пробелы также обычно несущественны для PHP.

Код HTML без PHP (листинг 1.4, строки 1-5) может быть размещен обычным образом, как если бы вы размечали обычную **HTML-страницу**. Чтобы получить такую же разметку из PHP (листинг 1.4, строка 7), нужно явно использовать необходимые HTML-тэги.

Разбиение на строки PHP-кода и данных, посылаемых в браузер

1. Откройте файл `first.php` в текстовом редакторе.
2. Вставьте новые строки до и после команды печати с помощью клавиши **Enter**.

Новые строки служат только для придания сценарию более структурированной и ясной формы.

3. В конце команды печати (строка 8) перед кавычками добавьте символ `\n` (листинг 1.5).

4. Сохраните сценарий, загрузите его на сервер и просмотрите с помощью браузера (рис. 1.5).

Листинг 1.5 Добавление пустых строк не влияет на вид страницы в браузере, но делает код более читабельным. Для каждого символа \n, вставленного в инструкцию печати, в HTML-коде появится новая строка (не путать с HTML-тэгом
, который вставляет новую строку в изображение страницы в браузере).

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7.
8. print ("<B><CENTER>Hello, world!</CENTER></B>\n");
9.
10. ?>
11. </BODY>
12. </HTML>
```

Символ \n посылает в браузер команду начать новую строку в HTML-коде. Можно считать, что это эквивалентно нажатию клавиши **Enter**.

Вышеописанные шаги могут сделать код PHP и HTML более читабельным, а использование промежутков не повлияет на вид страницы в браузере (рис. 1.5). Для этого необходимо использовать **HTML-тэги** (код для создания неразрывного пробела в браузере - , эквивалент нажатию клавиши **Enter** в HTML -
).



Дополнительные пробелы имеют значение Только в инструкции печати. В результате они попадают в HTML-код, однако затем обычно игнорируются браузером.



Для просмотра кода, посланного в браузер, используйте команду **View>Source** или **View>Page>Source**. Вы быстро увидите разницу между использованием



Рис. 1.5 Последовательное добавление новой пустой строки страницы в браузере выглядит, как прежде, так как символ \n – это не HTML-тэг, а пустые строки в PHP-коде (строки 7 и 9) не пересылаются в браузер

и неиспользованием новой строки (рис. 1.6 и 1.7). Достаточно сложно оценить преимущества применения пробелов в сценарии из двенадцати строк. Но по мере увеличения и усложнения сценариев значение промежутков будет ясно видно.

Существует мнение, что код HTML нужно сжимать как можно плотнее, избегая любых лишних промежутков. Считается, что это увеличивает скорость загрузки страницы, так как ее пустые места не передаются. Хотя идея и заслуживает внимания, она не очень применима на практике и для обучения.

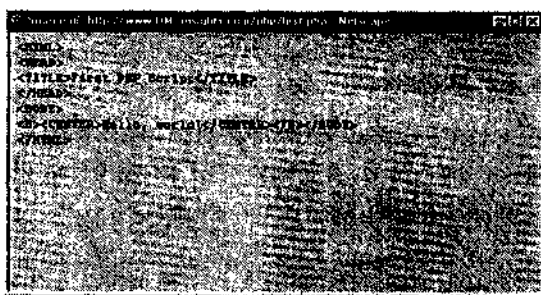


Рис. 1.6 т Просмотр исходного текста Web-страницы - хороший способ определить, где могут возникать проблемы форматирования. Это код нашего сценария до того, как мы добавили символ \п и пустые строки

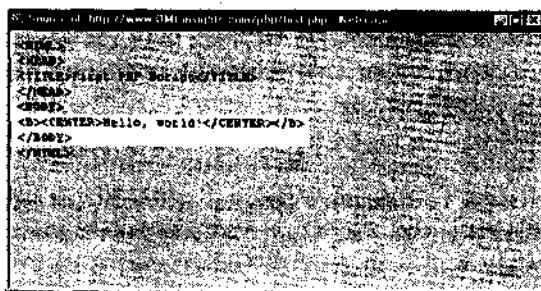


Рис. 1.7 ▼ Введя символ \п в инструкцию печати, мы поделили строку с текстом «Hello, world!» от других HTML-тэгов. При отправке более сложного HTML-кода в браузер использование новых строк помогает улучшить вид исходного текста

Добавление комментариев в сценарий

Каждый программист **понимает**, что делать заметки для себя - это спасательное средство, когда вы возвращаетесь к проекту для изменения, копирования или отладки фрагментов кода после большого перерыва. Использование заметок, или *комментариев*, помогает вспомнить, что вы думали в тот момент. Это не всегда просто сделать несколько месяцев спустя. При обработке сценария компьютер игнорирует комментарии, а PHP поддерживает три метода их создания.

Есть два способа закомментировать строку кода, поставив символы `//` или `#` в самое начало строки. Их можно также использовать для вставки комментария после строки PHP:

```
Print ("Hello."); // Просто приветствие.
```

Комментирование строки кода

1. Откройте сценарий `first.php` в текстовом редакторе.
2. Перед командой печати на строке 8 введите символ `//` или `#` (листинг 1.6).
3. Сохраните сценарий, загрузите его на сервер и просмотрите страницу с помощью браузера (рис. 1.8).

Листинг 1.6 т Использование символа `//` или `#` перед строкой кода означает, что эта строка закомментирована и не будет обработана сервером.

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7.
8. // print ("<B><CENTER>Hello,world!</CENTER></B>\n");
9.
10. ?>
11. </BODY>
12. </HTML>
```

Если все работает правильно, вы ничего не увидите в браузере. Не беспокойтесь, это не ошибка! Браузер не напечатал фразу «Hello, world!», так как не было команды на это действие (мы ведь закомментировали инструкцию с помощью символа `//` или `#`).

Используя обозначения `/*` до, а `*/` после сегмента кода, вы дадите команду серверу проигнорировать все, что попало между этими скобками, от слова до нескольких строк.

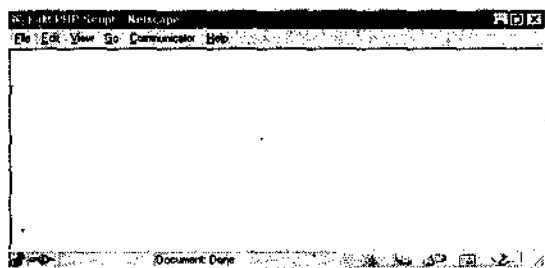


Рис. 1.8 т С закомментированной командой печати страница выглядит так, будто бы команды печати вообще не существует. Так как сервер не обрабатывает команду печати, браузер не получает текст «Hello, world!»

Комментирование нескольких строк кода

1. Откройте сценарий `first.php` в текстовом редакторе.
2. Удалите операторы `#` и `//` перед командой `print()`.
3. В любом месте перед командой `print()` на строке 8, но после открывающего PHP-тэга (`<?PHP`) на строке 6 наберите символ `/*`.
4. В любом месте после команды `print()` на строке 8 (после точки с запятой), но до закрывающего PHP-тэга (`?>`) на строке 10 введите символ `*/` (листинг 1.7).
5. Сохраните сценарий, загрузите его на сервер и просмотрите страницу с помощью браузера (рис. 1.9).

Листинг 1.7 Может показаться избыточным использование операторов `/* */` для комментирования одной строки кода, хотя в этом нет ничего страшного, а полученный результат тот же, что и при работе с операторами `#` или `//` (рис. 1.8 и 1.9).

```

1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. /*
8. print ("<B><CENTER>Hello, world!</CENTER></B>\n");
9. */
10. ?>
11. </BODY>
12. <HTML>

```



Посредством операторов `/*` и `*/` можно закомментировать как одну строку (в нашем примере), так и несколько, а операторами `//` и `#` - только одну строку.



Программисты комментируют код по-разному. Выберите понравившийся вам метод и придерживайтесь его в своей работе. Тот, кто программирует на JavaScript, по всей видимости, будет пользоваться операторами `//` и `/**/`. Программистам на Perl более известен символ `#`.



Рис. 1.9 ▼ Неважно, какие операторы используются для комментирования сценария, главное - использовать их правильно: `//` и `#` - для одной строки, а `/*` и `*/` - для любого количества строк. (Обещаю, это последний рисунок, на котором абсолютно ничего не изображено!)



Обратите внимание на то, что для комментариев кода в PHP нельзя использовать символы комментариев HTML (`<!--` и `-->`). PHP можно использовать для печати этих элементов на странице, в этом случае комментарий появится в исходном тексте HTML на клиентском компьютере (но не в окне браузера). Комментарии PHP никогда не видны на мониторе пользователя.



Так как закоментированный в PHP текст не пересылается в браузер (рис. 1.10), это хорошее место для хранения замечаний, видимых только программисту.



В хороших текстовых редакторах, таких как BBEdit, закоментированный код выделяется цветом (рис. 1.11). Очень полезная функциональность при работе с большими сценариями.

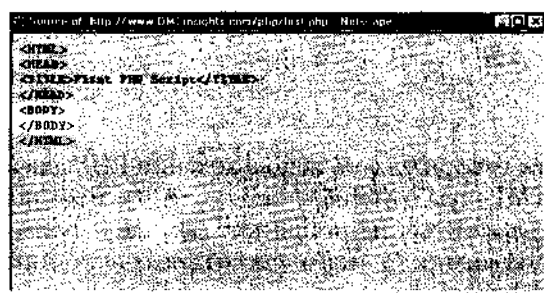


Рис. 1.10 ▼ Сравните этот код с кодом до того, как мы закоментировали команду печати (рис. 1.7), и вы увидите, что PHP-код не был передан в браузер. Закомментированный с помощью операторов `<!--` и `-->` HTML-код все же появляется в браузере, но не показывается им

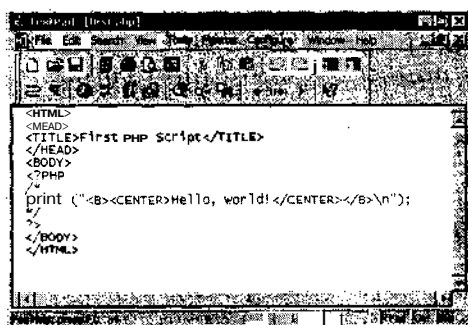


Рис. 1.11 ▼ Если текстовый редактор позволяет выделять логические блоки кода разным цветом, это значительно облегчает программирование. Вы видите, что закоментированный код помечен другим цветом, следовательно, этот конкретный код является неактивным

Глава 2

Переменные

В первой главе говорилось об использовании PHP для передачи простого текста и HTML-кода в Web-браузер, хотя это можно сделать и без PHP. Не стоит беспокоиться: книга научит вас, как использовать инструкцию `print ()` и другие возможности языка для создания действительно интересных вещей на вашем Web-сайте.

Для перехода от простых статических страниц к созданию динамических Web-приложений и интерактивных Web-сайтов вам необходимо научиться манипулировать данными. Для этих целей используют переменные. Это неотъемлемый инструмент языка PHP, а также JavaScript, Java, Perl и любого другого языка программирования.

Переменные позволяют временно хранить данные и манипулировать ими. Они наделяют любой **язык** программирования его истинной силой. Понимание того, что есть переменная, знание типов поддерживаемых в языке переменных и умение их использовать необходимы для работы. В этой главе анализируется само понятие «переменные, используемые в PHP», а в главах 4-6 говорится о том, что именно можно делать с различными типами переменных.

Что такое переменная

Переменная - это своего рода контейнер для данных. Как только данные сохранены в переменной (иначе говоря, как только переменной присвоено значение), они могут быть изменены, напечатаны в Web-браузере, сохранены в базе данных, посланы по электронной почте и т.п. (Под словом «напечатаны» имеется в виду то, что данные отправлены в Web-браузер, но это выполняется именно инструкцией `print`, поэтому приемлемы оба термина.)

Переменные гибки: можно поместить данные в этот «контейнер», извлечь их оттуда (что не влияет на значение самой переменной), поместить туда новые

данные, и этот цикл может продолжаться **столько**, сколько необходимо. При этом переменные в PHP - объекты непостоянные, то есть они существуют, или имеют значение, только внутри сценария. Как только вы переходите на новую страницу, переменные перестают существовать, если только вы специально не передали их на этот ресурс (подробнее см. в следующей главе).

Синтаксис переменных

В PHP все переменные обозначаются следующим образом: знак доллара (\$), за ним имя переменной. Имя должно начинаться либо с буквы (A-Z, a-z), либо с символа подчеркивания (_). Состоять оно может из любого количества букв, цифр, знаков подчеркивания или их комбинации. В имени переменной нельзя использовать пробелы. Вместо них для разделения слов обычно используется символ подчеркивания.

Необходимо помнить, что для имен имеет значение регистр. Поэтому \$variable и \$Variable - две разных переменных, хотя и не стоит создавать переменные с такими похожими именами. Необходимо сразу научиться давать переменным содержательные имена, а также использовать комментарии для указания их назначения (листинг 2.1). Таким образом вы сократите количество ошибок и сможете легко разобраться в проделанной когда-то работе. Например, \$FirstName - более осмысленное имя, чем \$FN.

Листинг 2.1 Всегда лучше включить в код программы слишком много комментариев, чем чересчур мало, ведь часто вполне очевидные при написании моменты становятся непонятными, если вернуться к ним несколько месяцев спустя.

```
1. <HTML>
2. <HEAD>
3. <TITLE>First PHP Script</TITLE>
4. </HEAD>
5. <BODY>
6. <?PHP
7. /* Эта страница печатает простое приветствие. */
8. print ("Hello, world!")
9. ?>
10. </BODY>
11. </HTML>
```

Если добавить комментариев о назначении переменной, то ваша программа станет более понятной. Вы можете посчитать, что \$first_name - более подходящее имя переменной, чем \$FirstName, так как в нем нет заглавных букв, а слова разделены для ясности. Не имеет значения, как вы будете называть переменные, важно быть последовательным в присвоении имен. Это поможет избежать элементарных ошибок при программировании.

В отличие от некоторых других языков в PHP нет необходимости объявлять переменную, то есть присваивать ей тип (об этом см. в разделе «Типы переменных»), и/или присваивать ей начальное значение, то есть инициализировать. В языке PHP переменная определяется сразу, как только вы используете ее в первый раз.

Типы переменных

В этой книге рассматриваются три самых распространенных типа переменных: *числа*, *строки* и *массивы*.

В языке РНР числа подразделяются на два вида: целые и с плавающей запятой (с плавающей запятой двойной точности, если совсем строго). Для наших целей мы будем называть оба вида просто числами. Так как в РНР нет строгого контроля типов переменных, объединение этих двух категорий в одну группу не повлияет на программирование.

В РНР используется еще один **тип** переменных - объекты, но это сложный вопрос, рассматривающийся в приложении С. Как только вы немного освоите описываемый язык, то поймете, **что** изучение объектов значительно облегчит написание кода, так как объектно-ориентированное программирование мощное средство, экономящее время.

Числа

Для простоты обучения мы объединили два вида *чисел* - целые и с плавающей запятой - в одну группу. *Целые числа* могут быть положительными или отрицательными, но не дробными. Числа с десятичным знаком (даже такие как 1.0) - это числа с *плавающей запятой*. Числа с *плавающей* запятой используются для дробных чисел, так как в РНР единственный способ выразить дробь - конвертировать ее в десятичный эквивалент. Так, число «1 1/4» будет записано как «1.25».

Примеры целых чисел:

1

1972

-1

Примеры чисел с плавающей точкой:

1.0

19.72

-1.0

Так как оба типа называются у нас числами, любая из вышеприведенных записей будет правильной.

Примеры неправильного обозначения чисел:

1 1/4

1972a

02.23.72

Дробь содержит два не используемых в числах знака: пробел и слеш (/). Второе значение неверно, потому что включает цифры и буквы. Это приемлемо для имени переменной, но не для ее значения. Третий пример неверен, так как в нем используются два десятичных знака. Если по каким-либо причинам (но не для вычислений) необходимо использовать одно из этих значений, можно обозначить его как строку.

Строки

Переменная является *строкой* (string), если состоит из знаков (некоторая комбинация букв, **цифр**, символов и пробелов), заключенных в одинарные или

двойные кавычки. Строки могут содержать любую комбинацию символов, включая имена других переменных.

Примеры верных значений строк:

```
"Hello, world!"
"Hello, $FirstName!"
"1 1/4"
"Hello/ world! How are you today?"
"02.23.72"
"1972"
```

Обратите внимание на то, что в **последнем** примере мы превратили целое число в переменную, взяв его в кавычки. На самом деле строка содержит символы «1972», а число равно 1972. Язык PHP позволяет использовать такие **числовые** строки непосредственно в **математических** выражениях.

Примеры неверных значений строк:

```
Hello, world!
"I said, "How are you?""
```

Первый пример неверен, так как текст не взят в кавычки. Второй пример достаточно коварен. Если присвоить строке такое значение, то возникнут проблемы, ведь при прочтении вторых кавычек PHP «предположит», что строка закончена, и последующий текст вызовет ошибку.

Вы можете поинтересоваться, как использовать кавычки внутри строки? Как было описано в главе 1, при использовании функции `print()` для создания кода HTML можно *экранировать* кавычки, поставив перед ними обратный слеш. Изменив строку на `"I said, \"How are you?\""`, вы дали команду PHP включить эти кавычки как часть значения строки, а не рассматривать их как индикаторы начала и конца строки. Поэтому, хотя и любая комбинация символов может быть включена в строку, специальные символы должны быть экранированы для **корректного** выполнения строки. Вместе с двойными кавычками следует также **экранировать** апостроф или одинарные кавычки (`'`), обратный слеш (`\`) и знак доллара (`$`).



Двойные кавычки имеют бесспорное преимущество перед одинарными: значение переменной будет распечатано только при использовании первых. Если вы работаете с одинарными кавычками, то строка `'Hello, $FirstName!'`; вызовет печать `Hello, $FirstName!` вместо, скажем, `Hello, Larry!` (если вы заранее присвоили переменной `$FirstName` значение `Larry`). Если экранировать знак доллара в двойных кавычках (`print "Hello, \$FirstName!";`), то снова будет распечатано имя переменной, а не ее значение (`Hello, $FirstName!`).



В главе 1 мы рассматривали, как создать новую строку с помощью символа `\n`. Хотя обратный слеш используется для экранирования некоторых специальных символов, в сочетании с некоторыми буквами он имеет особое значение. Так, комбинация `"\n"` означает новую строку, `"\r"` - возврат каретки, а `"\t"` - знак табуляции.

Массивы

Более подробно *массивы* будут рассмотрены в главе 6, так как они более сложны, чем числа или строки, но вкратце охарактеризуем их и здесь. Строка и число обычно содержат одно значение, а массив может иметь несколько определенных

для него значений. Разумно представить массив в виде списка значений. Другими словами, в него можно вложить множество строк и/или чисел. Разрешается вкладывать и множество массивов в один!

Стандартный массив PHP, состоящий из строк и чисел, в языке Perl также называется массивом. Создав состоящий из массивов массив, можно создать в PHP эквивалент используемого в Perl «хэша», также называемого ассоциативным или многомерным массивом. В PHP эти два типа - одномерные или многомерные массивы - не различаются.

Присвоение значений переменным

В начале этой главы мы упомянули, что в PHP нет необходимости инициализировать или объявлять переменные. Так как же присваивать им значения в сценариях? Значение переменной присваивается независимо от типа с помощью знака равенства (=). Он называется *оператором присваивания*, так как присваивает значение переменной, стоящей слева от него. В следующих нескольких главах говорится и о других типах операторов.

Например:

```
$number = 1;  
$floating_number = 1.2;  
$string = "Hello, World! ";
```

Тип переменной может изменяться «на лету», так как он не зафиксирован (PHP относится к языкам со слабым контролем типов, как и JavaScript):

```
$variable = 1;  
$variable = "Greetings";
```

Теперь, если распечатать значение переменной, мы получим Greetings.

Присваивание значений массиву будет рассмотрено в главе 6.

Вы можете явно присвоить переменной тип при первом ее использовании (присвоение типа - это то же, что и объявление переменной, когда точно указывается ее тип). Синтаксис для этого следующий:

```
$number = (integer) 5;  
$string = (string) "Hello, World!";
```

Честно говоря, даже после этого можно моментально менять тип переменной. Но это один из вариантов, используя который вы останетесь последовательным, если программируете и на других языках.

Предопределенные переменные

Важность *предопределенных переменных* нельзя недооценивать: во-первых, они иначе используются в программах, во-вторых, вы можете случайно создать переменную с таким же именем, а это, вероятно, приведет к проблемам.

Предопределенные переменные - специальный тип переменных, который используется и программой Web-сервера (например, Apache), и операционной

Глава 3

HTML-формы и PHP

Одна из областей применения переменных - их использование с HTML-формами. На Web-сайтах формы нужны для регистрации пользователей и их входа в систему, для обратной связи, электронной торговли и т.д. Даже на самом простом сайте непременно будут применяться формы HTML.

Часто программисты создают CGI-сценарии на языке Perl для обработки данных из этих форм, но того же результата гораздо проще достичь при помощи PHP. В отличие от CGI-сценариев (там необходимо написать сегмент кода, который будет извлекать информацию, посланную формой) PHP имеет встроенную поддержку для получения данных из HTML-форм. При этом проведение синтаксического анализа не является необходимым.

Данная глава посвящена основам создания HTML-форм и способам передачи данных из экранных форм в PHP-сценарий. Если вы не знаете, как разрабатывать экранные формы, обратитесь для более детального изучения этой темы, очень важной для дизайна Web-сайтов, к HTML-ресурсам.

Создание простой формы

Давайте создадим для HTML-формы страницу, где пользователи будут фиксировать свое имя, фамилию, адрес электронной почты и давать произвольный комментарий. Для этого в экранной форме нужно создать необходимые поля и соответствующие им переменные.

Создание HTML-формы

1. Откройте текстовый редактор и создайте новый документ.

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD><BODY></BODY></HTML>
```

2. В тело программы добавьте открывающий и закрывающий тэги для экранной формы (листинг 3.1).

```
<FORM ACTION="HandleForm.php"> </FORM>.
```

Тэги `<FORM>` задают начало и конец формы. Все элементы формы должны быть размещены между ними. Атрибут `ACTION` сообщает серверу, какая страница (или сценарий) получит данные из формы.

Листинг 3.1 т Каждая HTML-форма начинается и заканчивается тэгами `<FORM>` и `</FORM>`. Не забывайте использовать их. Также помните, что необходимо отправлять форму для обработки в соответствующий сценарий с помощью атрибута `ACTION`.

```
1. <HTML>
2. <HEAD>
3. <TITLE>HTML Form</TITLE>
4. </HEAD>
5. <BODY>
6. <FORM ACTION = "HandleForm.php">
7. </FORM>
8. </BODY>
9. </HTML>
```

3. Сохраните страницу как `form.html`.
4. Вставьте курсор между тэгами `<FORM>` и нажмите клавишу **Enter** для создания новой строки.
5. Теперь начнем добавлять в форму поля:

```
First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20xBR>
```

Будьте последовательны и присвойте каждому полю в форме логическое и наглядное имя. Для имени используются буквы, числа и символ подчеркивания (`_`). При работе внимательно следите за именами полей.

```
Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=40xBR>
```

Добавьте тэги `
`, чтобы форма выглядела более аккуратно в окне браузера.

```
E-mail Address <INPUT TYPE=TEXT NAME="Email" SIZE=60xBR>
```

```
Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40x/TEXTAREAxBR>
```

Текстовая область предоставляет пользователю больше места для ввода комментариев, чем текстовое поле. Однако в последнем можно ввести ограничение на количество вводимых символов, в то время как в текстовой области это сделать невозможно. При создании формы выбирайте тот тип поля, который более соответствует информации, вводимой пользователем.

6. На отдельной строке напечатайте новую строку.

```
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
```

Значение типа `SUBMIT` - это надпись на кнопке в окне браузера. Также можно использовать кнопки **Go!** или **Enter**.

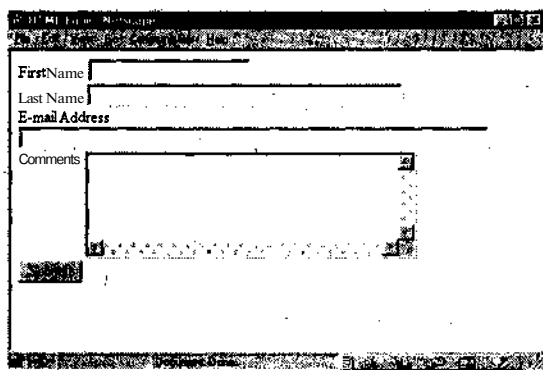


Рис. 3.1 ▼ Так выглядит форма в окне браузера, если все сделано правильно. Не забудьте добавить кнопку **Submit**

7. Сохраните сценарий (листинг 3.2), загрузите его на сервер и просмотрите в окне браузера (рис. 3.1). Так как это обычная страница, а не PHP-сценарий, ее можно увидеть и без сервера, в окне браузера прямо на вашем компьютере.

Листинг 3.2 ▼ Разрешается использовать любую комбинацию полей ввода в форме, главное, чтобы все они были определены внутри тэгов `<FORM>`, иначе не будут работать. Представленная в виде таблицы форма удобна в использовании и выглядит профессионально.

```

1. <HTML>
2. <HEAD>
3. <TITLE>HTML Form</TITLE>
4. </HEAD>
5. <BODY>
6. <FORM ACTION="HandleForm.php">
7.   First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20><BR>
8.   Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=40><BR>
9.   E-mail Address <INPUT TYPE=TEXT NAME="Email" SIZE=60><BR>
10.  Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
11.  <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12. </FORM>
13. </BODY>
14. </HTML>

```



В данном примере мы создали форму, вручную написав HTML-код. Можно делать то же самое в специальных HTML-редакторах, таких как Dreamweaver или GoLive, если вам это больше нравится.



Мы использовали стандартное расширение HTML (.html), так как создавали обычную HTML-страницу. Можно было применить расширение .php и получить тот же результат, хотя сам язык и не был бы использован. Напоминаю, что на PHP-странице все, что не ограничено PHP-тэгами `<?php` и `??`, трактуется как обычный код HTML.

Хотя мы и не рассматривали этот вопрос, я все же порекомендовал бы использовать в формах, особенно сложных, кнопку **Reset**. Создать ее можно следующим образом:

```
<INPUT TYPE=RESET NAME=RESET VALUE="RESET">
```

Убедитесь, что атрибут ACTION правильно указывает на существующий на сервере файл, иначе форма не будет обработана. В нашем случае мы определяем, что файл HandleForm.php находится в одном каталоге со страницей form.html.

Использование методов Get и Post

Опытный читатель наверняка заметил, что мы кое-что пропустили в открывающем тэге <FORM>, а именно атрибут METHOD. Этот атрибут указывает серверу, как передавать данные из формы в обрабатывающий сценарий. Я не упоминал об этом раньше, так как данная тема заслуживает отдельного обсуждения.

Для атрибута METHOD имеется два варианта: GET и POST. Я подозреваю, что многие программисты на HTML не совсем ясно понимают разницу между ними и не всегда знают, когда какой использовать. По сути, между этими вариантами нет большой разницы, особенно если вы начинаете их использовать впервые, так как оба приведут к желаемому результату.

Разница между вариантами GET и POST состоит только в том, как информация передается из формы в обрабатывающий сценарий. Метод GET посылает всю собранную информацию как часть адреса URL. Метод POST передает информацию так, что пользователь этого не видит. Например, при использовании метода GET, после передачи информации URL примет следующий вид: <http://www.DMCinsights.com/php/HandleForm.php?FirstName=Larry&LastName=Ullman>.

При использовании метода POST конечный пользователь увидит только такую запись: <http://www.DMCinsights.com/php/HandleForm.php>.

При выборе метода следует учитывать три фактора:

- метод GET ограничивает объем передаваемой информации;
- метод GET открыто пересылает введенную информацию в обрабатывающий сценарий, что может неблагоприятно сказаться на безопасности. Например, каждый человек, которому виден монитор вашего компьютера, может заметить введенный в форму пароль;
- страницу, сгенерированную формой с помощью метода GET, разрешается помечать закладкой, а сгенерированную методом POST - нет.

Для обработки форм мы будем использовать метод POST. Там, где метод GET предлагает дополнительные возможности вашему Web-сайту, мы обратимся к этому способу (см. раздел «Ввод данных вручную»). Оба метода успешно передают данные из формы, и, принимая окончательное решение о применении

того или другого варианта, вы обязательно должны учитывать три **вышеупомянутых** фактора.

Добавление метода в сценарий

1. Откройте страницу form.html в текстовом редакторе.
2. В открывающий тэг `<FORM>` вставьте `METHOD=POST` (строка 6, листинг 3.3).

Листинг 3.3 т Вам решать, какой метод - GET или POST - использовать. Главное, вообще не забывать обращаться к одному из них. Когда вы наберетесь опыта, то сможете легко определять, в какой ситуации удобней тот или иной метод, но это больше вопрос эстетический, чем функциональный.

```

1. <HTML>
2. <HEAD>
3. <TITLE>HTML Form</TITLE>
4. </HEAD>
5. <BODY>
6. <FORM ACTION="HandleForm.php" METHOD=POST>
7. First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20xBR>
8. Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=40xBR>
9. E-mail Address <INPUT TYPE=TEXT NAME="Email" SIZE=60xBR>
10. Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
11. <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12. </FORM>
13. </BODY>
14. </HTML>

```

3. Сохраните сценарий и загрузите его на сервер.
4. Просмотрите исходный текст страницы и убедитесь, что там есть все необходимые элементы (рис. 3.2).

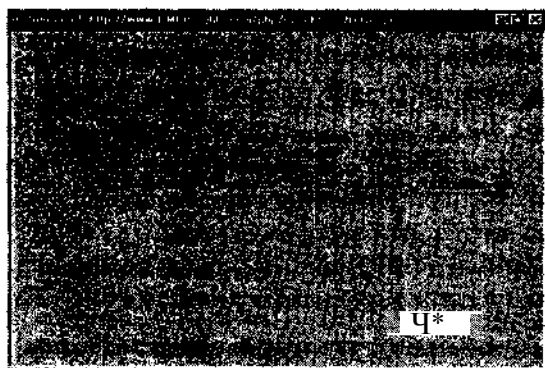


Рис. 3.2 т Просматривая исходный HTML-код в браузере, удастся **увидеть**, что и почему работает или нет. Как и всюду в программировании, **внесение** небольших корректив может многое изменить

Получение данных из формы в PHP

Вы создали форму. Теперь необходимо написать сценарий `HandleForm.php`, который будет получать и обрабатывать данные, введенные пользователем в экранной форме `form.html`. Вот здесь и станет очевидна простота и эффективность языка PHP.

Создание сценария `HandleForm.php`

1. Откройте текстовый редактор и создайте новый документ.

```
<HTML><HEAD><TITLE>Form Results</TITLE></HEAD><BODY>
-<?php /*Эта страница получает и обрабатывает данные, принятые
-от "form.html". */?></BODY></HTML>.
```

2. Это стандартный формат **PHP-страницы**. Мы добавили комментарий о цели создания сценария. Хотя страница `form.html` указывает, куда посылаются данные (через атрибут `ACTION`), мы должны сделать комментарий, указывающий обратное. Создайте новую строку после комментария, но до закрывающего **PHP-тэга**. Наберите следующий текст:

```
print ("Your first name is $FirstName.<BR>\n");
print ("Your last name is $LastName.<BR>\n");
print ("Your E-mail address is $Email.<BR>\n");
print ("This is what you had to say:<BR>\n $Comments<BR>\n");
```

3. Сохраните сценарий в файле с именем `HandleForm.php`.
4. Загрузите сценарий на сервер (листинг 3.4) и убедитесь, что он сохранен в одном каталоге со страницей `form.html`.
5. Протестируйте сценарий в **Web-браузере** (рис. 3.3 и 3.4).

Листинг 3.4 т. Взяв название элемента `NAME = "Name"` в **HTML-форме** и добавив знак `$`, вы получаете переменную, которая содержит значение, введенное пользователем в соответствующем поле формы. Это справедливо для любого типа ввода кода **HTML**, будь то `TEXT`, `TEXTAREA` или выбор из меню `SELECT`, и является одной из причин того, почему язык **PHP** так хорош для обработки **HTML-форм**. Для сравнения: **CGI-сценарии** требуют синтаксического анализа кода.

```
1. <HTML>
2. <HEAD>
3. <TITLE>Form Results</TITLE></HEAD>
4. <BODY>
5. <?php
6. /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7. print ("Your first name is $FirstName.<BR>\n");
8. print ("Your last name is $LastName.<BR>\n");
9. print ("Your E-mail address is $Email.<BR>\n");
10. print ("This is what you had to say:<BR>\n $Comments<BR>\n");
11. ?>
12. </BODY>
13. </HTML>
```

Рис. 3.3 ▼ Все, что пользователь вводит в HTML-форму, будет напечатано в браузере сценарием HandleForm.php (см. рис. 3.4)

Рис. 3.4 т Другой вариант использования описанной в главе инструкции print. Демонстрирует вашу первую динамически сгенерированную Web-страницу. В последующих главах говорится о том, как обрабатывать полученные данные, как отправлять их по электронной почте и сохранять в базе данных

Смысл этого упражнения в том, чтобы показать, как легко можно передать данные из HTML-формы в PHP-страницу. PHP-сценарий сохранит данные в соответствующих переменных, поэтому переменная \$FirstName получит значение, введенное пользователем в поле **First Name** (вы берете имя поля в HTML-форме, добавляете знак \$ и получаете переменную с соответствующим значением). Передача происходит автоматически, и в отличие от CGI в PHP нет необходимости в синтаксическом разборе входного потока.



Еще одно преимущество использования языка PHP для обработки HTML-форм состоит в том, что при передаче автоматически происходит экранирование всех специальных символов. Например, если вы введете комментарий I thought

"form.html" was too simple!, переменная `$Comments` будет равна `I thought \ "form.html\ " was too simple!`, и комментарий можно распечатать без осложнений (рис. 3.5).

Если вы хотите передать в сценарий предварительно заданное значение, используйте тип ввода `HIDDEN` в вашей HTML-форме. Например, строка `<INPUT TYPE = HIDDEN NAME = "ThisPage" VALUE = "form.html">`, вставленная между тэгами `FORM`, создаст в сценарии обработки переменную `$ThisPage` со значением `"form.html"`. Таким же образом, дав команду `print ("<INPUT TYPE=HIDDEN NAME=\ "FirstName\ " VALUE=\ "$FirstName\ ">");`, вы можете «продлить жизнь» переменной `$FirstName`, передавая ее значение в другие формы.

Хотя в PHP для вывода формы на экран и обработки данных из нее можно использовать один файл, мы отложим этот прием, пока не рассмотрим все языковые конструкции PHP. Пока же будем пользоваться отдельным файлом, который назван `HandleForm.php`.

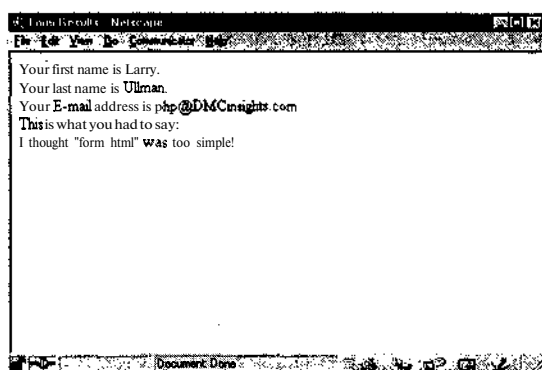


Рис. 3.5 Язык PHP автоматически экранирует специальные символы, введенные в HTML. Это необходимо при отправке данных обратно в браузер и сохранении их в базе данных

Ввод данных вручную

Необязательно передавать данные в сценарий только через HTML-форму, иногда возникает необходимость сделать это вручную. Если вы создали сценарий, печатающий имя пользователя в приветствии, вы можете создать переменную, которая содержит имя отдельно от приветствия. В этом случае легко изменить имя, не меняя приветствия, инструкцию `print` или сам сценарий. Напишем PHP-сценарий, чтобы наглядно показать этот процесс.

Создание PHP-сценария, выводящего на экран приветствие

1. Создайте новый документ в текстовом редакторе.

```
<HTML><HEAD><TITLE>Using Variables</TITLE></HEAD><BODY><?php
/* Эта страница может содержать строки с операторами print. */
```

Я вставил в сценарий два комментария на случай, если вдруг строка приветствия станет частью более крупной и сложной страницы.

```
print ("<H2><CENTER><B>Hello, $FirstName.</B></CENTER></H2><BR>\n");
/* Эта страница может содержать дополнительный код после оператора
print. */
?> </BODY></HTML>
```

2. Сохраните документ (листинг 3.5) в файле с именем `hello.php` и загрузите его на сервер.

Листинг 3.5 ■ Вместо написания страницы, автоматически выдающей приветствие «Hello, World!» или «Hello, Larry!», мы создали динамическую страницу, напечатав значение переменной. Теперь, как только изменится значение, вид результирующей страницы станет иным.

```
1. <HTML>
2. <HEAD>
3. <TITLE>Using Variables</TITLE></HEAD>
4. <BODY>
5. <?php
6. /* Эта страница может содержать строки с операторами print. */
7. print ("<H2><CENTER><B>Hello, $FirstName.</B></CENTER></H2><BR>\n");
8. /* Эта страница может содержать дополнительный код после оператора
   print. */
9. ?>
10. </BODY>
11. </HTML>
```

Если вы просмотрите сценарий через браузер, то увидите на экране только запись "Hello, .", так как переменная `$FirstName` не имеет значения. Есть два способа вручную задать это значение, без применения форм. Первый - использовать знание того, как метод GET передает данные в сценарий.

Использование метода GET без HTML-формы

1. Просмотрите сценарий `hello.php` в Web-браузере, зайдя по соответствующему адресу URL (в нашем случае это <http://www.DMCinsights.com/php/hello.php> (рис. 3.6)).
2. Добавьте в конец URL текст `?FirstName=Larry` (можете указать свое имя, главное, чтобы не было пробелов). Выше в данной главе («Получение данных из формы в PHP»), когда вы посылали переменную в сценарий через URL (то есть использовали метод GET в HTML-форме), мы уже использовали этот формат www.url.com/script.php?variable=value.
3. Перезагрузите страницу в браузере с новым адресом URL (рис. 3.7).

Если вы не видите адрес в окне браузера, значит, была допущена ошибка. Проверьте наличие вопросительного знака "?", отделяющего адрес файла от данных. Потом убедитесь в том, что правильно набрали переменную `FirstName` (учтите, что `firstname` - совсем другая переменная).

Второй способ предварительного присвоения значения переменной - напрямую в сценарии.

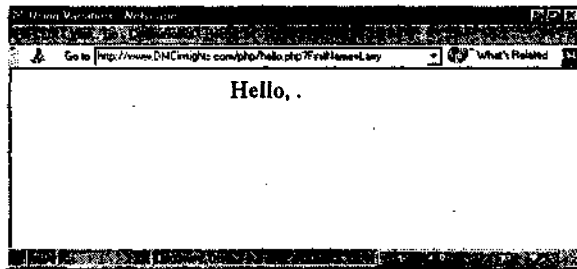


Рис. 3.6 Если переменной `$FirstName` не присвоено значение, в окне браузера видно такое странное сообщение

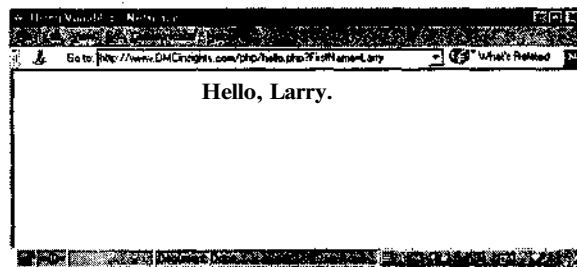


Рис. 3.7 Присвоив переменной `$FirstName` значение `Larry` или другое имя, вы создали динамическую страницу, которая подстраивается под имя пользователя

Присвоение значения переменной

1. Откройте файл `hello.php` в текстовом редакторе.
2. На строке перед инструкцией `print` добавьте запись `$FirstName = "Jude";` (листинг 3.6).

Листинг 3.6 Строка `$FirstName = "Jude";` задает значение "Jude" переменной "FirstName" для этой страницы.

1. `<HTML>`
2. `<HEAD>`
3. `<TITLE>Using Variables</TITLE></HEAD>`
4. `<BODY>`

```

5. <?php
6. $FirstName = "Jude";
7. /* Эта страница может содержать строки с операторами print. */
8. print ("<H2><CENTER><B>Hello, $FirstName</B></CENTER></H2><BR>\n");
9. /* Эта страница может содержать дополнительный код после оператора
   print. */
10. ?>
11. </BODY>
12. </HTML>

```

3. Сохраните страницу, загрузите ее на сервер и просмотрите с помощью браузера (рис. 3.8).
4. А теперь просмотрите эту страницу с помощью измененной версии URL, как это было сделано в последнем примере (<http://www.DMCinsights.com/php/hello.php?FirstName=Larry>) - рис. 3.9.

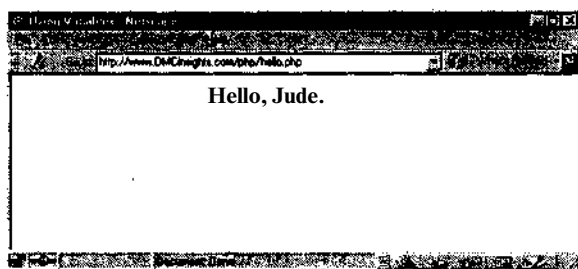


Рис. 3.8 ▼ Можно получить такой же результат, присвоив значение Jude переменной \$FirstName, фактически изменив строку `print` следующим образом: `print ("<H2><CENTER>Hello, Jude.</CENTER></H2>
\n");`. Все же значительно удобнее изменять значения с помощью переменных

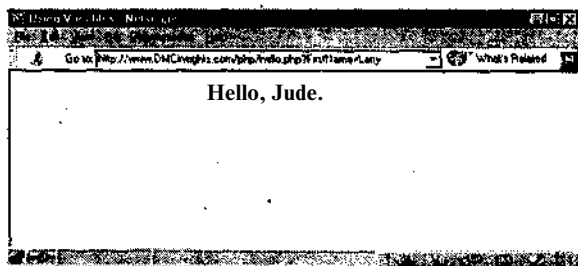


Рис. 3.9 ▼ Так как сама страница содержит строку, присваивающую значение "Jude" переменной \$FirstName, теперь не имеет значения, что вы передаете в сценарий через адрес URL, всегда будет напечатано имя Jude

Становится понятно, что происходит, когда переменной многократно присваиваются значения: только последнее присвоенное значение сохраняется в переменной и выводится на экран в сценарии `hello.php`. Будьте внимательны, чтобы не присвоить значение какой-нибудь переменной многократно!

Способ предварительного присвоения значений переменным становится более значимым по мере расширения и усложнения программы. Использование легко редактируемых переменных означает, что вам никогда не потребуется искать их в многочисленных строках кода для смены значения.

Таким же образом, добавив переменную к URL, можно связать одну страницу с другой динамически генерируемой страницей (такой как `hello.php`). Это можно сделать, просто включив в ссылки на вашей Web-странице значения переменных, где необходимо. Например, формой на Web-сайте может быть взято имя пользователя, которое затем передается на следующие связанные с формой страницы с помощью кода `hello.php`.



Если вы используете метод GET для передачи данных в сценарий, вы можете пересылать многочисленные значения, разделив пары `variable=value` (например, `FirstName=Larry`) с помощью амперсанда (&). Поэтому измененный адрес URL может выглядеть как `hello.php?FirstName=Larry&LastName=Ullman`.



Пробелы внутри значений, передаваемых как часть URL, должны быть заменены на знак плюс (+). В главе 5 обсуждается, как PHP может автоматически подготовить строку текста для передачи как части URL.



Хотя приведенный здесь пример - задание значения имени пользователя - может быть не очень практичным, создание переменной с адресом электронной почты в начале сценария позволит вам легко изменить адрес без необходимости искать его среди многочисленных строк кода. В главе 11 мы постоянно будем устанавливать параметры доступа в базу данных через переменные в начале сценариев, чтобы их можно было легко менять без необходимости вносить изменения во всех местах, где используются эти значения.

Глава

Использование чисел

Во второй главе мы обсудили различные типы переменных, способы присваивания им значений и области их применения. Данная глава посвящена числовым переменным - целым и с плавающей запятой. В последнем разделе главы будет рассмотрено несколько полезных стандартных функций для работы с числами.

Сложение, вычитание, умножение и деление

Со школы всем известно, что в основе арифметики лежат четыре операции: сложение, вычитание, умножение и деление. Мы используем эти действия в PHP-сценарии, созданном для расчета общей стоимости покупок. Сценарий может быть применен как основа прикладной программы shopping cart («корзина покупателя»).

Создание калькулятора стоимости покупок

1. Откройте текстовый редактор и создайте новый документ (листинг 4.1).

```
<HTML><HEAD><TITLE>Using Numbers</TITLE></HEAD><BODY><?php?>
-></BODY></HTML>
```

2. Между PHP-тэгами вставьте такую запись:

```
$Cost = 2000.00; $Tax = 0.06;
```

Мы вручную задаем стоимость товара, равную \$2,000.00. Запомним, в значении переменной не используется ни знак доллара, ни запятая. Также вручную задается ставка налога с продаж (6%). В коде мы используем десятичную дробь 0.06, чтобы пока не отвлекаться на операции с процентами. Оба числа - с плавающей запятой.

```
$TotalCost = $Cost * $Quantity;
```

Умножение в языке PHP обозначается звездочкой (*). Значение переменной `$Quantity` может передаваться в сценарий из экранной формы, как это и происходит в приложении shopping cart на Web-сайте. Вы можете использовать способы, продемонстрированные в главе 3, и создать такую форму, но здесь мы просто добавим это значение к указателю URL.

Листинг 4.1. Хотя вычисления довольно простые, не стесняйтесь добавлять комментарии, которые вы считаете необходимыми для описания этого процесса. Если вы собираетесь усовершенствовать свои навыки работы с HTML, создайте форму, которая берет информацию у пользователя (включая количество и скидку) и передает ее в сценарий.

```
1  <HTML>
2  <HEAD>
3  <TITLE>Using Numbers</TITLE>
4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу.
   $Discount необязательна. */
8  $Cost = 2000.00;
9  $Tax = 0.06;
10 $TotalCost = $Cost * $Quantity;
11 $Tax = $Tax + 1; // Налог ($Tax) 1.06.
12 $TotalCost = $TotalCost - $Discount;
13 $TotalCost = $TotalCost * $Tax;
14 $Payments = $TotalCost / 12;
15 // Печать результатов.
16 print ("You requested to purchase $Quantity widget(s) at \$$Cost
   each.\n<P>");
17 print ("The total with tax, minus your \$$Discount, comes to $");
18 print (" \n<P>You may purchase the widget(s) in 12 monthly
   installments of \$$Payments each.\n<P>");
19 ?>
20 </BODY>
21 </HTML>
```

```
$Tax = $Tax + 1; // Налог ($Tax) 1.06.
```

Для сложения используется знак плюс (+). Вы можете вычислить, сколько будет стоить товар, включая налог, добавив к проценту единицу и затем умножив это число на общую стоимость покупки. Для ясности добавим комментарий (его можно разместить в конце строки, как у меня, а можно - на следующей строке, допустимо вообще не делать комментариев). Одна из причин объединения обоих типов чисел в одну категорию переменных заключается в следующем: вы можете выполнять вычисления со смешанными типами переменных без каких-либо проблем.

```
$TotalCost = $TotalCost - $Discount;
```

Чтобы показать операцию вычитания, для которой используется знак минус (–), предположим, что можно использовать скидку, которая также будет добавлена к указателю URL или введена в форму.

```
$TotalCost = $TotalCost * $Tax;
```

Можно сделать вычисление с самой переменной для присвоения ей нового значения (вполне обычная практика), но обратите внимание, что первоначальное значение переменной теряется. Поэтому на данной строке первоначальное значение переменной `$TotalCost` заменено значением, полученным в результате умножения величины `$TotalCost` на `$Tax`.

```
$Payments = $TotalCost / 12;
```

Чтобы продемонстрировать операцию деления, предположим, что за товар будет уплачено в течение года. Поэтому мы разделили сумму покупки, включая налоги и соответствующие скидки, на 12 и нашли сумму ежемесячного платежа.

```
// Печать результатов.
```

Этот комментарий отделяет вычисления от передачи результатов в браузер.

```
print ("You requested to purchase $Quantity widget(s) at \$$Cost
      -each.\n<P>");
print ("The total with tax, minus your \$$Discount, comes to $");
print (" \n<P>You may purchase the widget(s) in 12 monthly installments
      -of \$$Payments each.\n<P>");
```

3. Сохраните сценарий как файл `numbers.php` и загрузите его на сервер.
4. Протестируйте сценарий в Web-браузере, не забудьте указать количество товара и скидки (рис. 4.1).

Поэкспериментируйте с этими значениями, в том числе опуская переменные, как на рис. 4.2, и посмотрите, правильно ли работает калькулятор.



Как можно было заметить, калькулятор использует числа, не совсем соответствующие реальным денежным значениям (см. рис. 4.1). В следующем разделе вы научитесь округлять числа.

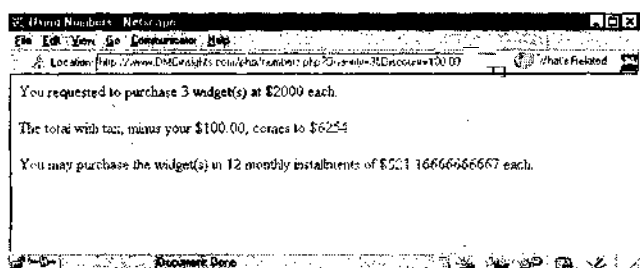


Рис. 4.1 ▼ Здесь изображен ваш калькулятор. Не забудьте добавить значения количества товара и скидки к URL или создать форму для передачи этих значений, в противном случае полученный результат будет аналогичен тому, что представлен на рис. 4.2

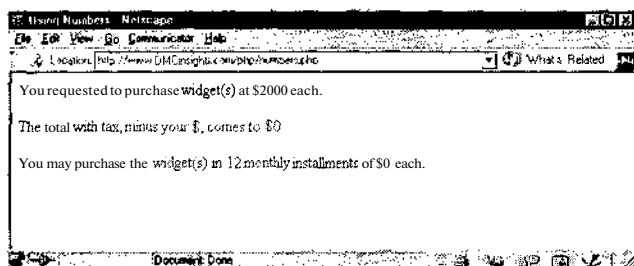


Рис. 4.2 т Соответствующие расчеты будут произведены даже без необходимых значений



Если вы хотите распечатать значение общей суммы до прибавления налога и вычитания скидки, есть два способа сделать это. Первый состоит в том, чтобы вставить соответствующую инструкцию `print()` сразу же за рассчитанным значением, перед тем как переменная `$TotalCost` была изменена. Второй способ – создать новые переменные, в которых будут храниться значения последующих вычислений (например, `$TotalWithTax` и `$TotalLessDiscount`).



Есть два способа напечатать цифру со знаком доллара, например \$2000.00. Первый – заэкранировать знак доллара, как это сделано в примере. Второй – вставить пробел между знаком доллара и именем переменной. В таком случае и знак доллара, и пробел будут переданы в браузер. Нельзя использовать переменную типа `$$Variable`, так как комбинация `$$` создает особый тип переменной, который в книге не рассматривается.

Форматирование чисел

Хотя ваш калькулятор уже достаточно удобен, есть одна финансовая проблема. Вряд ли вы решите попросить кого-либо осуществлять ежемесячные платежи по \$521.166666666667. Для создания более подходящего числа для таких платежей необходимо округлить денежные значения с точностью до цента. Для этого используется функция `printf()`, которая может распечатывать числа, отформатированные в соответствии с заданными условиями.

Для использования функции `printf()` необходимо сначала задать формат, а затем сами числа или строку с ними. Например, для печати значения переменной в форме числа с плавающей точкой с двумя цифрами после нее (в качестве примера возьмем число 1.02) необходимо написать:

```
printf ("%01.2f", $Amount);
```

Строка формата `"%01.2f"` дает команду напечатать слово `$Amount` с нулем для заполнения дополнительных мест, по крайней мере с одной цифрой слева от десятичной точки и с двумя – после нее. Функция `printf()` используется и для более сложных задач, но приведенного примера достаточно для получения допустимых денежных сумм.

Использование функции printf

1. Откройте файл numbers.php в текстовом редакторе.
2. Измените строку 17 следующим образом:

```
print ("The total with tax, minus your \$$Discount, comes to $");
printf ("%01.2f", $TotalCost);
```

При использовании функции форматной печати printf() стоит отделять эту строку от обычных команд print(), потому что способ ее вызова несколько отличается.

3. Измените строку 18 следующим образом:

```
print (" \n<P>You may purchase the widget(s) in 12 monthly
installments of $");
printf ("%01.2f", $Payments);
print (" each.\n<P>");
```

Сделайте с этой строкой то же, что со строкой 17. Из одной строки получится три, но результат будет выглядеть привлекательнее.

4. Сохраните файл (листинг 4.2), загрузите его на сервер и протестируйте в браузере (рис. 4.3).

Листинг 4.2 ▼ Функция print() напечатает числа, отформатированные в соответствии с заданными условиями.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Using Numbers</TITLE>
4 </HEAD>
5 <BODY>
6 <?php
7 /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8 $Cost = 2000.00;
9 $Tax = 0.05;
10 $TotalCost = $Cost * $Quantity;
11 $Tax = $Tax + 1; // Налог ($Tax) 1.06.
12 $TotalCost = $TotalCost - $Discount;
13 $TotalCost = $TotalCost * $Tax;
14 $Payments = $TotalCost / 12;
15 // Печать результатов.
16 print ("You requested to purchase $Quantity widget(s) at \$$Cost
   each.\n<P>");
17 print ("The total with tax, minus your \$$Discount, comes to $");
18 printf ("%01.2f", $TotalCost);
19 print (" \n<P>You may purchase the widget(s) in 12 monthly
   installments of $");
20 printf ("%01.2f", $Payments);
21 print (" each.\n<P>");
22 ?>
23 </BODY>
24 </HTML>
```



Функции `printf()` и `print()` аналогичны. Отличаются они тем, что первая сразу посылает результаты в браузер, а функция `printf()` помещает результирующую строку в переменную, не распечатывая ее:

```
$Amount = sprintf ("%01.2f", $Amount);
```

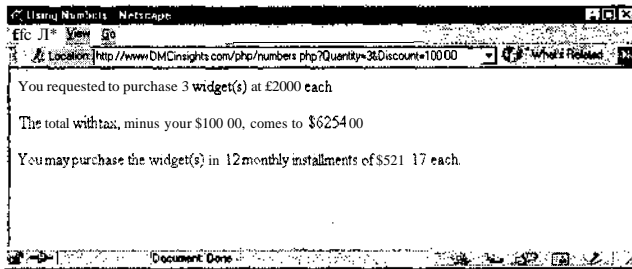


Рис. 4.3 Благодаря функции `printf()` обновленная версия сценария выдает более приемлемые цифры для ежемесячных платежей (рис. 4.1)

Инкремент и декремент

В PHP, как в Perl и большинстве других языков программирования, есть особые операторы, нужные для того, чтобы сократить длинные неуклюжие присвоения вроде `$Tax = $Tax + 1;`. Если необходимо увеличить значение переменной ровно на единицу (инкремент) или уменьшить на единицу (декремент), можно использовать операторы «++» и «--» соответственно.

Увеличение значения переменной на единицу

1. Откройте файл `numbers.php` в текстовом редакторе.
2. Измените строку 10 в листинге 4.2 следующим образом:
`$Tax++;`
3. Сохраните сценарий (листинг 4.3), загрузите его на сервер и протестируйте в браузере (рис. 4.4).

Листинг 4.3. С использованием оператора автоинкремента (++) сценарий стал немного аккуратнее, а математический результат – тот же.

```

1 <HTML>
2 <HEAD>
3 <TITLE>Using Numbers</TITLE>
4 </HEAD>
5 <BODY>
6 <?php
7 /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */

```

```

8  $Cost = 2000.00;
9  $Tax = 0.05;
10 $TotalCost = $Cost * $Quantity;
11 $Tax++; // Налог ($Tax) 1.06.
12 $TotalCost = $TotalCost - $Discount;
13 $TotalCost = $TotalCost * $Tax;
14 $Payments = $TotalCost / 12;
15 // Печать результатов.
16 print ("You requested to purchase $Quantity widget(s) at \$$Cost
each.\n<P>");
17 print ("The total with tax, minus your \$$Discount, comes to $");
18 printf ("%01.2f", $TotalCost);
19 print ("\n<P>You may purchase the widget(s) in 12 monthly
installments of $");
20 printf ("%01.2f", $Payments);
21 print (" each.\n<P>");
22 ?>
23 </BODY>
24 </HTML>

```

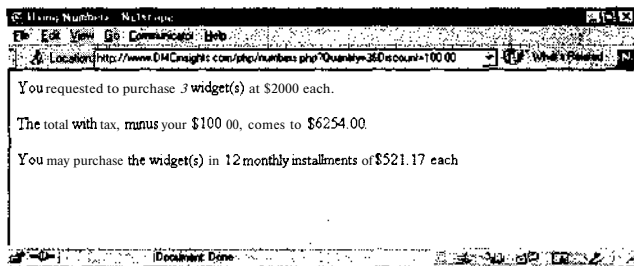


Рис. 4.4 т На результаты вычислений не влияет, как вы выполняете сложение - обычным способом или с помощью автоинкремента (сравни листинги 4.2 и 4.3)



Хотя с функциональной точки зрения не имеет значения, пишете ли вы `$Tax = $Tax + 1`; или сокращенно `$Tax++`, последний вариант, с оператором автоинкремента, более удачен и распространен.



В главе 6 говорится о том, как оператор автоинкремента используется в циклах.



Автоматически уменьшить переменную на единицу можно с помощью двойного знака минус «--» (оператор автодекремента): `$Number--`.

Совместное использование различных операторов

При обсуждении различного рода математических операторов неизбежно возникает вопрос *приоритета* - порядка выполнения действий при вычислениях. Например, какое значение будет иметь следующая переменная:

```
$Number = 10 - 4 / 2;
```

Будет ли результатом число 3 (сначала вычитаем, потом делим) или 8 (сначала делим, потом вычитаем)? Правильный ответ - 8, так как деление имеет приоритет над вычитанием. В приложении С представлен список приоритетов операторов в PHP (включая не упомянутые здесь).

Однако вместо заучивания большой таблицы символов я бы порекомендовал использовать скобки, которые всегда имеют приоритет над другими операторами:

```
$Number = (10 - 4) / 2;  
$Number = 10 - (4 / 2);
```

В первом примере переменная теперь равна 3, а во втором - 8. Использование скобок гарантирует, что вы не получите странных результатов вычислений.

Можно переписать сценарий, объединив многочисленные строки в одну, при этом поддерживая порядок вычислений с помощью скобок.

Листинг 4.4 Вместо выполнения вычислений на нескольких строках, мы сделали это на одной, не нарушив математических принципов. Используя скобки, можно не беспокоиться о приоритетах операторов.

```
1  <HTML>  
2  <HEAD>  
3  <TITLE>Using Numbers</TITLE>  
4  </HEAD>  
5  <BODY>  
6  <?php  
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.  
   $Discount обязательна. */  
8  $Cost = 2000.00;  
9  $Tax = 0.06;  
10 $Tax++; // Налог ($Tax) 1.06.  
11 $TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;  
12 $Payments = $TotalCost / 12;  
13 // Печать результатов.  
14 print ("You requested to purchase $Quantity widget(s) at \$$Cost  
   each.\n<P>");  
15 . print ("The total with tax, minus your \$$Discount, comes to $");  
16 printf ("%01.2f", $TotalCost);  
17 print (" \n<P>You may purchase the widget(s) in 12 monthly  
   installments of $");  
18 printf ("%01.2f", $Payments);  
19 print (" each.\n<P>");  
20 ?>  
21 </BODY>  
22 </HTML>
```

Установление приоритета с помощью скобок

1. Откройте файл numbers.php в текстовом редакторе.
2. Измените строку 13 следующим образом (листинг 4.4):

```
$TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;
```

Вполне можно выполнить все вычисления разом, если вы используете скобки для установки приоритетов. Другой вариант - выучить правила приоритетов в PHP, но намного легче использовать скобки.

3. Удалите строки 10 и 12. Так как вычисление происходит на одной строке, эти две больше не нужны.
4. Сохраните сценарий, загрузите его на сервер и протестируйте в браузере (рис. 4.5).



Следите за правильностью использования скобок при создании формул (каждая открывающая скобка должна иметь закрывающую).

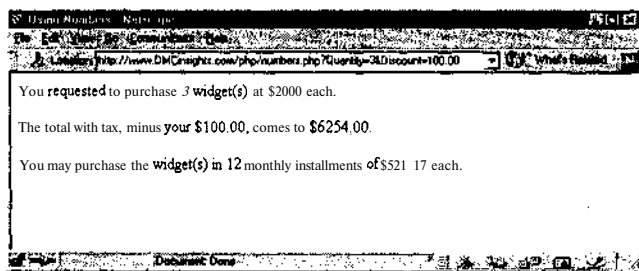


Рис. 4.5 ▼ Хотя мы и сжали вычисления, это не повлияло на результат. Если вы получаете другой результат или видите сообщение об ошибке, внимательно проверьте правильность расстановки скобок

Использование встроенных математических функций

В PHP есть ряд встроенных функций для обработки математических данных. Мы используем некоторые из них для улучшения сценария numbers.php, об остальных вы можете узнать из руководства по PHP.

Одна из встроенных математических функций, которую можно использовать в сценарии калькулятора, - это `round()`. Как видно из названия, данная функция округляет число до целого по стандартным математическим правилам: 0,5 и больше округляется до целого числа, меньше 0,5 - дробная часть просто отбрасывается. Подразумевается, что функция используется для округления дробных чисел, но даже попытка округлить целое число не вызовет никаких проблем (она просто ничего не будет делать, ведь результатом округления числа 2 будет 2). Некоторые примеры:

```
$Number = round(23.309); // $Number равно 23.  
$Number = round(23.51); // $Number равно 24.
```

В PHP 4.0 можно округлить число до определенного знака после запятой, добавив в уравнение второй параметр:

```
$Number = round(23.51, 1); // $Number равно 23.5.
```

На функцию `round()` похожи две другие функции. Первая, `ceil()`, округляет число до следующего целого, а вторая, `floor()`, - до предыдущего целого.

Еще одной функцией, которую можно использовать на нашей странице-калькуляторе, является `abs()`. Она возвращает абсолютное значение числа и работает следующим образом:

```
$Number = abs(-23); // $Number равно 23.
```

```
$Number = abs(23); // $Number равно 23.
```

На языке дилетантов абсолютное значение числа – всегда положительное число.

Две последние функции, которые будут здесь представлены, - это `srand()` и `rand()`. Последняя представляет собой генератор случайных чисел, а первая - функцию, инициализирующую `rand()`. Для получения действительно случайных чисел функцию `srand()` необходимо использовать до вызова `rand()`. В руководстве по PHP рекомендуется следующий код:

```
srand ((double) microtime() * 1000000);
```

```
$RandomNumber = rand();
```

При этом не поясняется, почему необходимо использовать именно такую строку кода для инициализации генератора случайных чисел, однако говорится, что создатели PHP 4.0 утверждают, что строка гарантирует действительно случайные числа.

В функции `rand()` можно использовать параметры минимума и максимума, если вы хотите, чтобы генерируемые числа относились к определенному диапазону.

```
$RandomNumber = rand (0, 10);
```

Из вышеупомянутых математических функций в сценарий `numbers.php` мы включим две - `abs()` и `round()` - для защиты от неверного ввода пользователя.

Выполнение действия

1. Откройте файл `numbers.php` в текстовом редакторе.
2. После строки 9 добавьте (листинг 4.5) следующую запись:

```
$Quantity = abs($Quantity);
```

```
$Discount = abs($Discount);
```

Если пользователь введет отрицательное количество или скидку, будет автоматически **предположено**, что имелось в виду положительное число, и для внесения соответствующего изменения будет использоваться функция `abs()`. Можно также округлять переменную `$Quantity`, если вы хотите продавать только целые единицы товара. Переписывая этот сценарий, вы можете проявить предусмотрительность и пойти **далее**, применив функцию `abs()` для переменной `$Payments`.

Листинг 4.5 т С помощью функций `round()` и `abs()` можно контролировать корректность денежных операций. Если необходимы целые числа или числа

с заданной точностью, используйте функцию `round()`, а если требуются положительные числа - `abs()`.

```

1  <HTML>
2  <HEAD>
3  <TITLE>Using Numbers</TITLE>
4  </HEAD>
5  <BODY>
6  <?php .
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 2000.00;
9  $Tax = 0.06;
10 $Quantity = abs($Quantity);
11 $Discount = abs($Discount);
12 $Tax++; // Налог ($Tax) 1.06.
13 $TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;
14 $Payments = round ($TotalCost, 2) / 12;
15 // Печать результатов.
16 print ("You requested to purchase $Quantity widget (s) at \$$Cost
   each.\n<P>");
17 print ("The total with tax, minus your \$$Discount, comes to $");
18 printf ("%01.2f", $TotalCost);
19 print (".\n<P>You may purchase the widget (s) in 12 monthly
   installments of $");
20 printf ("%01.2f", $Payments);
21 print (" each.\n<P>");
22 ?>
23 </BODY>
24 </HTML>

```

3. Измените строку 14, на которой рассчитывается переменная `$Payments`, следующим образом:

```
$Payments = round ($TotalCost, 2) / 12;
```

Чтобы увеличить правильность расчета, ежемесячные платежи будут вес- тись с точностью до центов (число `$TotalCost` округляется до двух знаков по- сле запятой).

4. Сохраните сценарий, загрузите его на сервер и протестируйте в браузере, введя отрицательное число для количества товара или скидки (рис. 4.6).

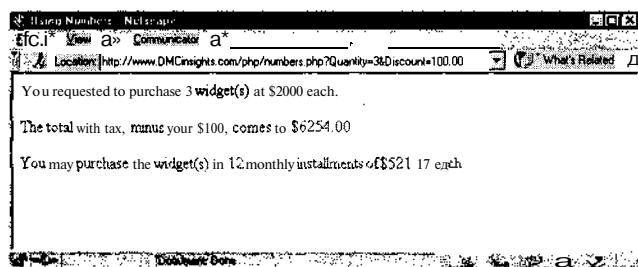


Рис. 4.6 т Включение функций `round()` и `abs()` в ваш сценарий сделает его более профессиональным, обеспечив корректность введенных значений и правильные результаты

Глава

Использование строк

Как упоминалось в главе 2, вторая категория переменных – это строки (strings), которые представляют собой набор символов, заключенных в одинарные или двойные кавычки. Строка может состоять из одной буквы, слова, предложения, параграфа или даже бессмысленного набора букв, чисел и символов (что часто используется для паролей). Строки – наиболее часто используемый тип переменных в PHP.

Пароли, имена, адреса электронной почты, комментарии и подобные вводимые в HTML-формы данные становятся строками вашего PHP-сценария. Вы могли заметить это при использовании страниц `form.html` и `HandleForm.php` в главе 3.

В данной главе рассматриваются основные встроенные функции PHP для обработки текстовых строк, приходящих из форм или создающихся иначе. Здесь будут также представлены некоторые распространенные приемы работы со строками: удаление концевых пробелов, соединение нескольких строк в одну, извлечение подстрок. В последующих главах мы также будем неоднократно возвращаться к способам манипулирования строками.

Удаление концевых пробелов

Из-за небрежности пользователя при вводе информации, а иногда из-за неаккуратного HTML-кода часто к строке-переменной добавляются лишние пробелы. Перед тем как применять подобные строки, эти пробелы обязательно нужно удалить. В таком случае вам удастся избежать множества проблем. Строка с невидимым дополнительным пробелом уже не будет совпадать с такой же строкой без пробела. Лишние пробелы, посланные в Web-браузер, могут исказить вид страницы, а пробелы, отправленные в базу данных, или cookie-файлы

вызвать еще более неприятные последствия. Например, если пароль содержит концевые пробелы, он не будет действителен при введении без пробелов.

Функция `trim()` автоматически отсекает лишние пробелы в начале и конце строки (но не в середине). Используется следующий формат функции `trim()`:

```
$String = " extra space before and after text";
$string = trim($String);
// Переменная $String теперь равна "extra space before and after text".
```

Вернемся к сценарию `HandleForm.php` из главы 3 и на этот раз внимательно и грамотно обработаем все полученные от пользователя строки.

Выполнение действия

1. Откройте сценарий `HandleForm.php` в текстовом редакторе (листинг 5.1).

Листинг 5.1 ▼ Первая версия сценария `HandleForm.php` совсем проста, так что в него нужно добавить некоторые дополнительные средства обработки полученных данных.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Form Results</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7 print ("Your first name is $FirstName.<BR>\n" );
8 print ("Your last name is $LastName.<BR>\n" );
9 print ("Your E-mail address is $Email.<BR>\n" );
10 print ("This is what you had to say:<BR>\n $Comments<BR>\n");
11 ?>
12 </BODY>
13 </HTML>
```

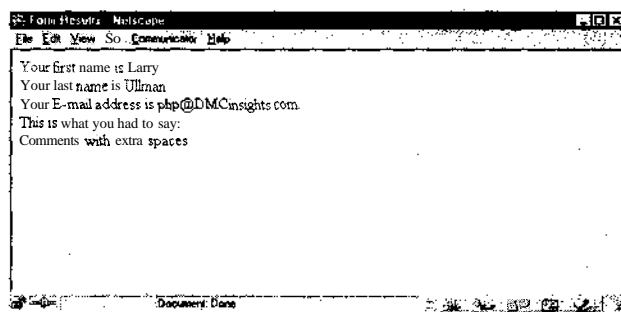


Рис. 5.1 ▼ Присмотревшись, можно увидеть лишние пробелы между именем и точкой в конце первых двух строк. Наличие таких незапланированных пробелов говорит о непредусмотрительности разработчика. На рис. 5.2, где представлен исходный текст HTML, пробелы видны четче

2. В листинг 5.1 после строки 6 (комментарий) добавьте следующую строку:

```
$FirstName = trim($FirstName);
```

Убрав концевые пробелы из переменной `$FirstName`, вы исключите передачу в браузер такого утверждения как "Your first name is _Larry_", содержащего лишние пробелы (рис. 5.1-5.2).

3. `$LastName = trim($LastName);`

```
$Email = trim($Email);
```

Приведение в порядок адреса электронной почты очень полезно, так как лишние пробелы могут сделать адрес нерабочим.

```
$Comments = trim($Comments);
```

Функция `trim()` работает одинаково независимо от того, откуда взята строка: из короткого фрагмента текста (как переменная `$FirstName` или `$Email`), из текстового окна HTML или параграфа из текстовой области.

4. Сохраните сценарий (листинг 5.2) все еще как `HandleForm.php`, загрузите его на сервер и протестируйте в браузере (рис. 5.3-5.5).

Листинг 5.2. Кроме отсеечения лишних пробелов от всех получаемых данных, я изменил также название страницы, хотя этого можно и не делать.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Form Results/Using Strings</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает И обрабатывает данные, принятые
   от "form.html." */
7 $FirstName = trim($FirstName);
8 $LastName = trim($LastName);
9 $Email = trim($Email);
10 $Comments = trim($Comments);
11 print "Your first name is $FirstName.<BR>\n";
12 print "Your last name is $LastName.<BR>\n";
13 print "Your E-mail address is $Email.<BR>\n";
14 print "This is what you had to say:<BR>\n $Comments<BR>\n";
15 ?>
16 </BODY>
17 </HTML>
```



Для удаления лишних пробелов только в начале строки или только в ее конце используются функции `ltrim()` и `rtrim()` соответственно. В остальном они применяются точно так же, как и функция `trim()`:

```
$String = rtrim($String);
$String = ltrim($String);
```

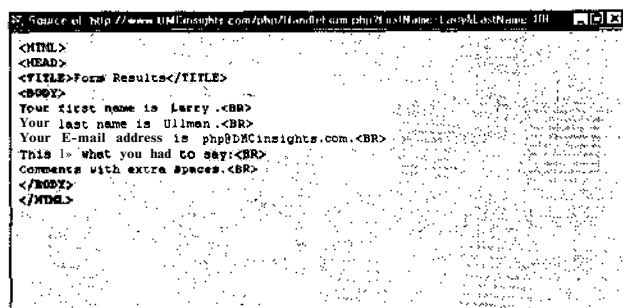


Рис. 5.2 т При просмотре исходного текста HTML отчетливо видны лишние пробелы, переданные из формы. В нашем случае это не критично, но иногда лишние пробелы могут разрушить дизайн HTML-страницы

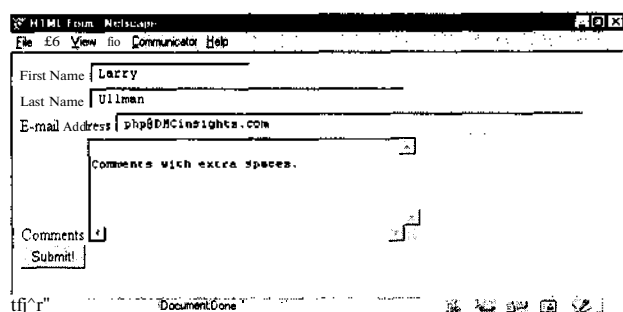


Рис. 5.3 т По разным причинам пользователи часто добавляют лишние пробелы при вводе информации в форму. В хорошей программе эта проблема решается без труда, а грамотный программист должен предвидеть такие ситуации заранее

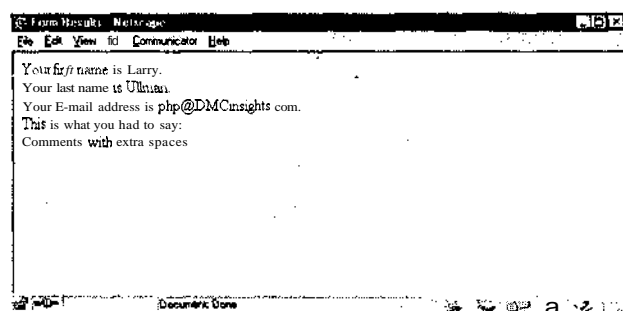


Рис. 5.4 т Грамотный стиль написания сценариев подразумевает изрядную долю недоверия к тому, что намеренно или случайно вводит пользователь. Теперь ваша страница выглядит более профессионально, так как вы привели в порядок введенную пользователем информацию

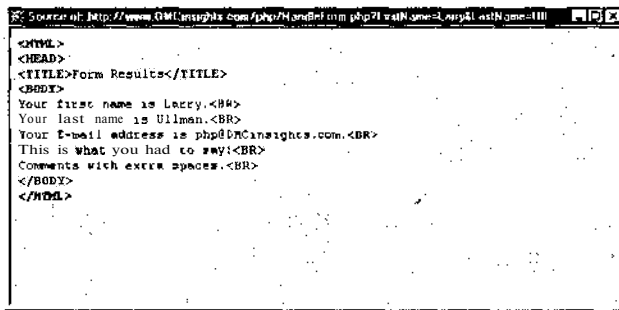


Рис. 5.5 Сравните исходный текст HTML-страницы, изображенной на рис. 5.3, с исходным текстом, представленным на рис. 5.2, и вы легко увидите разницу при использовании функции `trim()`

Соединение строк (сцепление, конкатенация)

Хотя термин *сцепление*, или *конкатенация*, не очень благозвучен, им обозначается чрезвычайно важный оператор - построение предложения из отдельных фраз, или соединение нескольких строк в одну. Мы будем часто сцеплять строки. Для этого предназначен оператор точка (`.`), который используется аналогично знаку плюс в арифметических выражениях:

```
$NewString = $aString . $bString;
```

Таким образом можно связать любое количество строк. К строкам с таким же успехом добавляются числа, которые становятся частью новой строки:

```
$NewString = $aString . $bString . $cNumber;
```

Это работает потому, что, как мы уже обсуждали, PHP относится к языкам со слабой типизацией, то есть переменные не привязаны строго к одному типу данных. Числовая переменная `$cNumber` будет автоматически преобразована в строку и добавлена к содержимому переменной `$NewString`.

Сценарий `HandleForm.php` содержит строки, которые так и хочется соединить с помощью вновь изученного оператора. Рекомендуется вводить данные имени и фамилии отдельно, как мы сделали в нашей форме. С другой стороны, иногда необходимо трактовать имя и фамилию как единое целое. Изменим сценарий соответствующим образом.

Использование сцепления в сценарии

1. Откройте сценарий `HandleForm.php` в текстовом редакторе (листинг 5.2).
2. Измените строку 11 следующим образом:

```
$Name = $FirstName . " " . $LastName;
```

Так как имя и фамилия будут объединены в единое целое, больше нет необходимости использовать отдельные инструкции (строки 11 и 12. листинг 5.2), поэтому вы заменяете данную строку и модифицируете другую. Сцепление

должно быть выполнено после инструкции `trim()`, так как с помощью этой функции невозможно удалить лишние пробелы в строке после сцепления имени и фамилии. Не забудьте вставить пробел между именем и фамилией, иначе они будут писаться слитно.

3. Измените строку 12 на следующую:

```
print ("Your name is $Name.<BR>\n");
```

4. Сохраните сценарий (листинг 5.3), загрузите его на сервер и протестируйте в браузере (рис. 5.6 и 5.7).

Листинг 5.3. Сцепление - одна из самых распространенных манипуляций со строками. Его можно представить как сложение строк.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Form Results/Using Strings</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7 $FirstName = trim($FirstName);
8 $LastName = trim($LastName);
9 $Email = trim($Email);
10 $Comments = trim($Comments);
11 $Name = $FirstName . " " . $LastName;
12 print ("Your name is $Name.<BR>\n");
13 print ("Your E-mail address is $Email.<BR>\n");
14 print ("This is what you had to say:<BR>\n $Comments<BR>\n");
15 ?>
16 </BODY>
17 </HTML>
```

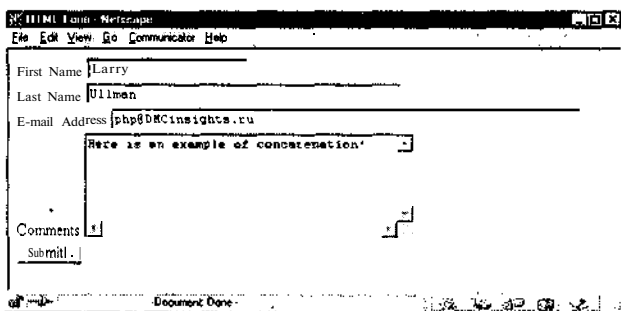


Рис. 5.6 Хотя сама HTML-форма и не изменится, сценарий `HandleForm.php` сможет создать полное имя пользователя из имени и фамилии, введенных раздельно (см. рис. 5.7)



Благодаря интерпретации языком PHP переменных такой же эффект может быть достигнут с помощью функции `$Name = "$FirstName $LastName"`; . Переменные, используемые внутри двойных кавычек, заменяются значениями этих переменных при обработке PHP. Однако формальный метод использования точки для

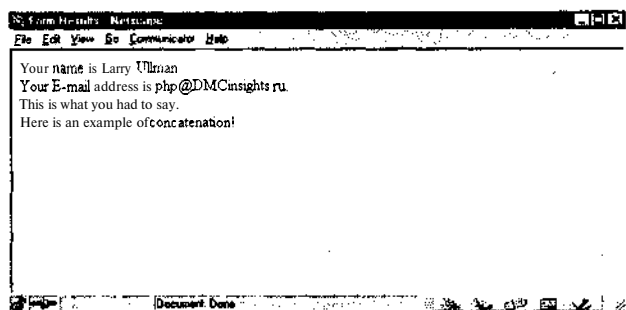


Рис. 5.7 Т с помощью простого сцепления переменных имени и фамилии мы получили еще одну переменную - полное имя пользователя

сцепления строк более широко распространен, и я бы порекомендовал следовать именно ему (тогда вам будет легче разобраться с кодом).



Можно также ввести запись `$FirstName = $FirstName . " " . $LastName;`, но в принципе этого делать не следует. Во-первых, после такой записи исходное значение переменной `$FirstName` будет перезаписано. Во-вторых, `FirstName` больше не будет подходящим именем для описания значения переменной. Программируя, всегда надо стараться использовать осмысленные имена переменных.

Кодирование и декодирование строк

В конце главы 3 было показано, как использовать метод GET для отправки данных в программу, добавляя параметры после URL-адреса сценария. Тогда этот метод применялся только для отправки значения, представляющего собой число или одно слово. Таким же образом этот процесс был дан и в главе 4. А что делать, если необходимо передать несколько слов как одно значение переменной?

Для таких случаев понадобится функция `urlencode()`. Как видно из ее имени, функция кодирует (encode) строку для того, чтобы она была правильно передана как часть URL. В частности, функция `urlencode()` заменяет пробелы на знаки плюс (+) и преобразовывает специальные символы, например апостроф, в более подходящие для URL знаки. Синтаксис функции выглядит следующим образом:

```
$String = urlencode($String);
```

Отправим только что созданную переменную `$Name` в страницу, которая приветствует пользователя по имени и фамилии.

Использование функции `urlencode`

1. Откройте сценарий `HandleForm.php` в текстовом редакторе (см. также листинг 5.3).

2. После строки 14 добавьте следующее:

```
$Name = urlencode($Name);
print ("<P>Click <A HREF=\"welcome.php?Name=$Name\"> here</A> to see
-your personalized greeting!\n");
```

3. Создайте новый документ с именем welcome.php. Вы можете написать код сами, опираясь на уже изученный материал, или скопировать его из листинга 5.5.

Новая страница получит значение переменной \$Name из сценария HandleForm.php. Используя это, покажем, как можно передавать значение из одной страницы на другую, а затем в следующую (из form.html в HandleForm.php и в welcome.php).

4. Сохраните оба сценария (листинг 5.4), загрузите их на сервер и протестируйте в браузере (рис. 5.8-5.14).

Листинг 5.4 Отметим следующее: во-первых, тэг HREF - это еще один элемент HTML, требующий использования кавычек, которых следует избегать в функции print(), во-вторых, в PHP при отправке в браузер переменная \$Name будет заменена ее значением (см. рис. 5.10).

```
1 <HTML>
2 <HEAD>
3 <TITLE>Form Results/Using Strings</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7 $FirstName = trim($FirstName);
8 $LastName = trim($LastName);
9 $Email = trim($Email);
10 $Comments = trim($Comments);
11 $Name = $FirstName " " . $LastName;
12 print ("Your name is $Name.<BR>\n");
13 print ("Your E-mail address is $Email.<BR>\n");
14 print ("This is what you had to say:<BR>\n $Comments<BR>\n");
15 $Name = urlencode($Name);
16 print ("<P>Click <A HREF=\"welcome.php?Name=$Name\"> here</A> to see
   your personalized greeting!\n");
17 ?>
18 </BODY>
19 </HTML>
```

Листинг 5.5 Как и наш первый сценарий «Hello, world!» (см. главу 1), страница welcome.php создает простое, немного отредактированное приветствие в браузере (рис. 5.11). Однако это приветствие персонализировано.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Welcome!</TITLE></HEAD>
4 <BODY>
5 <?php
6 print ("<B><CENTER>Hello, $Name.</CENTER></B>\n");
7 ?>
8 </BODY>
9 </HTML>
```

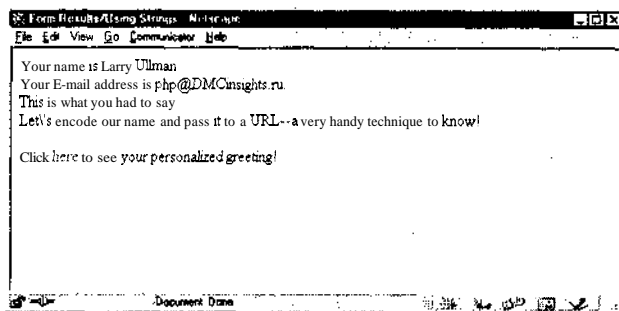



Рис. 5.8 ▼ Если не экранировать кавычки тэга HREF, последняя строка может быть отображена некорректно

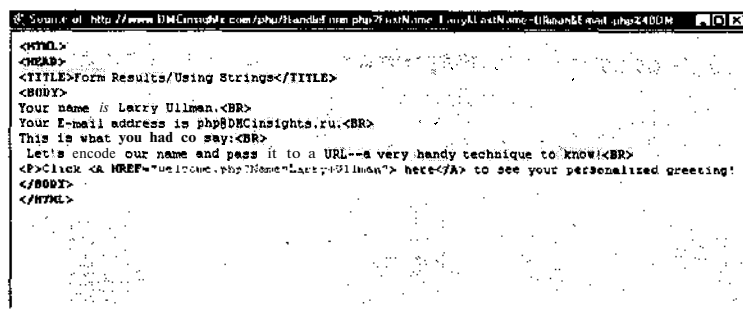


Рис. 5.9 т Это исходный текст HTML-страницы, сгенерированной на языке PHP. Обратите внимание на прикрепленное к ссылке закодированное имя. Если бы функция `urlencode()` не использовалась, результат был бы подобен тому, что представлен на рис. 5.12

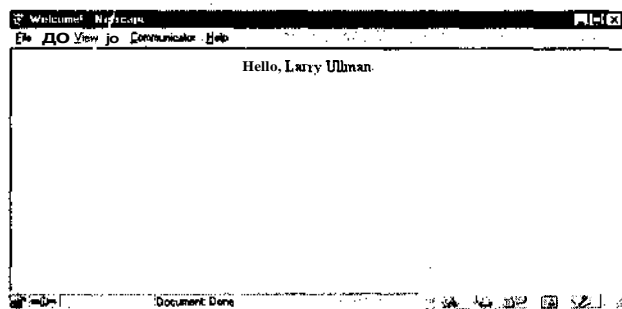


Рис. 5.10 т Эта динамически сгенерированная PHP-страница получает переменную не из формы, а из другого PHP-сценария. Распространенный и удобный прием

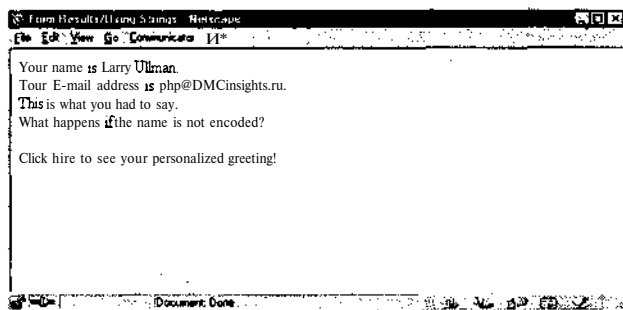


Рис. 5.11 ▼ Глядя на данную страницу, нельзя определить, правильно ли было закодировано значение полного имени. Однако это можно узнать из исходного текста HTML (рис. 5.12) или на самой странице welcome.php (рис. 5.13)

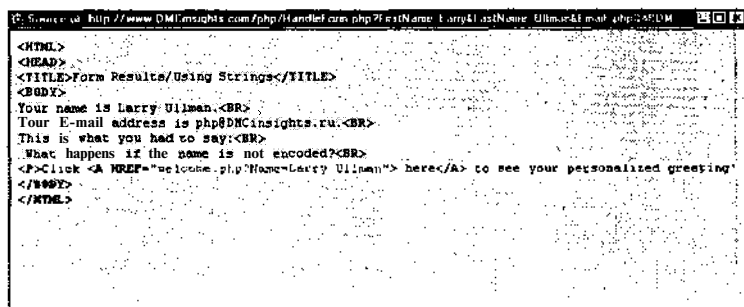


Рис. 5.12 ▼ Если между именем и фамилией есть пробел, то при передаче на другую страницу фамилия не будет отправлена. Вот почему необходимо кодировать переменные при передаче их через URL

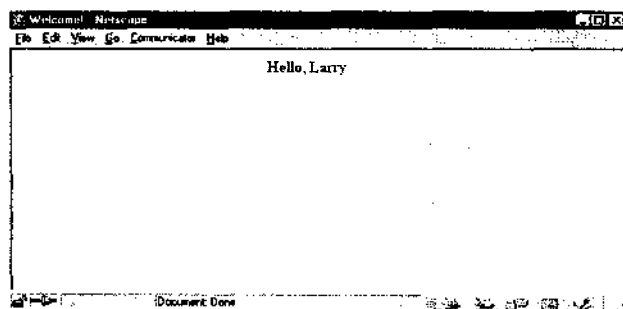


Рис. 5.13 ▼ Если PHP-страница распечатывает только первое слово из строки, значит, строка была послана в незакодированном виде

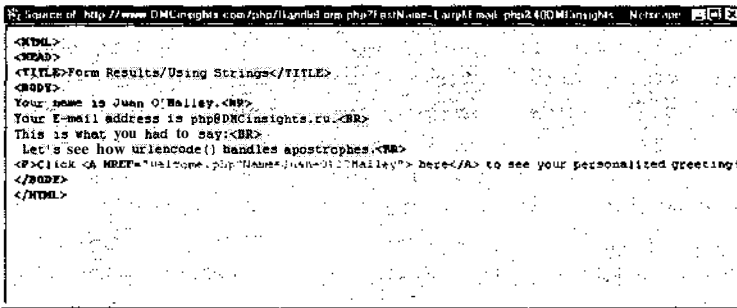


Рис. 5.14 Один из специальных символов, которые преобразует функция `urlencode()`, - это апостроф. Здесь мы видим его ASCII-эквивалент - %27



Функция `urldecode()` выполняет действие, обратное действию `urlencode()`: она берет закодированную для URL строку и преобразует ее обратно в обычную стандартную форму.



Запомните, что значения, посланные прямо из формы, автоматически кодируются в адресе URL перед отправкой и приходят в сценарий уже закодированными. Функция `urlencode()` используется только тогда, когда необходимо закодировать данные вручную.



В главе 11 мы рассмотрим функции, очень похожие на только что рассмотренную парочку `urlencode()` - `addslashes()` и `stripslashes()`. Первая подготавливает данные для ввода в базу данных, экранируя проблематичные символы (одинарные и двойные кавычки, обратный слеш). Вторая открывает эти самые символы вновь, удаляя символ экранирования. Синтаксис выглядит следующим образом:

```
$Data = addslashes($Data);
$Data = stripslashes($Data);
```

Как и с функцией `urlencode()`, перекодирование символов происходит автоматически при получении данных из формы в сценарий. Последний выполняет запись полученных переменных в базу данных.

Шифрование и дешифрование строк

Операции шифрования и дешифрования используются на большинстве сайтов, особенно тех, через которые ведется электронная торговля. Часто единственный способ защитить ценные данные - зашифровать их, то есть преобразовать в форму, в которой информацию трудно, порой практически невозможно распознать, не зная ключа. Пароли представляют собой данные, которые всегда необходимо шифровать. В зависимости от желаемого уровня безопасности можно также кодировать имена пользователей, адреса электронной почты, номера телефонов.

Заметим, однако, что тема защиты информации сложна и обширна. Мы не будем останавливаться на вопросах надежности многих сопутствующих программ и самого сервера. Только шифрование данных не гарантирует безопасности Web-сайта. Во-первых, приложив определенные усилия, зашифрованные данные иногда удастся взломать. Во-вторых, будучи серверной технологией, РНР может обеспечивать безопасность только после поступления информации в РНР-модуль на сервере. Безопасность данных, находящихся в пути между компьютером пользователя и сервером, надо обеспечивать другими методами. В приложении В вы найдете дополнительную информацию и ссылки на ресурсы по этому вопросу.

В данном разделе представлены три функции для шифрования и дешифрования строк. Одна из них будет продемонстрирована в действии и включена в сценарий.

Первая функция – `crypt()` – может использоваться для кодирования данных, но расшифровать их обратно не удастся. Если зашифровать с помощью этой функции пароль и хранить его в таком виде, то первоначальное значение пароля восстановить практически невозможно. Тем не менее даже эта функция одностороннего шифрования может быть использована в целом ряде случаев. Например, в Web-приложении она применяется для шифрования пароля пользователя при регистрации. Потом, когда пользователь еще раз входит в систему, вводимый им пароль снова шифруется, а две зашифрованные версии сравниваются. Синтаксис функции выглядит следующим образом:

```
$Data = crypt($Data);
```

Еще две функции, которые связаны с шифрованием, – это `encrypt()` и выполняющая обратное действие `decrypt()` (обратите внимание на то, что `crypt()` и `encrypt()` – две разные функции). Для использования этих функций в РНР-модуле должны быть установлены соответствующие расширения. К сожалению, в вопросы шифрования довольно грубо вмешиваются власть и политика. В США действуют ограничения на экспорт технологий шифрования и защиты информации, и две только что упомянутые функции подпадают под действие этих ограничений. Не удивляйтесь, если функции `encrypt()` и `decrypt()` не поддерживаются на сервере вашего провайдера. Тогда вам придется ограничиться только функцией `crypt()`. Впрочем, находчивый программист всегда найдет способ обойти препятствия.

Шифрование данных с помощью функции `crypt`

1. Откройте сценарий `HandleForm.php` в текстовом редакторе (листинг 5.4).
2. Замените строки 15 и 16, где для передачи имени пользователя на страницу `welcome.php` использована функция `urlencode()`, на следующие:

```
$CryptName = crypt($Name);  
print ("<P>This is the crypt() version of your name: $CryptName\n");
```

Нет необходимости зашифровывать URL-закодированную версию вашего имени, поэтому мы заменили данную часть сценария.

3. Сохраните сценарий (листинг 5.6), загрузите его на сервер и протестируйте в браузере (рис. 5.15).

Листинг 5.6 т Строки с функцией `crypt()` добавлены только для демонстрации ее работы. Хотя такое использование вряд ли актуально, пример показывает, как шифрование работает со строками.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Form Results/Using Strings</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7 $FirstName = trim($FirstName);
8 $LastName = trim($LastName);
9 $Email = trim($Email);
10 $Comments = trim($Comments);
11 $Name = $FirstName . " " . $LastName;
12 print ("Your name is $Name.<BR>\n");
13 print ("Your E-mail address is $Email.<BR>\n");
14 print ("This is what you had to say:<BR>\n $Comments<BR>\n");
15 $CryptName = crypt($Name);
16 print ("<P>This is the crypt() version of your name: $CryptName\n");
17 ?>
18 </BODY>
19 </HTML>
```



Аналогом функции `crypt()` является `md5()`, речь о которой пойдет ниже в этой главе. Более подробная информация об использовании `md5()`, а также о синтаксисе `encrypt()` и `decrypt()` содержится в приложении С.



Чтобы узнать, поддерживает ли ваш сервер конкретную функцию, надо просто вызвать ее. Если функция не поддерживается, вы получите сообщение об ошибке (рис. 5.16). Однако перед этим тщательно проверьте правописание и синтаксис. Так, если бы мы вызвали функцию, ошибочно переставив буквы `encript()`, появилось бы точно такое же сообщение, как на рис. 5.16.

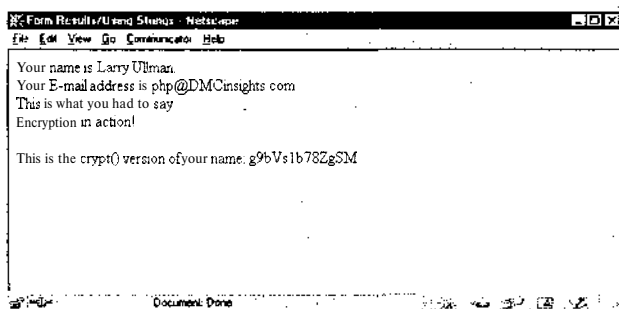


Рис. 5.15 т Функция `crypt()` возвращает уникальную 12-ти символьную строку на основании того, что зашифровано

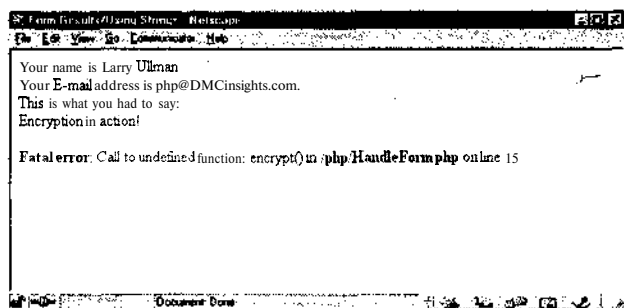


Рис. 5.16 ▼ Если немного изменить сценарий `HandleForm.php`, чтобы вызывалась функция `encrypt()`, результатом будет сообщение об ошибке. Оно означает, что сервер не поддерживает эту функцию



Будьте внимательны и не путайте шифрование строки с ее кодированием. Шифрование используется в целях безопасности и делает текст абсолютно не читаемым. Кодирование заменяет только определенные символы на эквиваленты, приемлемые для URL.

Извлечение части строки

В начале главы говорилось о том, как соединять строки с помощью оператора сцепления. Но из длинных строк можно также извлекать маленькие части. Рассмотрим две функции, с помощью которых можно делать это. Необходимо отметить, что для эффективного извлечения частей строки нужно иметь какую-то информацию о самой строке.

Функция `strtok()` извлекает подстроку, называемую *лексемой*, из строки на основе заранее установленного *разделителя* (обычно запятая или пробел). Например, если пользователи вводят имя и фамилию в одном поле, разделяя эти данные пробелом, то можно узнать имя с помощью следующего кода:

```
$FirstName = strtok($Name, " ");
```

Здесь `$Name` - это имя переменной, получаемой из формы с полным именем пользователя. Строка дает команду PHP извлечь из переменной `$Name` все до первого пробела. Если пользователи вводят полное имя в формате «фамилия, имя», фамилию можно узнать следующим образом:

```
$LastName = strtok($Name, ",");
```

К счастью, нам не нужно ломать над этими вариантами голову, так как в форме для ввода имени и фамилии мы предусмотрительно использовали разные поля.

Второй способ выделения подстрок - обратиться к *индексной позиции* символов внутри строки. Индексация строки означает нумерацию символов с начала строки, как если бы строка была массивом, а символы - ее элементами. В PHP, как и в большинстве языков программирования, индексация начинается с нуля.

Поэтому пронумерованная строка "Larry" будет выглядеть следующим образом: L на позиции 0, a - 1, r - 2, вторая r - 3, y - 4. Хотя длина строки "Larry" равна пяти, ее индекс «пробегаёт» значения от 0 до 4. Разобравшись с индексами, мы можем использовать функцию `substr()` для выделения нужного диапазона символов:

```
$SubString = substr($String,0,10);
```

Во-первых, необходимо указать строку (здесь это `$String`), из которой будет вырезаться подстрока. Во-вторых, с помощью индекса необходимо указать начало подстроки (если вы начинаете индексацию с нуля, значит, учитываться будет и первый символ). Третьим параметром вы указываете, из скольких символов будет состоять подстрока (10). Если строка состоит из меньшего количества символов, чем 10, подстрока закончится в конце строки.

Часто для определения длины строки используют функцию `strlen()`. Она рассчитывает длину строки, то есть количество содержащихся в ней символов:

```
$StringLength = strlen($String);
```

Еще раз напомним, что нумерация символов в строке начинается с нуля, так что индекс последнего символа в строке всегда равен `$StringLength-1`!

Давайте используем рассмотренные функции `substr()`, `strlen()` и `md5()` для создания простой программы генерации паролей. Известно, что, чем более случайно сочетание символов в пароле, тем он надежнее. Самые надежные пароли не имеют никаких ассоциаций, не встречаются даже в самых подробных словарях. Лучше всего, если пароль состоит из беспорядочного сочетания букв и цифр в разном регистре. Этот сценарий будет создавать новый случайный пароль каждый раз, когда вы перезагружаете страницу.

Создание генератора паролей в PHP

1. Начните создавать новый PHP-документ в текстовом редакторе со следующих строк:

```
<HTML><HEAD><TITLE>Password Generator</TITLE></HEAD><BODY><?php
$String = "This is the text which will be encrypted so that we may
-create random and move secure passwords!";
```

Здесь можно вставить любую строку. Конкретный текст не имеет значения для наших целей. Этот текст будет зашифрован для создания случайной строки, из которой взят пароль.

```
$Length = 8;
```

Установив длину пароля в виде переменной, легко менять одно это значение для получения паролей различной длины. Длина пароля ограничена 32 знаками, то есть длиной строки, зашифрованной с помощью функции `md5()`.

```
$String = md5($String);
```

Функция `md5()` похожа на `crypt()`, но генерирует строки длиной до 32 символов. Более подробную информацию по функции `md5()` можно найти в руководстве по PHP или на сайте, посвященном этому языку.

```
$StringLength = strlen($String);
```

Чтобы извлечь подстроку, необходимо знать длину зашифрованной строки. Для этого мы используем функцию `strlen()`, которая определяет количество символов в строке. Хотя точно известно, что при использовании `md5()` строка всегда будет состоять из 32 символов, лучше перестраховаться и применить функцию `strlen()`. В этом случае, даже если позже вы измените функцию (например, используете `encrypt()` вместо `md5()`), данная строка кода будет работать корректно.

```
srand ((double) microtime() * 1000000);
$Begin = rand(0, ($StringLength-$Length-1));
```

Необходимо определить исходную позицию для функции `substr()`. Функция `rand()` создает случайное число между минимумом (здесь это 0) и максимумом. Не забывайте перед этим использовать функцию `srand()`, иначе не получите действительно случайных результатов от `rand()`. Мы установили максимум как длину строки минус длина пароля минус единица. И вот почему: если зашифрованная строка состоит из 32 символов, а пароль – из восьми, то мы можем использовать функцию `substr()` и получить пароль из восьми символов максимум с 24-й позиции, или, учитывая нумерацию начиная с 0, с индекса номер 23. Мы взяли вычисление максимальной начальной позиции в скобки только для ясности, в функции `rand()` они не требуются.

Последний шаг – «заставить» функцию `substr()` извлечь подстроку нужной длины, начиная с вычисленной позиции.

```
$Password = substr($String, $Begin, $Length);
```

Вы прописываете, что переменная `$Password` равна подстроке, которая получена из переменной `$String`, начинающейся с индексной позиции `$Begin` и продолжающейся `$Length` символов.

```
print ("Your recommended password is:<P><BIG>$Password</BIG>\n");
```

А теперь уверенно распечатайте результат.

2. Закончите сценарий следующим кодом:

```
?></BODY></HTML>
```

3. Сохраните сценарий как `passwords.php` (листинг 5.7), загрузите его на сервер и протестируйте в браузере (рис. 5.17 и 5.18).

Листинг 5.7 ▼ Сценарий `passwords.php` содержит в сжатой форме то, чему вы научились до этого момента. И всего на 16 строках!

```
1 <HTML>
2 <HEAD>
3 <TITLE>Password Generator</TITLE></HEAD>
4 <BODY>
5 <?php
6 $String = "This is the text which will be encrypted so that we may
  create random and secure passwords!";
```



```

7  $Length = 8; // Измените это значение, чтобы установить длину пароля.
   32 символа - максимум.
8  $String = md5($String);
9  $StringLength = strlen($String);
10 srand ((double) microtime() * 1000000);
11 $Begin = rand(0, ($StringLength-$Length-1)); // Получить произвольную
   стартовую точку.
12 $Password = substr($String, $Begin, $Length);
13 print ("Your recommended password is:<P><BIG>$Password</BIG>\n");
14 ?>
15 </BODY>
16 </HTML>

```

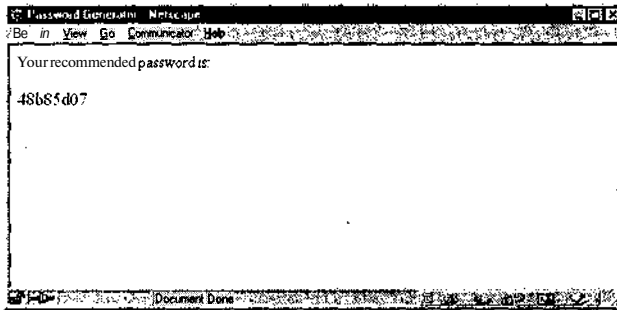


Рис. 5.17 ↑ Страница passwords.php создает надежные пароли

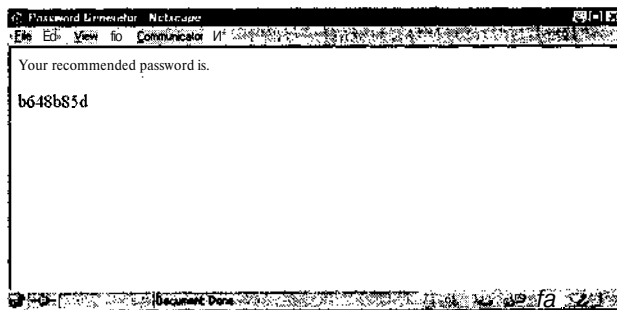


Рис. 5.18 ▼ Каждая перезагрузка страницы вызывает генерирование нового пароля. (В силу того что функция md5 () создает строку из 32 символов, количество уникальных паролей ограничено, но этот сценарий создаст десятки разных паролей.)



Вместо установки переменной `$Length` можно использовать метод GET для отправки в страницу значения длины строки, добавив `?Length=8` к URL (если вы хотите это попробовать, удалите седьмую строку).



Базы данных имеют следующий или подобный формат даты: YYYY-MM-DD. Так как известен точный формат извлекаемой из базы данных строки, то значения года, месяца и дня могут быть легко рассчитаны с помощью функции `substr ()`:

```
$Year = substr($Date,0,4);  
$Month = substr($Date,5,2);  
$Day = substr($Date,8,2);
```



Будьте внимательны при использовании функции `strlen()`. Например, нельзя писать `$pas-sword = strlen($Password);`, так как в этом случае реальное значение пароля будет потеряно и заменено числом, указывающим на количество символов, которое раньше имелось в переменной `$Password`.



Управляющие структуры

Поговорив об основных типах переменных и некоторых операциях с ними (кроме массивов, которые будут описаны в главе 7), займемся собственно программированием - изучим управляющие структуры: условные операторы, циклы, блоки. Здесь же будет завершено обсуждение различных операторов, используемых в РНР. В главах, посвященных переменным, уже шла речь об арифметических операторах и операторах присваивания.

Управляющие структуры - это основа языков программирования. Они позволяют задать параметр и затем в зависимости от его значения выполнить то или иное действие. Благодаря этому Web-сайты становятся более динамичными. Например, до обеда можно приветствовать пользователя словами вроде «Доброе утро!», а после обеда - «Добрый день!» В языке РНР есть два основных условных оператора - `if` и `switch`. Речь о них и пойдет в этой главе.

Также здесь представлены и две новые категории операторов - сравнения и логические. Они обычно используются в условных операторах.

В заключение мы поработаем с циклами, которые позволяют повторять действие определенное количество раз. Циклы экономят время программирования и особенно удобны при работе с массивами (см. главу 7). В языке РНР поддерживается два типа циклов - `while` (и его эквивалент `do...while`) и `for`.

Условный оператор `if`

Основной условный оператор в программировании - это стандартный `if` (раньше он назывался `if-then`, then теперь только подразумевается). Синтаксис этого оператора очень прост:

```
if (condition) {  
    statement(s);  
}
```

Условие должно быть заключено в круглые скобки. Раздел с инструкциями заключается в фигурные скобки и иногда называется *блоком операторов*. Здесь размещаются исполняемые команды, например печать строки, сложение двух чисел и т.п. Каждая отдельная инструкция (или команда) должна заканчиваться точкой с запятой. Ограничений на количество используемых инструкций не существует, равно как и на количество вложенных блоков и операторов. Обычно программисты располагают эти инструкции с отступом от начала строки с оператором `if`, показывая таким образом, что данный блок выполняется при определенном условии. Если не использовать точку с запятой после каждой инструкции, забыть поставить открывающую, закрывающую обычную или фигурную скобку, нечаянно поставить точку с запятой после любой из скобок, то программа будет выполняться с ошибками.

Для определения того, нужно ли выполнять инструкцию, в PHP используются понятия *истинный* (`true`) и *ложный* (`false`). Если значение условия истинно, то блок будет выполнен, в противном случае нет. В следующем разделе указанные понятия рассматриваются более подробно.

Перепишем программу-калькулятор (глава 4) так, чтобы она работала только при условии передачи в нее значения количества. Это предотвратит выполнение вычислений без наличия всех требуемых данных, что может вызвать неправильные результаты или ошибки на Web-страницах.

Создание условного оператора `if`

1. Откройте текущую версию `numbers.php` (листинг 6.1) в текстовом редакторе.

Листинг 6.1 В исходном сценарии `numbers.php` производились вычисления, а результаты распечатывались с учетом предположения о том, что значения `$Quantity` и `$Discount` были получены. Это не совсем удачная программистская практика, такой сценарий слишком легко «взломать».

```

1  <HTML>
2  <HEAD>
3  <TITLE>Using Numbers</TITLE>
4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 2000.00;
9  $Tax = 0.06;
10 $Quantity = abs($Quantity);
11 $Discount = abs($Discount);
12 $Tax++; // Налог ($Tax) составляет 1.06.
13 $TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;
14 $Payments = round ($TotalCost, 2) / 12;
15 // Печать результатов.
16 print ("You requested to purchase $Quantity widget(s) at \$$Cost
   each.\n<P>");

```

```

17 print ("The total with tax, minus your \$$Discount,' comes to $");
18 printf ("%01.2f", $TotalCost);
19 print (".\n<P>You may purchase the widget(s) in 12 monthly
    installments of $");
20 printf ("%01.2f", $Payments);
21 print (" each.\n<P>");
22 ?>
23 </BODY>
24 </HTML>

```

2. Измените название (строка 3) следующим образом:

```
<TITLE>Conditionals</TITLE>
```

3. Измените строки 10-23 так, чтобы в них применялся условный оператор if.

```

if ($Quantity) {
    $Quantity = abs($Quantity);
    $Discount = abs($Discount);
    $Tax++; // $Tax составляет 1.06.
    $TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;
    $Payments = round ($TotalCost, 2) / 12;
    // Печать результатов.
    print ("You requested to purchase $Quantity widget(s) at\$$Cost
    -each.\n<P>");
    print ("The total with tax, minus your \$$Discount, comes to $");
    printf ("%01.2f", $TotalCost);
    print (".\n<P>You may purchase the widget(s) in 12 monthly
    -installments of $");
    printf ("%01.2f", $Payments);
    print (" each.\n<P>");
}

```

В PHP простое использование имени переменной в качестве условия (как здесь с переменной \$Quantity) является эквивалентом высказывания «Если переменная \$Quantity существует, то есть имеет значение, отличное от нуля, ...». Таким образом, в PHP могут быть выполнены следующие строки, только если значение \$Quantity отлично от нуля.

4. Сохраните сценарий (листинг 6.2), загрузите его на сервер и протестируйте страницу в браузере с необходимой информацией \$Quantity и без нее (рис. 6.1 и 6.2 соответственно).

Листинг 6.2 Частое использование условного оператора if делает программирование более надежным, ведь перед переходом к соответствующим процессам проверяются определенные параметры. Здесь мы задаем следующее условие: вычисления производятся только после получения значения переменной \$Quantity.

```

1 <HTML>
2 <HEAD>
3 <TITLE>Conditionals</TITLE>

```

```

4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 2000.00;
9  $Tax = 0.06;
10 if ($Quantity) {
11     $Quantity = abs($Quantity);
12     $Discount = abs($Discount);
13     $Tax++; // $Tax составляет 1.06.
14     $TotalCost = (($Cost * $Quantity) - $Discount) * $Tax;
15     $Payments = round ($TotalCost, 2) / 12;
16     // Печать результатов.
17     print ("You requested to purchase $Quantity widget(s) at \$$Cost
   each.\n<P>");
18     print ("The total with tax, minus your \$$Discount, comes to $");
19     printf ("%01.2f", $TotalCost);
20     print (".\n<P>You may purchase the widget(s) in 12 monthly
   installments of $");
21     printf ("%01.2f", $Payments);
22     print (" each.\n<P>");
23 }
24 ?>
25 </BODY>
26 </HTML>

```

Если инструкция состоит только из одной строки, то нет технической необходимости ставить фигурные скобки и все можно разместить на одной строке:

```
if (condition) statement;
```

Об этом варианте разумно говорить только потому, что можно увидеть код в таком формате. Однако рекомендуется всегда придерживаться многострочного формата, показанного выше, для поддержки согласованности и минимизации ошибок.

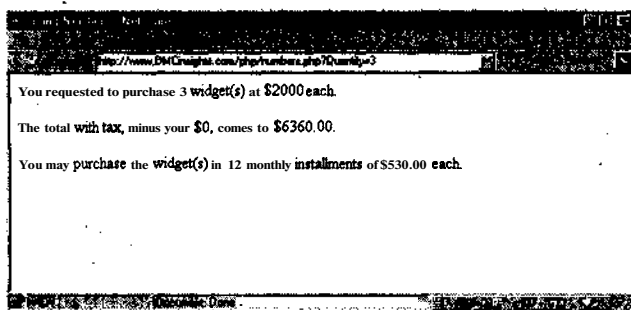


Рис. 6.1 т До тех пор пока страница получает значение переменной \$Quantity, она будет работать также, как и до добавления условия

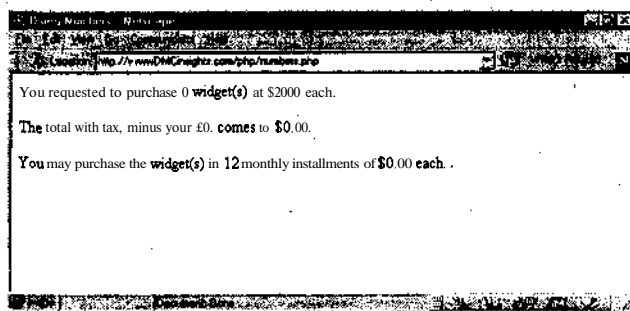


Рис. 6.2 ▼ Благодаря условному оператору `if` ваш сайт никогда больше не выдаст подобных результатов

В PHP вполне допустимо использовать вложенные условия, то есть задавать проверку следующего условия внутри выполняемого блока предыдущего. Главное, не забывать соответствующим образом закрывать условные выражения и не запутаться во вложенных блоках операторов.

Ниже в этой главе речь пойдет о том, как выполнить команды, если условие не удовлетворено. Тогда, если необходимое значение не получено, можно запрограммировать сценарий на запрос этого значения.

Чтобы определить, существует ли переменная, можно также использовать функцию `isset()`. В отличие от обращения к переменной по имени, как это делалось выше, функция `isset()` возвратит истинное значение, если переменная равна нулю:

```
$Quantity = 0;
if ($Quantity) { ... // FALSE
if (isset($Quantity)) { ... // TRUE
```

Другие операторы

В большинстве своем операторы PHP и типы переменных, которые в них используются, описывались в предыдущих главах. Это арифметические операторы сложения (+), вычитания (-), умножения (*) и деления (/), а также операторы инкремента (++) и декремента (--) для увеличения и уменьшения числового значения на единицу. Это также оператор присвоения (=), используемый для задания значения переменной любого типа. Мы обсудили и оператор сцепления (.), используемый для соединения строк.

Все указанные операторы хороши для присвоения значения переменной, но от них мало пользы, когда дело касается условных выражений. Для этого применяются операторы сравнения и логические операторы.

Операторы сравнения

В главе 2 был представлен оператор присвоения (знак равенства). При этом оговаривалось, что значение оператора несколько необычно. Выражение

`$Variable = 5;` означает не то, что переменная `$Variable` равна 5, а то, что ей присвоено значение 5.

При программировании условий часто необходимо знать, равна ли переменная конкретному значению (например, при проверке имени пользователя или пароля), что нельзя сделать с помощью одного знака равенства (тем более что он уже занят оператором присвоения). Для этих целей используется *оператор равенства* (`==`), состоящий из двух знаков равенства подряд.

```
$Variable = 5;  
$Variable == 5;
```

При совместном использовании этих строк кода первая строка присваивает переменной `$Variable` значение 5, а затем «говорит», что результат истинен, когда «видит», что значение `$Variable` равно 5. Это доказывает, какую значительную разницу вносит дополнительный знак равенства в код PHP и почему необходимо проводить четкую границу между операторами присвоения и сравнения.

Оператор *неравенства* в PHP представлен комбинацией из восклицательного знака и знака равенства (`!=`). Вообще восклицательный знак показывает отрицание значения, функцию *логическое нет*. `$Variable` означает, что переменная `$Variable` существует и имеет значение, отличное от нуля, а `!$Variable` — что переменная `$Variable` не существует, не имеет значения или ее значение равно нулю.

Остальные операторы сравнения аналогичны математическим эквивалентам: *меньше* (`<`), *больше* (`>`), *меньше или равно* (`<=`), *больше или равно* (`>=`).

Расширим функциональность программы расчета стоимости товара, переписав сценарий `numbers.php` так, чтобы скидка применялась только при покупке на сумму больше \$50.

Использование операторов сравнения

1. Откройте текущую версию `numbers.php` (листинг 6.2) в текстовом редакторе.
2. Снизьте цену единицы товара, чтобы было не так просто дойти до искомого рубежа в \$50, строка 8.

```
$Cost = 20.00;
```

3. После строки 13 измените выражение с `$TotalCost` следующим образом:

```
$TotalCost = ($Cost * $Quantity);
```

Так как скидка применима лишь для общей суммы покупки свыше \$50, то сначала необходимо отдельно рассчитать эту общую сумму.

4. Создайте условие, проверяющее, превысила ли общая сумма \$50.
if (`$TotalCost >= 50`) {
 `$TotalCost = $TotalCost - $Discount;`
}

Используя операторы сравнения внутри выражения условия, можно определить, что скидка будет предоставлена, только если значение `$TotalCost` больше или равно `$50` (не указывайте знак доллара в условии). Скидка вычитается, только если условие истинно.

5. Добавьте к общей сумме налог.

```
$TotalCost = $TotalCost * $Tax;
```

6. Оставшаяся часть сценария осталась неизменной, включая определение ежемесячных взносов и печать всех результатов.

7. Сохраните сценарий (листинг 6.3), загрузите его на сервер и протестируйте в браузере с помощью разных значений переменной `$Quantity` (рис. 6.3 и 6.4).

Листинг 6.3 т Операторы сравнения, такие как «меньше или равно» (`<=`), позволяют лучше задавать числовые условия в коде.

```
1  <HTML>
2  <HEAD>
3  <TITLE>Conditionals</TITLE>
4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 20.00;
9  $Tax = 0.06;
10 if ($Quantity) {
11     $Quantity = abs($Quantity);
12     $Discount = abs($Discount);
13     $Tax++; // $Tax составляет 1.06.
14     $TotalCost = ($Cost * $Quantity);
15     if ($TotalCost >= 50) {
16         $TotalCost = $TotalCost - $Discount;
17     }
18     $TotalCost = $TotalCost * $Tax;
19     $Payments = round ($TotalCost, 2) / 12;
20     // Печать результатов.
21     print ("You requested to purchase $Quantity widget(s) at \$$Cost
   each.\n<P>");
22     print ("The total with tax, minus your \$$Discount, comes to $");
23     printf ("%01.2f", $TotalCost);
24     print (".\n<P>You may purchase the widget(s) in 12 monthly
   installments of $");
25     printf ("%01.2f", $Payments);
26     print (" each.\n<P>");
27 }
28 ?>
29 </BODY>
30 </HTML>
```

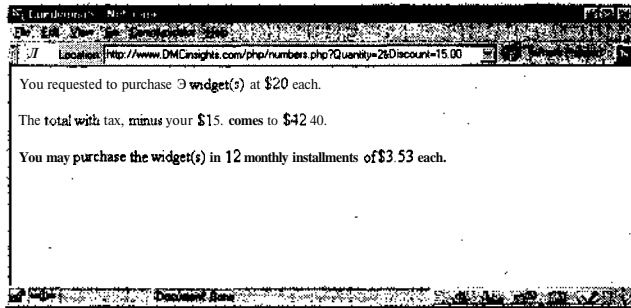


Рис. 6.3 т Скидка не была предоставлена, так как общая сумма покупки меньше \$50. Сравните с тем, что показано на рис. 6.4

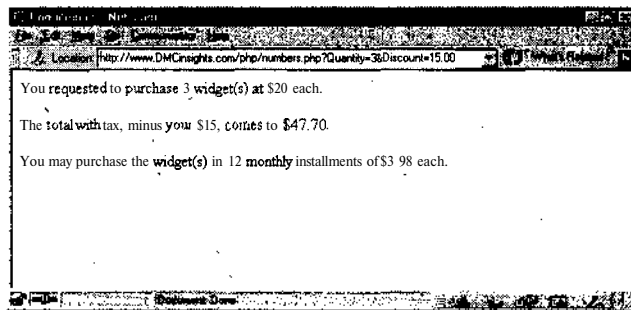


Рис. 6.4 т Пользователь заказал товар на общую сумму больше \$50, поэтому ему была предоставлена скидка



Если в выражениях с условным оператором `if` делается ошибка и вместо `$Variable == 5` пишется `$Variable = 5`, то соответствующие условию инструкции будут выполняться всегда. Это происходит потому, что оператор сравнения `$Variable == 5` может быть истинным или ложным, а оператор присвоения `$Variable = 5` всегда выдает истину, если справа от знака равенства стоит ненулевое значение.

Логические операторы

Логические операторы помогают создавать *логические выражения* — утверждения, которые имеют либо истинное, либо ложное значение. В PHP условие считается истинным, если это просто имя переменной, и данная переменная имеет значение, отличное от нуля:

```
$Variable = 5;
if ($Variable) { ...
```

Условие также истинно, если логически верно:

```
if (5>=3) { ...
```

Условие считается ложным, если относится к не имеющей значения переменной или если оператор сравнения выдает значение «ложь». Например, следующее условное выражение всегда будет ложным:

```
if (5<=3) { ...
```

Для построения из операторов логических выражений в PHP используют специальные логические операторы: два варианта *И* (AND и &&); два варианта *Или* (OR и || - две вертикальных черты); два варианта *Нем* (NOT и ! - восклицательный знак); а также *Или Нем* (XOR). Если имеется два варианта одного оператора (AND, OR и NOT), то они отличаются друг от друга только приоритетом (см. соответствующую таблицу в приложении С).

С помощью скобок и логических операторов можно создавать сколь угодно сложные условные выражения для оператора if. Для истинности выражения AND обе части условия должны быть верны. При использовании оператора OR хотя бы одна часть должна быть истинна, и тогда все условие верно. Следующие выражения истинны:

```
if ((5 <= 3) OR (5 >=3 )) { ...
```

```
if ((5 > 3) AND (5 < 10) ) { ...
```

Условия, приведенные ниже, ложны:

```
if ((5 != 5) AND (5 > 3 )) { ...
```

```
if ((5 != 5) OR (5 < 3) ) { ...
```

Создавая условные структуры, не забывайте о двух нюансах: во-первых, блок операторов **выполняется**, только если все условное выражение имеет истинное значение; во-вторых, используя скобки, можно не учитывать правила приоритета и использовать операторы по вашему выбору.

Чтобы показать работу логических операторов, добавим еще одно условие к сценарию numbers.php. Оно позволяет пользователю узнать, положена ли ему скидка.

Использование логических операторов

1. Откройте файл numbers.php в текстовом редакторе (листинг 6.3).
2. После того как впервые рассчитано значение \$TotalCost (листинг 6.3, строка 14), но перед условием if (\$TotalCost >= 50), наберите следующие строки:

```
if ( ($TotalCost < 50) AND ($Discount) ) {  
    print ("Your \$$Discount will not apply because the total value of  
    -the sale is under $50!\n<P>");  
}
```

Этим условием будут проверяться два момента: **во-первых**, превышает ли переменная \$TotalCost сумму \$50, **во-вторых**, существует ли ненулевое значение

скидки. Если оба значения истинны, сообщение будет распечатано. Если хоть одно из этих значений ложно, то все условие воспринимается как ложное (ведь оно регулируется оператором AND) и сообщение не распечатывается.

3. Поскольку это было единственное изменение в сценарии, теперь страницу можно сохранить (листинг 6.4), загрузить на сервер и протестировать в браузере (рис. 6.5 и 6.6).

Листинг 6.4 В этом сценарии логический оператор AND задает конкретное условие, при выполнении которого будет распечатано сообщение. При использовании оператора AND обе части условия должны быть истинны, чтобы все условие стало таким же.

```

1  <HTML>
2  <HEAD>
3  <TITLE>Conditionals</TITLE>
4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 20.00;
9  $Tax = 0.06;
10 if ($Quantity) {
11     $Quantity = abs($Quantity);
12     $Discount = abs($Discount);
13     $Tax++; // $Tax составляет 1.06.
14     $TotalCost = ($Cost * $Quantity);
15     if ( ($TotalCost < 50) AND ($Discount) ) {
16         print ("Your \$$Discount will not apply because the total value
           of the sale is under $50!\n<P>");
17     }
18     if ($TotalCost >= 50) {
19         $TotalCost = $TotalCost - $Discount;
20     }
21     $TotalCost .= $TotalCost * $Tax;
22     $Payments = round ($TotalCost, 2) / 12;
23     // Печать результатов.
24     print ("You requested to purchase $Quantity widget(s) at \$$Cost
           each.\n<P>");
25     print ("The total with tax, minus your \$$Discount, comes to $");
26     printf ("%01.2f", $TotalCost);
27     print (".\n<P>You may purchase the widget(s) in 12 monthly
           installments of $");
28     printf ("%01.2f", $Payments);
29     print (" each.\n<P>");
30 }
31 ?>
32 </BODY>
33 </HTML>

```



Еще одно общепринятое соглашение в программировании - английские слова TRUE (истинный) и FALSE (ложный) в программах писать заглавными буквами.



При написании длинных, сложных условных конструкций вы можете забыть поставить открывающую или закрывающую скобку, что приведет к ошибке или непредсказуемым результатам. Используйте какую-либо систему (например, пишите условные конструкции с отступом, как в наших сценариях), позволяющую создавать аккуратный и понятный код.

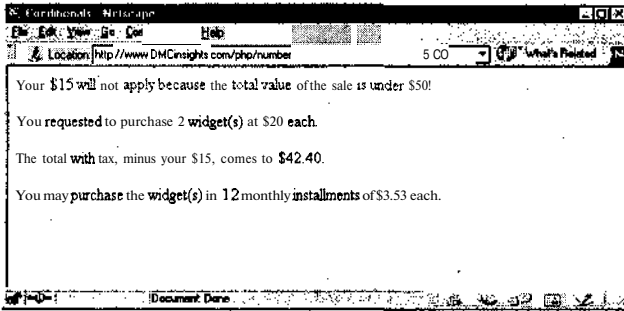


Рис. 6.5 ↑ Теперь пользователь получает сообщение о том, что к его покупке скидка не применяется. При этом калькулятор выглядит вполне профессионально

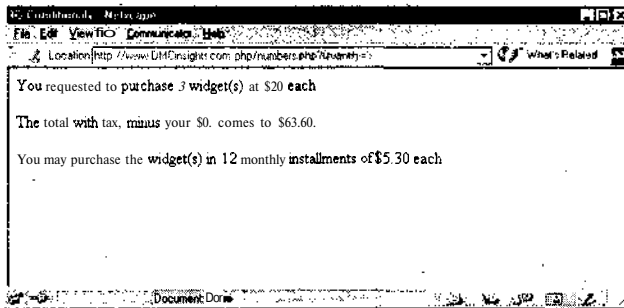


Рис. 6.6 ↓ Даже если стоимость покупки меньше \$50, пользователь не получает сообщения, так как переменная \$Discount не имеет значения

Использование оператора if-else

Следующий логический оператор – это if-else, иногда называемый if-then-else. Он позволяет задавать условие, при соблюдении которого будет выполнена одна инструкция, а в противном случае – другая.

```
if (condition) {  
    statement(s);  
}
```

```

    } else {
        statement(s)2;
    }

```

Необходимо запомнить, что при любом невыполнении условия `if` будет исполнен блок инструкций после `else`. Другими словами, инструкции после `else` Представляют собой действия по умолчанию, а блок инструкций после `if` – исключение из правила. Теперь можно переписать страницу `numbers.php`, вставив в условную конструкцию оператор `else`.

Использование оператора `else`

1. Откройте файл `numbers.php` в текстовом редакторе (листинг 6.4).
2. Сразу же после закрывающей фигурной скобки условной конструкции `if` (строка 30), напишите следующее:

```

    } else { print ("Please make sure that you have entered both
        → a quantity and an applicable discount and then resubmit.\n"); }

```

Теперь, если страница не получила значение количества, будет распечатано соответствующее сообщение об ошибке.

3. Сохраните сценарий (листинг 6.5), загрузите его на сервер и протестируйте в браузере (рис. 6.7).

Листинг 6.5 Часто имеет смысл использовать конструкцию `if-else` вместо простого оператора `if`, так как обычно нужно выполнять **какие-либо** действия и в том **случае**, если условие не выполнено.

```

1  <HTML>
2  <HEAD>
3  <TITLE>Conditionals</TITLE>
4  </HEAD>
5  <BODY>
6  <?php
7  /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8  $Cost = 20.00;
9  $Tax = 0.06;
10 if ($Quantity) {
11     $Quantity = abs($Quantity);
12     $Discount = abs($Discount);
13     $Tax++; // $Tax составляет 1.06.
14     $TotalCost = ($Cost * $Quantity);
15     if ( ($TotalCost < 50) AND ($Discount) ) {
16         print ("Your.\$Discount will not apply because the total value
           of the sale is under \$50!\n<P>");
17     }
18     if ($TotalCost >= 50) {
19         $TotalCost = $TotalCost - $Discount;
20     }
21     $TotalCost = $TotalCost * $Tax;
22     $Payments = round ($TotalCost, 2) / 12;

```

```

23 // Печать результатов.
24 print ("You requested to purchase $Quantity widget(s) at \$$Cost
    each.\n<P>");
25 print ("The total with tax, minus your \$$Discount, comes to $");
26 printf ("%01.2f", $TotalCost);
-27 print (".\n<P>You may purchase the widget(s) in 12 monthly
    installments of $");
28 printf ("%01.2f", $Payments);
29 print (" each.\n<P>");
30 } else {
31     print ("Please make sure that you have entered both a quantity and
        an applicable discount and then resubmit.\n"); }
32 ?>
33 </BODY>
34 </HTML>
35

```

Вы можете добавить условную конструкцию с оператором if-else к сценарию `numbers.php`, чтобы распечатывалось сообщение, если скидка не предоставляется. Также допустимо использовать конструкцию if-else для печати слова «штука» (widget) при заказе одного экземпляра и слова «штуки» (widgets) в остальных случаях.

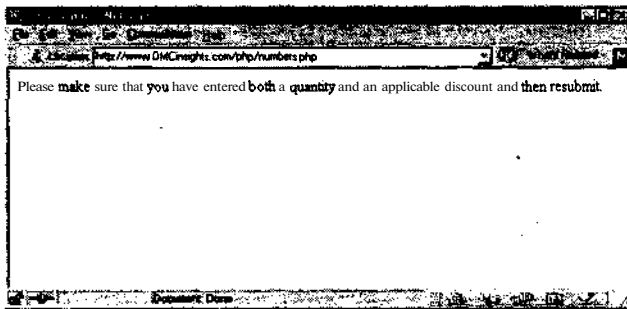


Рис. 6.7 Наш сайт стал еще более профессиональным: теперь он соответствующим образом «реагирует» на действия пользователя и системы. Условные конструкции помогают работать с предположениями, которые могут исполниться или нет

Использование конструкции if-elseif

Условная конструкция if-elseif (или if-elseif-else) совершенно аналогична оператору if (или if-else). Работает она точно так же, как if, добавляя проверку еще одного условного выражения и еще одного блока инструкций, и может быть расширена:

```

if (conditional) {
    statement(s);
}

```

```

    } elseif (conditional2) {
        statement(s)2;
    }

```

Другой пример:

```

if (conditional) {
    statement(s);
} elseif (conditional2) {
    statement(s)2;
} else {
    statement(s)3;
}

```

Мы создадим новую страницу `hello.php`, похожую на приведенную в главе 3. С помощью условной конструкции `if-elseif` и функции `date()` будет печататься усложненное приветствие пользователю.

Использование конструкции `elseif`

1. Создайте новый PHP-документ в текстовом редакторе.
2. Напишите HTML-заголовок и откройте PHP-раздел страницы.

```
<HTML><HEAD><TITLE>If-elseif Conditionals</TITLE><BODY><?php
```

3. Создайте главную условную конструкцию `if`.

```
if ($Username) {
```

Приветствие будет напечатано, только если известно имя пользователя.

4. `print ("Good ");`

С помощью этого кода первая часть приветствия печатается отдельно от трех последующих частей. Таким образом, если позже потребуется сменить конкретные слова, это можно сделать в одном месте.

5. `if (date("A") -== "AM") {`

Функция `date()` используется для определения любой конкретной информации о дате (дне недели, месяце и т.п.) исходя из полученного параметра. Здесь `date("A")` возвращает значение "AM" (до полудня) или "PM" (после полудня). Функция `date()` более подробно описана в главе 13.

6. `print ("morning, ");`
`} elseif ((date("H") >= 12) and (date("H") < 18)) {`

Функция `date("H")` возвращает время суток в формате AM/PM. Поэтому между полуднем и 18:00 будет использовано приветствие «Добрый день».

7. `print ("afternoon, ");`
`} else {`
`print ("evening, ");`

Если сейчас не утро и не день, значит, вечер или ночь. По умолчанию используется инструкция после else, если ни одно другое условие не было удовлетворено.

```
8. } // Закрывать if даты.  
   print ("{$Username}");  
   print ("!\n");  
   } else {  
       print ("Please log in.\n");
```

Если неизвестно имя пользователя, вы просите его зарегистрироваться перед тем, как продолжать работу.

```
9. } // Закрывать if имени пользователя.
```

Комментарии помогают ориентироваться в сложных и вложенных условных конструкциях, помогая закрывать их соответствующим образом.

10. Сохраните сценарий как `hello.php` (листинг 6.6), загрузите его на сервер и протестируйте в браузере (рис. 6.8 и 6.9).

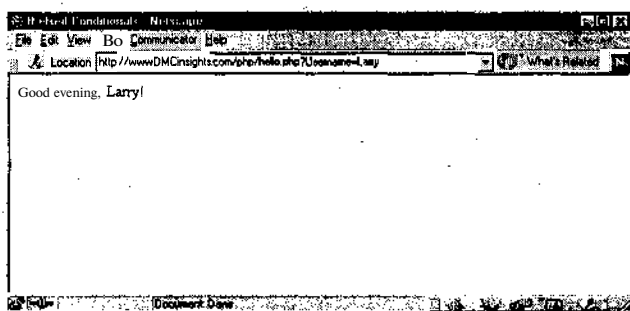


Рис. 6.8 т Вы видите такое приветствие, если страница получила значение имени из поля формы и время на сервере - между 18:00 и полуночью

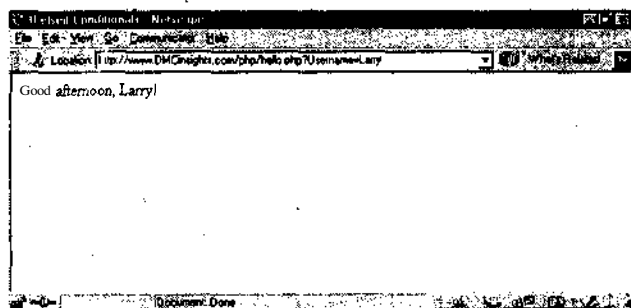


Рис. 6.9 т Вы видите такое приветствие, если страница получила значение имени из поля формы, а время на сервере - между полуднем и 18:00

Листинг 6.6 ▼ Мы вложили условие `if-elseif-else` в `if-else`. Это вполне приемлемо, главное, использовать правильный синтаксис (отступ последующих строк помогает не ошибиться).

```

1  <HTML>
2  <HEAD>
3  <TITLE>If-elseif Conditionals</TITLE>
4  <BODY>
5  <?php
6  if ($Username) {
7      print ("Good ");
8      if (date("A") == "AM") {
9          print ("morning, ");
10     } elseif ( ( date("H") >= 12 ) and ( date("H") < 18 ) ) {
11         print ("afternoon, ");
12     } else {
13         print ("evening, ");
14     } // Закреть if даты.
15         print ("{$Username}");
16         print ("!\n");
17     } else {
18         print ("Please log in.\n");
19     } // Закреть if имени пользователя
20 ?>
21 </BODY>
22 </HTML>

```



Оператор `else` всегда необходимо использовать в последней части условной конструкции, так как условие `else` выполняется только в том случае, если ни одно из предыдущих не было удовлетворено.



Как часть условной конструкции `if`, условия `elseif` можно использовать любое количество раз.



В PHP можно писать `elseif` в два слова, если вам так больше нравится:

```

if (condition) {
    statement(s);
} else if (condition2) {
    statement(s)2;
}

```



Функция `date()` очень полезна, но она отражает время на сервере, а не там, где находится пользователь. К тому же, если время на сервере установлено неверно, функция `date()` выдаст неправильные дату и время.

Условная конструкция `switch`

Если вы обнаружили, что ваши условные выражения `if-elseif-else` становятся все более сложными, самое время познакомиться с оператором `switch`. Он

поможет сэкономить массу времени и упростить программирование. Конструкция switch проверяет на совпадение значение переменной с некоторыми образцами строк. При совпадении выполняются следующие операторы:

```
switch ($Variable) {  
    case "value1":  
        statement(s)1;  
        break;  
    case "value1":  
        statement(s)2;  
        break;  
    default:  
        statement(s)3;  
        break;  
}
```

Важно понять, как работает конструкция switch. В PHP начинается последовательное сравнение значения переменной с указанными вариантами, и, когда найдено полное соответствие, выполняются следующие за двоеточием инструкции. Это происходит до тех пор, пока не закончится конструкция switch (закрывающая фигурная скобка) или не появится оператор break, после чего прекращается выполнение всех инструкций блока switch. Следовательно, каждый вариант (а также вариант по умолчанию) очень важно закрывать инструкцией break.

Вышеприведенной конструкцией switch выполняются точно те же проверки и операторы, что и аналогичной, но более громоздкой конструкцией:

```
if ($Variable == "value1") {  
    statement(s)1;  
} elseif ($Variable == "value2") {  
    statement(s)2;  
} else {  
    statement(s)3;  
}
```

Еще раз поясню порядок выполнения сравнений и соответствующих операторов в конструкции switch. Во-первых, сравнения производятся последовательно: с самого первого и далее вниз. Во-вторых, при первом же совпадении начинают выполняться указанные операторы вплоть до оператора break или до конца блока. Если совпадений не найдено, выполняются инструкции, идущие за специальным оператором default:.

В следующем разделе мы используем конструкцию switch вместе с циклом для создания **HTML-формы**, в которой можно будет выбрать день месяца. Здесь же, для демонстрации возможностей условия switch, мы напишем простой сценарий, который печатает сообщение на основе выбора, сделанного пользователем в HTML-форме.

Использование конструкции switch

1. Создайте новый HTML-документ в текстовом редакторе.
2. Начните со стандартного HTML-заголовка.

```
<HTML><HEAD><TITLE>HTML Contact Form</TITLE></HEAD><BODY>
```

3. Создайте форму, которая предоставляет пользователю возможность выбрать вариант связи с ним.

```
<FORM ACTION="HandleContact.php" METHOD=POST>
First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20xBR>
Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=20xBR>
How would you prefer to be contacted: <SELECT NAME="ContactHow">
<OPTION VALUE="">Select One:</OPTION>
<OPTION VALUE="Telephone">Telephone</OPTION>
<OPTION VALUE="Mail">Mail</OPTION>
<OPTION VALUE="E-Mail">E-Mail</OPTION>
<OPTION VALUE="Fax">Fax</OPTION>
</SELECT><BR>
```

Это может рассматриваться как часть системы с обратной связью для большего Web-приложения. Такой документ и обрабатывающая его страница определяют два первых этапа процесса обратной связи. Здесь пользователь вводит свое имя и выбирает способ контакта.

4. Создайте в форме окно для комментариев, затем закройте форму и HTML-документ.

```
Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
</FORM></BODY></HTML>
```

5. Сохраните форму как contact.html (листинг 6.7) и загрузите ее на сервер.

Листинг 6.7 т В этой HTML-форме для предоставления пользователю списка опций используется ниспадающее меню.

```
1  <HTML>
2  <HEAD>
3- <TITLE>HTML Contact Form</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleContact.php" METHOD=POST>
7  First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20xBR>-
8  Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=20xBR>
9  How would you prefer to be contacted: <SELECT NAME="ContactHow">
10 <OPTION VALUE="">Select One:</OPTION>
11 <OPTION VALUE="Telephone">Telephone</OPTION>
12 <OPTION VALUE="Mail">Mail</OPTION>
13 <OPTION VALUE="E-Mail">E-Mail</OPTION>
14 <OPTION VALUE="Fax">Fax</OPTION>
15 </SELECT><BR>
16 Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40x/TEXTAREAxBR>
```

```

17 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
18 </FORM>
19 </BODY>
20 </HTML>

```

6. Теперь необходимо создать страницу, которая начнет обрабатывать ввод пользователя со страницы contact.html.
7. Создайте новый PHP-документ в текстовом редакторе.
8. Начните со стандартного HTML-заголовка.

```
<HTMLxHEADxTITLE>Contact Information Request</TITLEx/HEADxBODY>
```

9. Создайте HTML-форму и откройте раздел PHP.

```
<FORM ACTION="HandleContact2.php" METHOD=POSTx?php
```

Чтобы получить больше информации от пользователя, мы обратимся к другой форме.

10. Сохраните полученные из сценария contact.html значения в невидимых элементах формы.

```

print ("<INPUT TYPE=HIDDEN NAME=\"FirstName\"
→VALUE=\"\$FirstName\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"LastName\"
→VALUE=\"\$LastName\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"Comments\"
→VALUE=\"\$Comments\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"ContactHow\"
→VALUE=\"\$ContactHow\">\n");

```

В главе 3 упоминалось, что можно передавать разнообразную информацию с помощью поля ввода с типом HIDDEN (скрытый). Здесь все данные, собранные из сценария contact.html, помещаются в скрытые элементы, чтобы быть переданными на следующую страницу - HandleContact2.php.

11. Создайте конструкцию switch, которая в зависимости от сделанного пользователем выбора в contact.html функционирует по-разному.

```

switch ($ContactHow) {
    case "Telephone":
        print("<B>Please enter a daytime phone number where you can be
        →reached:</B><BR>\n");
        print ("<INPUT TYPE=TEXT NAME=\"Telephone\" SIZE=10><BR>");
        print ("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
        break;
    case "Mail":
        print ("<B>Please enter your complete mailing address:
        </B><BR>\n");
        print ("<TEXTAREANAME=\"MailAddress\" ROWS=5 COLS=40>
        →<TEXTAREA><BR>\n");
        print ("<INPUTTYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
        break;
    case "E-Mail":
        print("<B>Please enter your E-Mail address:</B><BR>\n");

```

```

        print("<INPUT TYPE=TEXT NAME=\"E-Mail\" SIZE=40><BR>\n");
        print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
        break;
    case "Fax":
        print("<B>Please enter your Fax number:</B><BR>\n");
        print("<INPUT TYPE=TEXT NAME=\"Fax\" SIZE=10><BR>\n");
        print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
        break;
    default:
        print("<B>Please go back and select how you would prefer to be
        -contacted!</B><BR>\n");
        break;
}

```

Конструкция `switch` создана для вывода на дисплей текста, зависящего от значения переменной `$ContactHow`. Если пользователь хочет, чтобы с ним связались по телефону или по электронной почте, будет запрошен номер телефона или e-mail соответственно. Если переменная `$ContactHow` не имеет значения, то будет инициирован вариант по умолчанию и пользователь получит сообщение с просьбой вернуться на страницу `contact.html` и выбрать вариант контакта с ним.

12. Закройте раздел PHP, затем форму и HTML-страницу.

```
?></FORMx/BODYx/HTML>
```

13. Сохраните сценарий как `HandleContact.php` (листинг 6.8), загрузите его на сервер в один каталог с `contact.html` и протестируйте обе страницы в браузере (рис. 6.10-6.14).

Листинг 6.8 ▼ Конструкцией `switch` в этом сценарии используется значение переменной `$ContactHow`. Таким образом определяется, какую информацию запросить у пользователя: номер телефона, факса, адрес электронной почты или почтовый адрес. Типы скрытого ввода используются также для передачи других существующих значений.

```

1 <HTML>
2 <HEAD>
3   <TITLE>Contact Information Request</TITLEx/HEAD>
4 <BODY>
5 <FORM ACTION="HandleContact2.php" METHOD=POST>
6 <?php
7   // Передача принятого значения с помощью поля INPUT типа HIDDEN.
8   print("<INPUT TYPE=HIDDEN NAME=\"FirstName\" VALUE=\"\$FirstName\">\n");
9   print("<INPUT TYPE=HIDDEN NAME=\"LastName\" VALUE=\"\$LastName\">\n");
10  print("<INPUT TYPE=HIDDEN NAME=\"Comments\" VALUE=\"\$Comments\">\n");
11  print("<INPUT TYPE=HIDDEN NAME=\"ContactHow\" VALUE=\"\$ContactHow\">\n");

```

```

12
13 switch ($ContactHow) {
14     case "Telephone":
15         print("<B>Please enter a daytime phone number where you can be
16           reached:</B><BR>\n");
17         print("<INPUT TYPE=TEXT NAME=\"Telephone\" SIZE=10><BR>v");
18         print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
19         break;
20     case "Mail":
21         print("<B>Please enter your complete mailing address:</B>
22           <BR>\n");
23         print("<TEXTAREA NAME=\"MailAddress\" ROWS=5 COLS=40>
24           <TEXTAREA><BR>\n");
25         print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
26         break;
27     case "E-Mail":
28         print("<B>Please enter your E-Mail address:</B><BR>\n");
29         print("<INPUT TYPE=TEXT NAME=\"E-Mail\" SIZE=40><BR>\n");
30         print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
31         break;
32     case "Fax":
33         print("<B>Please enter your Fax number:</B><BR>\n");
34         print("<INPUT TYPE=TEXT NAME=\"Fax\" SIZE=10><BR>\n");
35         print("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Continue\">\n");
36         break;
37     default:
38         print("<B>Please go back and select how you would prefer to be
39           contacted!</B><BR>\n");
40         break;
41 }
42 }
43 </FORM>
44 </BODY>
45 </HTML>

```

First Name: Larry

Last Name: Ullman

How would you prefer to be contacted: E-Mail

Comments: I'm using the switch conditional to help handle an HTML form.

Submit

Рис. 6.10 Это HTML-форма, созданная файлом contact.html. Первый шаг к созданию системы с обратной связью

Рис. 6.11 ▼ Так как я попросил связываться со мной по электронной почте (см. рис. 6.10), на странице HandleContact.php появится запрос на мой адрес и на экран будет выведено текстовое поле, куда можно ввести искомую информацию

Рис. 6.12 т Это исходный HTML-код страницы, представленной на рис. 6.11. Обратите внимание на то, что в скрытых полях хранится ранее собранная информация, которая будет передана в сценарий HandleContact2.php

Рис. 6.13 т Если бы я выбрал для связи обычную почту, на экране появились бы запрос на почтовый адрес и поле для ввода нужного текста

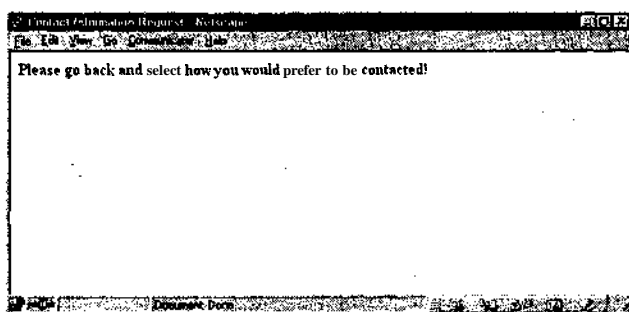


Рис. 6.14 Если пользователь не выберет вариант связи, он получит это сообщение и не сможет двигаться дальше



Мы еще не создали страницу `HandleContact2.php`, которая обрабатывала бы результаты сценария `HandleContact.php` (как было упомянуто, мы собирались разработать только два этапа данного процесса). Если вы хотите, чтобы страница `HandleContact.php` функционировала как следует, напишите простой PHP-сценарий, который будет печатать все полученные значения.



В конструкциях `switch` вариант по умолчанию можно не использовать (допустимо задать условие так, что ничего не произойдет, если ни один вариант не совпал с заданным значением). С другой стороны, если вы установили некий вариант по умолчанию, то он должен быть последним.



Если в конструкциях `switch` используются строки, помните, что они чувствительны к регистру и значение `Value` не будет соответствовать значению `value`.

Цикл while

Как было упомянуто в начале этой главы, циклы используются для многократного выполнения фрагмента кода. Можно создать ниспадающее меню, состоящее из дней месяца (цифры от 1 до 31). Допустимо распечатывать все значения массива. В этих и во многих других случаях удобно использовать *цикл*.

Первый из двух существующих в PHP циклов - цикл `while` - выполняет соответствующий блок операторов до тех пор, пока заданное условие истинно. Он проверяет значение условия перед каждой итерацией. Как только условие становится ложным, программа покидает цикл.

```
while (condition) {
    statement(s);
}
```

Для демонстрации цикла `while` мы создадим сценарий, который динамически генерирует ниспадающее меню даты (месяц, день, год) для HTML-формы.

Хотя форма сама по себе не выполняет никаких операций, вы увидите, как использовать PHP для быстрого создания и усовершенствования стандартного элемента HTML-формы.

Использование цикла while

1. Создайте новый PHP-документ в текстовом редакторе.

```
<HTML><HEAD><TITLE>Select Menu</TITLE></HEAD><BODY><?php
```

2. Задайте текущий год с помощью функции date ().

```
$Year = date ("Y");
```

При поступлении в функцию date () значения "Y" будет возвращаться текущий год. Мы используем это значение для вывода в меню текущего года и следующих десяти лет. В этом случае не придется менять форму каждый год.

3. Создайте HTML-форму, у которой будет ниспадающее меню с месяцами.

```
print ("<FORM ACTION=\"$_PHP_SELF\" METHOD=POST>\n");
print ("Select a month:<BR>\n");
print ("<SELECT NAME=Month><OPTION>Choose One</OPTION>\n");
print ("<OPTION VALUE=January>January</OPTION>\n");
print ("<OPTION VALUE=February>February</OPTION>\n");
print ("<OPTION VALUE=March>March</OPTION>\n");
print ("<OPTION VALUE=April>April</OPTION>\n");
print ("<OPTION VALUE=May>May</OPTION>\n");
print ("<OPTION VALUE=June>June</OPTION>\n");
print ("<OPTION VALUE=July>July</OPTION>\n");
print ("<OPTION VALUE=August>August</OPTION>\n");
print ("<OPTION VALUE=September>September</OPTION>\n");
print ("<OPTION VALUE=October>October</OPTION>\n");
print ("<OPTION VALUE=November>November</OPTION>\n");
print ("<OPTION VALUE=December>December</OPTION>\n");
```

4. С помощью цикла while создайте ниспадающее меню с днями.

```
print ("<P>Select a day:<BR>\n");
print ("<SELECT NAME=Day><OPTION>Choose One</OPTION>\n");
$Day = 1;
while ($Day <= 31) {
    print ("<OPTION VALUE=$Day>$Day</OPTION>\n");
    $Day++;
}
print ("</SELECT>\n");
```

Первым делом присвоим переменной \$Day значение 1. Это должно быть сделано до вызова цикла. Затем будет происходить автоматическая проверка того, равна ли переменная \$Day 31 или нет. Если да, то значение \$Day будет напечатано как опция в меню дней, а затем значение \$Day увеличится на 1. Этот процесс будет продолжаться, пока значение не составит 32. Тогда произойдет выход из цикла и начнется выполнение дальнейших инструкций сценария.

5. С помощью другого цикла while создайте ниспадающее меню с годами.

```
print ("<P>Select a year:<BR>\n");
print ("<SELECT NAME=Year><OPTION>Choose One</OPTION>\n");
$EndYear = $Year + 10;
while ($Year <= $EndYear ) {
    print ("<OPTION VALUE=$Year>$Year</OPTION>\n");
    $Year++;
}
print ("</SELECT>\n");
```

В ниспадающем меню будут сгенерированы текущий год и последующие десять лет. Первый был присвоен переменной \$Year ранее. Последний год, \$EndYear, получает значение переменной \$Year плюс ГО (это значение легко изменить для печати 5 или 15 лет). Цикл «знает», что, пока значение переменной \$Year меньше или равно значению переменной \$EndYear, он должен печатать значение переменной \$Year и затем увеличивать его на единицу.

6. Создайте кнопку **Submit**, закройте форму, PHP и HTML.

```
print ("<P><INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Go!\n">
-</FORM>\n");?></BODY></HTML>
```

Хотя данная страница была разработана как часть более сложной HTML-формы и поэтому ничего здесь не выполняется автоматически, всегда стоит быть последовательным. Вот мы и добавили кнопку **Submit**.

7. Сохраните страницу как **select.php** (листинг 6.9), загрузите ее на сервер и протестируйте в браузере (рис. 6.15 и 6.16).

Листинг 6.9 Два цикла while быстро сгенерируют два ниспадающих меню (см. рис. 6.16). С помощью функции date() мы разработаем цикл по годам. Он не позволит устареть нашему сценарию.

```
1  <HTML>
2  <HEAD>
3  <TITLE>Select Menu</TITLE></HEAD>
4  <BODY>
5  <?php
6  $Year = date ("Y");
7  // Создание формы.
8  print ("<FORM ACTION=\"$_PHP_SELF\" METHOD=POST>\n");
9  // Создание меню для выбора месяца.
10 print ("Select a month:<BR>\n");
11 print ("<SELECT NAME=Month><OPTION>Choose One</OPTION>\n");
12 print ("<OPTION VALUE=January>January</OPTION>\n");
13 print ("<OPTION VALUE=February>February</OPTION>\n");
14 print ("<OPTION VALUE=March>March</OPTION>\n");
15 print ("<OPTION VALUE=April>April</OPTION>\n");
16 print ("<OPTION VALUE=May>May</OPTION>\n");
17 print ("<OPTION VALUE=June>June</OPTION>\n");
18 print ("<OPTION VALUE=July>July</OPTION>\n");
```

```

19 print ("<OPTION VALUE=August>August</OPTION>\n");
20 print ("<OPTION VALUE=September>September</OPTION>\n");
21 print ("<OPTION VALUE=October>October</OPTION>\n");
22 print ("<OPTION VALUE=November>November</OPTION>\n");
23 print ("<OPTION VALUE=December>December</OPTION>\n");
24 print ("</SELECT>\n");
25 // Создание меню для вывода дня.
26 print ("<P>Select a day:<BR>\n");
27 print ("<SELECT NAME=Day><OPTION>Choose One</OPTION>\n");
28 $Day = 1;
29 while ($Day <= 31) {
30     print ("<OPTION VALUE=$Day>$Day</OPTION>\n");
31     $Day++;
32 }
33 print ("</SELECT>\n");
34 // Создание меню для вывода года.
35 print ("<P>Select a year:<BR>\n");
36 print ("<SELECT NAME=Year><OPTION>Choose One</OPTION>\n");
37 $EndYear = $Year + 10;
38 while ($Year <= $EndYear) {
39     print ("<OPTION VALUE=$Year>$Year</OPTION>\n");
40     $Year++;
41 }
42 print ("</SELECT>\n");
43 print ("<P><INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Go!\"></FORM>\n");
44 ?>
45 </BODY>
46 </HTML>

```



Можно использовать также цикл `do...while`, гарантирующий по крайней мере однократное выполнение инструкций, что не всегда удастся в цикле `while`:

```

do {
    statement(s);
} while (condition);

```

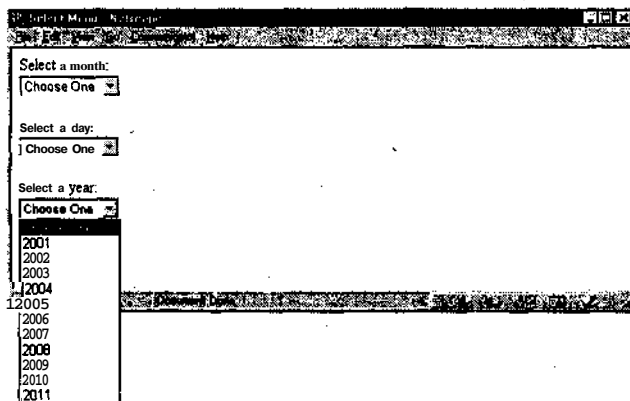


Рис. 6.15 ▾ Страница `select.php` генерирует три ниспадающих меню, позволяя пользователю выбрать месяц, день и год. В любое время этот код можно скопировать в более сложную HTML-форму

Дважды подумайте о том, какие элементы стоит включить в цикл, а какие нужно вынести за его пределы. Невозможность увеличить значение переменной \$Day или \$Year в цикле создаст бесконечный цикл (например, день всегда будет меньше или равен 31). В противоположность этому использование в цикле тэгов <SELECT> приведет к созданию многочисленных ниспадающих меню.

[illegible]

Рис. 6.16▼ Это исходный текст формы, изображенной на рис. 6.15. Обратите внимание на то, как **десяток** строк из листинга 6.9 (содержащего два цикла **while**) сгенерировали 40-50 строк HTML (меню выбора дня и года)



Так же, как и условная конструкция `if`, цикл `while` может быть размещен на одной строке, если имеется всего одна инструкция. Однако лучше не делать этого.

Цикл `for`

Цикл `for` предназначен для выполнения блока инструкций определенное количество раз (в отличие от `while`, который выполняется, пока условие не станет ложным). Для этого в цикле обычно используют специальную переменную, иногда называемую *счетчиком цикла*. Синтаксис цикла `for` более сложен, чем цикла `while`. Хотя они совпадают во многих отношениях, порой первый более пригоден для одних задач, а второй - для других.

```
for (начальное выражение; условие; завершающее выражение) {
    statement(s);
}
```

Для создания цикла необходимо указать три выражения и задать переменную - счетчик цикла. При инициализации цикла будет один раз выполнено начальное выражение. Затем проверяется условие, нужно ли выполнять сопутствующий блок инструкций. И наконец, завершающее выражение выполняется каждый раз, когда условие будет определено как истинное, но только после обработки всех инструкций из блока цикла. Поэтому для печати всех значений в массиве необходимо написать следующее:

```
for ($n = 0; $n < count($Array); $n++) {
    print (" $Array[$n]<BR>\n");
}
```

Как видите, все команды для организации цикла собраны в одном месте и точно соответствуют синтаксису такого же цикла в других языках программирования. Для лучшего понимания синтаксиса цикла `for` я переписал цикл `while` для переменной `$Day` из листинга 6.9 как цикл `for`. Исходный код был следующим:

```
$pay = 1;
while ($Day <= 31) {
    print ("<OPTION VALUE=$Day>$Day</OPTION>\n");
    $Day++;
}
```

Сначала было присвоено значение переменной `$Day`, затем задано условие (`$Day <= 31`). Если возвращаемое значение истинно, то выполняется инструкция `print()` и значение переменной `$Day` увеличивается на единицу. В цикле `for` тот же код выглядит следующим образом:

```
for ($Day = 1; $Day <= 31; $Day++;) {
    print ("<OPTIONVALUE=$Day>$Day</OPTION>\n");
}
```

Распространенный пример использования цикла for - печать всех простых чисел от 1 до 1000.

Написание цикла for

1. Создайте новый PHP-документ в текстовом редакторе.

```
<HTML><HEAD><TITLE>Prime Numbers</TITLE></HEAD><BODY><?php
```

2. Начните цикл for.

```
for ($n = 1; $n <= 1000; $n++) {
```

В соответствии с синтаксисом цикла for переменная \$n объявлена переменной цикла. Ей изначально присваивается значение 1. Затем циклом проверяется, чему равно значение переменной \$n: оно должно быть меньше или равно 1000. Другими словами, этот цикл выполняется 1000 раз. В конце концов, если условие цикла ($\$n \leq 1000$) истинно, обрабатывается блок инструкций в фигурных скобках, затем значение \$n увеличится на единицу и процесс начинается заново.

3. Теперь напишем само тело цикла.

```
if ( ($n == 1) OR ($n == 2) OR ($n == 3) OR ($n == 5) ) {  
    print("$n<BR>\n");  
} elseif (($n % 2 != 0) AND ($n % 3 != 0) AND ($n % 5 != 0)) {  
    print("$n<BR>\n");  
}
```

Число считается простым, если оно делится без остатка только на само себя и на единицу. Другими словами, если разделить простое число на любое другое, кроме него самого и единицу, то всегда будет оставаться остаток. Например, 4 делится на два без остатка, поэтому оно не простое, а 7 не делится без остатка на 2, 3 или 5, поэтому это простое число. Всем известно, что 1, 2, 3 и 5 - простые числа, поэтому, если значение переменной \$n равно одному из них, оно будет распечатано автоматически. Обратите внимание на то, что мы использовали оператор OR, так как, если одно из этих условий верно, значение \$n будет напечатано.

Если значение \$n не равно 1, 2, 3 или 5, то для определения того, простое ли это число, его необходимо проверить. Число делят на 2, 3 или 5 и смотрят, есть ли в результате остаток. При этом используется оператор остатка, или, как говорят математики, взятия по модулю (%). Данный оператор просто возвращает остаток от деления первого числа на второе. Хотя это не самый часто используемый оператор, здесь он бесценен. Мы также ограничились только проверкой на деление на 3 простых чисел в начале ряда, так что в наш список может попасть несколько «непростых» чисел, но более строгую проверку на простоту числа оставим для других курсов по программированию. Если после деления переменной \$n на 2, 3 или 5 возвращается отличный от нуля остаток, то перед нами, возможно, простое число, которое должно быть напечатано.

Обратите внимание на то, что мы использовали оператор AND. Следовательно, если одно из этих условий ложно, то анализируемое число - не простое.

4. Закройте цикл, PHP и HTML.

```
}
?>
</BODY>
</HTML>
```

5. Сохраните сценарий под именем primes.php (листинг 6.10), загрузите его на сервер и протестируйте в браузере (рис. 6.17).

Листинг 6.10 т Этот короткий сценарий - простой пример применения цикла for, повторяющего процесс 1000 раз.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Prime Numbers</TITLE></HEAD>
4 <BODY>
5 <?php
6 // Если вы хотите напечатать больше простых
  чисел, измените это значение.
7 for ($n = 1; $n <= 1000; $n++) {
8   if ( ($n == 1) OR ($n == 2) OR ($n == 3)
9     OR ($n == 5) ) {
10    print("$n<BR>\n");
11  } elseif ((($n % 2 != 0) AND ($n % 3 != 0)
12    AND ($n % 5 != 0)) {
13    print("$n<BR>\n");
14  } // Заккрытие IF.
15 } // Заккрытие FOR.
16 ?>
17 </BODY>
18 </HTML>
```

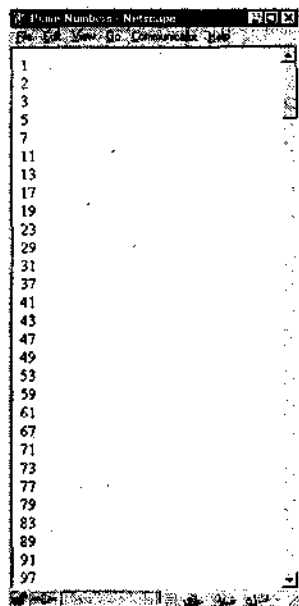


Рис. 6.17 т Это список простых чисел, автоматически рассчитанных и распечатанных циклом for. 16 строк листинга 6.10 создали приблизительно 300 строк в окне HTML

Хотя существует много задач, для решения которых можно применять как цикл for, так и while, постепенно приходит понимание того, в каких ситуациях более уместно употребить тот или иной цикл. Цикл while часто используется для извлечения данных из базы (см. главу 11), а цикл for - для работы с массивами (более подробная информация о них представлена в главе 7).

Не все числа, которые печатает сценарий 6.10, являются простыми (49, например). Чтобы устранить данный недостаток, надо проверять, делится ли переменная \$n на все найденные простые числа.

7 Глава

Массивы

Последний обсуждаемый в настоящем издании тип переменных - *массивы*. (Объекты не включены в книгу, но как только вы освоите PHP, то наверняка захотите узнать, как с ними работать.)

Массивы сложны, но очень **полезны**. Это набор многочисленных значений, собранных в одну переменную. Массив может состоять из чисел и/или строк (и/или других массивов), что позволяет этой одной переменной содержать гораздо больше информации, чем простая строка или число. Еще важнее то, что информация в массиве логично организована (проиндексирована по ключу) и может быть легко извлечена. Например, если необходимо создать список товаров овощного магазина, с помощью строк мы напишем такой код:

```
$Item1 = "apples"  
$Item2 = "bananas"  
$Item3 = "oranges"
```

Для каждого товара необходимо создавать новую строку, а работать с множеством этих строк как со списком **практически** невозможно. Лучше облегчить себе жизнь, поместив весь список в один массив (скажем, `$Items`). Подобный список разрешается пополнять, сортировать, в нем можно проводить поиск и т.д. Имеет смысл подробно рассмотреть синтаксис массивов.

Анализируемые в других главах типы переменных - числа и строки - имеют имя переменной и значение (например, переменная `$FirstName`, которая может иметь значение "Larry"). Массивы имеют имя, образованное по тем же правилам (знак доллара, за ним буква или символ подчеркивания, далее любая комбинация букв, цифр и символов подчеркивания), но рассматриваемая переменная в отличие от других типов может **содержать** множество элементов. Каждый элемент массива представляет собой пару значений - собственно значение элемента, а также значение индекса, или ключа (эти термины взаимозаменяемы). Через ключ осуществляется доступ к значению элемента. В качестве

ключа массива может использоваться число или строка - это зависит от того, как вы собираетесь использовать эту структуру.

Массив можно представить в виде таблицы с двумя столбцами. В первом столбце будут расположены индексы (в виде номера или названия строки), а во втором - значения этих строк. С помощью индекса легко определить значение, представленное в определенной ячейке таблицы (табл. 7.1 и 7.2).

По сути, формат массива похож на формат других переменных, за исключением выделения ключа квадратными скобками (`[]`) при обращении к конкретному элементу. Следовательно, запись `$Array` обозначает Массив как целое, а `$Array[0]` указывает на первый элемент массива. (Символы строки проиндексированы начиная с нуля, как было упомянуто в главе 5. Также с нуля можно нумеровать элементы массива, что и делается по умолчанию, если не указан другой способ индексации.)

Данная глава посвящена основным приемам работы с массивами. Вы познакомитесь с несколькими ключевыми понятиями и научитесь использовать их в сценариях.

Таблица 7.1 ■ Знакомая каждому электронная таблица - хорошая иллюстрация применения массива. Здесь изображен массив, состоящий из названий глав, индексом (ключом) к которым служат их номера

Index or Key	Value
1	Getting Started with PHP
2	Variables
3	HTML Forms and PHP
4	Using Numbers
5	Using Strings
6	Control Structures
7	Using Arrays

Таблица 7.2 ■ В этой табличной версии массива индексом (или ключом) являются слова, а не числа

Index or Key	Value
Chapter_1	Getting Started with PHP
Chapter_2	Variables
Chapter_3	HTML Forms and PHP
Chapter_4	Using Numbers
Chapter_5	Using Strings
Chapter_6	Control Structures
Chapter_7	Using Arrays

Создание массива

Формальный метод создания массива - использование функции `array()` со следующим синтаксисом:

```
$List = array ("apples", "bananas", "oranges");
```

В этом примере (так как мы не проиндексировали элементы) первый предмет, яблоки, автоматически получит индекс 0, второй – 1, третий – 2. Чтобы присвоить другие значения индекса, их надо явно указать при использовании той же функции `array()`:

```
$List = array (1=>"apples", 2=>"bananas", 3=>"oranges");
```

Значение индекса необязательно должно быть числом, можно использовать и слова. Такой способ индексации иногда более удобен при работе со списками. Создадим массив, в котором будут перечислены первые блюда дня на неделю.

Выполнение действия

1. Создайте новый PHP-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>Using Arrays</TITLE></HEAD><BODY>
```

3. Начните PHP-раздел сценария и при помощи функции `array()` создайте массив.

```
<?php
    $Soups = array(
        "Monday"=>"Clam Chowder",
        "Tuesday"=>"White Chicken Chili",
        "Wednesday"=>"Vegetarian"
    );
```

Это верный формат для инициализации (создания и присвоения значения) массива в PHP с использованием строк в качестве индексов.

4. Отправьте массив в Web-браузер.

```
print (" $Soups<P>\n");
```

5. Закройте PHP и HTML.

```
?></BODY></HTML>
```

6. Сохраните документ как `soups.php` (листинг 7.1), загрузите его на сервер и протестируйте в браузере (рис. 7.1).

Листинг 7.1 В данном случае использование пробелов для удобства чтения кода (при инициализации массива) не вызывает никаких отрицательных последствий.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Using Arrays</TITLE></HEAD>
4 <BODY>
5 <?php
6 $Soups = array(
7     "Monday"=>"Clam Chowder",
8     "Tuesday"=>"White Chicken Chili",
9     "Wednesday"=>"Vegetarian");
```

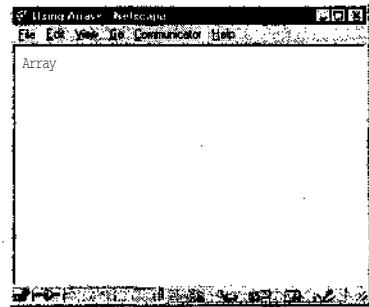


Рис. 7.1 Так как массив принципиально отличается от других типов переменных, результатом запроса на печать массива будет всего лишь печать слова `Array`. Не так полезно, как печатать отдельных элементов массива (об этом позже), зато наглядно подтверждает успешное его создание

```
10 print ("$Soups<P>\n");  
11 ?>  
12 </BODY>  
13 </HTML>
```



Традиция начинать любой индекс с нуля сложилась как в PHP, так и во многих других языках программирования. Поначалу это может показаться неестественным, но следующие два замечания несколько меняют дело. Во-первых, вы можете вручную начинать индексацию с единицы. А во-вторых, попробуйте забыть, что счет начинается с единицы. Выберите наиболее простой для вас способ. Впрочем, многие программисты привыкли к этой странной структуре.

Добавление элементов в массив

Если массив существует, в него можно добавлять дополнительные элементы. Это делается напрямую с помощью оператора присваивания (знак равенства) так же, как присвоение значения строке или числу. При этом можно не задавать ключ добавленного элемента, но в любом случае при обращении к массиву необходимы квадратные скобки. Добавляя два новых элемента к списку \$List, напомним:

```
$List[] = "pears";  
$List[] = "tomatoes";
```

Если ключ не задан, каждый элемент будет добавлен к существующему массиву и проиндексирован следующим порядковым номером. Если мы добавим новые элементы к массиву из предыдущего раздела, элементы которого имели индексы 1, 2 и 3, то у груш (pears) будет индекс 4, а у помидоров (tomatoes) - 5.

Когда вы явно задаете индекс, а значение с ним уже существует, то существовавшее в этом месте значение будет потеряно и заменено новым:

```
$List[3] = "pears";  
$List[4] = "tomatoes";
```

Теперь значение элемента с индексом 4 - «помидоры», а элемента «апельсины» (oranges) больше нет. Я бы посоветовал не указывать ключ при добавлении элементов в массив, если, конечно, вы не хотите специально переписать какие-либо существующие данные. Однако, если в качестве индексов используются строки, ключи нужно указывать обязательно, чтобы не потерять значения.

Мы попробуем добавить в массив новые элементы, переписав сценарий soups.php. Сначала распечатав исходные элементы массива, а затем - исходные вместе с добавленными, мы легко увидим произошедшие изменения.

Подобно тому как можно узнать длину строки (количество содержащихся в ней символов) с помощью функции strlen(), также нетрудно определить количество элементов в массиве с помощью функции count():

```
$HowMany = count($Array);
```

Выполнение действия

1. Откройте файл `soups.php` в текстовом редакторе.
2. После инициализации массива с помощью функции `array()` добавьте следующую запись:

```
$HowMany = count($Soups);
print ("The array contains $HowMany elements.<P>\n");
```

Функцией `count()` будет определено, сколько элементов содержится в массиве `$Soups`. Присвоив это значение переменной, его можно распечатать.

3. Добавьте в массив три дополнительных элемента.

```
$Soups["Thursday"] = "Chicken Noodle";
$Soups["Friday"] = "Tomato";
$Soups["Saturday"] = "Cream of Broccoli";
```

4. Пересчитайте элементы в массиве и распечатайте это значение.

```
$HowManyNow = count($Soups);
print ("The array now contains $HowManyNow elements.<P>\n");
```

5. Сохраните сценарий (листинг 7.2), загрузите его на сервер и протестируйте в браузере (рис. 7.2).

Листинг 7.2 t Можно напрямую добавлять по одному элементу в массив, присваивая каждому элементу значение с помощью соответствующего оператора. С помощью функции `count()` удастся узнать, сколько элементов содержится в массиве.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Using Arrays</TITLE></HEAD>
4 <BODY>
5 <?php
6 $Soups = array(
7   "Monday"=>"Clam Chowder",
8   "Tuesday"=>"White Chicken Chili",
9   "Wednesday"=>"Vegetarian");
10 $HowMany = count($Soups);
11 print ("The array contains $HowMany
    elements.<P>\n");
12 $Soups["Thursday"] = "Chicken Noodle";
13 $Soups["Friday"] = "Tomato";
14 $Soups["Saturday"] = "Cream of
    Broccoli";
15 $HowManyNow = count($Soups);
16 print ("The array now contains
    $HowManyNow elements.<P>\n");
17 ?>
18 </BODY>
19 </HTML>
```

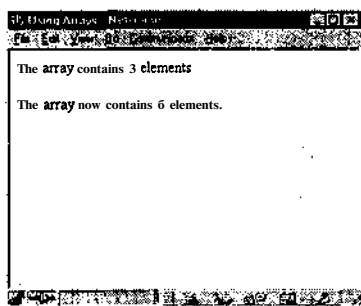


Рис. 7.2 ▼ Простой способ удостовериться в том, что новые компоненты были успешно добавлены в массив, — сосчитать элементы до и после их добавления

В PHP 4.0 появилась новая функция, позволяющая добавлять один массив к другому. Эту операцию можно также назвать слиянием или сцеплением массивов. Функция `array_merge()` вызывается следующим образом:

```
$NewArray = array_merge ($OneArray, $TwoArray);
```

Вы можете переписать страницу `soups.php` с использованием этой функции, если работаете с сервером, на котором установлен PHP 4.0.

Объединение двух массивов

1. Откройте файл `soups.php` в текстовом редакторе, если он еще не открыт.
2. После инициализации массива `$Soups` сосчитайте его элементы и напечатайте результат.

```
$HowMany = count ($Soups);  
print ("The \ $Soups array contains $HowMany elements.<P>\n");
```

3. Создайте второй массив, сосчитайте его элементы и также напечатайте результат.

```
$Soups2 = array(  
    "Thursday"=>"Chicken Noodle",  
    "Friday"=>"Tomato",  
    "Saturday"=>"Cream of Broccoli");  
$HowMany2 = count ($Soups2);  
print ("The \ $Soups2 array contains $HowMany2 elements.<P>\n");
```

4. Объедините два массива в один.

```
$TheSoups = array_merge ($Soups, $Soups2);
```

Проследите, чтобы массивы были расположены именно в этом порядке (`$Soups`, потом `$Soups2`), то есть элементы четверга и пятницы должны быть добавлены к элементам понедельника - среды, а не наоборот.

5. Сосчитайте элементы нового массива и напечатайте результат.

```
$HowMany3 = count ($TheSoups);  
print ("The \ $TheSoups array contains  
-$HowMany3 elements.<P>\n");
```

6. Закройте PHP и HTML-документ.

```
?></BODY></HTML>
```

7. Сохраните файл (листинг 7.3), загрузите его на сервер и протестируйте в браузере (рис. 7.3).

Листинг 7.3 т Функция `Array_merge()` является новой. Это одна из нескольких дополнительных функций PHP 4.0, предназначенная для работы с массивами. Используя массивы, можно значительно сэкономить время.

```
1 <HTML>  
2 <HEAD>
```

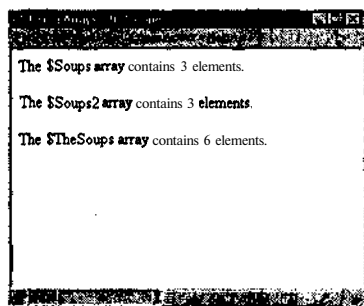


Рис. 7.3 т Функция `count()` используется здесь для определения того, все ли элементы обоих массивов были включены в новый массив

```

3 <TITLE>Using Arrays</TITLE></HEAD>
4 <BODY>
5 <?php
6 $Soups = array(
7     "Monday"=>"Clam Chowder", "Tuesday"=>"White Chicken Chili",
8     "Wednesday"=>"Vegetarian"
9 );
10 $HowMany = count($Soups);
11 print ("The \${$Soups} array contains $HowMany elements.<P>\n");
12 $Soups2 = array(
13     "Thursday"=>"Chicken Noodle",
14     "Friday"=>"Tomato",
15     "Saturday"=>"Cream of Broccoli"
16 );
17 $HowMany2 = count($Soups2);
18 print ("The \${$Soups2} array contains $HowMany2 elements.<P>\n");
19 $TheSoups = array_merge ($Soups, $Soups2);
20 $HowMany3 = count ($TheSoups);
21 print ("The \${$TheSoups} array contains $HowMany3 elements.<P>\n");
22 ?>
23 </BODY>
24 </HTML>

```

Будьте внимательны при добавлении элементов в массив напрямую. Правильно это делается так: `$Array[] = "Add This";` или `$Array[1] = "Add This";`, а неправильно - вот так: `$Array = "Add This";`. Если вы забыли поставить скобки, то добавленное значение уничтожит существующий массив, превратив его в простую строку или число.

В PHP 4.0 есть несколько новых функций для работы с массивами. Не все они рассмотрены в книге. Однако полная информация по этому вопросу содержится в руководстве по языку PHP, которое можно найти на сайте PHP. Будьте внимательны и не используйте новые функции, присущие только PHP 4.0, если на вашем сервере работает PHP 3.x.

Доступ к элементам массива

Независимо от того, как был создан массив, есть только один способ извлечения из него конкретного элемента (точнее, его значения) - обращение к массиву с указанием индекса этого элемента. Один из вариантов работы с массивами - **присвоение** значения Конкретного элемента отдельной переменной с помощью соответствующего оператора:

```
$Total = $Array[1];
```

Однако можно напрямую обратиться к значению конкретного элемента и использовать его во многих случаях так же, как и обычную переменную, указывая индекс в кавычках или без них:

```
print ("The total of your order comes to $Array[Total]");
```

При печати из массива придется опускать кавычки, в которые обычно заключается индекс, так как они конфликтуют с другими кавычками в самой инструкции `print()`. Следующий фрагмент кода вызовет сообщение об ошибке:

```
print ("The total of your order comes to $Array["Total"]");
```

А это пример использования кавычек, которые не вызовут никаких осложнений:

```
$Array["Name"] = trim($Array["Name"]);
```

К сожалению, функциональность, делающая массивы такими полезными, - способность хранить множество значений в одной переменной - несет с собой и дополнительные требования, которые не свойственны другим типам переменных. Так, для доступа к элементам массива необходимо знать их ключи. Если массив был создан из строк как массив `$Soups`, обращение к функции `$Soups[1]` не даст никаких результатов. И, так как переменные чувствительны к регистру, вызов `$Soups["monday"]` вернет пустую строку, потому что запись `Clam Chowder` имеет другой индекс: `$Soups["Monday"]`.

К счастью, существует быстрый и легкий способ доступа ко всем значениям массива: использовать цикл вместе с функцией `each()`. Последняя извлекает из массива значения ключа и собственно элемента массива. Если вызвать функцию `each()` столько раз, сколько элементов содержит массив, то мы пройдем по всем его элементам:

```
for ($n = 0; $n < count($Array); $n++) {
    $Line = each ($Array);
    print ("Key is equal to $Line[key].<BR>Value is equal
    -to $Line[value].");
}
```

В этом примере функция `each()` создает пару значений, то есть массив из двух элементов с именем `$Line`, который содержит ключ и значение для массива `$Array` в его текущем положении. Можно представить, что в массиве есть некий **внутренний** указатель. При первом использовании функции `each()` он находится на самом первом элементе массива `$Array`. Функция `each()` извлечет значения ключа и элемента, затем присвоит их элементу `$Line[key]` (а также `$Line[0]`) и `$Line[value]` (то же самое, что и `$Line[1]`) соответственно. После этого указатель будет передвинут на следующий элемент. При втором вызове функции `each()` извлеченным окажется второй набор значений и т.д. Нам остается только обернуть вокруг функции `each()` цикл `for()`, чтобы пройти по все элементам массива.

Теперь можно переписать сценарий `soups.php`, опираясь на только что полученные знания. Вместо простого вывода на экран монитора количества элементов массива мы будем обращаться к реальным значениям.

Вывод значений любого массива на экран монитора

1. Создайте новый PHP-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок (листинг 7.4).

```
<HTML><HEAD><TITLE>Using Arrays</TITLE></HEAD><BODY>.
```


Листинг 7.4 т Цикл - это наиболее распространенный способ доступа ко всем элементам массива. В этом сценарии функция `each()` определяет ключи и значения массива, которые затем выводятся в браузере.

```

1  <HTML>
2  <HEAD>
3  <TITLE>Using Arrays</TITLE></HEAD>
4  <BODY>
5  <?php
6  $Soups = array(
7      "Monday"=>"Clam Chowder",
8      "Tuesday"=>"White Chicken Chili",
9      "Wednesday"=>"Vegetarian",
10     "Thursday"=>"Chicken Noodle",
11     "Friday"=>"Tomato",
12     "Saturday"=>"Cream of Broccoli"
13 );
14 for ($n = 0; $n < count($Soups); $n++) {
15     $Line = each ($Soups);
16     print ("{$Line[key]}'s soup is {$Line[value]}<P>\n");
17 }
18 ?>
19 </BODY>
20 </HTML>

```

3. Откройте PHP-раздел страницы и иницилируйте массив `$Soups`.

```

<?php
$Soups = array(
    "Monday"=>"Clam Chowder",
    "Tuesday"=>"White Chicken Chili",
    "Wednesday"=>"Vegetarian",
    "Thursday"=>"Chicken Noodle",
    "Friday"=>"Tomato",
    "Saturday"=>"Cream of Broccoli"
);

```

4. Начните цикл `for` для доступа к каждому элементу массива.

```
for ($n = 0; $n < count($Soups); $n++) {
```

Этот цикл присваивает значение 0 счетчику цикла `$n`. Затем проверяется величина `$n`. Если она меньше, чем количество элементов в массиве, то цикл выполняется и значение переменной `$n` увеличивается на единицу.

5. С помощью функции `each()` извлеките ключи и значения, затем распечатайте их.

```

$Line = each ($Soups);
print ("{$Line[key]}'s soup is {$Line[value]}<P>\n");

```

Цикл присвоит ключи и значения массива `$Soups` массиву `$Line` через функцию `each()`. Затем он распечатает ключ и значение.

6. Закройте цикл, PHP и HTML.

```
}?></BODY></HTML>
```

7. Сохраните страницу как `soups.php`, загрузите ее на сервер и протестируйте в браузере (рис. 7.4).



Лучше было бы создать переменную для значения `count ($Array)` и использовать ее в заголовке цикла `for ()`. В этом случае длина массива при каждой итерации пересчитываться не будет.

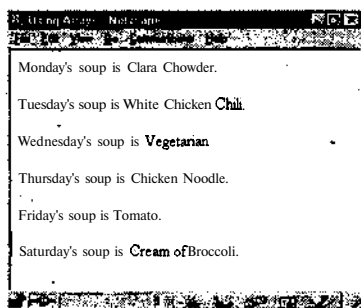


Рис. 7.4 ▼ Выполнение цикла для каждого элемента массива генерирует эту страницу. Благодаря функции `each ()` в PHP можно осуществлять доступ к каждому ключу и значению, не зная о них заранее

Сортировка массивов

В PHP поддерживается несколько способов сортировки массивов (под *сортировкой* мы имеем в виду распределение по алфавиту для строк и разброс в арифметическом порядке для чисел). При сортировке массива необходимо иметь в виду, что он состоит из многих пар ключей и значений. Поэтому можно сортировать массив на основе значений либо ключей. Допустимо также распределить значения и оставить им имеющиеся ключи или присвоить новые.

Сортировка только по значениям элементов без учета ключей выполняется с помощью функции `sort ()`. В обратном порядке и снова без учета ключей значения распределяют с помощью функции `rsort ()`. Синтаксис функций выглядит следующим образом:

```
sort ($Array);  
rsort ($Array);
```

После вызова функций изменится порядок элементов в массиве `$Array`. По сути, из тех же элементов создается другой массив. Если массив был проиндексирован по умолчанию, числами начиная с нуля и заканчивая величиной `count ($Array) - 1`, то о новых значениях ключей можно не заботиться. Однако,

если мы проиндексировали массив строками или числами в особом порядке, то желательно не потерять связь между ключами и соответствующими значениями. Для таких массивов имеется специальный термин - ассоциативные массивы, или хэши. Оказывается, в языке PHP все массивы ассоциативные, так что при их преобразованиях в большинстве случаев необходимо заботиться о сохранении связи между ключом и значением. Исключение составляют только две вышеупомянутые функции для индексирования по умолчанию. Во всех остальных случаях необходимо использовать похожие, но немного другие функции, например `asort()` и `arsort()`. Как и первые две функции, эти сортируют массив по значению в прямом и обратном порядке, сохраняя соответствие между ключами и значениями.

Для сортировки по ключам с сохранением соответствия между ключами и значениями используется функция `ksort()`. Для сортировки по ключам в обратном порядке также с сохранением соответствия между ключами и значениями из оригинального массива имеется функция `krsort()`.

Функция `shuffle()` случайным образом меняет порядок значений в массиве, сохраняя, однако, соответствие между ключами и значениями элементов.

Чтобы показать сортировку массивов, создадим список студентов и отметок, которые они получили за контрольную работу, затем оформим этот список по отметкам и по именам.

Выполнение действия

1. Откройте текстовый редактор и создайте новый PHP-документ.
2. Начните со стандартного кода документов HTML и PHP.

```
<HTML><HEAD><TITLE>Sorting Arrays</TITLE></HEAD><BODY><?php
```

3. Создайте массив.

```
$Grades = array(  
    "Richard"=>"95",  
    "Sherwood"=>"82",  
    "Toni"=>"98",  
    "Franz"=>"87",  
    "Melissa"=>"75",  
    "Roddy"=>"85"  
);
```

4. Напечатайте заголовок, а затем каждый элемент массива с помощью цикла.

```
print ("Originally, the array looks like this:<BR>");  
for ($n = 0; $n < count($Grades); $n++) {  
    $Line = each ($Grades);  
    print ("{$Line[key]}’s grade is {$Line[value]}.<BR>\n");  
}
```

5. Отсортируйте массив по значениям в обратном порядке. Так в начале окажутся фамилии студентов, получивших высшие отметки.

```
arsort($Grades);
```

Мы определяем, кто получил лучшие отметки, поэтому необходимо использовать вместо функции `asort()` функцию `arsort()`. Первая функция сортирует массив в числовом порядке, и при ее использовании мы получили бы последовательность 75, 82, 85 и т.д., а не желаемую 98, 95, 87,...

Необходимо также использовать функцию `arsort()`, а не `rsort()`, чтобы сохранить взаимосвязь ключей со значениями. Просто `rsort()` эту взаимосвязь не сохраняет, то есть имена студентов будут утрачены. Если вам нужен только список оценок, можете попробовать только функцию `rsort()`.

6. Верните указатель массива в исходное положение с помощью функции `reset()`.

```
reset($Grades);
```

Функция `reset()` возвращает указатель на первый элемент массива `$Grade`, с которого начинается новый цикл. Это необходимо делать, так как предыдущий цикл перемещает указатель в конец массива.

7. Снова распечатайте массив (с заголовком) с помощью другого цикла.

```
print ("<P>After sorting the array by key using arsort(),
-the array looks like this:<BR>");
for ($n = 0; $n < count($Grades); $n++) {
    $Line= each ($Grades);
    print ("$Line[key]'s grade is $Line[value].<BR>\n");
}
```

8. Теперь отсортируйте массив по ключу для получения списка имен студентов в алфавитном порядке и снова восстановите цикл.

```
ksort($Grades);
reset($Grades);
```

Функция `ksort()` упорядочит массив по ключам (в нашем случае - в алфавитном порядке), при этом взаимосвязь ключей и значений будет сохранена.

9. В очередной раз распечатайте заголовок и массив.

```
print ("<P>After sorting the array by key using ksort(),
-the array looks like this:<BR>");
for ($n = 0; $n < count($Grades); $n++) {
    $Line= each ($Grades);
    print ("$Line[key]'s grade is $Line[value].<BR>\n");
}
```

10. Закройте сценарий стандартными тэгами PHP и HTML.

```
?></BODY></HTML>
```

11. Сохраните сценарий как `sort.php` (листинг 7.5), загрузите его на сервер и протестируйте в браузере (рис. 7.5).

Листинг 7.5 т Для сортировки массивов в PHP имеется много различных функций, включая используемые здесь `arsort()` и `ksort()`.

```
1 <HTML>
2 <HEAD>
```

```

3 <TITLE>Sorting Arrays</TITLE>
4 <BODY>
5 <?php
6 $Grades = array(
7     "Richard"=>"95",
8     "Sherwood"=>"82",
9     "Toni"=>"98",
10    "Franz"=>"87",
11    "Melissa"=>"75",
12    "Roddy"=>"85"
13 );
14 print ("Originally, the array looks like
this:<BR>");
15 for ($n = 0; $n < count($Grades); $n++) {
16     $Line = each ($Grades);
17     print ("{$Line[key]}’s grade is
{$Line[value]}.<BR>\n");
18 }
19 arsort($Grades);
20 reset($Grades);
21 print ("<P>After sorting the array by
value using arsort(), the array looks
like this:<BR>");
22 for ($n = 0; $n < count($Grades); $n++) {
23     $Line = each ($Grades);
24     print ("{$Line[key]}’s grade is
{$Line[value]}.<BR>\n");
25 }
26 ksort($Grades);
27 reset($Grades);
28 print ("<P>After sorting the array by key using ksort(), the array
looks like this:<BR>");
29 for ($n = 0; $n < count($Grades); $n++) {
30     $Line= each ($Grades);
31     print ("{$Line[key]}’s grade is {$Line[value]}.<BR>\n");
32 }
33 ?>
34 </BODY>
35 </HTML>

```

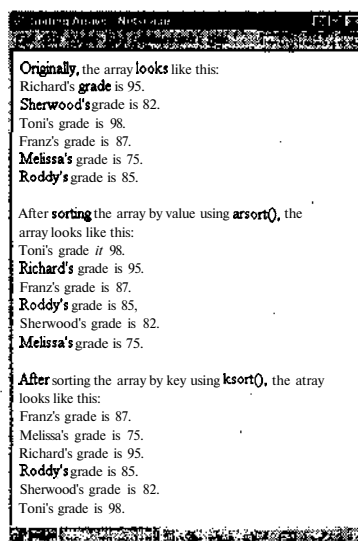


Рис. 7.5 т Сортировку массива можно выполнить несколькими способами, при этом результаты будут различными. При выборе функции прежде всего решите, хотите ли вы сохранить связь ключей и их значений

Преобразование строк и массивов

Теперь, когда вы имеете представление и о массивах, и о строках, уместно поговорить о двух функциях для перехода из одного представления в другое. Первая функция, `implode()`, превращает массив в строку. Вторая, `explode()`, выполняет обратное действие, разбивает строку и создает массив. Эти функции удобно использовать в целом ряде случаев:

- иногда необходимо превратить массив в строку для добавления этого значения к адресу URL (с массивом это сделать нелегко);
- порой нужно превратить строку в массив для того, чтобы сохранить информацию в базе данных;

- вам может понадобиться сделать строку массивом, чтобы разбить **текстовое** поле с разделителями в виде запятых (скажем, область поиска по **ключевому** слову в форме) на отдельные части.

Синтаксис функции `explode()` выглядит так:

```
$Array = explode ($Separator, $String);
```

Разделитель `$Separator` - это строка символов или один символ, обозначающий конец одного элемента массива и начало другого. В таких случаях **говорят**, что разделитель делит строку на поля, которые затем станут элементами массива. Обычно для этого используется запятая или пробел. Код будет выглядеть так:

```
$Array = explode (" ", $String);
```

Другой возможный вариант:

```
$Array = explode. (" ", $String);
```

Для обратного преобразования массива в строку необходимо указать **разделитель**, а все остальное будет сделано автоматически:

```
$String = implode($Glue, $Array);
```

```
$String = implode(" ", $Array);
```

Другой допустимый вариант:

```
$String = implode(" ", $Array);
```

Чтобы показать возможное использование функций `explode()` и `implode()`, создадим **HTML-форму**, в которую пользователь вводит строку из имен, **разделенных** запятыми. Затем PHP-скрипт превратит строку в массив для того, чтобы можно было отсортировать список. И наконец, будет создана и возвращена **строка**, в которой имена расположены в алфавитном порядке.

Выполнение преобразования

1. Создайте новый HTML-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD><BODY>
```

3. Создайте новый HTML-документ для ввода **текста**.

```
<FORM ACTION="HandleList.php" METHOD=POST>
Enter the words you want alphabetized with each
individual word separated by a space:<BR>
<INPUT TYPE=TEXT NAME="List" SIZE=80xBR>
```

В подобных случаях важно дать пользователю подробную подсказку о том, какую информацию требуется ввести. Например, если в качестве разделителя в списке он применяет запятые, то обработать строку соответствующим образом будет невозможно (после создания обоих сценариев замените пробелы запятыми и оцените **результаты**).

4. Создайте кнопку **Submit**, затем закройте форму и HTML-страницу.

```
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
</FORM></BODY></HTML>
```

5. Сохраните сценарий как list.html (листинг. 7.6) и загрузите его на сервер.

Листинг 7.6 т Это простая HTML-форма, куда пользователь может ввести список слов. Включение в форму подробных инструкций - разумная практика Web-дизайна.

```
1 <HTML>
2 <HEAD>
3 <TITLE>HTML Form</TITLE>
4 </HEAD>
5 <BODY>
6 <FORM ACTION="HandleList.php" METHOD=POST>
7 Enter the words you want alphabetized with each individual word
  separated by a space:<BR>
8 <INPUT TYPE=TEXT NAME="List" SIZE=80xBR>
9 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
10 </FORM>
11 </BODY>
12 </HTML>
```

А теперь создадим страницу HandleList.php, где будут обрабатываться данные, полученные от пользователя из формы list.html.

6. Создайте новый HTML-документ в текстовом редакторе.
7. Напишите HTML-заголовок и откройте PHP-раздел страницы.

```
<HTML><HEAD><TITLE>Alphabetizing Example</TITLE></HEAD><BODY><?php
```

8. Превратите введенную пользователем и переданную в сценарий строку \$List в массив.

```
$Array = explode (" ", $List);
```

Эта строка кода создает новый массив \$Array из строки \$List. Каждый пробел между словами в строке \$List указывает на конец одного слова и начало следующего. Таким образом, первое слово устанавливается в \$Array[0], затем идет пробел в строке \$List, затем второе слово устанавливается в \$Array[1], и так далее до конца строки.

9. Отсортируйте массив в алфавитном порядке.

```
sort ($Array);
```

Так как нет необходимости сохранять связь между ключами и значениями в массиве \$Array, можно использовать функцию sort() вместо asort(), которую мы применяли выше. Для очень большого массива это дало бы некоторую экономию времени - функция sort() работает немного быстрее.

10. Создайте новую строку из отсортированного массива.

```
$NewList = implode ("<BR>", $Array);
```

ГЛАВА 7 ▼ Массивы

Нам надо распечатать новый список. Но, поскольку распечатать массив не так просто, как строку, сначала преобразуем массив `$Array` в строку `$NewList`. Новая строка будет начинаться со значения `$Array[0]`, за ним пойдет HTML-тэг `
`, затем значение `$Array[1]`, снова тэг `
` и т.д. Использование тэга вместо пробела или запятой придаст списку более читабельную форму при отображении в браузере.

11. Распечатайте новую строку в браузере.

```
print ("An alphabetized version of your list is:<BR>$NewList");
```

12. Закройте PHP-раздел и HTML-страницу.

```
?></BODY></HTML>
```

13. Сохраните страницу как `HandleList.php`, загрузите ее на сервер в один каталог с `list.html` (листинг. 7.7) и протестируйте оба сценария в браузере (рис. 7.6 и 7.7).

Листинг 7.7 ▼ Простые, но действенные функции `explode()` и `implode()` позволяют быстро и легко отсортировать список переданных слов практически любого размера с помощью двух строк кода.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Alphabetizing Example</TITLE></HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "list.html". */
7 $Array = explode (" ", $List);
8 sort ($Array);
9 $NewList = implode ("<BR>", $Array);
10 print ("An alphabetized version of your list is:<BR>$NewList");
11 ?>
12 </BODY>
13 </HTML>
```

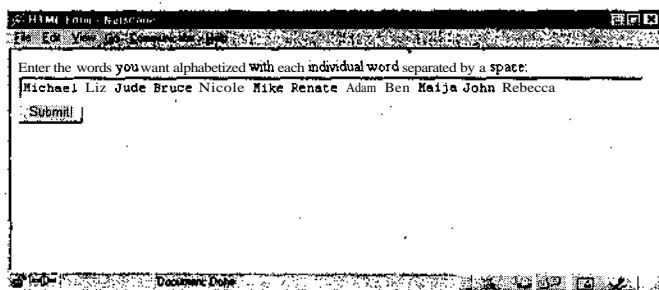


Рис. 7.6 ▼ HTML-формой берется список слов, которые будут отсортированы в алфавитном порядке в сценарии `HandleList.php` (рис. 7.7)

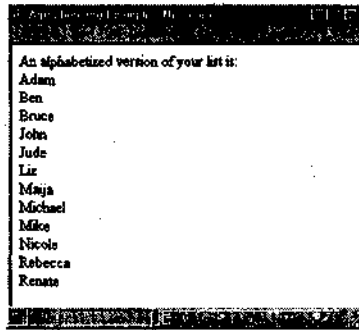


Рис. 7.7 ▼ Тот же список, отсортированный в алфавитном порядке. Запрограммировать этот процесс легко и просто, но он был бы невозможен без существования массивов

Функция `join()`, которая также возможна в коде, идентична функции `implode()`. Есть еще несколько функций, имеющих несколько равноправных названий-синонимов. Вы можете использовать то имя функции, которое вам больше нравится.

Создание массива в экранной форме

На протяжении этой главы мы создавали массивы только внутри PHP-скрипта. Однако можно создать массив и послать его в PHP-сценарий непосредственно из HTML-формы. Кодирование в таком случае только немного сложнее, чем было до этого. Перепишем страницу с обратной связью из главы 3, чтобы она сразу генерировала массив вместо отдельных переменных.

Создание массива через HTML-форму

1. Создайте новый HTML-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD><BODY>
```

3. Откройте HTML-форму.

```
<FORM ACTION="HandleForm.php" METHOD=POST>
```

4. Создайте три окна для ввода текста с массивами для атрибутов имени.

```
First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20><BR>
Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
E-mail Address <INPUT TYPE=TEXT NAME="Array[Email]" SIZE=60><BR>
```

В исходной форме, созданной раньше, сценарий `HandleForm.php` получал переменные `$FirstName` и `$LastName`. Теперь он примет `$Array[$FirstName]`,

`$Array[$LastName]` и т.д., что автоматически создаст проиндексированный соответствующим образом массив. Мы **опустили** кавычки в ключе массива (было бы `$Array["$LastName"]`), что вполне приемлемо и помогает избежать синтаксических ошибок.

5. Создайте текстовую область как часть массива.

```
Comments <TEXTAREA NAME="Array[Comments]" ROWS=5 COLS=40>
-</TEXTAREA><BR>
```

6. Сохраните сценарий как `form.html` (листинг 7.8) и загрузите его на сервер.

Листинг 7.8 ▼ Это HTML-форма с массивом для ввода имен. Нет необходимости специфицировать ключи для каждого **поля**, можно просто пометить каждое поле `Array[]`, и тогда имя будет размещено в массиве `Array[0]`, фамилия - в `Array[1]`, адрес электронной почты - в массиве `Array[2]` и комментарии - в `Array[3]`.

```
1 <HTML>
2 <HEAD>
3 <TITLE>HTML Form</TITLE>
4 </HEAD>
5 <BODY>
6 <FORM ACTION="HandleForm.php" METHOD=POST>
7 First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20xBR>
8 Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
9 E-mail Address <INPUT TYPE=TEXT NAME="Array[Email]" SIZE=60xBR>
10 Comments <TEXTAREA NAME="Array[Comments]" ROWS=5 COLS=40>
    </TEXTAREA><BR>
11 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12 </FORM>
13 </BODY>
14 </HTML>
```

Необходимо также написать новую страницу `HandleForm.php` и отразить в ней изменения, произошедшие в сценарии `form.html`.

7. Создайте новый PHP-документ в текстовом редакторе.
8. Напишите стандартный HTML-заголовок, затем открывающий PHP-тэг.

```
<HTML><HEAD><TITLE>Form Results/Using Arrays</TITLE></HEAD><BODY>x?php
```

9. Возьмите элементы имени и фамилии и объедините их в новый элемент массива.

```
$Array["Name"] = $Array["FirstName"] . " " . $Array["LastName"];
```

Мы добавили еще один компонент информации в массив, но данные по-прежнему можно обрабатывать с помощью только одной переменной.

10. Распечатайте переданные имя и фамилию в браузере, чтобы подтвердить успешное получение данных из формы.

```
print ("Your full name is $Array[Name].<BR>\n");
```

11. Сохраните страницу как `HandleForm.php` (листинг 7.9), загрузите ее на сервер в один каталог с `form.html` и протестируйте оба сценария в браузере (рис. 7.8-7.10).

Листинг 7.9 ▼ Не имеет значения, получает PHP набор строк из HTML-формы или один массив, как в этом случае. Однако при использовании массива остается меньше переменных, которыми можно манипулировать.

```

1  <HTML>
2  <HEAD>
3  <TITLE>Form Results/Using Arrays</TITLE></HEAD>
4  <BODY>
5  <?php
6  /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7  $Array["Name"] = $Array["FirstName"] . " " . $Array["LastName"];
8  print ("Your full name is $Array[Name].<BR>\n");
9  ?>
10 </BODY>
11 </HTML>

```

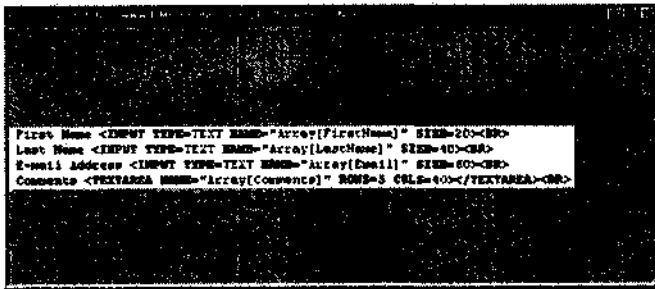


Рис. 7.8 ▼ Это исходный текст страницы `form.html`, где для ввода имен теперь используется массив

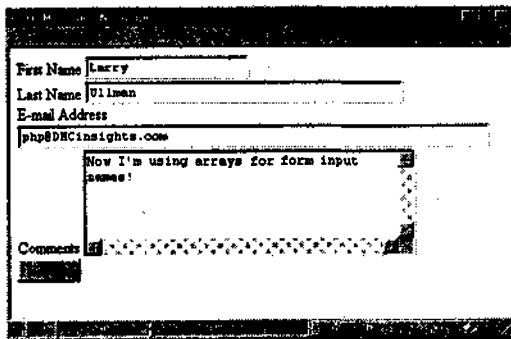


Рис. 7.9 ▼ Пользователю все равно, какие имена вы применяете в вашей форме, так как обычно он видит только такой экран. Поэтому выбирайте тот способ, который облегчит программирование, в том числе создавайте массивы прямо в экранной форме

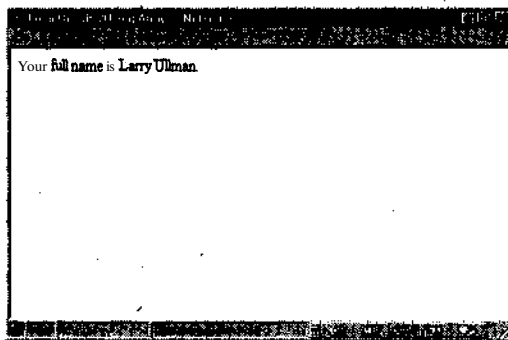


Рис. 7.10 ▼ Такое сообщение появляется как подтверждение успешной передачи данных из формы (рис. 7.9) на страницу `HandleForm.php`



Применение массивов в HTML-формах **очень** удобно при использовании многочисленных **триггерных** кнопок или ниспадающего меню, когда пользователь может выбрать несколько вариантов.

Создание многомерных массивов

Многомерные массивы могут быть вам известны по другим языкам программирования. Если нет, то это еще одна тема, которую вы наверняка захотите освоить, как только овладеете основами программирования на языке PHP.

Массивы настолько универсальны, что не стоит ограничивать значения элементов в них только строками и числами. Если в качестве значения в массиве снова задать массив, мы получим многомерный массив. Например:

```
$Array1 = array("apples", "bananas", "oranges");
$Array2 = array("steaks", "hamburgers", "pork chops");
$List = array("fruits"=>$Array1, "meats"=>$Array2,
  "other"=>"peanuts", "cash"=>30.00);
```

Многомерный массив `$List` состоит теперь из одной строки (peanuts), одного числа с плавающей точкой (30.00) и двух массивов (fruits и meats).

Обращение к элементам многомерного массива может быть более трудным, чем в простом массиве. Ключевым моментом здесь становится добавление индексов по мере необходимости. Поэтому в нашем примере строка `bananas` находится по такому адресу:

```
$List["fruits"][1].
```

Сначала мы указываем на элемент (в данном случае массив `$Array1`) массива `$List` с помощью переменной `["fruits"]`. Затем представляем элемент массива исходя из его положения. Это второй элемент, следовательно, мы используем индекс `[1]`.

Допустимо создать многомерный массив в одной инструкции с помощью набора вложенных вызовов функции `array()`. Но я не рекомендовал бы делать этого, так как возможность синтаксической ошибки значительно увеличивается по мере того, как возрастает уровень вложенности инструкции. Если вас заинтересовали многомерные массивы и вы испытываете необходимость в их использовании, то сначала всегда задавайте отдельные массивы, а затем объединяйте их в один.

Распечатать значение из многомерного массива достаточно сложно. Во всех версиях PHP ранее четвертой сделать это напрямую件 невозможно. В PHP 4.0 с помощью фигурных скобок разрешается печатать из многомерных массивов в строке:

```
print ("The value I want to print is {Array[index1][index2]}.");
```

Впрочем, вариант

```
print ("The value I want to print is $Array[index1][index2].");
```

будет работать не во всех версиях.



Регулярные выражения

Если бы нужно было знать только две особенности регулярных **выражений**, они были бы следующими: во-первых, регулярные выражения очень помогают писать хорошие программы, во-вторых, их легко понять, но порой трудно добиться правильной работы. Однако, как только вы поймете правила написания регулярных выражений, эти знания окупятся сторицей. Используя регулярные выражения, со временем вы значительно повысите **качество** программирования и сэкономите время.

В данной главе дается определение регулярных выражений, описываются способы их **создания** и приводятся примеры, демонстрирующие их возможности.

Что такое регулярные выражения

Одной главы, посвященной регулярным выражениям, явно недостаточно для того, чтобы понять все преимущества. Однако этой информации хватит, если вы только начинаете использовать данный инструмент в своей работе. Дополнительные сведения о регулярных выражениях представлены в **приложении С**.

Воспринимайте регулярные выражения как усовершенствованную систему шаблонов. Сначала пишется шаблон, затем с помощью одной из встроенных функций РНР он применяется к текстовой строке (регулярные выражения используются исключительно для работы со **строками**). В РНР имеется две группы функций, использующих регулярные выражения для сопоставления с шаблонами, и две - для нахождения соответствий шаблонам и замены одного текста другим. В обеих группах одна из функций реагирует на **регистр**, другая нет.

Для начала зададим простой шаблон и найдем соответствующие ему фрагменты строк. Затем рассмотрим более сложные шаблоны и завершим обучение тем, что будем находить нужные соответствия и выполнять замены.

Некоторые текстовые редакторы, в частности BBEdit для Macintosh, TextPad для Windows и Emacs для UNIX, позволяют использовать регулярные выражения для нахождения фрагментов строк и их так называемой контекстной замены сразу в нескольких документах (рис. 8.1). Вероятно, это еще одна причина для изучения регулярных выражений и, возможно, один из критериев при выборе текстового редактора.



Рис. 8.1 В стандартном диалоговом окне **Find** некоторых текстовых редакторов можно включить опцию использования регулярных выражений (в том числе одновременно в нескольких файлах или каталогах), нажав кнопку **Use Grep**. В программе BBEdit есть несколько встроенных шаблонов. Кроме того, вы можете хранить в этом редакторе созданные регулярные выражения

В руководстве по PHP, имеющемся в любом дистрибутиве или архиве с исходниками, рассмотрены различия в использовании регулярных выражений в языках PHP и Perl. Специалисты по Perl, скорее всего, сначала захотят ознакомиться с этим разделом руководства.

Создание простого шаблона

Перед тем как применять встроенные функции регулярных выражений PHP, необходимо создать шаблон, который будет использоваться функцией для поиска совпадений. В PHP есть ряд правил создания шаблонов. Многие из них аналогичны применяемым в языках Perl, C и Java. Эти правила можно использовать по отдельности или в любой комбинации для создания как совсем простых, так и очень сложных шаблонов.

Чтобы понять, как создавать шаблоны, сначала проанализируем используемые для этого особые символы, или метасимволы. Затем обсудим, как группировать символы, и охарактеризуем классы. Шаблон определяется комбинацией символов, групп и классов.

Литералы

Первый тип символов, используемый для создания шаблона - это *литерал*. Он обозначает только то, что обозначает, и не более того. Например, шаблону «a» будет соответствовать только буква «a», шаблону «ab» - только «ab» и т.д.

Литералы позволяют находить точные соответствия, но, если бы регулярные выражения использовались только для этого, они были бы гораздо менее полезны (тогда для работы со строками было бы достаточно нескольких

функций, см. главу 5). Для создания общих шаблонов можно также применять специальные символы, имеющие в регулярных выражениях особенное значение (см. ниже).

Метасимволы

Более сложной структурой, чем литералы, являются *метасимволы*. Это специальные символы, **имеющие** более широкое значение. Так, если «а» означает просто «а», то первый метасимвол, точка (`.`), отождествляется с любым отдельным символом (а, 1, % и т.д.). Достаточно просто, однако обратите внимание на следующее: если вы хотите использовать метасимвол буквально, его необходимо заэкранировать, как мы уже экранировали кавычки в команде `print()`. Например, точке, используемой в тексте (знак препинания без специального значения), будет соответствовать символ `«\.»`.

Есть три метасимвола, позволяющие находить многочисленные соответствия: «а*» отождествляется с нулем или любым количеством букв «а» (а, аа, ааа и т.д.); «а+» отождествляется с одной или более «а» (а, аа, ааа и т.д., но хотя бы одна должна быть); и «а?» отождествляется с нулем или одной буквой «а». Каждый метасимвол меняет смысл стоящей перед ним буквы. Программисты, уверенно работающие с регулярными выражениями, могут менять смысл фразы. В силу того что указанные метасимволы (*, +, ?) позволяют «размножить» стоящее перед ними выражение, их иногда называют символами-умножителями.

Для поиска определенного количества одной буквы необходимо поместить в фигурные скобки диапазон (`{ }`), указав либо конкретное число, либо минимум, либо и минимум и максимум. Например, «а{3}» будет соответствовать только «ааа»; «а{3,}» - ааа, аааа и т.д. (три или больше букв «а»); «а{3,5}» - ааа, аааа и ааааа (между тремя и пятью буквами включительно).

Знак «крышечка» (`^`, тот, что обычно представлен на клавиатуре над цифрой 6) будет соответствовать строке, начинающейся со следующей за знаком буквы. Этот знак можно условно представить как псевдосимвол, с которого начинается каждая строка.

Знак доллара (`$`) отождествляется со строкой, которая заканчивается предшествующей этому символу буквой. Его также можно представить как самый последний символ в строке, обозначающий переход на новую строку. Такой символ в отличие от «крышечки» (`^`) действительно существует, он вставляется в текст, как только вы нажимаете клавишу **Enter**. Следовательно, запись `^а.` будет соответствовать любой строке из двух символов, начинающейся на букву «а», а запись `.а$` - любой двухсимвольной строке, заканчивающейся буквой «а». Следовательно, знак `^а$` будет отождествляться только со строкой из одного символа «а», являясь эквивалентом литерала «а» на отдельной строке.

В регулярных выражениях также используется знак вертикальной **черты** (`|`) как эквивалент слова «или». Следовательно, запись `a|b` будет соответствовать строкам `a` или `b`, а запись `gre|au` будет отождествляться с обоими вариантами написания названия цвета (по-английски и `greu`, и `grau` означают «серый»). С помощью вертикальной черты в шаблонах выполняется чередование.

Вышеприведенные примеры наглядно показывают, как работают метасимволы. Прежде всего вам необходимо привыкнуть, что означают **различные** символы и как они используются.

При использовании фигурных скобок для указания количества знаков всегда необходимо задавать минимум поиска: `a{3}` и `a{3,}` - приемлемые варианты в отличие от записи `a{,3}`.

Специальные символы (`^`, `.`, `[]`, `$`, `()`, `|`, `*`, `?`, `{}`, `\`) в шаблоне **должны** быть экранированы посредством поставленного перед ними обратного слеша. Это справедливо для метасимволов и для группирующих символов (разнообразных скобок). Обратный слеш можно также использовать для отождествления с символом новой строки (`\n`), символом табуляции (`\t`) и некоторыми другими. По сути, таким образом мы создаем метасимвол из литерала.

В табл. С.2 (приложение С) приведен полный список метасимволов и их комбинаций, которые используются в регулярных выражениях.

Сопоставление с шаблонами

В PHP есть две встроенные функции, предназначенные специально для сопоставления с шаблоном в строке: `ereg()` и `eregi()`. Разница между ними состоит в том, что функция `ereg()` реагирует на регистр, а `eregi()` - нет, то есть не различает строчные и прописные буквы. Как правило, широко используется именно функция `eregi()`, если нет особой необходимости в более точном совпадении (например, в целях безопасности, как с паролями). Значение обеих функций будет истинным, если произошло совпадение с шаблоном, и ложным в противном случае. Ниже представлены два способа использования этих функций:

```
ereg("pattern", "string");  
или  
$Pattern = "pattern";  
$String = "string";  
eregi($Pattern, $String);
```

Далее мы будем присваивать шаблон переменной, как во втором примере, чтобы обратить большее внимание на сам шаблон как основу любого регулярного выражения.

Создадим новый сценарий HandleForm.php, **который** работает в связке с файлом form.html (глава 7). Сценарием HandleForm.php будет проверяться правильность переданного адреса электронной почты (листинг 8.1).

Листинг 8.1 т Исходная форма form.html была создана для ввода адреса электронной почты, комментариев пользователя, а также его имени и фамилии.

```

1  <HTML>
2  <HEAD>
3  <TITLE>HTML Form</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleForm.php" METHOD=POST>
7  First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20><BR>
8  Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
9  E-mail Address <INPUT TYPE=TEXT NAME="Array[Email]" SIZE=60><BR>
10 Comments <TEXTAREA NAME="Array[Comments]" ROWS=5 COLS=40>
    </TEXTAREA><BR>
11 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12 </FORM>
13 </BODY>
14 </HTML>

```

Проверка соответствия шаблону с помощью функции eregi

1. Создайте новый сценарий HandleForm.php в текстовом редакторе.
2. Начните со стандартного заголовка HTML и PHP.

```

<HTML>
<HEAD>
<TITLE>Using Regular Expressions</TITLE></HEAD>
<BODY>
<?php
/* Эта страница получает и обрабатывает данные, принятые
от "form.html". */

```

3. Задайте необходимые условия.

```

if (($Array["FirstName"]) AND ($Array["LastName"])) {
    $Array["Name"] = $Array["FirstName"] . " " . $Array["LastName"];
} else {
    print ("Please enter your first and last names.<BR>\n");
}

```

Для проверки правильности введенного имени можно было бы использовать регулярное выражение, но, так как оно состоит только из букв, вполне "достаточно проверить лишь то, что переменная имеет значение. Затем, если пользователь ввел имя и фамилию, эти данные будут объединены в один элемент и переданы в массив. Если имя или фамилия пропущены, на экране появится запрос на ввод нужной информации.

4. Следующий шаг - создание шаблона для адреса электронной почты.

```
SPattern = ".+@.+\..+";
```

Это шаблон для проверки правильности адреса электронной почты. Он довольно прост, но отлично справится с задачей (впрочем, для сравнения далее будет разработан более сложный **шаблон**).

Первая часть шаблона означает, что адрес электронной почты должен начинаться по крайней мере с одного произвольного знака (.) до знака коммерческое «а» (@). Из второй части шаблона следует, что далее следует символ @, обязательный во всех адресах электронной почты. Третья часть означает, что необходим по крайней мере еще один знак. Четвертая часть требует наличия точки, а последняя - использования еще по крайней мере одного знака, завершающего строку (это не должна быть **точка**).

5. Используйте шаблон для проверки переданного адреса электронной почты.

```
if (ereg($Pattern, $Array["Email"])) {  
    print ("Your information has been received!<BR>\n");  
} else {  
    print ("Please enter a valid email address!\n");  
}
```

Для применения функции `ereg()` необходимо передать в нее шаблон, заданный выше, и переменную `$Array["Email"]` из файла `form.html`. Если значение переменной соответствует шаблону, условие признается истинным и будет напечатано соответствующее сообщение. В противном случае пользователь получит запрос на ввод Правильного адреса электронной почты. Так как адреса электронной почты никак не реагируют на регистр, мы используем функцию `ereg()`, а не `ereg()`.

6. Сохраните сценарий (листинг 8.2), загрузите его на сервер и протестируйте в браузере (рис. 8.2–8.4).

Листинг 8.2 ▼ Этот сценарий не только проверяет, ввел ли пользователь адрес электронной почты в HTML-форму, но и **подтверждает** его правильность.

```
1  <HTML>  
2  <HEAD>  
3  <TITLE>Using Regular Expressions</TITLE></HEAD>  
4  <BODY>  
5  <?php  
6  /* Эта страница получает и обрабатывает данные, принятые  
   * от "form.html". */  
7  if (($Array["FirstName"]) AND ($Array["LastName"])) {  
8      $Array["Name"] = $Array["FirstName"] . " " . $Array["LastName"];  
9  } else {  
10     print ("Please enter your first and last names.<BR>\n");  
11 }  
12 $Pattern = ".+@.+\.+\.+";  
13 if (ereg($Pattern, $Array["Email"])) {  
14     print ("Your information has been received!<BR>\n");  
15 } else {  
16     print ("Please enter a valid email address!\n");  
17 }  
18 ?>  
19 </BODY>  
20 </HTML>
```

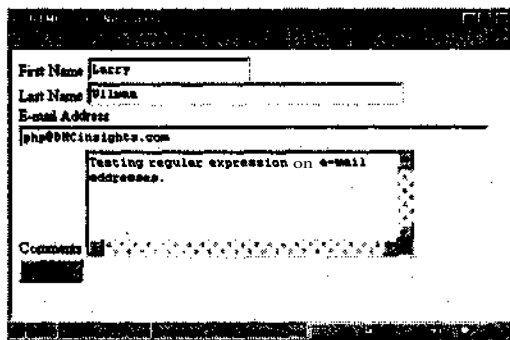


Рис. 8.2 ▼ Форма `form.html` берет введенные пользователем данные и посылает их на страницу `HandleForm.php` (рис. 8.3) для проверки с помощью регулярных выражений

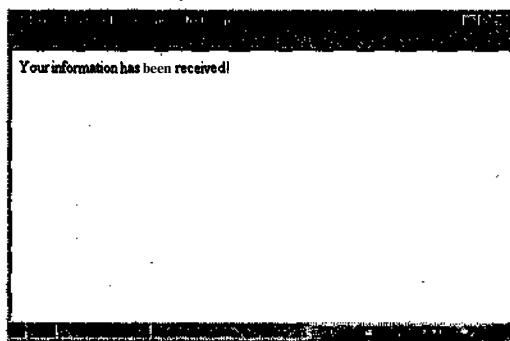


Рис. 8.3 т Сценарием `HandleForm.php` создается такое сообщение, если пользователь ввел соответствующий шаблону адрес электронной почты. Сравните с рис. 8.4

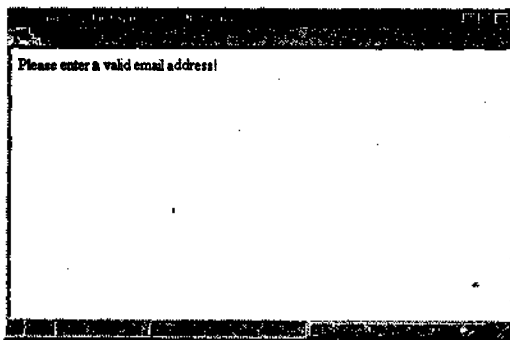


Рис. 8.4 ▼ На экране сообщение об ошибке, так как был введен неправильный адрес электронной почты `php@DMCinsights`

Хотя написание и использование своих регулярных выражений и доказывает вашу преданность делу программирования, разработано много работающих конструкций (включая шаблон для проверки адресов электронной почты), которые можно загрузить с сайтов, посвященных языку PHP и перечисленных в приложении С.

Я присвоил шаблон переменной и затем обратился к переменной в функции `eregi()`. Можно, однако, поместить шаблон прямо в функцию:

```
if (eregi(".*@.\.+\.+", $Array["Email"])) {...
```

Проверка формы, идентичная только что запрограммированной нами, может осуществляться на машине-клиенте с помощью языка JavaScript и встроенных в него регулярных выражений. В зависимости от обстоятельств можно использовать регулярные выражения в одном или другом языке, а можно и в обоих.

Создание более сложных шаблонов

Если вы поняли, как создавать шаблоны с помощью литералов и метасимволов, то можно переходить к изучению групп и классов, позволяющих создавать более сложные шаблоны.

Группирование

С помощью все тех же обычных круглых скобок можно группировать уже знакомые нам основные символы в более сложные шаблоны. Группирование не-сложно: `(abc)` соответствует только записи `abc`, `(trout)` - только `trout`. Приведенные примеры достаточно спорны, поскольку `abc` и так отождествляется только с `abc`. Только начав использовать метасимволы со скобками, можно увидеть, как группирование влияет на шаблоны.

По существу, использование скобок - это способ создания нового литерала. С логической точки зрения, если `a` - литерал, которому соответствует только буква «а», то `(abc)` - это литерал, которому соответствует **только** запись «abc». Следовательно, метасимволы, соответствующие множеству символов (вы помните, что их еще называют умножителями), относятся не только к предшествующему литералу, но и ко всей группе. Другими словами, если `a{3}` отождествляется с записью «aaa», `(abc){3}` равносильно строке `abcabcabc`. Более наглядный пример: `bon+` соответствует фрагменту текста, начинающемуся с буквосочетания «bon», за которым следует одна или больше букв «n» (скажем, `bonnet`). Однако запись `(bon)+` соответствует **строке**, начинающейся с букв «bon», за которой следует один или больше слогов «bon» (`bonbon`, например). Скобками выделяется и контролируется группа символов, позволяя создавать настоящие регулярные выражения.

Подобным образом, в то время как запись `yes|no` отождествляется с «yes» или «no» (`yes` плюс `s` или `n`, плюс `o`), то `(Yes) I (No)` полностью соответствует только одному из этих слов.

Классы

Вне зависимости от **того**, как вы комбинируете символы и метасимволы в регулярные выражения, они будут полезны только для поиска конкретных слов. А если необходимо найти все слова, состоящие из четырех строчных **букв**, или все числа в строке? В этом случае определяются и используются классы (**формальное название** - классы **символов**).

Классы создаются с помощью размещения символов (литералов) в **квадратных** скобках (`[]`). Например, можно определить класс из всех гласных английского алфавита с помощью записи `[aeiou]`. Допустимо использовать дефис для задания диапазона следующих подряд символов, например: `[a-z]` - любая строчная буква, `[A-Z]` - любая прописная буква, `[A-Za-z]` - вообще любая буква английского языка, `[0-9]` - любая цифра. Следует отметить, что классы в шаблонах всегда будут отождествляться только с одним символом из класса или диапазона, а такой шаблон, как `[a-z]{3}` соответствует группе из 3 любых строчных букв: `abc`, `aaa` и т.п.

Знак «крышечка», который, как мы уже знаем, обычно привязывает шаблон к началу строки, при задании класса символов в квадратных скобках имеет **особенное** значение. Если знак «крышечка» указан сразу за левой квадратной **скобкой**, то «крышечка» используется для исключения группы символов. Так, `[^a]` соответствует любому знаку, кроме буквы «a», а запись `[^0-9]` задает любой **нецифровой** символ. Как вы думаете, что будет обозначать такой класс: `[^^^]`? Правильно, это любой знак, кроме самой «крышечки».

В РНР есть несколько предопределенных классов, которые можно спокойно использовать при составлении регулярных выражений: `[[:alpha:]]` соответствует любой букве (аналог `[A-Za-z]`), `[[:digit:]]` - любой цифре (эквивалент `[0-9]`), `[[:alnum:]]` - **любой** букве и цифре (**так же**, как и `[A-Za-z0-9]`).

Определяя собственные классы **символов**, а также используя встроенные в РНР (см. табл. С.3 приложения С), можно создавать любые шаблоны для **обработки** текста и получаемой от пользователя информации.

Примеры шаблонов

Изучив предыдущие разделы, вы можете самостоятельно создавать полезные шаблоны. Примеры, приведенные ниже, облегчат вам эту задачу.

```
«^([0-9]{5}){1}([0-9]{4})?$$$»
```

Данный шаблон предназначен для проверки правильности ввода почтового индекса, принятого в США. Индекс должен состоять из пяти цифр, затем могут идти дефис и еще четыре цифры. Начало строки обозначено знаком вставки, а в первых скобках **указано**, что необходимо именно пять цифр (за классом в фигурных скобках указано количество используемых символов). Во вторых скобках сначала вы видите дефис, затем класс и точное количество используемых символов этого класса. Вопросительный знак показывает, что

данная часть индекса необязательна (то есть или еще дефис и 4 цифры, или больше **ничего**). Знак доллара обозначает, что это конец строки, то есть требуется, чтобы в строке больше не было никаких символов.

А вот более сложный шаблон для проверки адресов электронной почты:

```
«^([0-9a-z]+)(-[0-9a-z\._-]+)@(-[0-9a-z\._-]+)\.([0-9a-z]+)»
```

Так как для поиска соответствий мы используем только функцию `ereg()`, следовательно, не стоит беспокоиться о регистре букв, то классы символов будут включать только строчные английские буквы.

Вернемся к предыдущему сценарию (листинг 8.2) и зададим более точную формулу для проверки электронного адреса. По сравнению с разобранным **выше примером**, в новом скрипте (листинг 8.3) шаблон в переменной `$Pattern` в конце строки включает пробел. Таким образом, если адрес электронной почты находится внутри любого другого текста, слова после адреса не включаются в шаблон.

Листинг 8.3 ▼ В сценарий `HandleForm.php` был вставлен новый шаблон для комплексной проверки правильности адреса электронной почты.

```
1  <HTML>
2  <HEAD>
3  <TITLE>Using Regular Expressions</TITLE></HEAD>
4  <BODY>
5  <?php
6  /* Эта страница получает и обрабатывает данные, принятые
   от generated by "form.html". */
7  if (($Array["FirstName"]) AND ($Array["LastName"])) {
8      $Array["Name"] = $Array["FirstName"] . " " .
        $Array["LastName"];
9  } else {
10     print ("Please enter your first and last names.<BR>\n");
11 }
12
13 $Pattern = "^([0-9a-z]+)(-[0-9a-z\._-]+)@(-[0-9a-z\._-]+)\.([0-9a-
   z]+) ";
14 if (ereg($Pattern, $Array["Email"])) {
15     print ("Your information has been received!<BR>\n");
16 } else {
17     print ("Please enter a valid email address!\n");
18 }
19 ?>
20 </BODY>
21 </HTML>
```

Обратные ссылки

Понятие «обратная **ссылка**» станет особенно важным, когда мы начнем извлекать из текста нужные фрагменты, а также будем проводить **контекстную** замену одной подстроки на другую.

Подробно разобранный выше шаблон почтового индекса состоит из двух групп, заключенных в скобки: `{[0-9]{5}}` и `(-[0-9]{4})`. При сопоставлении регулярного выражения и обрабатываемого текста в РНР автоматически нумеруются заключенные в скобки группы начиная с единицы. Используя обратные ссылки, можно обратиться к каждой отдельной группе символов с помощью двойного обратного следа (`\\`), поставленного перед соответствующим числом. Например, если с помощью этого шаблона **проверить** почтовый код **94710-0001**, обратная ссылка номер `\\1` даст число **94710**, а обратная ссылка `\\2` будет содержать запись **«-0001»** (конечно, без кавычек. Я взял этот фрагмент в кавычки, только чтобы показать, что дефис является частью шаблона и найденной подстроки). В следующем разделе приведен пример практического использования обратных ссылок.

Сопоставление с шаблоном и его замена

Несмотря на то что функции `ereg()` и `eregi()` следует использовать для проверки правильности строки, мы можем Поднять свои программы на качественно новый уровень, применяя шаблон для нахождения определенного фрагмента, а затем замены его на другой шаблон или конкретный текст. Синтаксис этих функций выглядит следующим образом:

```
ereg_replace("pattern", "replace", "string");
```

или:

```
$Pattern = "pattern";
$Replace = "replace";
$string = "string";
eregi_replace($Pattern, $Replace, $String);
```

Использовать указанные функции допустимо, если есть желание превратить вводимый пользователем адрес сайта (URL) в синтаксически правильную HTML-ссылку, инкапсулировав его тэгами ``. Сделаем это, модифицировав файл `form.html` (листинг 8.1) и сценарий `HandleForm.php` (листинг 8.3).

Использование шаблона с целью проверки и последующей замены с помощью функции `eregi_replace`

1. Откройте файл `form.html` в текстовом редакторе. Слегка изменим страницу `form.html` для того, чтобы она принимала адрес URL и описание.
2. Замените строку 9 (листинг 8.4), которая берет адрес электронной почты, на следующую:

```
URL <INPUT TYPE=TEXT NAME="Array{URL}" SIZE=60xBR>
```

Листинг 8.4 т Страница `form.html`, слегка модифицированная для ввода URL, описания вместо адреса электронной почты и комментария.

```
1 <HTML>
2 <HEAD>
```



```

3  <TITLE>HTML Form</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleForm.php" METHOD=POST>
7  First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20><BR>
8  Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
9  TOL <INPUT TYPE=TEXT NAME="Array[URL]" SIZE=60><BR>
10 Description <TEXTAREA NAME="Array[Description]" ROWS=5 COLS=40></
    TEXTAREA><BR>
11 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12 </FORM>
13 </BODY>

```

3. Замените строку 10 с комментариями на следующую:

```

Description <TEXTAREA NAME="Array[Description]" ROWS=5 COLS=40>
•</TEXTAREA><BR>

```

4. Сохраните сценарий и загрузите его на сервер.

А теперь подредактируем страницу HandleForm.php.

5. Откройте текущую версию HandleForm.php в текстовом редакторе (листинг 8.3).
6. Замените строку 12 таким кодом:

```
$Pattern = "(http://)?([^\s:]+)([^\s:]*\.\s?&=)";
```

Это не очень строгий шаблон, созданный для распознавания URL. Действительно, данный шаблон больше подходит для поиска и замены, чем для проверки правильности введенных данных, так как позволяет вставить в адрес URL некоторые потенциально опасные для вашего сайта символы.

Шаблон содержит три группы: http://, массив URL и замыкающую часть URL. Адрес URL может начинаться с записи http://, а может и нет. Для проверки этого пишем http, ставим двоеточие и два слеша. Вопросительный знак указывает, что раздел необязателен.

Вторая часть URL состоит из разных символов, кроме пробелов: букв, цифр, дефисов, знаков подчеркивания, точек и т.д. И наконец, в последней части имеются буквенно-цифровые символы, амперсанта и знаки равенства. Эта часть шаблона будет проверять последнюю часть URL типа .com/php/.

7. Следующая строка выглядит так:

```
$Replace = "<a href=\"http://\\2\\3\" target=\"_new\">\\2\\3</a>";
```

Здесь мы определили заменяемый текст. С помощью обратной ссылки можно взять найденный фрагмент строки и вставить его в новую строку. Так как фрагмент http:// является необязательным, самый легкий способ обеспечения единообразия - опустить его при замене (обратите внимание, что ссылки \\1 не существует) и затем всегда вставлять запись http://, чтобы все ссылки

были одного формата. В новую строку сначала добавляем тэг `<a href=`, затем вставляем правильный адрес URL, еще раз помещаем вторую и третью части ссылки и закрываем HTML-тэг ``.

```
8. $Array["URL"] = eregi_replace($Pattern, $Replace, $Array["URL"]);
```

В этой строке программы происходит замена. Мы записываем новую измененную строку на старое место, запрограммировав тем самым необходимое редактирование введенной строки. Теперь при отсылке URL в браузер, как в следующей строке, адрес будет появляться как активная ссылка. Эта измененная строка может быть также сохранена в базе данных или в файле, и тогда ее удастся использовать в дальнейшем.

```
9. print "Your submission-$Array[URL]-has been received!<BR>\n");
```

Наконец, мы завершаем условную инструкцию `if`. Сохраните сценарий, загрузите его на сервер (листинг 8.5) и протестируйте в браузере (рис. 8.5-8.7). Если сценарий работает правильно, то он берет переданный пользователем URL, проверяет его правильность и превращает в активную ссылку. Подобный прием вам наверняка пригодится при работе со многими Web-приложениями.

Листинг 8.5 т Функция `eregi_replace()` автоматически превратит переданный пользователем адрес URL в активную ссылку. Это возможно благодаря использованию обратных ссылок.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Using Regular Expressions</TITLE>x/HEAD>
4 <BODY>
5 <?php
6 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7 if (($Array["FirstName"]) AND ($Array["LastName"])) {
8     $Array["Name"] = $Array["FirstName"] . " " . $Array["LastName"];
9 } else {
10     print ("Please enter your first and last names.<BR>\n");
11 }
12 $Pattern = "(http://)?([^\s:]+)([^\s:]*\.\s?|&=)";
13 $Replace = "<a href=\"http://\2\3\" target=\"_new\">\2\3</a>";
14 $Array["URL"] = eregi_replace($Pattern, $Replace, $Array["URL"]);
15 print ("Your submission-$Array[URL]-has been received!<BR>\n");
16 ?>
17 </BODY>
18 </HTML>
```

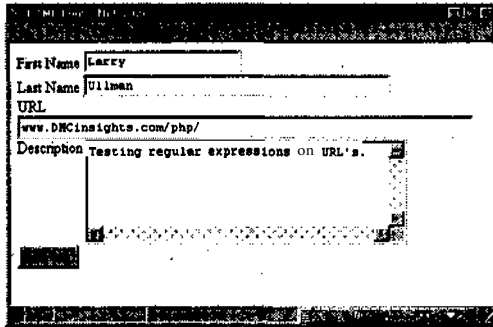


Рис. 8.5 ▾ Страница `form.html` требует ввести адрес URL, который будет проверен и поправлен с помощью регулярных выражений (рис. 8.6)

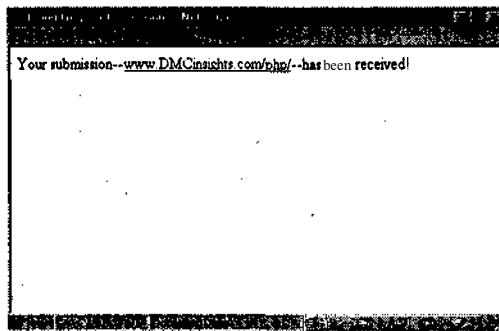


Рис. 8.6 ▴ С помощью функции `eregi_replace()` переданный URL (с префиксом `http://` или без него) можно превратить в активную ссылку

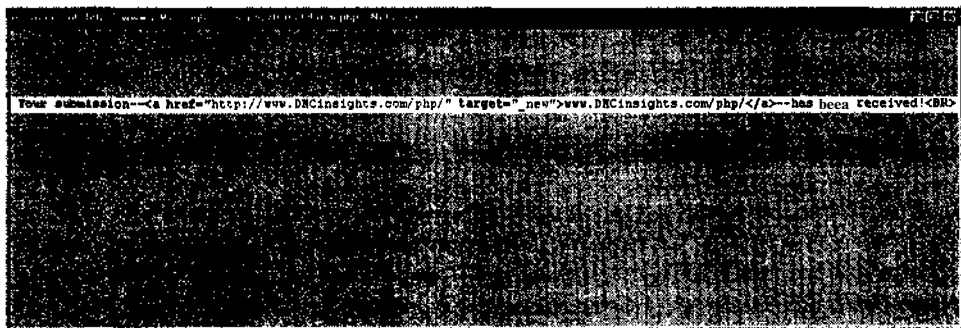


Рис. 8.7 ▴ Исходный текст страницы показывает HTML-теги `<A HREF>`, добавленные к введенному пользователем адресу URL

Глава

Создание функций

Функции так же важны для написания программ, как и регулярные выражения, но их легче понять и применять. Функции, речь о которых шла выше (`phpinfo()`, `count()`, `ereg_replace()`), и многие другие, являются стандартными и представлены в любой реализации языка **PHP**. Данная глава посвящена созданию своих собственных функций. **Функции**, изначально встроенные в **PHP** или созданные программистом, работают совершенно одинаково.

Создание функций может сэкономить много времени при написании программ. Более того, это серьезный шаг к созданию солидной **библиотеки PHP**-кода, который можно использовать в будущих **Web**-приложениях.

В данной главе говорится о том, как создавать функции **для** выполнения **определенных** задач. Вы научитесь передавать информацию в функцию и получать обратно результат, а также узнаете, как функции и переменные работают вместе.

Создание и использование простых функций

При программировании определенные фрагменты кода часто многократно используются как в одном сценарии, так и в нескольких. Оформив эти подпрограммы в виде отдельной функции, вы сэкономите время и облегчите программирование, особенно при работе с быстро растущими сайтами. Если функция создана, она будет выполнять заданные в ней действия при каждом вызове так же, как функция `print()` посылает текст в браузер при каждом ее использовании.

Синтаксис для создания определяемых пользователем функций следующий:

```
function FunctionName () {  
    statement(s);  
}
```

При именовании функции нужно руководствоваться теми же самыми правилами, что и при присвоении имен переменным, за исключением **использования** знака доллара. При определении и вызове функций знак \$ не нужен. Имена функций должны быть значимыми, как и названия переменных (например, CreateHeader - более подходящее имя функции, чем Function1). Помните, что нельзя использовать пробел, так как в этом случае имя **функции** будет состоять из **двух** слов, что приведет к ошибке (знак подчеркивания является общепринятой заменой пробела: Create_Header - правильное имя функции).

В область, ограниченную фигурными скобками, можно вставлять любой **PHP-код**, в том числе вызовы других функций. Количество инструкций в функции может **быть** любым, но не забывайте после каждой инструкции ставить точку с запятой.

При написании функции желательно придерживаться тех же правил **форматирования**, что **использовались** на протяжении всей книги, хотя это и не требуется синтаксисом. Главное, чтобы все **необходимые** элементы были на месте. К ним относятся: слово function, имя функции, открывающая и закрывающая круглые скобки, открывающая и закрывающая фигурные скобки и последовательность инструкций.

Инструкции функции традиционно размещаются с отступом от начала предыдущей строки, это делает код более ясным. В любом случае выберите формат, который вам больше нравится (он должен быть синтаксически и логически правильным), и придерживайтесь его.

Функция, создаваемая вами, вызывается с помощью обращения к ней, как и любая встроенная функция. Строка кода FunctionName(); иницирует выполнение включенных в эту функцию инструкций.

Обратимся к генерирующему пароли сценарию (глава 5) и перепишем его с использованием функции.

Создание и вызов базовой функции

1. Откройте файл passwords.php в текстовом редакторе (листинг 9.1).

Листинг 9.1 Это исходный сценарий для создания паролей, в котором не используются функции. Создадим первую функцию с помощью кода, составляющего основу данной страницы.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Password Generator</TITLE></HEAD>
4 <BODY>
5 <?php
6 $String = "This is the text which will be encrypted so that we may
   create random and secure passwords!";
7 $Length = 8; // Измените это значение, чтобы установить длину пароля.
   32 символа - максимум.
8 $String = md5($String);
9 $StringLength = strlen($String);
10 srand ((double) microtime() * 1000000);
```

```

11 $Begin = rand(0,($StringLength-$Length-1)); // Pick an arbitrary
    starting point.
12 $Password = substr($String, $Begin, $Length);
13 print ("Your recommended password is:<P><BIG>$Password</BIG>\n");
14 ?>
15 </BODY>
16 </HTML>

```

2. Поменяйте стандартный HTML-заголовок на тэг <?php.

Мы должны определить функцию до того, как она будет вызвана (это требуется в PHP 3.x), поэтому напомним функцию в самом начале сценария, до HTML-кода.

3. function CreatePassword () {

Назовем функцию CreatePassword. Имя соответствует назначению функции и легко запоминается.

4. Теперь поместим PHP-код (строки 6–13) из сценария passwords.php в функцию. Чтобы было видно, что эти строки принадлежат функции, я бы рекомендовал разместить их с отступом от строки с именем функции (листинг 9.2).

```

$String = "This is the text which will be encrypted so
→that we may create random and secure passwords!";
$Length = 8; // Измените это значение, чтобы установить длину пароля.
→32 символа - максимум.
$String = md5($String);
$StringLength = strlen($String);
srand ((double) microtime() * 1000000);
$Begin = rand(0,($StringLength-$Length-1)); // Pick an
→Получение произвольной точки старта".
$Password = substr($String, $Begin, $Length);
print ("Your recommended password is:
→<P><BIG>$Password</BIG><P> \n");

```

5. Закройте функцию на следующей строке фигурной скобкой (}).

Пропуск открывающей или закрывающей скобки - распространенная причина ошибок, поэтому тщательно соблюдайте правила синтаксиса.

6. Закройте PHP-код тэгом ?>.

Мы закрываем PHP-раздел страницы, так как собираемся писать HTML-код. При желании можно оставить раздел PHP открытым и послать код в браузер с помощью функции print (). В таком случае пропустите этот пункт.

7. Напишите стандартный HTML-заголовок.

```

<HTML><HEAD><TITLE>Password Generator within a Function</TITLE></HEAD>
-<BODY> -

```

8. Откройте новый PHP-раздел страницы тэгом `<?php`.

В HTML-документ можно вставлять несколько разделов PHP-кода, это вполне обычная практика.

9. `CreatePassword()`;

Если функция уже определена и вы хотите ее использовать, просто вызовите ее по имени (будьте внимательны и не ошибитесь в написании). Не забывайте про скобки.

10. `?></BODY></HTML>`

Закройте второй PHP-раздел и HTML-тэги.

11. Сохраните сценарий, загрузите его на сервер (листинг 9.2) и протестируйте в браузере (рис. 9.1).

Листинг 9.2 т Размещение функции в самом начале сценария - хороший способ выделить ее. Подобным образом размещение инструкций функции с отступом помогает установить их принадлежность к функции. Затем одна строка кода в основном теле сценария выполняет многочисленные инструкции.

```

1  <?php
2  function CreatePassword () {
3      $String = "This is the text which will be encrypted so that we may
        create random and secure passwords!";
4      $Length = 8; // Измените это значение, чтобы установить длину
        пароля. 32 символа - максимум.
5      $String = md5($String);
6      $StringLength = strlen($String);
7      srand ((double) microtime() * 1000000);
8      $Begin = rand(0,($StringLength-$Length-1)); // Pick an arbitrary
        starting point.
9      $Password = substr($String, $Begin, $Length) /•
10     print ("Your recommended password is: <P><BIG>$Password</BIG><P>
        \n");
11 } // Конец функции CreatePassword.
12 ?>
13 <HTML>
14 <HEAD>
15 <TITLE>Password Generator within a Function</TITLE></HEAD>
16 <BODY>
17 <?php
18 CreatePassword(); У/ Вызов функции.
19 ?>
20 </BODY>
21 </HTML>
```

Если на вашем сервере установлен PHP 3.x, сначала необходимо определить функцию, и только после этого вызывать ее. Несмотря на то что в PHP 4.0 это ограничение снято, лучше располагать функции в самом начале сценария. Это гарантирует создание функции до ее вызова.

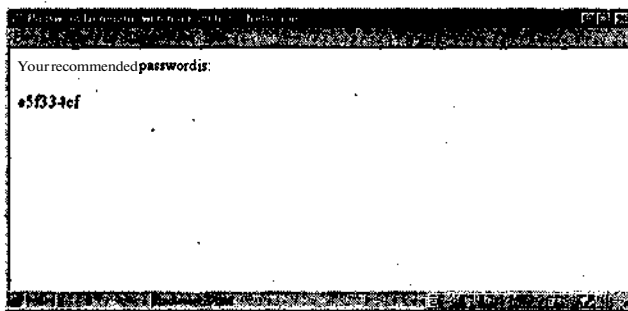


Рис. 9.1 т Поместив несколько строк сценария `password.php` в отдельную функцию, мы начали писать более структурированные, модульные программы. Это первый шаг к созданию сложных динамических сайтов, пусть даже мы получили тот же видимый результат, что и до создания функции

Имена создаваемых вами функций, так же как и имена встроенных функций PHP, не чувствительны к регистру. Поэтому не имеет никакого значения, напишете ли вы `createpassword` или `CreatePassword`.

Создание и вызов функций, принимающих аргументы

Хотя разработать такую простую функцию, как была описана в примере выше, - весьма полезно, создать функцию, **принимающую** данные в виде параметров и соответствующим образом их обрабатывающую, - еще лучше. То, что принимает функция, называется *аргументом*. Количество аргументов, которые может принять функция, не ограничено. Мы уже обращали внимание на концепцию аргумента: функция `print()` принимает строку как аргумент, а затем посылает ее в браузер.

Синтаксис написания функции, принимающей аргументы, выглядит следующим образом:

```
function FunctionName ($Argument1, $Argument2 и т.д.) {
    statement(s);
}
```

Аргументы представлены в форме переменных, которым присваивается значение, посылаемое в функцию, когда ее вызывают. Функции, принимающие аргументы, вызываются так же, как и функции без аргументов. Однако необходимо помнить, что требуется передать соответствующее количество аргументов. Это можно сделать либо с помощью передачи переменных:

```
FunctionName ($Variable1, $Variable2 и т.д.);
```

либо поместив значения в кавычки:

```
FunctionName ("Value1", "Value2" и т.д.);
```


А вот и еще один из возможных вариантов:

```
FunctionName ($Variable1, "Value2" и т.д.);
```

Обратите внимание на следующее: аргументы передаются строго в порядке поступления, то есть первое значение в функции равно первому значению в строке вызова, второе - второму и т.д. Имена аргументов функции и передаваемых в нее параметров никак не связаны.

Это справедливо и для случаев, когда значение не было передано вовсе. В таких случаях считается, что значение не определено, и используется специальное значение null. Это не математический нуль, который сам по себе является значением, а именно неопределенное значение. То же самое происходит, если **функция** принимает четыре аргумента, а передано только три - четвертый аргумент будет не определен.

Чтобы показать работу функций, принимающих **аргументы**, обратимся к странице hello.php (глава 6). Поместим код приветствия в простую функцию, которая принимает аргумент, - имя пользователя.

Выполнение действия

1. Откройте файл hello.php в текстовом редакторе (листинг 9.3).

Листинг 9.3 т. Существующая страница hello.php создает привязанное ко времени суток приветствие.

```
1  <HTML>
2  <HEAD>
3  <TITLE>If-elseif Conditionals</TITLE></HEAD>
4  <BODY>
5  <?php
6  if ($Username) {
7      print ("Good ");
8      if (date("A") == "AM") {
9          print ("morning, ");
10     } elseif ( ( date("H") >= 12 ) and ( date("H") < 18 ) ) {
11         print ("afternoon, ");
12     } else {
13         print ("evening, ");
14     } // Заккрытие if даты.
15     print ("{$Username}");
16     print ("!\n");
17 } else {
18     print ("Please log in.\n");
19 } // Заккрытие if имени пользователя.
20 ?>
21 </BODY>
22 </HTML>
```

2. Начните с открывающего PHP-тэга <?php, затем возьмите строки 7-16 из исходного сценария (листинг 9.3) и поместите их в функцию в самом начале программы.

```
function GreetUser ($TheUser) {
    print ("Good ");
```

```

if (date("A") == "AM") {
    print ("morning, ");
} elseif ((date("H")>12) and (date("H")<18)) {
    print ("morning, ");
} else {
    print ("evening, ");
} // Заккрытие if даты.
print ("{$TheUser}");
print ("!\n");
} // Конец функции GreetUser.

```

Мы только дали функции имя. Сама же она состоит из кода, используемого ранее. Переменная `$TheUser` будет получать значение `$Username`, посылаемое в функцию при ее вызове (см. строку 21 листинга 9.4).

3. Закройте PHP-раздел и создайте HTML-заголовок.

```
?><HTML><HEAD><TITLE>The GreetUser Function</TITLE></HEAD><BODY>
```

4. Снова откройте PHP-раздел и поместите туда условную конструкцию `if-else`, а также вызов новой функции.

```

<?php
if ($Username) {
    GreetUser ($Username); // Вызов функции.
} else {
    print ("Please log in.\n");
} // Заккрытие if имени пользователя.

```

Условная конструкция `if ($Username)` используется здесь для того, чтобы функция вызывалась, только если существует, переменная `$Username`.

5. Закройте PHP и HTML.

```
?></BODY></HTML>
```

6. Сохраните сценарий (листинг 9.4), загрузите его на сервер и протестируйте в браузере (рис. 9.2). Не забудьте отправить в сценарий значение имени пользователя, добавив его к адресу URL или через HTML-форму. В противном случае вы получите результат, подобный тому, что представлен на рис. 9.3.

Листинг 9.4 Здесь показана функция, которая принимает аргумент, передаваемый в нее при вызове. Функция вызывается, только если переменная `$Username` имеет значение.

```

1 <?php
2 function GreetUser ($TheUser) {
3     print ("Good ");
4     if (date("A") == "AM") {
5         print ("morning, ");
6     } elseif ((date("H")>12) and (date("H")<18)) {

```

```
7      print ("morning, ");
8  } else {
9      print ("evening, ");
10 } // Закрытие if даты.
11 print ("{$TheUser}");
12 - print ("!\n");
13 } // Конец функции GreetUser.
14 ?>
15 <HTML>
16 <HEAD>
17 <TITLE>The GreetUser Function</TITLE></HEAD>
18 <BODY>
19 <?php
20 if ($Username) {
21     . GreetUser ($Username); // Вызов функции.
22 } else {
23     print ("Please log in.\n");
24 } // Закрытие if имени пользователя.
25 ?>
26 </BODY>
27 </HTML>
```

В главе 13 говорится о том, как помещать определенную информацию во внешние файлы (их можно будет использовать в нескольких сценариях). Часто внешний файл - самое удачное место для хранения функций, таких как GreetUser. В таком случае они доступны из любого раздела сайта.

Если вы посылаете число в функцию как аргумент, оно необязательно должно быть заключено в кавычки, как строка. С другой стороны, не повредит использовать кавычки для передачи аргументов.

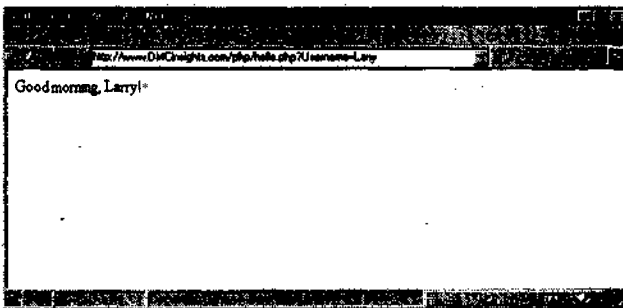


Рис. 9.2 Созданная нами функция для приветствия пользователя в соответствии с временем суток, хотя ее результат и похож на приветствие из оригинальной страницы hello.php, может быстро генерировать многочисленные приветствия без переписывания кода

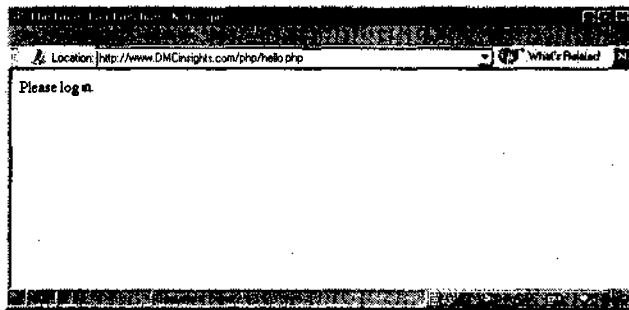


Рис. 9.3 т Все лучше и лучше понимая принципы программирования на языке PHP, не забывайте о простых фактах. Вы получите такое **сообщение**, если забыли послать имя пользователя в страницу `hello.php` (сравните с рис. 9.2)

Создание и использование функций, возвращающих значение

Мы научились создавать функции, которые принимают аргументы. Следующий шаг - заставить функцию вернуть результат, или значение. Для этого необходимо знать и применять следующее: во-первых, в функции надо использовать инструкцию `return`; во-вторых, при вызове функции нужно как-то обработать возвращаемое значение. Обычно оно присваивается переменной, но можно, например, просто распечатать результат. Ниже приведен основной формат функции, которая принимает аргумент и возвращает значение:

```
function FunctionName ($Argument) {
    statement(s);
    return $Value;
}
```

Обычно данная функция используется со строкой кода наподобие представленной ниже:

```
$Value = FunctionName($Variable);
```

Обратите внимание на **то**, что мы присвоили возвращаемое значение функции переменной. Создадим две функции на основе страницы `numbers.php` (глава 4, модифицированный вид - глава 6).

Выполнение действия

1. Откройте файл `numbers.php` в текстовом редакторе (листинг 9.5).

Листинг 9.5 т Это последний вариант страницы `numbers.php` (глава 6). Мы разделим инструкции этого сценария на отдельные модули, создав две функции.

```
1 <HTML>
2 <HEAD>
```

```

3 <TITLE>Conditionals</TITLE>
4 </HEAD>
5 <BODY>
6 <?php
7 /* Переменная $Quantity должна быть передана в эту страницу из формы.
   $Discount необязательна. */
8 $Cost = 20.00;
9 $Tax = 0.06;
10 if ($Quantity) {
11     $Quantity = abs($Quantity);
12     $Discount = abs($Discount);
13     Tax++; // $Tax составляет 1.06.
14     $TotalCost = ($Cost * $Quantity);
15     if ( ($TotalCost < 50) AND ($Discount) ) {
16         print ("Your \$$Discount will not apply because the total value
17             of the sale is under $50!\n<P>");
18     }
19     . if ($TotalCost >= 50) {
20         $TotalCost = $TotalCost - $Discount;
21     }
22     $TotalCost = $TotalCost * $Tax;
23     $Payments = round ($TotalCost, 2) / 12;
24     // Печать результатов.
25     print ("You requested to purchase $Quantity widget(s) at \$$Cost
26         each.\n<P>");
27     print ("The total with tax, minus your \$$Discount, comes to $");
28     printf ("%01.2f", $TotalCost);
29     print (".\n<P>You may purchase the widget(s) in 12 monthly
30         installments of $");
31     printf ("%01.2f", $Payments);
32     print (" each.\n<P>");
33 } else {
34     print ("Please make sure that you have entered both a quantity and
35         an applicable discount and then resubmit.\n");
36 }
37 ?>
38 </BODY>
39 </HTML>

```

2. Откройте первый PHP-раздел и создайте функцию CalculateTotal(), которая содержит строки 14-21 из исходной страницы.

```

<?php
function CalculateTotal ($HowMany, $Price, $TaxRate, $Savings) {
    $TaxRate++; // $TaxRate is now worth 1.06.
    $TheCost = ($Price * $HowMany);
    if ( ($TheCost < 50) AND ($Savings) ) {
        print ("Your \$$Savings will not apply because the total value
        -of the sale is under $50!\n<P>");
    }
    if ($TheCost >= 50) {
        -$TheCost = $TheCost - $Savings;
    }
    $TheCost = $TheCost * $TaxRate;
    return $TheCost;
}

```

Эта функция берет четыре аргумента (количество купленного товара (поштучно), цена за штуку, ставка налога и возможная скидка), проводит необходимые вычисления для определения полной стоимости и возвращает это значение.

3. Создайте вторую функцию.

```
function CalculatePayments ($Amount, $NumberPayments) {
    $Payments = round($Amount, 1) / $NumberPayments;
    $Payments = sprintf ("%01.2f", $Payments);
    return $Payments;
}
```

Функция `CalculatePayments()` возьмет два аргумента – полную стоимость и количество платежей – и выполнит простой подсчет. Сначала будет определен размер платежа с помощью формулы, которая изначально писалась для этих целей в файле `numbers.php`.

Затем функцией `sprintf()` будет отформатировано значение переменной `$Payments`. Как уже упоминалось в главе 4, эта функция работает точно так же, как и `printf()`, только ничего не отправляет в браузер, а выдает строковое значение, которое можно снова присвоить той же переменной. В конце концов, функция `sprintf()` возвращает значение переменной `$Payments`.

Размещение в отдельной функции всех вычислений приносит двойную выгоду: во-первых, позже функцию будет легче найти и модифицировать, так как она расположена в самом начале сценария, а не спрятана где-то в коде; во-вторых, если возникнет необходимость снова использовать действие в сценарии, это можно будет сделать без дублирования кода.

4. Закройте первый PHP-раздел и создайте стандартный HTML-заголовок.

```
?><HTML><HEAD><TITLE>Calculation Functions</TITLE></HEAD><BODY>
```

5. Откройте новый PHP-раздел и задайте переменные.

```
<?php
$Cost = 20.00;
$Tax = 0.06;
```

Мы не изменили значение переменной `$Cost`, но при желании вы можете задать свое собственное значение как этой переменной, так и переменной `$Tax`.

6. Напишем основной PHP-сценарий.

```
if ($Quantity) {
    $Quantity = abs($Quantity);
    $Discount = abs($Discount);
    $TotalCost = CalculateTotal ($Quantity, $Cost,$Tax, $Discount);
    // Печать результатов.
    print ("You requested to purchase $Quantity widget(s)
    -at \$$Cost each.\n<P>");
    print ("The total with tax, minus your \$$Discount, comes to $");
```

```

printf ("%01.2f", $TotalCost);
print (".\n<P>You may purchase the widget (s) in 12
-monthly installments of $");
print (CalculatePayments($TotalCost, "12"));
print (" each.\n<P>");
} else {
print ("Please make sure that you have entered both a
-quantity and an applicable discount and then resubmit.\n");
}

```

За исключением вызовов функций этот раздел кода полностью идентичен тому, что представлен в главе 6. Но теперь мы повысили наглядность страницы, выделив вычисления в отдельные функции. Также данный раздел демонстрирует два способа использования **функций**, возвращающих значения. На строке 31 переменной присваивается значение, которое возвращается функцией CalculateTotal(), а на строке 37 немедленно распечатывается результат вызова функции CalculatePayments().

7. Закройте PHP-раздел и HTML.

```
?></BODY></HTML>
```

8. Сохраните сценарий (листинг 9.6), загрузите его на сервер и протестируйте в браузере (рис. 9.4). Не забудьте послать в сценарий количество единиц товара (и значение **скидки**, если хотите), добавив его к адресу URL или через HTML-форму. Если количество единиц товара будет неизвестно, на экране появится сообщение «Please make sure», сгенерированное строкой 40..

Листинг 9.6 Обе функции в этом сценарии возвращают значения. При вызове функции CalculateTotal() возвращаемое значение присваивается переменной. При вызове функции CalculatePayments() возвращаемое значение передается в браузер.

```

1  <?php
2  function CalculateTotal ($HowMany, $Price, $TaxRate, $Savings) {
3      $TaxRate++; // $TaxRate is now worth 1.06.
4      $TheCost = ($Price * $HowMany);
5      if ( ($TheCost < 50) AND ($Savings) ) {
6          print ("Your \$$Savings will not apply because the total value
              of the sale is under $50! \n<P>");
7      }
8      if ($TheCost >= 50) {
9          $TheCost = $TheCost - $Savings;
10     }
11     $TheCost = $TheCost * $TaxRate;
12     return $TheCost;
13 } // Конец функции CalculateTotal.
14 function CalculatePayments ($Amount, $NumberPayments) {
15     $Payments = round($Amount, 2) / $NumberPayments;
16     $Payments = sprintf ("%01.2f", $Payments);
17     return $Payments;
18 } // Конец функции CalculatePayments.

```

```

19 ?>
20 <HTML>
21 <HEAD>
22 <TITLE>Calculation Functions</TITLE>
23 </HEAD>
24 <BODY>
25 <?php
26 $Cost = 20.00;
27 $Tax = 0.06;
28 if ($Quantity) {
29     $Quantity = abs($Quantity);
30     $Discount = abs($Discount);
31     $TotalCost = CalculateTotal ($Quantity, $Cost, $Tax, $Discount);
32     // Печать результатов.
33     print ("You requested to purchase $Quantity widget(s) at \$$Cost
34     each.\n<P>");
35     print ("The total with tax, minus your \$$Discount, comes to $");
36     printf ("%01.2f", $TotalCost);
37     print (".\n<P>You may purchase the widget(s) in 12 monthly
38     installments of $");
39     print (CalculatePayments($TotalCost, "12"));
40     print (" each.\n<P>");
41 } else {
42     print ("Please make sure that you have entered both a quantity and
43     an applicable discount and then resubmit.\n");
44 }
45 ?>
46 </BODY>
47 </HTML>

```

В функции CalculatePayments() допустимо заменить строку sprintf() на printf(), убрать инструкцию return, и результат будет сразу же распечатан. Хотя мы не сделали этого в нашем сценарии, вполне разумно, чтобы функция распечатывала результаты.

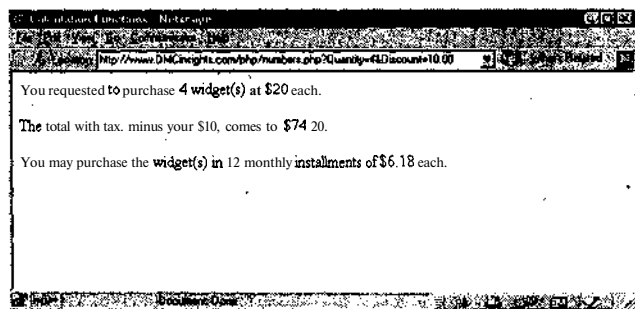


Рис. 9.4 ▼ Хотя результаты аналогичны тем, что удавалось получать и ранее, рассмотренные функции могут быть использованы в любых сценариях по всему сайту. Создание подобных функций ускорит выполнение будущих проектов, так как код можно использовать многократно

Только одна инструкция `return` будет выполнена в функции, хотя функция может иметь многочисленные инструкции `return`. Допустим, вы решили создать функцию, которая бы проверяла выполнение условия и возвращала соответствующее значение. В этом случае вы бы написали такой код:

```
if (условие) {  
    return TRUE;  
} else {  
    return FALSE;  
}
```

Результат, возвращаемый функцией, был бы истинным или ложным в зависимости от того, **выполнено** ли условие.

Инструкция `return` возвращает только одно значение. Для получения множественных значений необходимо использовать массивы. О том, где найти более подробную информацию о получении множественных значений, говорится в приложении С.

Переменные и функции

Когда в главе 2 мы обсуждали использование переменных, то не затронули понятие *области их действия*. Возвратимся к теме переменных и обсудим, как работают вместе переменные и функции.

Область действия переменных и глобальные переменные

Во втором разделе данной главы, говоря о передаваемых в функцию параметрах, мы отметили, что переменные могут быть отправлены в функцию в качестве *аргументов*. Однако можно использовать и переменную из функции, если это *глобальная* переменная. Переменная становится таковой с помощью инструкции `global`. Рассмотрим область действия, или область видимости переменной. *Область действия переменной* - это та часть программы, в которой переменная существует. По умолчанию переменные, которые вы создаете в сценарии, имеются, пока работает ваш сценарий. И наоборот, переменные среды (такие как `$OSTYPE`) существуют на сервере от момента запуска до выключения.

Впрочем, функции создают новую область видимости. Переменные функции - аргументы функции и любые переменные, *определенные* внутри функции, - существуют только в этой функции и недоступны извне. Другими словами, это локальные переменные с локальной областью действия. Подобным образом к внешней переменной можно обратиться, только передав ее в функцию как аргумент или же с помощью инструкции `global`. Последняя приблизительно означает: «Я хочу, чтобы эта переменная в функции была такой же, как и вне функции». Другими словами, глобальная инструкция превращает локальную переменную с областью видимости внутри функции в глобальную переменную

с областью действия в пределах всей программы. Любые изменения глобальной переменной в функции автоматически передаются в переменную с тем же именем, когда она используется вне функции (после **вызова** функции, конечно), без помощи команды `return`.

Синтаксис инструкции `global` следующий:

```
function FunctionName ($Argument) {  
    global $Variable;  
    statement(s);  
}
```

Рассмотрим подробнее, что же дает нам инструкция глобализации. Обычная переменная, даже если ее имя внутри функции полностью совпадает с таким же именем в основной части программы, **представляет** собой совершенно другую структуру (возможно, с другим значением, нежели переменная вне функции). Пусть строка вызова функции выглядит как `FunctionName ($Value1);`, а функция определена как `FunctionName ($Argument1)`. В момент обращения к функции PHP устанавливает значение `$Argument1` равным значению `$Value1`, и далее первое используется в теле функции. Если вы измените значение `$Argument1` внутри функции, это никак не отразится на значении `$Value1`. Если бы имена переменных случайно совпали, это бы ничего не изменило: они все равно являются двумя разными переменными. Область действия одной - внутри функции, другой - вне функции. Поэтому, чтобы избежать путаницы, при написании функций мы предусмотрительно использовали различные имена переменных в строке определения функции и в строке ее вызова.

Мы затронули эту тему, потому что на самом деле необязательно применять разные имена. Для удобства можно использовать одно и то же имя в строке функции и в строке вызова (тогда легко запомнить передаваемые **аргументы**), но помните, что это разные переменные. То, что происходит со значением переменной внутри функции, остается в функции. И наоборот: изменения переменной вне функции не действуют на переменную с тем же именем внутри функции. Но, если вы сделали переменную глобальной с помощью инструкции `global`, она становится одной и той же переменной и внутри, и снаружи.

Использование инструкции `global`

1. Откройте файл `numbers.php` в текстовом редакторе (листинг 9.6).
2. Уберите аргументы `$Price` и `$TaxRate` из функции `CalculateTotal`, чтобы строка 2 выглядела так:

```
function CalculateTotal ($HowMany, $Savings) {
```

Переменные `$Price` и `$TaxRate` будут введены в функцию как `$Cost` и `$Tax` с помощью инструкции `global`, поэтому нет необходимости использовать их как аргументы.

3. Добавьте две глобальные инструкции.

```
global $Cost;  
global $Tax;
```

Они «Прикажут» функции использовать переменные \$Cost и \$Tax, которые уже существуют вне ее.

4. Отредактируйте оставшуюся часть функции, заменив переменную \$Price на \$Cost, \$TaxRate на \$Tax, а \$TheCost на \$TotalCost.

```
$Tax++; // $Tax составляет 1.06.  
$TotalCost = ($Cost * $HowMany);  
if ( ($TotalCost < 50) AND ($Savings) ) {  
    print ("Your \$$Savings will not apply because  
    -the total value of the sale is under $50!\n<P>");  
}  
if ($TotalCost >= 50) {  
    $TotalCost = $TotalCost - $Savings;  
}.  
$TotalCost = $TotalCost * $Tax;  
return $TotalCost;
```

Так как теперь функцией используются разные имена переменных, необходимо соответствующим образом изменить вычисления. Чтобы лучше понять область действия переменной, мы поменяли также переменную \$TheCost на \$TotalCost. Запомните, что \$TotalCost в функции - это совсем не та переменная, как переменная с тем же именем, но находящаяся вне функции.

5. Ниже в сценарии, после строки \$Tax = 0.06; (строка 29), распечатайте текущее значение переменной. Оно будет меняться в ходе дальнейшего выполнения сценария.

```
print ("The tax value is currently \$$Tax .\n<P>");
```

Для показа того, что инструкция global вводит переменную в функцию и что любые реализованные в функции изменения применяются глобально, распечатаем значение \$Tax до и после вызова функции.

6. Измените строку вызова функции CalculateTotal (), чтобы переменные \$Tax и \$Cost больше не передавались как аргументы.

```
$TotalCost = CalculateTotal ($Quantity, $Discount);
```

-Так как функция принимает теперь только два аргумента, передача четырех аргументов вызовет ошибку.

7. Еще раз распечатайте значение переменной \$Tax.

```
print ("After calling the function, the tax value  
-is now \$$Tax .\n<P>");
```

Если бы не было инструкции global, то значения, распечатанные здесь и выше, были бы одинаковыми. Но, поскольку глобальная переменная \$Tax была модифицирована в функции, распечатанные значения будут разными.

8. Сохраните сценарий (листинг 9.7), загрузите его на сервер и протестируйте в браузере (рис. 9.5). Не забудьте отправить в сценарий значение количества (и значение скидки, если захотите), добавив его к адресу URL или через HTML-форму.

Листинг 9.7 Так как функции CalculateTotal требуются значения переменных \$Cost и \$Tax, они легко могут быть включены с помощью инструкции global. Помните, переменные больше нельзя передавать как аргументы – это может запутать программиста и вызвать определенные ошибки в работе приложения.

```

1  <?php
2  function CalculateTotal ($HowMany, $Savings) {
3      global $Cost;
4      global $Tax;
5      $Tax++; // $Tax составляет 1.06.
6      $TotalCost = ($Cost * $HowMany);
7      if ( ($TotalCost < 50) AND ($Savings) ) {
8          print ("Your \$$Savings will not apply because the total value
          of the sale is under $50!\n<P>") ;
9      }
10     if ($TotalCost >= 50) {
11         $TotalCost = $TotalCost - $Savings;
12     }
13     $TotalCost = $TotalCost * $Tax;
14     return $TotalCost;
15 } // Конец функции CalculateTotal.
16 function CalculatePayments ($Amount, $NumberPayments) {
17     $Payments = round($Amount, 2) / $NumberPayments;
18     $Payments = sprintf ("%01.2f", $Payments);
19     return $Payments;
20 } // Конец функции CalculatePayments.
21 ?>
22 <HTML>
23 <HEAD>
24 <TITLE>Calculation Functions</TITLE>
25 </HEAD>
26 <BODY>
27 <?php
28 $Cost = 20.00;
29 $Tax = 0.06;
30 print ("The tax value is currently \$$Tax . \n<P>");
31 if ($Quantity) {
32     $Quantity = abs($Quantity);
33     $Discount = abs($Discount);
34     $TotalCost = CalculateTotal ($Quantity, $Discount);
35     print ("After calling the function, the tax value is now \$$Tax
        .\n<P>");
36     // Печать результатов.
37     print ("You requested to purchase $Quantity widget (s) at \$$Cost
        each.\n<P>");
38     print ("The total with tax, minus your \$$Discount, comes to $");
39     printf ("%01.2f", $TotalCost);
40     print (".\n<P>You may purchase the widget (s) in 12 monthly
        installments of $");
41     print (CalculatePayments($TotalCost, "12"));
42     print (" each.\n<P>");
43 } else {

```

```

44     print ("Please make sure that you have entered both a quantity
and an applicable discount and then resubmit.\n");
45 }
46 ?>
47 </BODY>
48 </HTML>

```

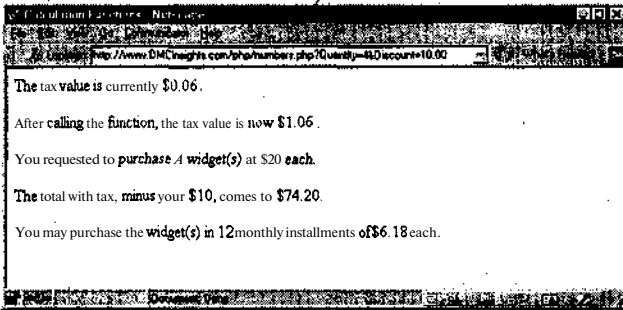


Рис. 9.5 ▼ Так как переменная `$Tax` внутри функции - это та же переменная, что и одноименная переменная `$Tax` вне функции (результат действия инструкции `global`), любое изменение ее значения внутри функции отражается и вне функции

Задание значений аргументов по умолчанию

При написании функций можно задавать аргументам значения по умолчанию. Функция будет использовать это значение, только если не получит аргумент, который заменит значение по умолчанию своим фактическим значением при вызове. Другими словами, задание аргументу значения по умолчанию делает этот аргумент **необязательным**, опциональным при вызове функции.

В качестве примера возьмем функцию `CalculatePayments()` (листинг 9.7). Как количество ежемесячных платежей можно задать значение **12** по умолчанию, создав функцию и используя следующий синтаксис:

```
function CalculatePayments ($Amount, $NumberPayments = "12") {
```

Вызов функции с помощью кода `CalculatePayments ($Amounts)`; не приведет ни к каким проблемам, а строка `CalculatePayments ($Amounts, "24")`; заменит значение переменной `$NumberPayments` со значения по умолчанию, равного 12, на фактическое, равное 24.

Значение аргумента по умолчанию используется, если логика программы подразумевает определенное значение, которое все же можно изменить. Однако необходимо помнить, что аргументы по умолчанию должны всегда быть написаны после других стандартных аргументов (то есть не имеющих значения по умолчанию). Это делается потому, что значения аргументам в PHP присваиваются напрямую в порядке их получения из строки вызова. Следовательно,

невозможно опустить значение для первого аргумента и задать значение **второму** (это показывает, что вы послали одно значение, которое будет автоматически присвоено первому аргументу, а не второму). Рассмотрим пример того, как не надо вызывать функцию. Допустим, функция определена следующим образом:

```
function CalculateTotal ($HowMany, $Price = "20.00", $TaxRate = "0.06") {
```

Мы вызовем ее такой строкой:

```
CalculateTotal (3, "0.07");
```

При этом мы собираемся задать аргументу `$HowMany` значение 3, желая оставить показатель `$Price` равным 20.00 и надеясь изменить величину `$TaxRate` на 0.07. Увы, у нас ничего не выйдет. Переменная `$HowMany` действительно получит значение 3, `$Price` - 0.07, а `$TaxRate` останется равным 0.06. Помнится, мы хотели получить несколько иной результат. Добиться нужного результата можно следующим образом:

```
CalculateTotal (3, "20.00", "0.07");
```

Теперь в свете того, что мы узнали о задании значений аргументов по умолчанию, изменим страницу `numbers.php` (листинг 9.7).

Написание функции, которая использует значения по умолчанию

1. Откройте файл `numbers.php` в текстовом редакторе (листинг 9.7).
2. Теперь добавьте значение по умолчанию переменной `$Savings` в функцию `CalculateTotal ()`.

```
function CalculateTotal ($HowMany, $Savings = "0") {
```

3. Мы указали 0 как значение по умолчанию переменной `$Savings`. Если никакой аргумент не будет передан в функцию, это будет **значить**, что скидка не применяется. Отредактируйте вторую функцию, задав **12** как **значение по умолчанию аргумента** `$NumberPayments`.

```
function CalculatePayments ($Amount, $NumberPayments = "12") {
```

Если в функцию `CalculatePayments ()` **отправлено** два аргумента, вместо значения по умолчанию аргумент `$NumberPayments` получит второе значение.

4. Сотрите две строки в основном теле **сценария**, которые распечатывают значение `$Tax`. Они больше не нужны.
5. Измените вызов функции `CalculatePayments ()` так, чтобы аргументы для количества платежей больше не посылались.

```
print (CalculatePayments($TotalCost));
```

Вызов функции `CalculatePayments` **О** передает теперь только значение `$TotalCost` как аргумент, так как `$NumberPayments` имеет значение **по умолчанию**.

Здесь можно было бы передать и значение `$NumberPayments`, если бы вы решили использовать другое количество ежемесячных платежей.

6. Сохраните сценарий (листинг 9.8), загрузите его на сервер и протестируйте в браузере (рис. 9.6). Не забудьте послать в сценарий значение количества (и значение скидки, если захотите), добавив его к адресу URL или через HTML-форму.

Листинг 9.8 Обновленная страница `numbers.php`. Задание значения по умолчанию в функции делает аргумент опциональным.

```
1  <?php
2  function CalculateTotal ($HowMany, $Savings = "0") {
3      global $Cost;
4      global $Tax;
5      $Tax++; // $Tax составляет 1.06.
6      $TotalCost = ($Cost * $HowMany);
7      if ( ($TotalCost < 50) AND ($Savings) ) {
8          print ("Your \$$Savings will not apply because the total
9              value of the sale is under $50!\n<P>");
10     }
11     if ($TotalCost >= 50) {
12         $TotalCost = $TotalCost - $Savings;
13     }
14     $TotalCost = $TotalCost * $Tax;
15     return $TotalCost;
16 } // Конец функции CalculateTotal.
17 function CalculatePayments ($Amount, $NumberPayments = "12") {
18     $Payments = round($Amount, 2) / $NumberPayments;
19     $Payments = sprintf ("%01.2f", $Payments);
20     return $Payments;
21 } // Конец функции CalculatePayments.
22 ?>
23 <HTML>
24 <HEAD>
25 <TITLE>Calculation Functions</TITLE>
26 </HEAD>
27 <BODY>
28 <?php
29 $Cost = 20.00;
30 $Tax = 0.06;
31 if ($Quantity) {
32     $Quantity = abs($Quantity);
33     $Discount = abs($Discount);
34     $TotalCost = CalculateTotal ($Quantity, $Discount);
35     // Печать результатов.
36     print ("You requested to purchase $Quantity widget(s) at \$$Cost
37         each.\n<P>");
38     print ("The total with tax, minus your \$$Discount, comes to $");
39     printf ("%01.2f", $TotalCost);
40     print (".\n<P>You may purchase the widget(s) in 12 monthly
41         installments of $");
42     print (CalculatePayments($TotalCost));
```

```

40     print (" each.\n<P>");
41 } else {
42     print ("Please make sure that you have entered both a quantity
        and an applicable discount and then resubmit.\n");
43 }
44 ?>
45 </BODY>
46 </HTML>

```

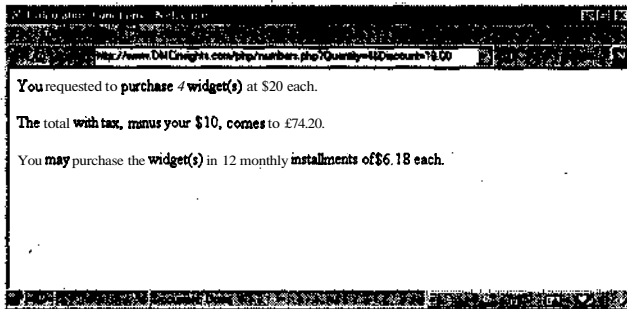


Рис. 9.6 т Даже если конечный пользователь, просматривая ваши страницы в браузере, никогда не сможет объяснить разницу между теми сценариями, в которых используются функции, и теми, где их нет (сравните с рис. 9.4), ваша программа стала гораздо проще и удобнее



Во всех примерах этой главы мы посылаем аргументы в функции по значению. Из этого следует, что в функцию посылается копия значения переменной, а не сама переменная. Можно также передавать аргументы по ссылке, что позволяет модифицировать переменную в функции, не делая ее глобальной, но это слишком сложная тема, которая не рассматривается в данной книге. Некоторая дополнительная информация содержится в приложении С.



В разных школах используются разные соглашения по именам аргументов функций. С одной стороны, применение в функции того же имени, что и имя переменной вне функции, позволяет легче запомнить, как соотносятся переменные. С другой стороны, использование тех же имен аргументов в функциях может сделать их излишне привязанными к остальному скрипту. Допустим и гибридный метод, при котором в начало имен функций просто добавляется строчная буква *f*, например *\$fHowMany* и *\$fTaxRate*. Некоторые программисты обозначают глобальные переменные как *\$gVariable*. Присваивая имена, важно оставаться последовательным и не отступать от принятой ранее логики.

Глава

Файлы и каталоги

Чтобы уровень ваших Web-приложений был действительно высок, вы обязательно должны уметь сохранять и восстанавливать накапливаемые в процессе работы данные.

В РНР есть два основных способа хранения данных: в файлах (или в каталогах) и в базах данных. В этой главе обсуждается первый способ, в следующей - второй. Время, которое потребуется на изучение обоих методов, не будет потрачено зря.

Данные лучше хранить в файлах: **во-первых**, тогда вам не нужно будет учиться работать с базами данных (это непросто), **во-вторых**, в большинстве своем **провайдеры** берут дополнительную плату за доступ к базам **данных**, в то время как доступ к файлам - бесплатный. Хотя система с базой данных, безусловно, более мощная, чем база **файлов**, **вы** не поверите, сколько всего можно сделать, просто сохраняя и извлекая информацию из обычных текстовых документов на сервере.

В этой главе говорится о правах доступа к файлам, о том, как вводить записи в файл и читать его, как создавать каталоги, осуществлять загрузку данных из HTML-формы и выполнять другие стандартные задачи с файлами и каталогами (переименование, удаление и т.д.).

Права доступа к файлам

Поговорим о правах доступа к файлам. Данная тема, так же как безопасность и регулярные выражения, слишком обширна. В книге дается только тот материал, который позволит вам осваивать программирование на языке РНР дальше. Если же вы хотите узнать о правах доступа к файлам больше, обратитесь к приложению С, где представлена некоторая дополнительная информация.

С помощью прав доступа к файлам и каталогам определяется, кто и что с ними может делать. Различают три варианта действий и с файлом, и с каталогом: написание, чтение, исполнение (файлы действительно могут создаваться

как исполняемые, а в каталоге право на исполнение означает, что можно посмотреть его содержимое). Более того, указанные варианты могут задаваться отдельно для **владельца** файла, то есть того, кто поместил его на **сервер**, для группы, которой принадлежит файл (ее назначает администратор сервера), и для всех остальных пользователей.

Обычно владельцу файла по умолчанию предоставляются права на запись и чтение, а остальным – только на чтение. Исполнение файла – один из основных вопросов безопасности, но, к счастью, оно не влияет на сценарии PHP (на языке Perl можно создать исполняемый файл, на PHP нельзя, так как интерпретатор этого языка работает в виде модуля в составе сервера). Разрешение на запись в файле тоже может быть вопросом безопасности, и такие действия стоит санкционировать только в крайнем случае.

- Изучая материал данной главы, мы будем работать с текстовым файлом `data.txt`, размещенным на сервере. В зависимости от конфигурации сервера (например, вы работаете на сервере провайдера или на своем собственном) стоит заранее установить соответствующие права доступа к этому файлу. Если файл не дает право делать то, что требует сценарий PHP, вы увидите сообщение об ошибке (рис. 10.1).

Создадим файл `data.txt`.

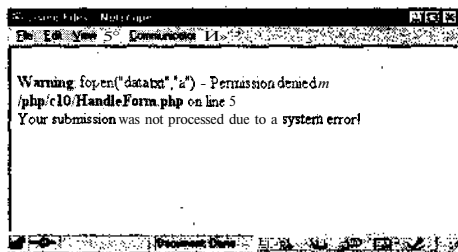


Рис. 10.1 ▼ Сообщение «В доступе отказано...» – результат попытки выполнить с файлом некоторую операцию, не разрешенную сервером. Здесь вы видите отказ функции `fopen()`, с помощью которой файл пытаются открыть для записи в него информации

Создание файла `data.txt`

Для выполнения действия сделайте следующее:

1. Откройте текстовый редактор и создайте новый документ.
2. Сохраните файл как `data.txt`, ничего в нем не печатая.
3. Загрузите файл на сервер.

Эти действия могут показаться странными, но для того, чтобы задавать права доступа к файлу, последний уже должен существовать. Обычно сначала задают права доступа к пустому файлу, и только в процессе работы Web-приложения в него пишутся данные.

В зависимости от ситуации можно использовать несколько способов задания прав доступа к файлу. В нашем примере всем пользователям будет предоставлено

право записи в файл data.txt и чтения из него (но не исполнения). Узнайте у своего провайдера, как задавать права доступа к файлам. Скорее всего, это будет один из следующих вариантов:

- некоторые провайдеры предоставляют пользователям возможность работать с панелями управления на основе Web. С помощью этих средств разрешается задавать не только права доступа к файлам и каталогам, но и многие другие параметры хоста;
- если у вас есть возможность выполнять команды в командном процессоре, зайдите с помощью клиента Telnet в каталог, где хранится файл data.txt, а затем посредством команды chmod (на сервере UNIX) измените права доступа к нему (рис. 10.2);
- разрешается изменять права доступа к файлу с помощью FTP-клиента (рис. 10.3);
- если вы работаете на своем сервере под Windows NT, разрешается изменять права доступа к файлу, щелкнув по нему правой кнопкой мыши, выбрав команду Свойства, а затем вкладку Безопасность.

```
* chmod 0666 data.txt
* ls -l
-rw-rw-rw- ullman 375 data.txt
```

Рис. 10.2 ▼ Посредством команды chmod 0666 data.txt запустив сессию Telnet на сервере и войдя в каталог, где хранится файл data.txt, можно изменить права доступа к этому файлу. Проверить права можно, напечатав строку ls -l. Команда показывает, что все три категории* пользователей имеют право на запись в файл и его чтение, имя владельца - ullman, размер файла - 375 байт

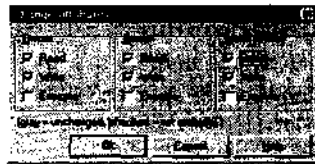


Рис. 10.3 т. Файловая оболочка Windows Commander позволяет задать права доступа с помощью этого временного рабочего окна



Если вы знакомы с интерпретатором командного языка и командой chmod, то наверняка понимаете, что означает число 0666 (рис. 10.2). А вот объяснение для тех, кто этого не знает. Ноль - это обязательный префикс, обозначающий восьмеричную систему счисления, а каждая шестерка соответствует правам записи (4) плюс чтения (2). Первая шестерка - права владельца, вторая - права группы, третья - права всех остальных пользователей. Сравните: число 0777 позволяет всем категориям пользователей записывать (4), считывать (2) и исполнять (1). Это относится только к операционным системам семейства UNIX (Linux, Solaris).

Запись данных в файл

Чтобы считывать информацию из файла, сначала ее надо записать туда. Запись в файл на сервере состоит из трех этапов: открытие файла, собственно запись данных, закрытие файла. К счастью, в PHP для выполнения каждого из этих действий есть встроенные функции:

```
$FileName = "data.txt";
$FilePointer = fopen ($FileName, "mode");
```

```
fwrite ($FilePointer, "data to be written");
fclose ($FilePointer);
```

Чтобы работа с файлами была удобной, сначала я всегда присваиваю отдельной переменной имя файла и путь к нему, то есть точные координаты файла на сервере. Указывая путь к файлу, пользуйтесь обычными правилами обозначения пути (например, если файл находится в каталоге files, являющемся подкаталогом текущего каталога, запись гласит: files/data.txt). Затем необходимо создать указатель файла, который присваивается соответствующим образом поименованной переменной \$FilePointer и используется в PHP для обращения к открытому файлу.

При открытии файла **большое** значение имеет используемый режим работы с ним (параметр mode). Выбор режима зависит от того, что вы собираетесь делать с этим файлом. Самый простой режим - **a+**. Он позволяет писать в файл и считывать из него. Этот режим создаст файл, если тот не существует, и автоматически добавит данные в конец файла. Более строгий режим **r** позволит только считывать информацию из файла. В таблице C.4 (приложение C) приведен полный список возможных режимов.

Функция `fwrite()` запишет новые данные, указанные как второй аргумент при вызове функции, в заданный **файл** в соответствии с выбранным режимом. И наконец, вы закрываете файл, снова обратившись к его указателю.

Создадим форму, которая сохраняет переданные пользователями адреса URL в отдельном файле.

Запись во внешний файл

1. Откройте страницу form.html (глава 8) в текстовом редакторе (листинг 10.1).

Листинг 10.1 ▼ Это существующая версия страницы form.html. Можно удалить строки для имени и фамилии, так как они здесь не понадобятся (см. листинг 10.2):

```
1 <HTML>
2 <HEAD>
3 <TITLE>HTML Form</TITLE>
4 </HEAD>
5 <BODY>
6 <FORM ACTION="HandleForm.php" METHOD=POST>
7 First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20><BR>
8 Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40xBR>
9 URL <INPUT TYPE=TEXT NAME="Array[URL]" SIZE=60xBR>
10 Description <TEXTAREA NAME="Array[Description]" ROWS=5 COLS=40>
    </TEXTAREA><BR>
11 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12 </FORM>
13 </BODY>
14 </HTML>
```

Я сделаю небольшие изменения в форме, а вы можете пропустить данный шаг, так как это не отразится на конечном результате.

2. Удалите строки для имени и фамилии (листинг 10.2).

Листинг 10.2 ▼ Я удалил ненужные строки для имени и фамилии из формы form.html, чтобы страница выглядела аккуратнее. Впрочем, этого можно было и не делать, в силу того что имена двух важных полей ввода – Array [URL] и Array [Description] остались неизменными.

```

1  <HTML>
2  <HEAD>
3  <TITLE>HTML Form</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleForm.php" METHOD=POST>
7  URL <INPUT TYPE=TEXT NAME="Array[URL]" SIZE=60xBR>
8  Description <TEXTAREA NAME="Array[Description]" ROWS=5 COLS=40>
   </TEXTAREA><BR>
9  <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
10 </FORM>
11 </BODY>
12 </HTML>

```

3. Сохраните страницу и загрузите ее на сервер.

Напишем новый сценарий HandleForm.php. Он будет обрабатывать данные, сгенерированные формой form.html.

4. Создайте новый PHP-документ в текстовом редакторе.

5. Начнем с определения функции PHP.

```

<?php
function WriteToFile ($URL, $Description) {

```

Функцией WriteToFile() используются два аргумента (адрес URL и его описание), которые пользователь ввел в форму form.html.

6. Называем файл и открываем его.

```

$TheFile = "data.txt";
$Open = fopen ($TheFile, "a");

```

С помощью этих двух строк мы указываем имя файла и открываем его только для записи (создаем файл, если его нет, или дописываем новые данные в конец в противном случае). Указанный здесь файл data.txt был создан ранее (см. предыдущий раздел), там же задавались права доступа к нему.

7. Создайте условную конструкцию на основе функции fopen().

```

if ($Open) {
    fwrite ($Open, "$URL\t$Description\n");
    fclose ($Open);
    $Worked = TRUE;
} else {
    $Worked = FALSE;
}

```

Если файл открыт успешно, новые данные будут добавлены к старым. Формат данных следующий: URL, затем знак табуляции (созданный с помощью обратного слеша и "буквы «t» - \t), описание и знак новой строки (созданный с помощью \n). После этого файл закрывается.

8. Возвратите значение \$Worked и закройте функцию.

```
return $Worked;
} // Конец функции WriteToFile.
```

Если файл открыт успешно, значение переменной \$Worked становится истинным и возвращается, чтобы показать успешное выполнение функции.

9. Закройте первый PHP-раздел и создайте стандартный HTML-заголовок.

```
?>
<HTML><HEAD><TITLE>Using Files</TITLE></HEAD><BODY>
```

10. Откройте основной PHP-раздел.

```
<?php
```

11. Создайте Шаблон регулярного выражения:

```
$Pattern = "(http://)?([^\s:]+)([^\s:]*\.\.\s?/?=&=)";
```

Регулярные выражения используются для проверки правильности адреса URL, который затем будет превращен в активную ссылку.

12. Создайте условную конструкцию.

```
if (ereg($Pattern, $Array["URL"])) {
    $Replace = "<a href=\"http://\2\3\"
    -target=\"_new\">\2\3</a>";
    $Array["URL"] = eregi_replace($Pattern,
    ->$Replace, $Array["URL"]);
    $CallFunction = WriteToFile ($Array["URL"],
    ->$Array["Description"]);
}
```

Если переданный пользователем адрес URL соответствует шаблону, то он будет изменен с помощью функции `ereg_replace()` и уже знакомой нам обратной ссылки. Затем вызывается функция `WriteToFile()`.

13. По результатам вызова функции будет напечатано сообщение.

```
if ($CallFunction) {
    print ("Your submission-$Array[URL]-has
    -been received!<BR>\n");
} else {
    print ("Your submission was not processed
    -due to a system error!<BR>\n");
}
```

Переменная \$CallFunction равна возвращенному из функции значению \$Worked. Если нужный файл откроется, то значение \$Worked и, следовательно, \$CallFunction, будет истинным, в противном случае - ложным. Пользователь получит соответствующее сообщение.

14. Закройте условную инструкцию, PHP и HTML.

```

    } else {
        print ("Please enter a valid Web address!\n");
    }
?>
</BODYx/HTML>

```

15. Сохраните сценарий как **HandleForm.php** (листинг 10.3), загрузите его на сервер (в один каталог с файлами **form.html** и **data.txt**) и протестируйте в браузере (рис. 10.4-10.6).

Листинг 10.3 В этом сценарии показано несколько известных вам приемов, которые стоит использовать в работе. Функция написана как первый элемент сценария. Передаваемые **пользователем** данные проверяются на правильность с помощью регулярных выражений. При возникновении каких-либо проблем в браузер отправляется соответствующее сообщение.

```

1  <?php
2  function WriteToFile ($URL, $Description) {
3  /* Функция WriteToFile принимает два аргумента URL и описание,
   которые будут записаны в файл. */
4      $TheFile = "data.txt";
5      $Open = fopen ($TheFile, "a");
6      if ($Open) {
7          fwrite ($Open, "$URL\t$Description\n");
8          fclose ($Open);
9          $Worked = TRUE;
10     } else {
11         $Worked = FALSE;
12     }
13     return $Worked;
14 } // Конец функции WriteToFile.
15 ?>
16 <HTML>
17 <HEAD>
18 <TITLE>Using Files</TITLE></HEAD>
19 <BODY>
20 <?php
21 /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
22 $Pattern = " (http://)?([^\s:]+)([^\s:]*\.\.?&=)";
23 if (ereg($Pattern, $Array["URL"]) ) {
24     $Replace = "<a href=\"http://\2\3\" target=\"_new\">\2\3</a>";
25     $Array["URL"] = eregi_replace($Pattern, $Replace, $Array["URL"] ) ;
26     $CallFunction = WriteToFile ($Array["URL"],
27     $Array["Description"] ) ;
28     if ($CallFunction) {
29         print ("Your submission-$Array[URL]-has been received!<BR>\n" ) ;
30     } else {
31         print ("Your submission was not processed due to a system
32         error!<BR>\n");

```

```

31     }
32 } else {
33     print ("Please enter a valid Web address!\n" );
34 }
35 ?>
36 </BODY>
37 </HTML>

```

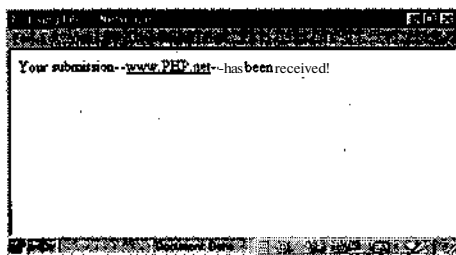


Рис. 10.4 т Такое сообщение появится, если пользователь ввел правильный Web-адрес и может записывать данные в файл

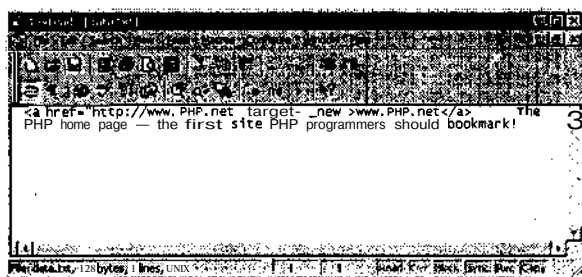


Рис. 10.5 т После хотя бы однократного просмотра страниц form.html и выполнения сценария HandleForm.php вы можете сохранить файл data.txt на вашем компьютере и просмотреть его в текстовом редакторе. Обратите внимание на то, что HTML-тэг A HREF, сгенерированный функцией eregi_replace(), тоже записывается в файл



В целях предосторожности можно использовать функцию `is_writable()`. Она определяет, позволит ли сервер записать данные в файл, перед тем как попытаться открыть его. Включить функцию `is_writable()` в сценарий можно следующим образом (это начало сценария):

```

$TheFile = "data.txt";
if (is_writable ($TheFile)) {
    $Open = fopen ($TheFile, "a");
}

```



Работая на серверах Windows, не забывайте экранировать обратные слешы в пути к файлу. Вместо них можно использовать обычный слеш (/). Например, нужно

написать `c:\\php\\data.txt` или `c:/php/data.txt`, но не `c:\php\data.txt`, так как обратный слеш - это знак экранирования следующего символа. Сказанное не относится к серверам UNIX, так как на них слеш всегда используется при указании полного пути к файлу.

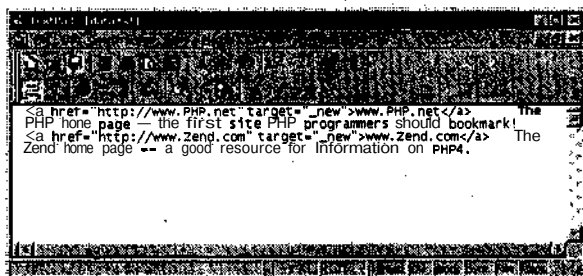


Рис. 10.6 Каждая дополнительная передача формы `form.html` добавит новую строку информации в файл `data.txt`

Чтение файла

Теперь, когда мы создали сценарий, который записывает данные в файл, самое время разработать сценарий, с помощью которого удастся считывать эту информацию. Считывать информацию из файла так же легко, как и записывать ее туда. Здесь тоже есть три этапа: открытие файла, собственно считывание информации, закрытие файла.

```
$FileName = "data.txt";  
$FilePointer = fopen ($FileName, "mode");  
$Array = file ($FileName);  
fclose ($FilePointer);
```

Встроенная функция `file()` - ценный инструмент PHP. С ее помощью информация из файла считывается и помещается в массив. Каждый элемент массива состоит из отдельной строки файла, а номер строки становится индексом в массиве. Если файл `data.txt` содержит две строки информации, каждая из которых заканчивается знаком новой строки (рис 10.6, не обращайте внимание на перескакивание длинной строки на следующую - знака новой строки там нет), то соответствующий массив будет состоять из двух элементов. Первый элемент отождествляется с первой строкой файла `data.txt`, второй - со второй. Как только данные переданы в массив, их можно легко обрабатывать или распечатывать.

Создадим сценарий, который выведет на одну страницу переданные пользователем и полученные ранее адреса URL.

Выполнение действия

1. Создайте новый документ в текстовом редакторе.
2. Начните со стандартного HTML-заголовка.

```
<HTML><HEAD>
<TITLE>Storing URLs in an External File</TITLE>
</HEAD><BODY>
```

Поскольку эта страница действует как форма и одновременно обрабатывает результаты последней, напомним сценарий немного по-другому. Страница будет начинаться не с функций, как делалось раньше.

3. Откройте PHP-раздел и скопируйте функцию WriteToFile() из сценария HandleForm.php (листинг 10.3).

```
<?php
function WriteToFile ($URL, $Description) {
    /* Функция WriteToFile принимает два аргумента URL и описание,
    →которые будут записаны в файл. */
    $TheFile = "data.txt";
    $Open = fopen ($TheFile, "a");
    if ($Open) {
        fwrite ($Open, "$URL\t$Description\n");
        fclose ($Open);
        $Worked = TRUE;
    } else {
        $Worked = FALSE;
    }
    return $Worked;
} // Конец функции WriteToFile.
```

Нет необходимости менять какие-либо строки. Налицо одно из преимуществ использования функций!

4. Создайте новую функцию, которая будет извлекать данные из файла.

```
function ReadFromFile () {
    /* Функция ReadFromFile отображает всю информацию,
    →сохраненную в файле. */
    $TheFile = "data.txt";
    $Open = fopen ($TheFile, "r");
    if ($Open) {
        print ("URLs currently listed in the data file:<P>\n");
        $Data = file ($TheFile);
        for ($n = 0; $n < count($Data); $n++) {
            $GetLine = explode("\t", $Data[$n]);
            print ("{$GetLine[0]}<BR>\n{$GetLine[1]}<P>\n");
        }
        fclose ($Open);
        print ("<HR><P>\n");
    } else {
        print ("Unable to read from data.txt!<BR>\n");
    }
} // Конец функции ReadFromFile.
```

Функция начинается с указания имени файла, затем осуществляется попытка его открытия в режиме **r**, чтобы данные могли считываться. Если файл успешно открыт, функция выполняется дальше.

Затем печатается заголовок, а данные файла превращаются в массив **\$Data**. Все элементы массива - строки (в формате: **URL**, символ табуляции, описание, символ новой строки). Затем каждая строка превращается в новый массив **\$GetLine** с помощью выполнения цикла, в котором из функции **\$Data** поочередно извлекаются все **элементы-строки**. В цикле функция **explode()** создает из каждой строки массив **\$GetLine**. При этом используется **пара** элементов, отделенных друг от друга знаком табуляции. Используя этот новый массив, можно отдельно распечатать части оригинальной строки.

И наконец, файл **закрывается**, код **HTML** отсылается в браузер. **Завершается** выполнение условной конструкции. При этом, если файл не был открыт, генерируется сообщение об ошибке.

5. Напишите третью функцию, которая при каждом вызове будет создавать **HTML-форму**.

```
function CreateForm () {
    /* Функция CreateForm отображает форму, */
    print ("Add a URL to the data file:\n");
    print ("<FORM ACTION=\"urls.php\" METHOD=POST>\n");
    print ("URL <INPUT TYPE=TEXT NAME=\"Array[URL]\"
    -SIZE=60><BR>\n");
    print ("Description <TEXTAREA NAME=\"Array[Description]\"
    -ROWS=5 COLS=40></TEXTAREA><BR>\n");
    print ("<INPUT TYPE=HIDDEN NAME=\"BeenSubmitted\"
    -VALUE=\"TRUE\">\n");
    print ("<INPUT TYPE=SUBMIT NAME=\"SUBMIT\"
    -VALUE=\"Submit!\"></FORM>\n");
} // Конец функции CreateForm.
```

Эта функция похожа на страницу **form.html**, но есть и два новых приема. Во-первых, тэг **ACTION** теперь посылает данные из формы в сценарий **urls.php**. Во-вторых, мы добавили скрытое поле, которое задает значение **\$BeenSubmitted** как истинное. Вы скоро поймете, зачем это было сделано.

6. Напишите четвертую функцию, которая будет обрабатывать форму,

```
function HandleForm () {
    global $Array;
```

Данная функция создана на основе сценария **HandleForm.php**. Ее первое действие - получение доступа к внешнему массиву **\$Array**, который содержит данные формы, с помощью инструкции **global**.

7. Основу функции составляет тело предыдущей страницы **HandleForm.php** (листинг 10.4).

```
$Pattern = "(http://)?([^\s:]+)([a-z:]+\.\s?&=)";
if (ereg($Pattern, $Array["URL"])) {
    $Replace = "<a href=\"http://\2\3\"
```

```

target="_new">\\2\\3</a>";
$Array["URL"] = eregi_replace($Pattern, $Replace,
$Array["URL"]);
$CallFunction = WriteToFile ($Array["URL"],
$Array["Description"]);
if ($CallFunction) {
    print ("Your submission-$Array[URL]-has
    -been received!<P><HR><P>\n");
} else {
    print ("Your submission was not processed
    -due to a system error!<BR>\n");
}
} else {
    print ("Please enter a valid Web address!\n");
}
} // Конец функции HandleForm.

```

Функцией проверяется правильность ввода с помощью регулярного выражения. Если был введен неправильный адрес URL, распечатывается сообщение об ошибке. При условии того что адрес верен, начинается выполнение строки \$CallFunction и соответствующих условных инструкций.

8. Создайте условную конструкцию, которая будет определять, обрабатывать ли форму.

```

if ($BeenSubmitted) {
    HandleForm();
}
ReadFromFile();
CreateFormO;

```

Если значение \$BeenSubmitted истинно (то есть форма уже была заполнена и сценарий вызван еще раз), то данные будут обработаны. Если значение \$BeenSubmitted ложно (это означает, что скрипт работает впервые, а пользователь пока ничего не ввел в форму), функция HandleForm() вызвана не будет. В любом случае имеющиеся данные выводятся из файла на экран, а для того чтобы пользователь смог ввести другой адрес URL, создается специальная форма.

9. Закройте PHP и HTML.

```
?></BODY></HTML>
```

10. Сохраните страницу как urls.php (листинг 10.4), загрузите ее на сервер и протестируйте в браузере (рис. 10.7-10.9).

Листинг 10.4 т Сценарий стал проще и логичней, когда мы создали четыре функции: для чтения файла, для записи в него, для создания и обработки формы (по одной на каждую операцию). Основное тело сценария в этом случае будет состоять из простой условной инструкции, вызывающей функции.

```

1 <HTML>
2 <HEAD>

```

```

3 <TITLE>Storing URLs in an External File</TITLE>
4 </HEAD>
5 <BODY>
6 <?php
7 function WriteToFile ($URL, $Description) {
8 /* Функция WriteToFile принимает два аргумента URL и описание,
   которые будут записаны в файл. */
9   $TheFile = "data.txt";
10  $Open = fopen ($TheFile, "a");
11  if ($Open) {
12    fwrite ($Open, "$URL\t$Description\n");
13    fclose ($Open);
14    $Worked = TRUE;
15  } else {
16    $Worked = FALSE;
17  }
18  return $Worked;
19 } // Конец функции WriteToFile.
20
21 function ReadFromFile () {
22 /* Функция ReadFromFile отображает всю информацию, сохраненную
   в файле. */
23   $TheFile = "data.txt";
24   $Open = fopen ($TheFile, "r");
25   if ($Open) {
26     print ("URLs currently listed in the data file:<P>\n");
27     $Data = file ($TheFile);
28     for ($n = 0; $n < count($Data); $n++) {
29       $GetLine = explode ("\t", $Data[$n]);
30       print ("{$GetLine[0]<BR>\n{$GetLine[1]<P>\n");
31     }
32     fclose ($Open);
33     print ("<HR><P>\n");
34   } else {
35     print ("Unable to read from data.txt!<BR>\n");
36   }
37 } // Конец функции ReadFromFile.
38
39 function CreateForm () {
40 /* Функция CreateForm отображает форму. */
41   print ("Add a URL to the data file:\n");
42   print ("<FORM ACTION=\"urls.php\" METHOD=POST>\n");
43   print ("URL <INPUT TYPE=TEXT NAME=\"Array[URL]\" SIZE=60><BR>\n");
44   print ("Description<TEXTAREA NAME=\"Array[Description]\" ROWS=5
   COLS=40></TEXTAREA><BR>\n");
45   print ("<INPUT TYPE=HIDDEN NAME=\"BeenSubmitted\"
   VALUE=\"TRUE\">\n");
46   print ("<INPUT TYPE=SUBMIT NAME=\"SUBMIT\" VALUE=\"Submit!\">
   </FORM>\n");
47 } // Конец функции CreateForm.
48
49 function HandleForm () {
50   global $Array;

```

```

51 $Pattern = "(http://)?([^\s:]+)([a-z:]\.|-_?/=&=)";
52 if (ereg($Pattern, $Array["URL"])) {
53     $Replace = "<a href=\"http://\2\3\" target=\"_new\">\2\3
    </a>";
54     $Array["URL"] = eregi_replace($Pattern, $Replace,
    $Array["URL"]);
55     $CallFunction = WriteToFile ($Array["URL"],
    $Array["Description"]);
56     if ($CallFunction) {
57         print ("Your submission-$Array[URL]-has been
    received!<P><HR><P>\n");
58     } else {
59         print ("Your submission was not processed due to a system
    error!<BR>\n");
60     }
61 } else {
62     print ("Please enter a valid Web address!\n");
63 }
64 } // Конец функции HandleForm.
65
66 /* Следующее условие устанавливает, обрабатывать ли форму
    в зависимости от того, имеет ли переменная $BeenSubmitted значение
    TRUE. */
67 if ($BeenSubmitted) {
68     HandleForm();
69 }
70 ReadFromFile();
71 CreateForm();
72 ?>
73 </BODY>
74 </HTML>

```

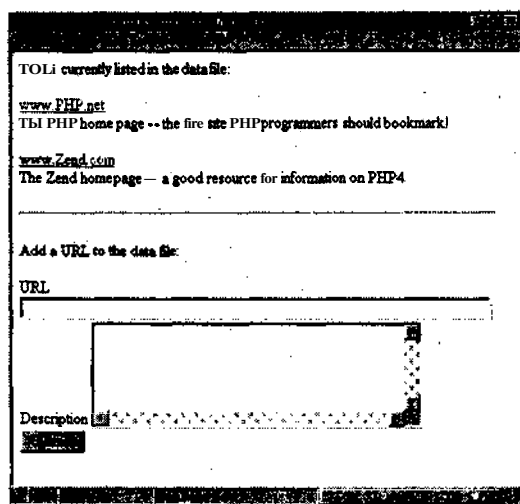


Рис. 10.7 т Когда пользователь попадает на страницу первый раз, значение `$BeenSubmitted` не является истинным, поэтому функция `HandleForm()` не вызывается (см. листинг 10.4). Как только форма отправлена, результаты будут обработаны, а ответ напечатан (рис. 10.8)

Your submission--www.DMCinsights.com/php/--has been received!

URLs currently listed in the data file:

www.php.net
The PHP home page -- the first site PHP programmers should bookmark!

www.zend.com
The Zend home page -- a good resource for information on PHP4.

www.DMCinsights.com/php/
A home page that accompanies this text

Add a URL to the data file:

URL

Description

Рис. 10.8 т После того как пользователь передал форму, сценарий `urls.php` сначала обработает данные, затем извлечет информацию из файла и создаст новую форму

```

<HTML>
<HEAD>
<TITLE>Storing URLs in an External File</TITLE>
<META>
<BODY>
Your submission--<a href="http://www.DMCinsights.com/php/" target="new">www.DMCinsights.com/php/</a>--has been received!<br>
URLs currently listed in the file<br>
<a href="http://www.php.net" target="new">www.php.net</a><br>
The PHP home page -- the first site PHP programmers should bookmark!<br>
<a href="http://www.zend.com" target="new">www.zend.com</a><br>
The Zend home page -- a good resource for information on PHP4.<br>
<a href="http://www.DMCinsights.com/php/" target="new">www.DMCinsights.com/php/</a><br>
A home page that accompanies this text
Add a URL to the data file:
<FORM ACTION="urls.php" METHOD="POST">
URL <INPUT TYPE="TEXT" NAME="url">
Description <TEXTAREA NAME="description">
<INPUT TYPE="HIDDEN" NAME="beenSubmitted" VALUE="TRUE">
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit">
</FORM>
</BODY>
</HTML>

```

Рис. 10.9 т Так выглядит выдаваемый сценарием `urls.php` HTML-код после того, как форма была передана. Обратите внимание на то, что все адреса URL становятся активными ссылками и что формой используется скрытое значение, не видимое пользователю. Последнее обстоятельство делает программу более функциональной

Каталоги

Чтение файла и запись в него на сервере – только часть процесса хранения данных. Наверняка вы захотите использовать и каталоги. *Каталог* можно представить в виде папки – это подраздел жесткого диска, где можно хранить файлы и другие каталоги. По умолчанию все посещаемые вами Web-страницы попадают в ваш так называемый «домашний» каталог, в котором можно создать другие «базы» для хранения изображений, картинок, данных и т.д.

В PHP каталог создается с помощью команды `mkdir("path", "permissions");`

Путь указывает имя и местонахождение каталога, второй параметр определяет права доступа к каталогу (формат тот же, что у команды `chmod`: 0, затем три восьмеричные цифры, например 0666).

Разработаем сценарий, который при регистрации нового пользователя будет создавать для него новый каталог.

Создание нового каталога

1. Сначала создадим простую страницу регистрации с именем пользователя и паролем. Откройте текстовый редактор и начните работу над новым HTML-документом.

```
<HTML><HEAD><TITLE>Registration Form</TITLE></HEAD><BODY>
```

2. Создайте форму со сценарием `HandleNewUser.php` в качестве атрибута ACTION и используя метод POST.

```
<FORM ACTION="HandleNewUser.php" METHOD=POST>
```

Так как в форму вводится пароль, для пересылки потребуется именно метод POST, а не GET, который менее безопасен.

3. Создайте одно поле для текста, а другое для пароля.

```
Username <INPUT TYPE=TEXT NAME="Array[Username]" SIZE=15xBR>
Password <INPUT TYPE=PASSWORD NAME="Array[Password]" SIZE=15><BR>
```

Не забудьте указать PASSWORD как тип ввода для пароля, иначе вводимый текст не будет скрыт (рис. 10.10).

4. Создайте кнопку **Submit**, затем закройте форму и HTML-документ.

```
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
</FORM></BODY></HTML>
```

5. Сохраните страницу как `NewUser.html` (листинг 10.5) и загрузите ее на сервер.

Листинг 10.5 ▼ Это самая простая регистрационная форма с двумя полями. В ваших приложениях она, скорее всего, будет более развернутой, хотя этапы обработки данных останутся такими же.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Registration Form</TITLE>
```



```

4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleNewUser.php" METHOD=POST>
7  Username <INPUT TYPE=TEXT NAME="Array[Username]" SIZE=15><BR>
8  Password <INPUT TYPE=PASSWORD NAME="Array[Password]" SIZE=15><BR>
9  <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
10 </FORM>
11 </BODY>
12 </HTML>

```

6. Создайте новый пустой документ в текстовом редакторе.
7. Сохраните документ как users.txt и загрузите его на сервер в один каталог со страницей NewUser.html.
8. Задайте такие права доступа, которые бы позволили всем пользователям записывать данные в файл users.txt и читать из него.

Третий шаг – разработать каталог, в котором автоматически будут созданы все остальные.

9. С помощью клиента FTP или Telnet (также допустимо использование панели управления на Web-основе, предоставленной провайдером) создайте новый каталог users в одной директории с объектами NewUser.html и users.txt.
10. Задайте такие права доступа, которые бы позволили всем пользователям делать записи в каталог users, читать его и производить в нем поиск (подсказка - права доступа 0777).

А теперь напишем сценарий HandleNewUser.php, который будет обрабатывать информацию из формы и создавать новый каталог.

11. Откройте текстовый редактор и создайте новый PHP-документ.
12. Начните с открытия PHP-раздела и написания новой функции.

```

<?php
function MakeDirectoryName ($Username) {
/* Функция MakeDirectoryName принимает в качестве аргумента имя
пользователя, на базе которого будет сгенерировано имя директории. */
srand ((double) microtime() * 1000000);
$Name = rand() . $Username;
return $Name;
} // End of MakeDirectoryName Function.

```

Каталог будет дано имя с помощью переменной \$Username и случайного числа. Имя сгенерируется инициализацией функции srand() текущим временем, и затем на основе этого будет выбрано случайное число. Присоединение его к переменной \$Username обеспечит уникальность каждого имени, которое затем возвращается функцией.

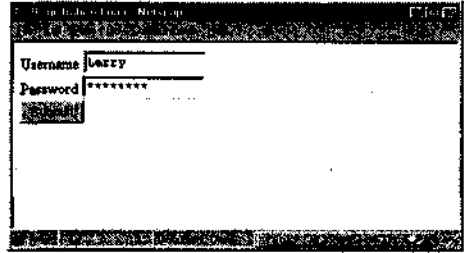


Рис. 10.10 ▼ Перед вами простая регистрационная форма. Тип ввода пароля скрывает вводимую пользователем информацию

13. Скопируйте функцию WriteToFile() из листинга 10.4 и слегка измените ее:

```
function WriteToFile ($Username, $Password) {
    $TheFile = "users.txt";
    $Open = fopen ($TheFile, "a");
    if ($Open) {
        $Password = md5 ($Password);
```

Измените имя текстового файла (теперь это users.txt) и добавьте строку для шифрования пароля перед его сохранением. В целях безопасности всегда лучше хранить зашифрованный пароль.

14. Определите новое имя каталога.

```
$Directory = "users/" . MakeDirectoryName ($Username);
```

Это имя создается с помощью вызова функции MakeDirectoryName() и соглашения о том, что новый каталог будет подкаталогом каталога users.

15. Добавьте новые данные в файл.

```
fwrite ($Open, "$Username\t$Password\t$Directory\n");
fclose ($Open);
```

16. Попробуйте создать каталог и задайте условие.

```
if (!(mkdir ($Directory, "0777"))) {
    $Directory = FALSE;
}
} else {
    $Directory = FALSE;
}
return $Directory;
} // Конец функции WriteToFile.
```

Переменной \$Directory присваивается значение имени нового каталога, только если файл успешно открыт. Если не удастся открыть файл или каталог не может быть создан, значение переменной \$Directory будет ложным. Это указание на то, что функция не выполнила свою работу. Результат, выдаваемый функцией, - FALSE.

17. Закройте PHP-раздел и HTML-заголовок.

```
?><HTML><HEAD><TITLE>Using Directories</TITLE></HEAD><BODY>
```

18. Откройте второй PHP-раздел и создайте главную условную конструкцию.

```
if (($Array[Username]) && ($Array[Password])) {
    $Check = WriteToFile ($Array[Username], $Array[Password]);
    if ($Check) {
        print ("Your request was successfully processed!<BR>\n");
    } else {
        print ("Your request was not processed due to
        -a system error!<BR>\n");
    }
} else {
    print ("Please enter a Username and Password!</n");
}
```

Эта часть кода вам уже должна быть знакома. Конструкцией проверяется, заполнены ли оба поля формы. Если да, то данные записываются в файл и создается новый каталог. На основе возвращаемого функцией значения отправляется соответствующее сообщение.

19. Закройте этот PHP-раздел и саму HTML-страницу.

```
?></BODY></HTML>
```

20. Сохраните страницу как `HandleNewUser.php` (листинг 10.6), загрузите ее на сервер (туда же, где находятся каталог `users`, файл `users.txt` и страница `NewUser.html`) и протестируйте результаты в браузере (рис. 10.11 и 10.12).

Листинг 10.6 ▼ Хотя здесь вы видите только две функции, можно написать и третью, предназначенную исключительно для создания каталога. Для этого нужно выделить условную конструкцию `mkdir()` из функции `WriteToFile()`.

```

1  <?php
2  function MakeDirectoryName ($Username) {
3  /* функция MakeDirectoryName принимает в качестве аргумента имя
   * пользователя, на базе которого будет сгенерировано имя директории. */
4      srand ((double) microtime() * 1000000);
5      $Name = rand() . $Username;
6      return $Name;
7  } // Конец функции MakeDirectoryName.
8
9  function WriteToFile ($Username, $Password) {
10 /* функция WriteToFile принимает два аргумента и описание, которые
   * будут записаны в файл. */
11     $TheFile = "users.txt";
12     $Open = fopen ($TheFile, "a");
13     if ($Open) {
14         $Password = md5 ($Password);
15         $Directory = "users/" . MakeDirectoryName ($Username);
16         fwrite ($Open, "$Username\t$Password\t$Directory\n");
17         fclose ($Open);
18         if (!(mkdir ($Directory, "0777"))) {
19             $Directory * FALSE;
20         }
21     } else {
22         $Directory = FALSE;
23     }
24     return $Directory;
25 } // Конец функции WriteToFile.
26 ?>
27 <HTML>
28 <HEAD>
29 <TITLE>Using Directories</TITLE></HEAD>
30 <BODY>
31 <?php
32 /* Эта страница получает и обрабатывает данные, принятые
   * от "NewUser.html". */
33

```

```

34 if (($Array[Username]) && ($Array[Password])) {
35     $Check = WriteToFile ($Array[Username], $Array[Password]);
36     if ($Check) {
37         print ("Your request was successfully processed!<BR>\n");
38     } else {
39         print ("Your request was not processed due to a system
        error!<BR>\n");
40     }
41 } else {
42     print ("Please enter a Username and Password!\n");
43 }
44 ?>
45 </BODY>
46 </HTML>

```

Проверить работу страницы также можно, **заглянув** в каталог users на сервере с помощью FTP или Telnet и **посмотрев**, есть ли новые подкаталоги.

Наверняка когда-нибудь вы захотите разработать систему, гарантирующую уникальность имени пользователя. Сделать это достаточно просто: перед созданием каталога ваш сценарий должен **проверить**, не соответствует ли только что введенное имя пользователя одному из тех, что есть в списке. Если соответствий не найдено, имя пользователя принимается. В противном случае нужно выдать сообщение с просьбой ввести другое имя пользователя.

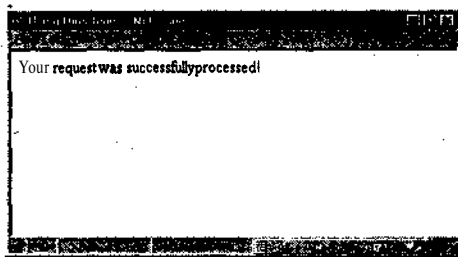


Рис. 10.11 т Если все работает, как задумано, пользователь получит только такое сообщение

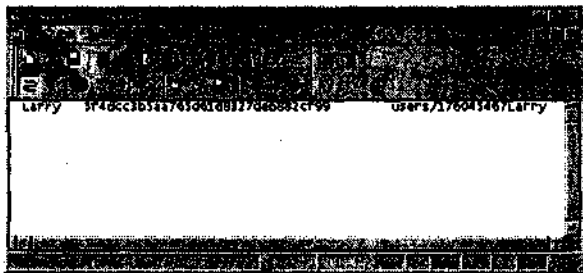


Рис. 10.12 т Файл users.txt содержит три разделенных знаком табуляции поля с информацией: имя пользователя, зашифрованный пароль и имя каталога пользователя

Загрузка файла на удаленный компьютер

Надеюсь, мы наглядно показали, как легко обрабатывать HTML-формы, пользуясь языком PHP. Независимо от типа передаваемых данных в PHP есть все необходимые средства для их проверки, правки и долговременного хранения. Выполнить загрузку файла через HTML-форму также легко.

Чтобы дать пользователю возможность загрузить файл, необходимо провести два изменения в стандартной HTML-форме. Во-первых, начальная строка формы должна содержать код `ENCTYPE="multipart/form-data"`, который указывает серверу, что ожидается получение файла или других данных. Во-вторых, элемент `<INPUT TYPE=FILE NAME=NAME>` используется для создания в форме поля, в котором вводится точный адрес файла.

Элемент `INPUT TYPE=FILE` позволяет пользователю указать тот файл на своем компьютере, который при отправке будет загружен на сервер. Как только это произошло, можно обрабатывать файл с помощью PHP.

При загрузке файла сервер помещает его во временный каталог. Ваша задача после прибытия файла - сохранить его в постоянном каталоге. Функция `copy()` используется для копирования файла в новое место:

```
copy ("SourceName", "DestinationName");
```

Затем с помощью функции `unlink()` необходимо удалить временный файл.

Напишем очень простой сценарий, который загружает файл и сохраняет его в каталоге `users`. Так же, как и сценарий `urls.php`, он создает HTML-форму и обрабатывает ее.

Использование PHP для загрузки файлов на сервер

1. Создайте новый PHP-документ в текстовом редакторе.
2. Начните со стандартного HTML-заголовка.

```
<HTML><HEAD><TITLE>Handling File Uploads</TITLE></HEAD><BODY>
```

3. Откройте PHP-раздел и создайте условную конструкцию, которая будет проверять, нужно ли загружать файл.

```
<?php  
if ($File) {
```

Информация о загружаемом файле будет храниться в переменной `$File`. Если эта переменная существует (то есть имеет значение), ее надо обрабатывать.

4. Распечатайте имя файла и его размер.

```
print ("File name: $File_name<P>\n");  
print ("File size: $File_size<P>\n");
```

Когда файл загружен, создается несколько новых связанных с ним переменных. Взяв имя основной переменной (в данном случае `$File`) и добавив суффикс `_name` или `_size`, вы получаете от сервера соответствующую информацию.

5. Попробуйте скопировать **файл** в каталог **users** и распечатать сообщение о результатах этого действия.

```
if (copy ($File, "users/$File_name")) {
    print ("Your file was successfully uploaded!<P>\n");
} else {
    print ("Your file could not be copied.<P>\n");
}
```

Команда `copy` принимает два аргумента - имя файла, откуда и куда **необходимо** копировать. У нас при копировании оригинальный файл хранится в переменной `$File`, а конечный пункт либо абсолютен (например, `c:/php/data.txt`), либо относителен к текущему каталогу (например, `"php/data.txt"`). Здесь мы использовали относительную ссылку, запросив поместить файл в каталог `users`, который находится в одном каталоге с этим скриптом.

6. Удалите файл и закройте условную конструкцию.

```
    unlink ($File);
}
```

Вполне вероятно, что позже файл будет автоматически удален сервером. Но лучше во всем быть последовательным и сразу после копирования стереть файл. Для удаления файла используется функция `unlink()`.

7. Распечатайте **HTML-форму** для загрузки файла на сервер.

```
print ("Upload a file to the server:\n");
print ("<FORM ACTION=\"FileUpload.php\" METHOD=POST
-ENCTYPE=\"multipart/form-data\">\n");
print ("File <INPUT TYPE=FILE NAME=\"File\" SIZE=20><BR>\n");
print ("<INPUT TYPE=SUBMIT NAME=\"SUBMIT\"
-VALUE=\"Submit!\"></FORM>\n");
```

Не забудьте добавить код `ENCTYPE` к открывающему тэгу формы. Строка `INPUT TYPE=FILE` не требует объяснений.

8. Закройте PHP и HTML.

```
?></BODY></HTML>
```

9. Сохраните файл как `FileUpload.php` (листинг 10.7), загрузите его на сервер (вместе с каталогом `users`) и протестируйте в браузере (рис. 10.13 и 10.4).

Листинг 10.7 т Этот простой сценарий показывает, как легко в PHP обрабатываются **HTML-формы**. Загрузка файла на сервер состоит из трех этапов: изменение соответствующим образом HTML-формы, перемещение файла в нужное место с помощью функции `copy()` и удаление файла посредством функции `unlink()`.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Handling File Uploads</TITLE>
4 </HEAD>
```

```

5 <BODY>
6 <?php
7 /* Следующее условие устанавливает, обрабатывать ли форму
   в зависимости от того, присутствует ли $File. */
8 if ($File) {
9     print ("File name: $File_name<P>\n");
10    print ("File size: $File_size<P>\n");
11    if (copy ($File, "users/$File_name")) {
12        print ("Your file was successfully uploaded!<P>\n");
13    } else {
14        print ("Your file could not be copied.<P>\n");
15    }
16    unlink ($File);
17 }
18
19 print ("Upload a file to the server:\n");
20 print ("<FORMACTION=\"FileUpload.php\" METHOD=POST
   ENCTYPE=\"multipart/form-data\">\n");
21 print ("File <INPUT TYPE=FILE NAME=\"File\" SIZE=20><BR>\n");
22 print ("<INPUT TYPE=SUBMIT NAME=\"SUBMIT\" VALUE=\"Submit!\">
   </FORM>\n");
23 ?>
24 </BODY>
25 </HTML>

```

Мы решили использовать созданный ранее каталог **users**, поскольку известно, что все пользователи могут записывать в него некую информацию. Если вы попытаетесь скопировать файл в каталог, запись данных в который неразрешена, результат этого действия будет выглядеть так же, как на рис. 10.15.

Максимальный размер загружаемого файла зависит от нескольких факторов. Во-первых, от **ограничений**, предусмотренных на **сервере**. Во-вторых, ограничения могут накладываться и в самом языке PHP (в конфигурационном файле `php.ini`). В-третьих, вы можете задать максимальный размер файла, написав следующее:

```
<INPUT TYPE=HIDDEN NAME="MAX_FILE_SIZE" VALUE="2048">
```

В форме перед вводом слова **FILE** значение выражено в байтах.

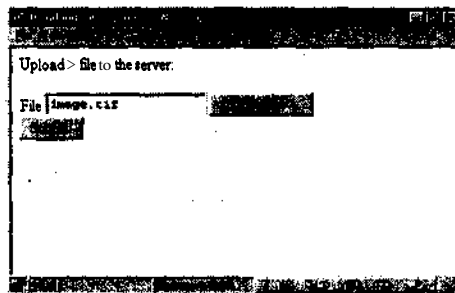


Рис. 10.13 Тип ввода **FILE** создает кнопку, с помощью которой можно найти нужный файл на своем компьютере. При выборе файла его имя автоматически появится в поле

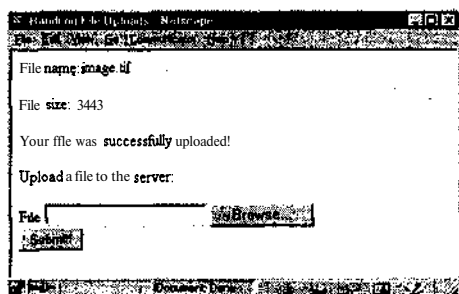


Рис. 10.14 ▼ Успешная обработка файла выдает на экран пользователя соответствующее сообщение. Вы можете одновременно узнать имя файла и получить информацию о его размере в байтах

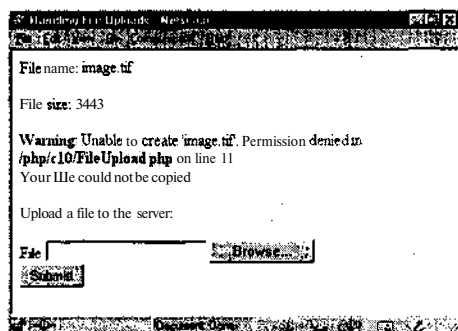


Рис. 10.15 т При попытке скопировать файл в каталог, который не **позволяет** этого делать, генерируется сообщение об ошибке. Однако понятно, что файл был успешно загружен на сервер, так как страница показывает имя файла и его размер. Просто не удастся скопировать файл в предложенный каталог



Тип файла, который присваивается переменной `$File_type` при загрузке переменной `$File`, - это то же самое, что и его тип MIME (многоцелевые расширения электронной почты в сети Internet). Последний используется в приложениях электронной почты и в браузерах для указания того, какой программой обрабатывать файл. К типам MIME относятся `image/jpeg` или `text/html`.

Переименование и удаление файлов и каталогов

В PHP есть еще несколько полезных встроенных функций для работы с файлами и каталогами. К ним относятся переименование и удаление файлов, а также получение списка файлов в каталоге. Обсудим синтаксис данных функций, а затем посмотрим, как они работают в контексте типичного сценария PHP.

Функция `rename()` имеет следующий синтаксис:

```
rename ("старое имя", "новое имя");
```

Она применима как к файлам, так и к каталогам.

Еще одна функция, о которой говорится в этом разделе, - `filesize()`. Она определяет размер файла в байтах. Это значение может быть присвоено переменной или распечатано.

```
$Number = filesize ("filename");
```

Один из удобных приемов работы - возможность получить список файлов в каталоге. Такой сценарий позволяет просматривать каталоги без помощи FTP и может быть использован для создания интерактивного хранилища документов. Чтение каталога похоже на чтение файла. Сначала открываем каталог,

затем просматриваем файлы по очереди, после чего закрываем каталог. Вместо указателя файла, который использовался при чтении файла, здесь применен указатель каталога, но идея та же.

```
$Handle = opendir ("path");
readdir ($Handle);
closedir ($Handle);
```

Так как функция `readdir()` последовательно показывает файлы по одному, ее надо поместить в цикл:

```
while (readdir ($Handle)) {
    statements;
}
```

Мы используем эти приемы в одном сценарии, который создается панелью управления файлами на основе обычного браузера для просмотра и работы с каталогами.

Создание панели управления каталогами

1. Откройте текстовый редактор и начните новый PHP-документ.
2. Создайте стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>Viewing Files in a Directory</TITLE>
-</HEAD><BODY>
```

3. Начните с таблицы, затем откройте PHP-раздел.

```
<TABLE BORDER=0 WIDTH="60%" CELSPACING=2
-CELLPADDING=2 ALIGN=CENTER>
<?php
```

Чтобы страница выглядела аккуратно, разместим данные в таблице.

4. Напишем несколько условных конструкций, проверяющих, нужно ли выполнять определенные действия исходя из заданных пользователем указаний. Нажав на кнопки в форме, в массивы `$Delete` (удаление файлов) и `$Rename` (переименование файлов) вы запишете списки файлов для этих операций. Если задана переменная `$Upload` (загрузка файлов), то она также будет содержать имя файла.

```
if ($Upload) { // Обработка загрузки файла.
    print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Uploaded
-file name: $File_name</TD></TR>\n");
    print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Uploaded
-file size: $File_size</TD></TR>\n");
    if (copy ($File, "users/$File_name")) {
        print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
-$File_name, was successfully uploaded!</TD></TR>\n");
    } else {
        print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
-$File_name, could not be copied.</TD></TR>\n");
    }
}
```

```

    }
    unlink ($File);
    print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;&nbsp;&nbsp;</TD></TR>\n");
}

```

Переменная \$Upload будет задана, если пользователь хочет загрузить файл на сервер. Следовательно, если переменная \$Upload существует, загружаемый файл будет обработан так, как мы видели выше.

Последняя инструкция print, которая создает пустой ряд в таблице, используется исключительно в эстетических целях, как и в последующих двух условиях. Пустая строка сделает Web-страницу менее загроможденной.

```

5. if ($Delete) { // Handle file deletions.
    for ($I = 0; $I < count ($Delete); $I++) {
        if ( unlink ("users/$Delete[$I]") ) {
            print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                -$Delete[$I], was successfully deleted!</TD></TR>\n");
        } else {
            print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                -$Delete[$I], could not be deleted.</TD></TR>\n");
        }
    }
    print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;&nbsp;&nbsp;</TD></TR>\n");
}

```

Переменная \$Delete используется для определения того, нужно ли удалять какие-либо файлы. Так как желательно удалять несколько файлов сразу, то был создан массив. Цикл обработает каждый элемент массива, удаляя файлы по очереди.

```

6. if ($Rename) { // Handle file renaming.
    for ($n = 0; $n < count ($Rename); $n++) {
        $OldFilename = $Rename[$n];
        $Old = "users/$OldFilename";
        $New = "users/$NewName[$OldFilename]";
        if ( rename ($Old, $New) ) {
            print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                -$Rename[$n], was successfully renamed!</TD></TR>\n");
        } else {
            print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                -$Rename[$n], could not be renamed.</TD></TR>\n");
        }
    }
    print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;&nbsp;&nbsp;</TD></TR>\n");
}

```

Механизм работы с элементами массива для переименования файлов такой же, как и при удалении файлов. Как только вы присвоили новое и старое имена файлов, вызывается функция rename (), которая и выполняет указанные изменения.

7. Создайте HTML-форму, не забыв включить код ENCTYPE для загрузки файла на сервер.

```
print("<FORM ACTION=\"files.php\" METHOD=POST
-ENCTYPE=\"multipart/form-data\">\n");
```

8. Распечатайте заголовки таблицы.

```
print("<TR><TD><B>File Name</B></TD><TD><B>File
-Size</B></TD><TD><B>Delete</B></TD><TD><B>Rename</B>
-Enter the New Name in the Box)</TD></TR>\n");
```

9. Напишите код, позволяющий считывать информацию из каталога.

```
$Open = opendir("users");
while ($Files = readdir($Open)) {
    $Filename = "users/" . $Files;
    if (is_file($Filename)) {
        $Size = filesize("users/$Files");
        print("<TR><TD>$Files</TD><TD>$Size</TD><TD><INPUT
-TYPE=CHECKBOX NAME=\"Delete[]\" VALUE=\"\"$Files\">
-</TD><TD><INPUT TYPE=CHECKBOX NAME=\"Rename[]\"
-VALUE=\"\"$Files\"><INPUT TYPE=TEXT NAME=
-\"NewName{$Files}\"></TD></TR>\n");
    }
}
closedir($Open);
```

Здесь задан цикл для обращения к каждому файлу (и каталогу), расположенному в каталоге users. В этом случае мы хотим работать только с файлами, пропуская каталоги. Вот почему использовалась функция проверки типа `is_file()`, которая обеспечивает создание списка файлов.

10. Создайте в форме опцию загрузки на сервер.

```
print("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;</TD></TR>\n");
print("<TR><TD COLSPAN=4 ALIGN=CENTER><INPUT TYPE=CHECKBOX
-NAME=\"Upload\" VALUE=\"Yes\">Upload a file to theserver:<INPUT
-TYPE=FILE NAME=\"File\" SIZE=20></TD></TR>\n");
print("<TR><TD COLSPAN=4 ALIGN=CENTER><INPUT TYPE=SUBMIT
-NAME=\"SUBMIT\" VALUE=\"Submit!\"></FORM></TD></TR>\n");
```

11. Закройте PHP и HTML.

```
?></TABLE></BODY></HTML>
```

12. Сохраните сценарий как files.php (листинг 10.8), загрузите его на сервер (в одно место с каталогом users) и протестируйте в браузере (рис. 10.16 и 10.17).

Листинг 10.8 т Вместо функций на этой странице используются условные конструкции, а результаты похожи. Вы можете проверить свои знания, переписав этот сценарий с использованием функций, особенно раздел, который выводит на экран список файлов каталога.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Viewing Files in a Directory</TITLE>
```

```

4  </HEAD>
5  <BODY>
6  <TABLE BORDER=0 WIDTH="60%" CELSPACING=2 CELLPADDING=2 ALIGN=CENTER>
7  <?php
8  /* Сценарий отображает информацию о файлах в директории и позволяет
   пользователь удалять, загружать и переименовывать файлы. */
9
10 if ($Upload) { // Обработка загрузки файла.
11     print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Uploaded file name:
        $File_name</TD></TR>\n");
12     print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Uploaded file size:
        $File_size</TD></TR>\n");
13     if (copy ($File, "users/$File_name")) {
14         print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file, $File_name,
            was successfully uploaded!</TD></TR>\n");
15     } else {
16         print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file, $File_name,
            could not be copied.</TD></TR>\n");
17     }
18     unlink ($File);
19     print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;</TD></TR>\n");
20 }
21
22 if ($Delete) { // Handle file deletions.
23     for ($i = 0; $i < count ($Delete); $i++) {
24         if ( unlink ("users/$Delete[$i]") ) {
25             print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                $Delete[$i], was successfully deleted!</TD></TR>\n");
26         } else {
27             print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                $Delete[$i], could not be deleted.</TD></TR>\n");
28         }
29     }
30     print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;</TD></TR>\n");
31 }
32
33 if ($Rename) { // Handle file renaming.
34     for ($n = 0; $n < count ($Rename); $n++) {
35         $OldFilename = $Rename[$n];
36         $Old = "users/$OldFilename";
37         $New = "users/$NewName[$OldFilename]";
38         if ( rename ($Old, $New) ) {
39             print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                $Rename[$n], was successfully renamed!</TD></TR>\n");
40         } else {
41             print ("<TR><TD COLSPAN=4 ALIGN=CENTER>Your file,
                $Rename[$n], could not be renamed.</TD></TR>\n");
42         }
43     }
44     print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;</TD></TR>\n");
45 }
46
47 // Начало формы.
48 print ("<FORM ACTION=\"files.php\" METHOD=POST ENCTYPE=\"multipart/
   form-data\">\n");

```

```

49 print ("<TR><TD><B>File Name</B></TD><TD><B>File Size</B>
    </TD><TD><B>Delete</B></TD><TD><B>Rename</B> Enter the New Name in
    the Box)</TD></TR></TR>\n"); •
50
51 // Чтение файлов из каталога.
52 $Open = opendir ("users");
53 while ($Files = readdir ($Open)) {
54     $Filename = "users/" . $Files;
55     if (is_file ($Filename)) {
56         $Size = filesize ("users/$Files");
57         print ("<TR><TD>$Files</TD><TD>$Size</TD><TD><INPUT
            TYPE=CHECKBOX NAME=\"Delete[]\" VALUE=\"\"$Files\">
            </TD><TD><INPUT TYPE=CHECKBOX NAME=\"Rename []\" 
            VALUE=\"\"$Files\"><INPUT TYPE=TEXT NAME= \"NewName[$Files]\">
            </TD></TR>\n");
58     }
59 }
60 closedir ($Open);
61
62 // Вывод поля для загрузки файлов.
63 print ("<TR><TD COLSPAN=4 ALIGN=CENTER>&nbsp;&nbsp;&nbsp;&nbsp;</TD></TR>\n");
64 print ("<TR><TD COLSPAN=4 ALIGN=CENTER><INPUT TYPE=CHECKBOX
    NAME=\"Upload\" VALUE=\"Yes\">Upload a file to theserver:<INPUT
    TYPE=FILE NAME=\"File\" SIZE=20></TD></TR>\n");
65 print ("<TR><TD COLSPAN=4 ALIGN=CENTER><INPUT TYPE=SUBMIT
    NAME=\"SUBMIT\" VALUE=\"Submit!\"></FORM></TD></TR>\n");
66 ?>
67 </TABLE>
68 </BODY>
69 </HTML>

```

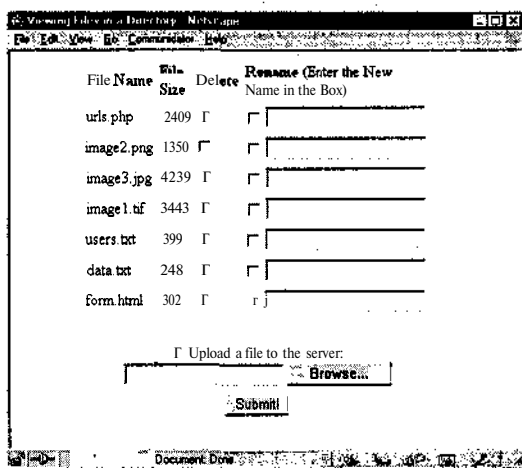


Рис. 10.16 Пользователь видит такую страницу, если посещает ресурс впервые. Триггерные кнопки позволяют пользователю удалять и переименовывать файлы, а также загружать их на сервер

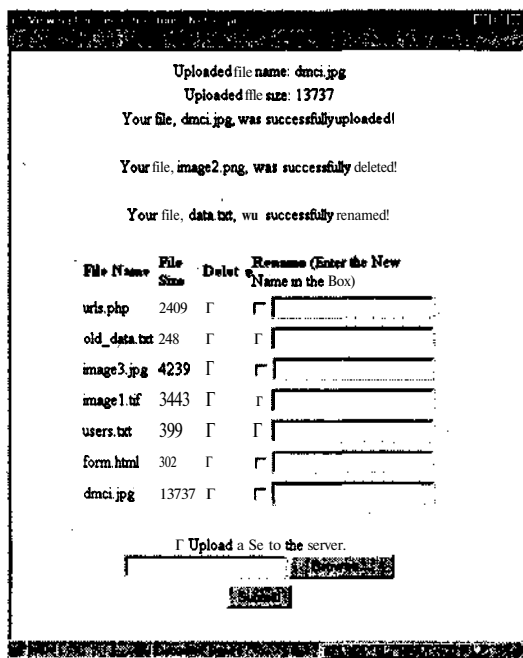


Рис. 10.17 т Здесь один файл загружен, другой удален, а третий переименован. После каждой успешной операции выдается соответствующий отчет, а на экран выводится обновленный список содержимого каталога

При желании с помощью страниц **NewUser.html** и **HandleNewUser.php** можно копировать файл **files.php** (с некоторыми изменениями) в каждый новый каталог пользователя. В этом случае при регистрации пользователь работает со своей панелью управления, с помощью которой разрешается управлять хранящимися файлами.

Узнать о других функциях для работы с файлами и каталогами можно в руководстве по PHP, в разделе «Каталоги и файловая система».

Глава

Базы данных

Как ни странно это может звучать, Internet не был бы тем, чем он является сейчас, если бы не существовало баз данных. Язык PHP также не стал бы настолько популярным и полезным, если бы не встроенная поддержка различных типов баз данных.

База данных (БД) представляет собой набор таблиц из столбцов и строк, в которых хранится информация. На сайтах электронной коммерции базы данных используются для хранения спецификаций продуктов и информации о клиентах, информационные сайты содержат в БД статьи и новости.

В настоящее время имеется множество серверов баз данных или систем управления базами данных (СУБД), которые работают на различных платформах. (С технической точки зрения, СУБД - это программное обеспечение, которое обеспечивает интерфейс с собственно базой данных. Однако термины «база данных» и «СУБД» все больше используются как синонимы. Во избежание путаницы мы будем их различать.) Лучшей СУБД для любой операционной системы считается Oracle. Впрочем, стоимость системы Oracle настолько высока, что позволяет использовать ее только в больших и хорошо финансируемых проектах. В среде Windows и Windows NT обычно используются SQL-сервер или СУБД Access. Возможно, это хорошие программы, но они не переносятся на другие платформы.

В этой главе в качестве СУБД используется MySQL (рис. 11.1).

Система MySQL адаптирована для большинства платформ. Возможно, она не такая мощная, как другие SQL-сервера, однако обладает замечательной скоростью и достаточной функциональностью для выполнения большинства задач. Для серверов UNIX система MySQL, как правило, бесплатна, что делает ее самой распространенной СУБД для создания Web-приложений. Если вы работаете на сервере провайдера, узнайте, какую СУБД вам могут предложить (часто за отдельную плату). Если вы работаете на своем сервере, подумайте

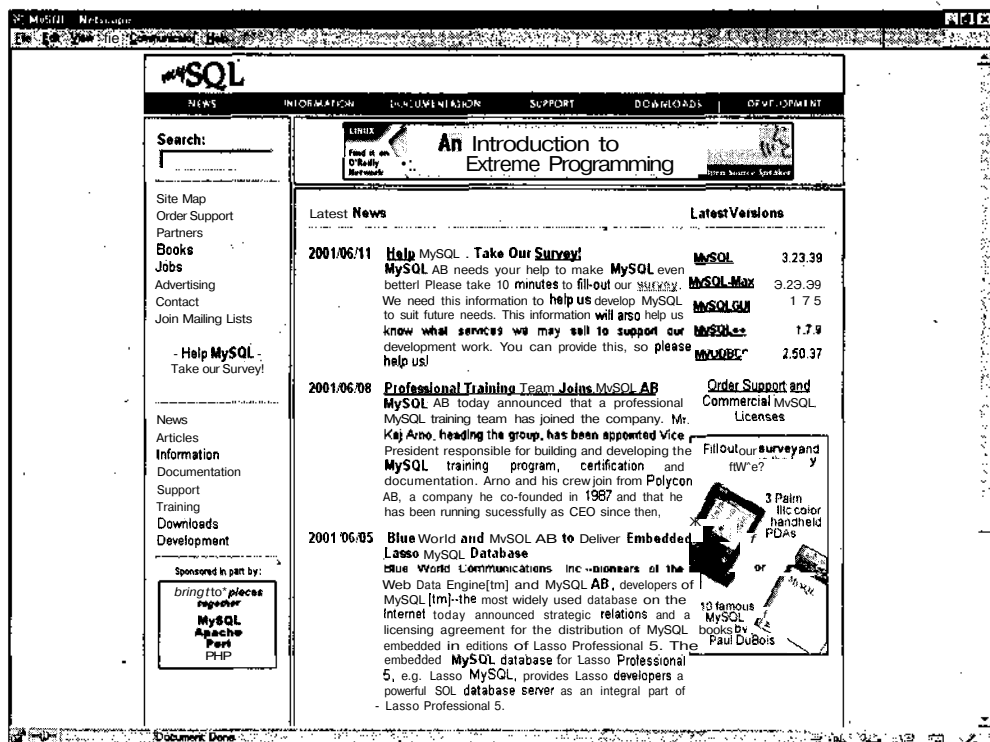


Рис. 11.1 ▼ Систему MySQL можно загрузить с сайта MySQL (www.MySQL.com). Здесь также содержится документация по установке и использованию данного программного обеспечения, а также даются сведения по лицензированию этого продукта

об установке MySQL (см. www.MySQL.com или другой сайт, откуда можно загрузить это программное обеспечение).

Базы данных создаются, обновляются и просматриваются с помощью языка SQL (языка структурированных запросов). В SQL удивительно мало команд, за что его одновременно и хвалят, и ругают. Он был задуман таким образом, чтобы на нем писали, как на английском языке. Поэтому иногда приходится ломать голову, чтобы создать из горстки имеющихся команд инструкции посложней. Помните, что использование языка SQL в Web-приложении увеличивает возможность возникновения ошибок (фактически появляется еще один язык программирования), поэтому тщательно проверяйте свою работу с базой данных.

Разработаем простую базу данных, куда будут записываться данные, поступающие от пользователя по каналам обратной связи. Представленного здесь материала будет достаточно, чтобы вы смогли начать работать самостоятельно. См. также приложение С.

Так как мы будем работать с системой MySQL, все функции в этой главе будут специфическими функциями MySQL. Например, для подсоединения к базе

данных в MySQL используется функция `mysql_connect()`. В другой СУБД (имеется в виду PostgreSQL) такую же работу выполняет функция `pg_connect()`. Как видите, очень похоже, но, если вы используете другую СУБД, вам необходимо обратиться к руководству по PHP (доступному на www.PHP.net) и найти там соответствующие имена функций.

Соединение с сервером и создание базы данных

Работая с файлами и каталогами (глава 10), при открытии объекта мы сначала создавали на него указатель. Затем этот указатель использовался во всех операциях с файлом или каталогом. При чтении списка файлов в каталоге мы познакомились с еще одним важным понятием - последовательно перемещаемым указателем на имя файла при каждом вызове функции `readdir()`. Подобный указатель используется и при работе с базами данных. Сначала необходимо соединиться с сервером БД (в нашем случае с сервером MySQL). Затем это соединение будет использоваться как точка доступа для последующих команд. Синтаксис MySQL для соединения с сервером следующий:

```
$Link=mysql_connect("host","user","password");
```

Связь устанавливается с помощью трех аргументов: хост, который почти всегда обозначается как `localhost`, имя пользователя и пароль. Два последних параметра определяют ваши права доступа к БД.

Права доступа к базе данных - вопрос еще более сложный, чем права доступа к файлам. Необходимо понимать, что разные категории пользователей имеют разные права доступа. Например, администратор СУБД может создавать новые и удалять старые базы данных (в СУБД могут быть десятки баз данных), а администратор нижнего уровня может только создавать и модифицировать таблицы в одной единственной БД. Обычно пользователю разрешено только читать информацию из таблиц, но не модифицировать их.

Ваш провайдер наверняка предоставит вам частичные права администратора - управление одной базой данных, но не всей СУБД - и создаст для вас исходную БД. Если вы работаете на своем сервере или имеете права администратора, то сможете создавать сколько угодно новых баз данных.

Новая база данных создается с помощью следующей функции:

```
mysql_create_db("databasename",$Link);
```

Обратите внимание на то, что значение аргумента `$Link` мы получили при соединении с БД и дальше используем его для работы с сервером БД так же, как применяли указатель файла при работе с файлом.

После завершения работы с СУБД рекомендуется отключить соединение:

```
mysql_close($Link);
```

Создадим новую базу данных, что потребует наличия у вас прав администратора. Если провайдер ограничивает ваши права доступа, он должен создать исходную БД по вашему запросу, и в этом случае вы можете сразу перейти к нашему следующему разделу, «Создание таблицы».

Подсоединение к MySQL и создание базы данных

1. Откройте новый PHP-документ в текстовом редакторе.
2. Начните со стандартного HTML-заголовка.

```
<HTML><HEAD><TITLE>Creating a Database</TITLE></HEAD><BODY>
```

3. Откройте PHP-раздел сценария и задайте переменные базы данных.

```
<?php
$Host = "localhost";
$User = "user";
$Password = "password";
$DBName = "NewDatabase";
```

Присвоив эти значения переменным, позже вы сможете легко изменить сценарий для работы с другими базами. Если вы работаете на сервере провайдера, он должен предоставить вам имя пользователя и пароль. Пробелы в имени базы данных не используются, как и в именах переменных и функций.

4. Установите связь с базой данных.

```
$Link = mysql_connect ($Host, $User, $Password);
```

Этой строкой сценария будет устанавливаться связь с СУБД MySQL на сервере с помощью имени хоста, имени пользователя и пароля. Если введенное имя пользователя или пароль не соответствуют определенным в базе данных, в момент выполнения сценария вы получите сообщение об ошибке.

5. Попробуйте создать новую базу данных и распечатайте сообщение о результате своих действий.

```
if (mysql_create_db ($DBName, $Link)) {
    print ("The database, $DBName, was successfully
    -created!<BR>\n");
} else {
    print ("The database, $DBName, could not be created!<BR>\n");
}
```

Когда база данных создана, вы получаете сообщение об этом (рис. 11.2). Если по какой-то причине база создана не была, вы увидите несколько сообщений об ошибках MySQL, а также сообщение «Не может быть создана!», сгенерированное этой условной конструкцией (рис. 11.3).

6. Отключите связь с системой MySQL, закройте PHP-раздел и HTML.

```
mysql_close ($Link);
?></BODY></HTML>
```

Рис. 11.2 ▼ Это HTML-форма для ввода данных в базу. Введенная информация будет сохранена в таблице Feedback в БД NewDatabase

Рис. 11.3 т Посмотрите, как автоматически экранирован апостроф. Это сделано для того, чтобы при отправке запроса в базу данных не возникло проблем. Если бы апостроф не был экранирован, SQL бы «подумал», что последний столбец состоит только из строки "Now I", и произошла бы ошибка

Необязательно закрывать сеанс связи с MySQL, так как это будет сделано автоматически сразу же после завершения выполнения сценария. Но я бы рекомендовал **все-таки** быть последовательным и осуществить этот шаг.

7. Сохраните сценарий как CreateDB.php (листинг 11.1), загрузите его на сервер и протестируйте в браузере.

Листинг 11.1 т Создание новой базы данных состоит из трех этапов: соединение с сервером, использование функции `mysql_create_db()` и закрытие соединения. У меня вошло в привычку задавать параметры базы данных - хост, имя пользователя, пароль и имя базы данных - как переменные, для того чтобы позже их легко можно было изменить.

```
1 <HTML>
2 <HEAD>
```

```

3 <TITLE>Creating a Database</TITLE>/HEAD>
4 <BODY>
5 <?php
6 // Установка значения переменных для доступа к базе данных.
7 $Host = "localhost";
8 $User = "user";
9 $Password = "password";
10 $DBName = "NewDatabase";
11
12 $Link = mysql_connect ($Host, $User, $Password);
13 if (mysql_create_db ($DBName, $Link)) {
14     print ("The database, $DBName, was successfully created!<BR>\n");
15 } else {
16     print ("The database, $DBName, could not be created!<BR>\n");
17 }
18 mysql_close ($Link);
19 ?>
20 </BODY>
21 </HTML>

```



В PHP поддерживается большинство баз данных, включая dBase, FilePro, mSQL, MySQL, Oracle, PostgreSQL и SyBase. Если вы используете базу данных, которая не поддерживается напрямую (например, Access или MS SQL-сервер), вам необходимо использовать набор функций ODBC (открытый интерфейс доступа к базам данных) в PHP и подключить соответствующие ODBC-драйвера к указанным СУБД. Более подробная информация содержится в приложении С.



Использование PHP с системой MySQL стало таким распространенным явлением, что появилось два специальных термина, обозначающие сервера, сконфигурированные с PHP и MySQL: LAMP (операционная система Linux, сервер Apache, СУБД MySQL, PHP) и WAMP (операционная система Windows, сервер Apache, СУБД MySQL, PHP).

Создание таблицы

После создания исходной базы данных разумно приступить к генерированию отдельных таблиц в ней. БД может содержать много таблиц, каждая из которых состоит из столбцов и строк.

Создадим простую таблицу, где будут храниться данные. Для этого используем язык SQL. Составляем SQL-запрос и обращаемся с **НИМ** к базе данных следующим образом:

```

$Query="текст запроса в формате SQL";
mysql_db_query ("DatabaseName", $Query, $Link);

```

Так как SQL похож на обычный английский язык, запрос на создание базы данных будет выглядеть так:

```

$Query="CREATE table TABLENAME(column1, column2 и т.д.)";

```

Каждому столбцу, отделенному от других запятыми, необходимо присвоить имя и тип. Типичные типы - TEXT (текст) и INT (целое число). Теория реляционных БД требует, чтобы первый столбец создавался как первичный ключ, который мог бы однозначно идентифицировать всю строку. Поэтому запрос на создание таблицы должен быть таким:

```
$Query="CREATE table NewTable(id INT PRIMARY KEY, information TEXT)";
```

Первичный ключ таблицы - это специальный столбец с уникальными значениями, используемыми для обращения к табличным строкам. Этот столбец индексируется в базе данных для того, чтобы вы могли быстрее передвигаться по таблице. Последняя может иметь только один первичный ключ, который я обычно задаю как автоматически увеличивающееся на единицу целое число. Первая строка будет иметь ключ 1, вторая - 2 и т.д. В дальнейшем с помощью первичного ключа (подобно индексу в массиве) можно извлекать значения из этой строки таблицы.

Более подробные сведения о SQL содержатся на сайте MySQL (рис. 11.1). Однако предоставленной здесь информации вполне достаточно для выполнения основных задач при работе с базами данных.

Создадим таблицу, в которой будут храниться данные, передаваемые из HTML-формы. В следующем разделе этой главы мы напишем сценарий, который будет вставлять отправленные материалы в созданную здесь таблицу.

Создание новой таблицы

1. Создайте новый PHP-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>Creating a Table</TITLE></HEAD><BODY>
```

3. Откройте PHP-раздел сценария и задайте переменные для доступа к базе данных.

```
<?php
$Host = "localhost";
$User = "user";
$Password = "password";
$DBName = "NewDatabase";
$TableName = "Feedback";
```

Обратите внимание на то, что к списку из листинга 11.1 мы добавили еще одну переменную - \$TableName. Она будет содержать имя таблицы с данными. Так как мы собираемся хранить информацию о пользователях, назовем нашу таблицу Feedback (обратная связь). Внимание! В MySQL имена таблиц и столбцов являются регистрозависимыми.

4. Установите связь с сервером.

```
$Link = mysql_connect ($Host, $User, $Password);
```

5. Напишите запрос.

```
$Query = "CREATE table $TableName (id INT UNSIGNED NOT
-NULL AUTO_INCREMENT PRIMARY KEY, FirstName TEXT, LastName
-TEXT, EmailAddress TEXT, Comments TEXT)";
```

Разобьем запрос на части. Сначала создаем новую таблицу, написав `CREATE table $TableName` (имя переменной `$TableName` будет заменено значением переменной `$TableName` при выполнении запроса). Затем в скобках перечисляем все столбцы, отделяя их друг от друга запятыми.

Первый столбец в таблице называется `id`. Это будет целое число без знака (запись `INT UNSIGNED` означает, что целое число может быть только положительным). Слова `NOT NULL` указывают, что данный столбец не может иметь пустое значение. При добавлении новой строки (`AUTO_INCREMENT`) значения автоматически увеличиваются на единицу и исполняют функцию первичного ключа.

Следующие четыре столбца будут текстовыми: первый содержит имя, второй - фамилию, третий - адрес электронной почты, четвертый - комментарии.

6. Обратитесь к базе данных с запросом и напечатайте соответствующее сообщение о результатах его выполнения.

```
if (mysql_db_query ($DBName, $Query, $Link)) {
    print ("The query was successfully executed!<BR>\n");
} else {
    print ("The query could not be executed!<BR>\n");
}
```

Если запрос выполнен успешно (значение условной конструкции истинно), вы увидите сообщение (рис. 11.4). Если таблица не была создана в результате ошибки в SQL, результат будет подобен тому, что представлен на рис. 11.5. Когда же таблица не была создана из-за отсутствия определенных прав доступа, появится сообщение, как на рис. 11.3.

7. Закройте соединение с системой MySQL, закройте PHP-раздел и HTML.

```
mysql_close ($Link);
?></BODY></HTML>
```

8. Сохраните сценарий как `CreateTable.php` (листинг 11.2), загрузите его на сервер и протестируйте в браузере.

Листинг 11.2 т Создание таблицы и большинства других запросов к базе данных обеспечивается написанием соответствующего запроса и использованием функции `mysql_db_query()`.

```
1 <HTML>
2 <HEAD>
3 <TITLE>Creating a Table</TITLE></HEAD>
4 <BODY>
5 <?php
6 // Установка значений переменных для доступа к базе данных.
7 $Host = "localhost";
8 $User = "user";
9 $Password = "password";
```

```

10 $DBName = "NewDatabase";
11 $TableName = "Feedback";
12
13 $Link = mysql_connect ($Host, $User, $Password);
14 $Query = "CREATE table $TableName (id INT UNSIGNED NOT NULL
    AUTO_INCREMENT PRIMARY KEY, FirstName TEXT, LastName TEXT,
    EmailAddress TEXT, Comments TEXT)";
15 if (mysql_db_query ($DBName, $Query, $Link)) {
16     print ("The query was successfully executed!<BR>\n");
17 } else {
18     print ("The query could not be executed!<BR>\n");
19 }
20 mysql_close ($Link);
21 ?>
22 </BODY>
23 </HTML>

```



Необязательно писать SQL-запросы, периодически используя заглавные буквы, как в приведенном примере. Но такой стиль помогает легко отличать SQL от имен таблиц и столбцов. Имена таблиц и столбцов являются регистрозависимыми, а SQL-команды в основном нет.

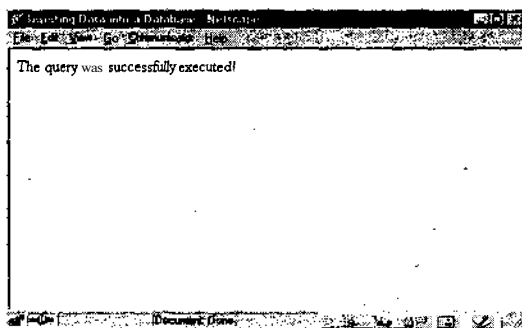


Рис. 11.4 ▼ Эта страница разработана для пользователя, и она не поможет вам при разработке сайта. Однако, посмотрев исходный текст, можно увидеть, каким был SQL-запрос (рис. 11.5)

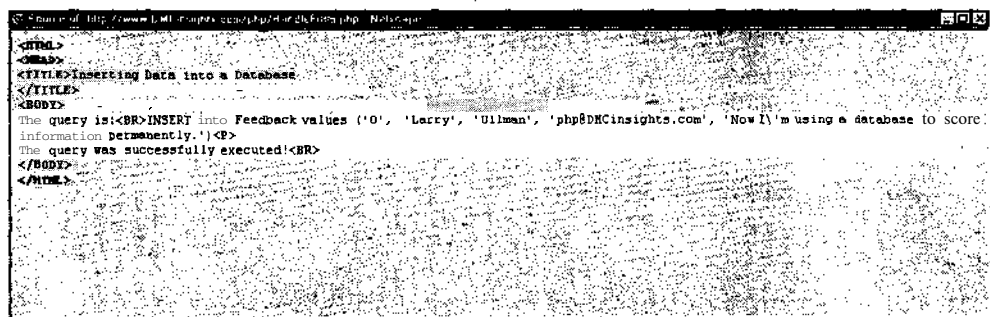


Рис. 11.5 ▼ При разработке сайта вы можете печатать SQL-запросы как HTML-комментарии и проверять, что именно делает код

Отправка данных

Как уже упоминалось выше, в созданной нами базе в табличном виде будут храниться данные пользователей для будущего просмотра. В предыдущем разделе мы создали таблицу с пятью столбцами: ключ, имя, фамилия, адрес электронной почты и комментарии. Процесс добавления информации в таблицу похож на создание таблицы: используется та же самая функция, а вот SQL-запрос будет другим.

```
$Query="INSERT into $TabName values ('value1','value2','value3',etc.)";
mysql_db_query("DatabaseName",$Query,$Link);
```

Запрос начинается со строки `INSERT into $TabName values`. Затем в скобках перечисляются значения столбцов, взятые в одинарные кавычки и разделенные запятыми. Количество значений должно точно совпадать с имеющимися в таблице столбцами, иначе запрос не будет работать. После этого запрос направляется в MySQL с помощью функции `mysql_db_query()`.

Чтобы наглядно показать, как это работает, используем HTML-форму, в которую пользователь вводит имя, фамилию, адрес электронной почты и комментарии. PHP-сценарий, обрабатывающий форму, поместит полученную информацию в базу данных.

Ввод данных в БД из HTML-формы

1. Создайте новый HTML-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок:

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD><BODY>
```

3. Создайте форму.

```
<FORM ACTION="HandleForm.php" METHOD=POST>
```

4. Создайте четыре текстовых поля.

```
First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20><BR>
Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
E-mail Address <INPUT TYPE=TEXT NAME="Array[Email]" SIZE=60><BR>
Comments <TEXTAREA NAME="Array[Comments]" ROWS=5 COLS=40>
</TEXTAREA><BR>
```

Вы можете сделать форму более эстетичной, чем эта, но не забудьте записать имена переменных ввода - они потребуются в странице `HandleForm.php`.

5. Добавьте кнопку **Submit**, закройте форму и HTML-страницу.

```
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
</FORM></BODY></HTML>
```

6. Сохраните страницу как `form.html` (листинг 11.3) и загрузите ее на Web-сервер.

Листинг 11.3 т Варианты этой формы рассматривались в предыдущих главах. Очень важно помнить названия полей ввода в HTML-формах, чтобы можно было сослаться на них в PHP.

```

1  <HTML>
2  <HEAD>
3  <TITLE>HTML Form</TITLE>
4  </HEAD>
5  <BODY>
6  <FORM ACTION="HandleForm.php" METHOD=POST>
7  First Name <INPUT TYPE=TEXT NAME="Array[FirstName]" SIZE=20xBR>
8  Last Name <INPUT TYPE=TEXT NAME="Array[LastName]" SIZE=40><BR>
9  E-mail Address <INPUT TYPE=TEXT NAME="Array[Email]" SIZE=60xBR>
10 Comments <TEXTAREA NAME="Array[Comments]" ROWS=5 COLS=40>
    </TEXTAREA><BR>
11 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
12 </FORM>
13 </BODY>
14 </HTML>

```

Напишем сценарий `HandleForm.php`, который берет сгенерированные формой данные и помещает их в базу.

1. Создайте новый PHP-документ в текстовом редакторе.
2. Напишите стандартный HTML-заголовок.

```
<HTML><HEAD><TITLE>Inserting Data into a Database</TITLE></HEAD><BODY>
```

3. Откройте PHP-раздел страницы и на всякий случай приведите в порядок входящие данные, избавившись от лишних пробелов.

```

<?php
$Array["FirstName"] = trim ($Array["FirstName"]);
$Array["LastName"] = trim ($Array["LastName"]);
$Array["Email"] = trim ($Array["Email"]);
$Array["Comments"] = trim ($Array["Comments"]);

```

4. Задайте переменные для доступа в базу данных.

```

$Host = "localhost";
$User = "user";
$Password = "password";
$DBName = "NewDatabase";
$TableName = "Feedback";

```

5. Подсоединитесь к системе MySQL, затем напишите запрос.

```

$Link = mysql_connect ($Host, $User, $Password);
$query = "INSERT into $TableName values ("0", "$Array[FirstName]",
-"$Array[LastName]", "$Array[Email]", "$Array[Comments])";

```

Запрос начинается с обязательного кода `INSERT into $TableName values`. Затем перечисляются пять значений (по одному для каждого столбца), каждое из которых взято в одинарные скобки и отделено от последующего запятой.

Так как столбцу `id` был задан автоматический инкремент, вы можете использовать в качестве значения 0. Это значение недопустимо для данного поля, поэтому оно будет автоматически исправлено, а у столбца `id` появится следующий порядковый номер.

6. Чтобы упростить отладку запроса, продублируйте его, напечатав в браузере следующее:

```
print ("The query is:<BR>$Query<P>\n");
```

Если при работе с базой возникли какие-либо трудности, прежде всего проверьте логику и правильность написания запроса. Вряд ли вы захотите, чтобы пользователь видел запрос, но этот прием можно с успехом использовать на этапе отладки программы.

7. Все функции обращения к БД, кроме действий собственно с данными, выдают значение «истина» при успешном выполнении и «ложь» во всех остальных случаях. Используйте это, создав условную конструкцию, которая выполняет определенные действия исходя из результатов запроса.

```
if (mysql_db_query ($DBName, $Query, $Link)) {
    print ("The query was successfully executed!<BR>\n");
} else {
    print ("The query could not be executed!<BR>\n");
}
```

8. Закройте сеанс связи с MySQL, PHP-раздел и HTML.

```
mysql_close ($Link);
?></BODY></HTML>
```

9. Сохраните сценарий как `HandleForm.php` (листинг 11.4), загрузите его на сервер в один каталог с файлом `form.html` и протестируйте обе страницы в браузере (рис. 11.2 и 11.3).

Листинг 11.4 т Инструкция запроса на добавление информации в базу данных достаточно проста. Однако не забывайте, что количество значений в скобках должно соответствовать числу столбцов в таблице.

```
1  <HTML>
2  <HEAD>
3  <TITLE>Inserting Data into a Database</TITLE></HEAD>
4  <BODY>
5  <?php
6  /* Эта страница получает и обрабатывает данные, принятые
   от "form.html". */
7  // Удаление пробелов в начале и в конце строк.
8  $Array["FirstName"] = trim ($Array["FirstName"]);
9  $Array["LastName"] = trim ($Array["LastName"]);
10 $Array["Email"] = trim ($Array["Email"]);
11 $Array["Comments"] = trim ($Array["Comments"]);
12
13 // Установка значения переменных для доступа к базе данных.
14 $Host = "localhost";
```

```

15 $User = "user";
16 $Password = "password";
17 $DBName = "NewDatabase";
18 $TableName = "Feedback";
19
20 $Link = mysql_connect ($Host, $User, $Password);
21 $Query = "INSERT into $TableName values ("0", "$Array[FirstName]",
    "$Array[LastName]", "$Array[Email]", "$Array[Comments]")";
22 print ("The query is:<BR>$Query<P>\n");
23 if (mysql_db_query ($DBName, $Query, $Link)) {
24     print ("The query was successfully executed!<BR>\n");
25 } else {
26     print ("The query could not be executed!<BR>\n");
27 }
28 mysql_close ($Link);
29 ?>
30 </BODY>
31 </HTML>

```



Если вы хотите проверять инструкции SQL так, чтобы пользователь явно не видел их, добавьте в вашу страницу следующую строку кода: `print ("<!--The query is $Query-->\n")`; . Теперь запрос будет выглядеть как HTML-комментарий, который можно просмотреть, открыв исходный текст (рис. 11.4 и 11.5).

Извлечение данных

Хотя при изучении данного раздела снова будет использоваться функция `mysql_db_query()`, извлечение данных отличается от сохранения тем, что извлекаемая информация должна быть присвоена переменной. Рассмотрим этот процесс поэтапно.

Данные из базы можно прочитать с помощью простого запроса:

```
$Query="SELECT * from $TableName";
```

Звездочка эквивалентна понятию «все» (то есть требуется выбрать все из столбца `$TableName`). Этой короткой инструкции часто вполне достаточно для извлечения данных.

Однако запрос можно ограничить, указав только часть полей, например `SELECT FirstName, Comments from $TableName`. Этот запрос дает задание извлечь информацию только из этих двух столбцов (имя и комментарий).

Другой способ ограничить запрос - написать что-нибудь вроде `SELECT * from $TableName where (FirstName='Larry')`. Здесь мы запрашиваем информацию из всех столбцов, но только из тех строк, где в столбце с указанием имени имеется значение Larry. Хорошие примеры того, как в SQL эффективно и гибко используется всего несколько терминов.

Основное отличие извлечения данных от записи их в базу состоит в том, что запрос необходимо обрабатывать по-другому. Я предпочитаю присваивать результаты запроса переменной:

```
$Result=mysql_db_query($DBName, $Query, $Link);
```

Как сказал бы непрофессионал, этой переменной теперь известен результат запроса. Для извлечения информации необходимо поместить переменную \$Result в цикл, последовательно пройдя по всем строкам полученного результата запроса.

```
while($Row=mysql_fetch_array($Result)) {
    statements;
}
```

Каждая итерация цикла будет превращать следующую строку информации из запроса (сохраненную в переменной \$Result) в массив с названием \$Row. Этот процесс будет продолжаться, пока не **останется** больше строк с информацией. Лучший способ понять данную систему - написать сценарий, который считывает информацию из таблицы Feedback (не забудьте заполнить таблицу данными с помощью формы form.html).

Извлечение данных из таблицы

1. Создайте новый PHP-документ в текстовом редакторе.
2. Начните со стандартного HTML-заголовка.

```
<HTML><HEAD><TITLE>Retrieving Data from a Database</TITLE></HEAD><BODY>
```

3. Откройте PHP-раздел страницы и задайте переменные для доступа в базу данных.

```
<?php
// Установка значения переменных для доступа к базе данных.
$Host = "localhost";
$user = "user";
$password = "password";
$dbName = "NewDatabase";
$tableName = "Feedback";
```

4. Соединитесь с сервером базы данных, напишите и выполните запрос.

```
$Link = mysql_connect ($Host, $User, $Password);
$query = "SELECT * from $TableName";
$result = mysql_db_query ($dbName, $query, $Link);
```

Этот запрос прост и полезен. Как было упомянуто выше, его результаты будут сохранены в переменной, которую мы позже поместим в цикл.

5. Создайте HTML-таблицу для вывода на экран результатов запроса.

```
print ("<TABLE BORDER=1 WIDTH=\"%75%\" CELSPACING=2
-CRLLPADDING=2 ALIGN=CRENTER>\n");
print ("<TR ALIGN=CRENTER VALIGN=TOP>\n");
print ("<TD ALIGN=CRENTER VALIGN=TOP>Name</TD>\n");
print ("<TD ALIGN=CRENTER VALIGN=TOP>Email Address</TD>\n");
print ("<TD ALIGN=CRENTER VALIGN=TOP>Comments</TD>\n");
print ("</TR>\n");
```

Так как мы извлекаем информацию из таблицы и выводим ее на экран в виде кода HTML, стоит поместить данные и в HTML-таблицу. Тогда страница будет выглядеть аккуратно.

6. Создайте цикл, извлекающий строки из базы данных, пока они не кончатся.

```
while ($Row = mysql_fetch_array ($Result)) {
```

Цикл помещает в переменную \$Row массив, состоящий из первой строки таблицы в переменной \$Result. Затем исполняются соответствующие команды (шаг 7). При обращении к строке `mysql_fetch_array ($Result)` на очередном проходе цикла переменная \$Row получает следующую строку. Цикл функционирует, пока не останется строк с информацией.

7. Распечатайте информацию из базы данных в виде HTML-таблицы.

```
print ("<TR ALIGN=CENTER VALIGN=TOP>\n");
print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[FirstName]
-$Row[LastName]</TD>\n");
print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[EmailAddress]</TD>\n");
print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[Comments]</TD>\n");
print ("</TR>\n");
```

Так как была использована функция `mysql_fetch_array()`, вы можете обращаться к каждому отдельному столбцу в извлеченной из базы данных строке точно так же, как к массиву. Ключи массива – названия столбцов таблицы, то есть имя, фамилия, адрес электронной почты и комментарии (конечно, названия полей даются на английском языке и с обязательным соблюдением регистра букв).

8. Закройте строку HTML-таблицы и цикл `while`.

```
print ("</TR>\n");
}
```

Еще раз подчеркнем, что этот цикл получает строку данных из таблицы БД, присваивает ее массиву \$Row, затем печатает строку HTML-таблицы.

9. Отключитесь от системы MySQL, закройте HTML-таблицу, PHP-раздел и саму HTML-страницу.

```
mysql_close ($Link);
print ("</TABLE>\n");
?></BODY></HTML>
```

10. Сохраните сценарий как `DisplayDB.php` (листинг 11.5), загрузите его на сервер и протестируйте в браузере (рис. 11.6).

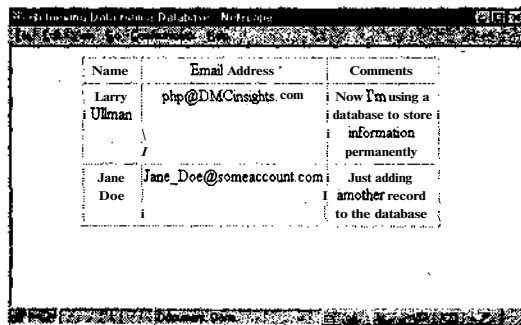
Листинг 11.5 т SQL-запрос на извлечение всех данных из таблицы очень прост, но, чтобы получить все данные* необходимо создать цикл.

```
1 <HTML>
2 <HEAD>
```

```

3 <TITLE>Retrieving Data from a Database</TITLE></HEAD>
4 <BODY>
5 <?php
6 // Установка значения переменных для доступа к базе данных.
7 $Host = "localhost";
8 $User = "user";
9 $Password = "password";
10 $DBName = "NewDatabase";
11 $TableName = "Feedback";
12
13 $Link = mysql_connect ($Host, $User, $Password);
14
15 $Query = "SELECT * from $TableName";
16 $Result = mysql_db_query ($DBName, $Query, $Link);
17
18 // Создание таблицы.
19 print ("<TABLE BORDER=1 WIDTH=75% CELLSPACING=2 CELLPADDING=2
20 ALIGN=CENTER>\n");
21 print ("<TR ALIGN=CENTER VALIGN=TOP>\n");
22 print ("<TD ALIGN=CENTER VALIGN=TOP>Name</TD>\n");
23 print ("<TD ALIGN=CENTER VALIGN=TOP>Email Address</TD>\n");
24 print ("<TD ALIGN=CENTER VALIGN=TOP>Comments</TD>\n");
25 print ("</TR>\n");
26
27 // Получение результатов из базы данных.
28 while ($Row = mysql_fetch_array ($Result)) {
29     print ("<TR ALIGN=CENTER VALIGN=TOP>\n");
30     print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[FirstName]
31     $Row[LastName]</TD>\n");
32     print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[EmailAddress]</TD>\n");
33     print ("<TD ALIGN=CENTER VALIGN=TOP>$Row[Comments]</TD>\n");
34     print ("</TR>\n");
35 }
36 mysql_close ($Link);
37 print ("</TABLE>\n");
38 ?>
39 </BODY>
40 </HTML>

```



Name	Email Address	Comments
Larry Ullman	php@DMConights.com	Now I'm using a database to store information permanently
Jane Doe	Jane_Doe@someaccount.com	Just adding another record to the database

Рис. 11.6 ▼ С помощью PHP можно извлекать данные из БД и создавать динамические Web-страницы, что невозможно сделать только средствами HTML

Глава 12

Использование cookie

Сookie - один из инструментов Internet, который меньше всего понимают и незаслуженно оговаривают. Мало кто понимает, какую реальную пользу может принести это средство. Когда cookie еще не было, навигация по сайту **не** оставляла никаких следов. Хотя ваш браузер отслеживает, какие страницы вы посетили, позволяя с помощью кнопки Назад возвращаться на эти страницы, а также окрашивая ссылки, по которым вы переходили на ресурсы, другим цветом, на сервере не ведется учет того, кто что видел. Все это еще справедливо для сайтов, где не применяются cookie, и для пользователей, которые отключили возможность использования cookie в своем браузере (рис. 12.1).

Если бы сервер не мог отслеживать действия пользователя, то в виртуальных магазинах не удавалось бы оформлять заказы. Если бы не было идентификаторов cookie или они были бы отключены в браузере, то люди не могли бы пользоваться такими службами, как Hotmail, где требуется регистрация пользователя.

Cookie используются сервером для хранения выборочной информации о пользователе на его компьютере. При первом обращении к Web-сайту информация о визите каким-то образом кодируется и позволяет идентифицировать этого пользователя во время последующих сессий. Представьте cookie в виде уникального номера: вы сообщаете серверу свое имя, и он присваивает вам идентификатор. Обратившись к cookie, сервер будет «знать», кто вы.

Это подводит нас к еще одному вопросу безопасности. Об инструменте cookie отзываются плохо, поскольку пользователи верят, что с его помощью о них можно узнать слишком многое. Однако cookie могут получать только ту

¹ Этот термин очень специфичен, его трудно перевести на русский язык. Можно только приблизительно передать его смысл как персональный идентификатор. Оставим английское написание слова «cookie» - читается «куки», ударение на первом слоге. - Прим. науч. ред.

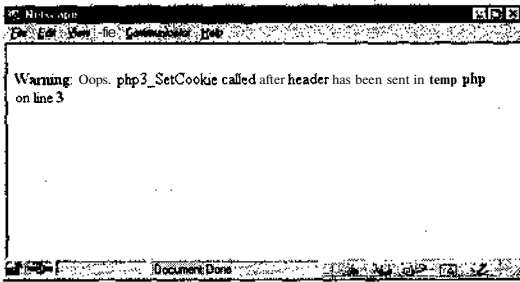


Рис. 12.2т Вы увидите такое детальное сообщение, если функция `setcookie()` вызвана после того, как что-либо, даже пустая строка, было послано в Web-браузер

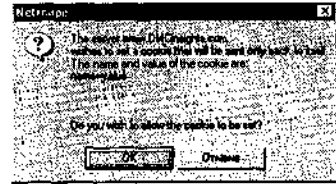


Рис. 12.3 т Если ваш браузер настроен так, чтобы выдавать предупреждение перед приемом cookie, вы будете каждый раз получать такое сообщение

Извлечь значение из cookie можно, обратившись к имени cookie как к переменной (знак доллара, затем имя), точно так же, как мы обращаемся к элементу HTML-формы как к переменной на странице обработки. Например, для извлечения значения cookie, созданного строкой `setcookie("UserName", "Larry");`, используется переменная `$UserName`.

Для примера создания cookie мы напишем сценарий, который позволит пользователю задавать цвета текста и фона страницы.

Отправка и извлечение cookie в PHP

1. Создайте новый PHP-документ в текстовом редакторе, начав со стандартного открывающего PHP-тэга.

```
<?php
```

2. Напишите условную инструкцию, которая отправит cookie, как только форма будет передана.

```
if ($BeenSubmitted) {
```

Как и раньше, переменная `$BeenSubmitted` будет использоваться для определения того, была ли передана форма. Если значение истинно, PHP обработает форму.

3. Задайте cookie, а затем цвета для страницы.

```
setcookie("BGColor", "$NewBGColor");
setcookie("TextColor", "$NewTextColor");
$BGColor = $NewBGColor;
$TextColor = $NewTextColor;
```

Если форма передана, тэг PHP отправит две переменных cookie со значениями цвета текста и фона. Сценарий заменит текущие значения (переменные `BGColor` и `TextColor`) на выбранные (`NewBGColor` и `NewTextColor`), моментально отразив эти изменения.

4. Допишите условную конструкцию.

```
} else {
    if (!$BGColor) {
```

Т ГЛАВА 12 ▼Использование cookie

```
$BGColor = "WHITE";  
}  
if (!$TextColor) {  
    $TextColor = "BLACK";  
}  
}
```

Если форма не была передана, PHP присвоит переменным значения по умолчанию.

5. Закройте первый PHP-раздел и создайте HTML-заголовок.

```
?>  
<HEAD>  
<TITLE>User Customization</TITLE>  
</HEAD>
```

6. Вставьте еще один PHP-раздел для печати тэга <BODY> с соответствующими значениями цвета фона и текста.

```
<?php  
print ("<BODY BGCOLOR=$BGColor TEXT=$TextColor>\n");  
?>
```

7. Напечатайте простое предложение, которое покажет цвет текста.

Currently your page looks like this!

8. Создайте HTML-форму, которая будет вызывать сама себя.

```
<FORM ACTION="cookies.php" METHOD=POST>
```

Для имени сценария все же удобнее использовать значение встроенной переменной \$PHP_SELF, которая всегда содержит это имя.

9. Создайте два ниспадающих меню, в которых пользователь сможет выбрать цвет фона и текста.

```
Select a new background color:  
<SELECT NAME="NewBGColor">  
<OPTION VALUE=WHITE>WHITE</OPTION>  
<OPTION VALUE=BLACK>BLACK</OPTION>  
<OPTION VALUE=BLUE>BLUE</OPTION>  
<OPTION VALUE=RED>RED</OPTION>  
<OPTION VALUE=GREEN>GREEN</OPTION>  
</SELECT>  
Select a new text color:  
<SELECT NAME="NewTextColor">  
<OPTION VALUE=WHITE>WHITE</OPTION>  
<OPTION VALUE=BLACK>BLACK</OPTION>  
<OPTION VALUE=BLUE>BLUE</OPTION>  
<OPTION VALUE=RED>RED</OPTION>  
<OPTION VALUE=GREEN>GREEN</OPTION>  
</SELECT>
```

Каждое ниспадающее меню имеет 5 опций, представленных в виде слов. При желании в эти меню можно добавить другие цвета. Набор цветов ограничен только палитрой HTML.

10. Задайте скрытую переменную, которая будет показывать, передана ли форма.

```
<INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
```

Эта переменная сообщает сценарию, что пользователь уже сделал свой выбор.

11. Создайте кнопку Submit, закройте форму и HTML-страницу.

```
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
</FORM>
</BODY>
</HTML>
```

12. Сохраните сценарий как `cookies.php` (листинг 12.1), загрузите его на сервер и протестируйте в браузере (рис. 12.4-12.8).

Листинг 12.1. Две переменных cookie используются для хранения информации о выбранных пользователем цветах фона и текста. Переданные из формы значения будут присвоены переменным на странице, чтобы требуемые изменения произошли немедленно.

```
1  <?php
2  if ($BeenSubmitted) {
3      setcookie("BGColor", "$NewBGColor");
4      setcookie("TextColor", "$NewTextColor");
5      $BGColor = $NewBGColor;
6      $TextColor = $NewTextColor;
7  } else {
8      if (!$BGColor) {
9          $BGColor = "WHITE";
10     }
11     if (!$TextColor) {
12         $TextColor = "BLACK";
13     }
14 }
15 ?>
16 <HEAD>
17 <TITLE>User Customization</TITLE>
18 </HEAD>
19 <?php
20 print ("<BODY BGCOLOR=$BGColor TEXT=$TextColor>\n");
21 ?>
22 Currently your page looks like this!
23 <FORM ACTION="cookies.php" METHOD=POST>
24 Select a new background color:
25 <SELECT NAME="NewBGColor">
26 <OPTION VALUE=WHITE>WHITE</OPTION>
```

```

27 <OPTION VALUE=BLACK>BLACK</OPTION>
28 <OPTION VALUE=BLUE>BLUE</OPTION>
29 <OPTION VALUE=RED>RED</OPTION>
30 <OPTION VALUE=GREEN>GREEN</OPTION>
31 </SELECT>
32 Select a new text color:
33 <SELECT NAME="NewTextColor">
34 <OPTION VALUE=WHITE>WHITE</OPTION>
35 <OPTION VALUE=BLACK>BLACK</OPTION>
36 <OPTION VALUE=BLUE>BLUE</OPTION>
37 <OPTION VALUE=RED>RED</OPTION>
38 <OPTION VALUE=GREEN>GREEN</OPTION>
39 </SELECT>
40 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
41 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
42 </FORM>
43 </BODY>
44 </HTML>

```



Значение cookie автоматически кодируется с помощью переменной `urlencode()` при отправке и декодируется при поступлении в PHP-страницу. То же самое происходит со значениями, посылаемыми HTML-формами.

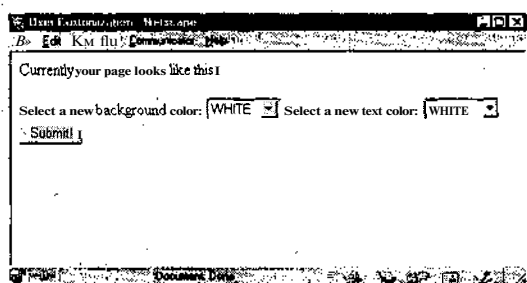


Рис. 12.4 т Так выглядит страница `cookies.php` при первом посещении. Для фона и текста используются цвета по умолчанию: белый и черный соответственно

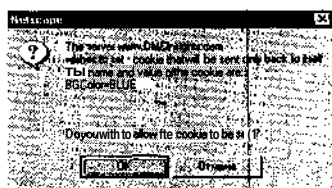


Рис. 12.5 т Такое сообщение пользователь получит при первом вызове функции `setcookie()`, если он установил в своем браузере опцию предупреждения об отправке cookie. Данный идентификатор загружает значение BLUE в переменную с именем BGColor

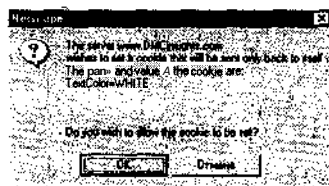


Рис. 12.6 т Вторая отсылаемая переменная cookie называется `TextColor` и имеет значение WHITE. Если пользователь не выбрал соответствующую опцию в меню Preferences, сообщения появляться не будут

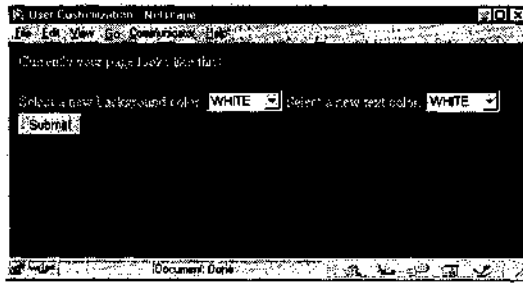


Рис. 12.7 После получения значений из HTML-формы страницей `cookies.php` будут отправлены две полученные переменные cookie для долговременного хранения этой информации, а изменения отразятся на странице



Рис. 12.8 По исходному тексту страницы можно отследить, как менялись значения цвета



Важно помнить, что значения cookie будут всегда иметь приоритет перед значениями, отправляемыми формой. В сценарии `cookies.php`, например, могут возникнуть проблемы, если в форме использовать имена полей `BGColor` и `TextColor`, поскольку они будут переписаны значениями cookie с такими же именами.

Добавление параметров в cookie

Хотя имени переменной и ее значения в функции `setcookie()` вполне достаточно в большинстве случаев, разрешается добавлять в функцию и другие аргументы. Функция может принимать до шести аргументов, каждый из которых накладывает какое-то ограничение на переменную cookie:

```
setcookie("name", "value", "expiration", "path", "domain", "secure");
```

Аргумент `expiration` используется, чтобы можно было задать срок существования cookie. Если он не определен, cookie будет функционировать, пока пользователь не закроет браузер. Обычно время жизни cookie задается в секундах,

начиная с настоящего момента. Эта строка кода задаст время жизни cookie в один час (60 секунд умножить на 60 минут) с настоящего момента:

```
setcookie("name","value",time()+3600);
```

В данном примере время жизни рассчитывается как значение `time()` плюс 3600, оба слагаемых даны в секундах. Аргумент не взят в кавычки (нам надо отправить не строку `time() + 3600`, а именно результат этого сложения).

Аргументы `path` и `domain` используются для ограничения cookie пределами конкретного домена или определенного каталога на сайте. Например, можно задать срок жизни cookie, а затем указать, что переменная будет существовать, только пока пользователь работает со своим каталогом домена:

```
setcookie("name","value",time()+3600,"/user/");
```

Аргумент `secure` заставляет посылать cookie только по защищенному протоколу HTTPS. Единица означает, что должен быть использован защищенный протокол, нуль указывает на то, что это необязательно. Защищенные протоколы обычно используются на сайтах электронной коммерции.

```
setcookie("name","value",time()+3600,"","1");
```

Как и в других функциях, принимающих аргументы, все значения необходимо передавать по порядку. В последнем примере мы не указали путь и домен, оставив кавычки пустыми. Таким образом, мы сохранили соответствующее количество аргументов, указав при этом, что необходимо использовать протокол HTTPS.

Добавим срок жизни в существующую страницу `cookies.php`, чтобы заданные пользователем параметры сохранились после закрытия браузера.

Задание срока жизни cookie

1. Откройте файл `cookies.php` в текстовом редакторе (листинг 12.1).
2. Во второй и третьей строках укажите срок жизни, равный нескольким дням или больше:

```
setcookie("BGColor", "$NewBGColor", time() + "10000000");
setcookie("TextColor", "$NewTextColor", time() + "10000000");
```

Если задать срок жизни как `time() + "10000000"`, cookie будет существовать около 116 дней (60 секунд x 60 минут x 24 часа x 115 дней \approx 10000000).

3. Сохраните сценарий (листинг 12.2), загрузите его на сервер и протестируйте в браузере (рис. 12.9 и 12.10).

Листинг 12.2 т Добавив аргумент срока жизни в cookies, мы обеспечили их существование даже после того, как пользователь закроет Web-браузер.

```
1 <?php
2 if ($BeenSubmitted) {
```

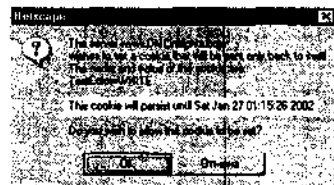


Рис. 12.9т Добавление аргумента срока жизни отражается в сообщении о cookie, которое получает пользователь

```

3      setcookie("BGColor", "$NewBGColor", time() + "10000000");
4      setcookie("TextColor", "$NewTextColor", time() + "10000000");
5      $BGColor = $NewBGColor;
6      $TextColor = $NewTextColor;
7  } else {
8      if (!$BGColor) {
9          $BGColor .= "WHITE";
10     }
11     if (!$TextColor) {
12         $TextColor = "BLACK";
13     }
14 }
15 ?>
16 <HEAD>
17 <TITLE>User Customization</TITLE>
18 </HEAD>
19 <?php
20 print ("<BODY BGCOLOR=$BGColor TEXT=$TextColor>\n");
21 ?>
22 Currently your page looks like this!
23 <FORM ACTION="cookies.php" METHOD=POST>
24 Select a new background color:
25 <SELECT NAME="NewBGColor">
26 <OPTION VALUE=WHITE>WHITE</OPTION>
27 <OPTION VALUE=BLACK>BLACK</OPTION>
28 <OPTION VALUE=BLUE>BLUE</OPTION>
29 <OPTION VALUE=RED>RED</OPTION>
30 <OPTION VALUE=GREEN>GREEN</OPTION>
31 </SELECT>
32 Select a new text color:
33 <SELECT NAME="NewTextColor">
34 <OPTION VALUE=WHITE>WHITE</OPTION>
35 <OPTION VALUE=BLACK>BLACK</OPTION>
36 <OPTION VALUE=BLUE>BLUE</OPTION>
37 <OPTION VALUE=RED>RED</OPTION>
38 <OPTION VALUE=GREEN>GREEN</OPTION>
39 </SELECT>
40 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
41 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
42 </FORM>
43 </BODY>
44 </HTML>

```

В определенных версиях браузеров Netscape и Internet Explorer возникают проблемы с cookie, в которых не использованы все аргументы. Если вы считаете, что подобные проблемы могут случиться и на вашем сайте, передавайте все аргументы, используя пустые кавычки для значений по умолчанию:

```
setcookie("BGColor", "$NewBGColor", time() + "10000000", "", "", "");
```

На самом деле разрешается задавать любые сроки жизни cookie. Однако стоит придерживаться следующих советов: если переменная cookie должна действовать только до конца сессии, не задавайте этот параметр; если желательно хранить настройки пользователя до его следующих посещений сайта, задайте время жизни

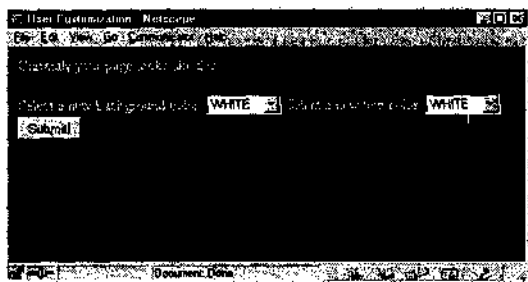


Рис. 12.10 ▼ Так как мы задали время жизни cookie, равное нескольким месяцам, предпочтения пользователя, которые сохранены в переменных cookie, будут действительными даже после того, как пользователь покинет сайт, а позже снова его посетит. Если бы cookies не были сохранены, пользователь увидел бы цвета по умолчанию и, заходя на эту страничку, должен был бы каждый раз задавать свои предпочтения заново

cookie, равное нескольким месяцам; если же cookie может подвергнуться риску тщательно охраняемую информацию, время жизни должно составлять час или меньше, чтобы cookies долго не хранились после того, как пользователь покинул сайт.



В целях безопасности срок жизни cookie можно устанавливать равным пяти или десяти минутам и задавать переменную заново каждый раз, когда пользователь заходит на новую страницу. Таким образом, идентификатор будет существовать, пока пользователь активен, и автоматически уничтожится через пять или десять минут после последнего действия пользователя.

Удаление cookie

Хотя переменная cookie автоматически пропадает, когда пользователь закрывает браузер или когда срок ее действия истекает, порой возникает необходимость удалить cookie вручную. В частности, на сайтах, работающих с зарегистрированными пользователями, все cookie удаляются, когда пользователь отменяет свою регистрацию на этом ресурсе.

Хотя функция `setcookie()` может принимать до шести аргументов, фактически для удаления требуется только один - имя cookie. Присвоение переменной cookie пустого значения - то же самое, что удаление cookie с тем же именем. Например, если для создания cookie `UserName` необходимо написать такую строку:

```
setcookie("UserName", "Larry");
```

то для удаления переменной `UserName` нужна следующая запись:

```
setcookie("UserName", "");
```


Ради предосторожности можно также задать срок действия, который уже истек.

```
setcookie("UserName","",time()-60);
```

Чтобы продемонстрировать эту возможность, добавим кнопку **Reset** на страницу cookies.php. Посланные ранее cookie будут стерты, и восстановятся цвета по умолчанию.

Выполнение действия

1. Откройте последнюю версию файла cookies.php в текстовом редакторе (листинг 12.2).
2. Добавьте условную инструкцию в уже существующую условную конструкцию if (\$BeenSubmitted) (листинг 12.3, строки 3-8).

```
if ($Reset) {  
    setcookie("BGColor", "", time()- "100");  
    setcookie ("TextColor", "" , time()-"100");  
    $BGColor = "WHITE";  
    $TextColor = "BLACK";  
} else {
```

Если форма отправлена, PHP сначала проверит, истинно ли значение переменной \$Reset. При положительном результате проверки сценарий удалит существующие cookies, установив пустые значения cookie с теми же именами. Сценарий также вернет значения цветов по умолчанию для данной страницы.

3. Закончите условную инструкцию if (\$Reset).

```
    setcookie("BGColor", "$NewBGColor", time()+ "1000000");  
    setcookie ("TextColor", "$NewTextColor", time()+ "1000000");  
    $BGColor = $NewBGColor;  
    $TextColor = $NewTextColor;  
}
```

Если форма отправлена, а значение переменной \$Reset ложно, форма должна быть обработана так же, как представлено в листинге 12.2.

4. В HTML-форму добавьте триггерную кнопку, чтобы пользователь мог вернуть исходные цвета (листинг 12.3, строка 47).

```
<P><INPUT TYPE=Checkbox NAME=Reset VALUE=TRUE>Check this box to reset  
the colors.</P>
```

Эта кнопка будет сообщать, возвращать ли исходные значения. Я добавил два тэга параграфа, чтобы HTML-форма выглядела эстетичней.

5. Сохраните сценарий (листинг 12.3), загрузите его на сервер и протестируйте в браузере (рис. 12.11-12.13).

Листинг 12.3 Для восстановления всех исходных значений посылаются пустые переменные cookie с именами существующих. В HTML-форму добавляется также триггерная кнопка **Reset**.

```

1  <?php
2  if ($BeenSubmitted) {
3      if ($Reset) {
4          setcookie("BGColor", "", time()- "100");
5          setcookie ("TextColor", "" , time()-"100");
6          $BGColor = "WHITE";
7          $TextColor = "BLACK";
8      } else {
9          setcookie("BGColor", "$NewBGColor", time()+ "1000000");
10         setcookie ("TextColor", "$NewTextColor", time()+ "1000000");
11         $BGColor = $NewBGColor;
12         $TextColor = $NewTextColor;
13     }
14 } else {
15     if (!$BGColor) {
16         $BGColor = "WHITE";
17     }
18     if (!$TextColor) {
19         $TextColor = "BLACK";
20     }
21 }
22 ?>
23 <HEAD>
24 <TITLE>User Customization</TITLE>
25 </HEAD>
26 <?php
27 print ("<BODY BGCOLOR=$BGColor TEXT=$TextColor>\n");
28 ?>
29 Currently your page looks like this!
30 <FORM ACTION="cookies.php" METHOD=POST>
31 Select a new background color:
32 <SELECT NAME="NewBGColor">
33 <OPTION VALUE=WHITE>WHITE</OPTION>
34 <OPTION VALUE=BLACK>BLACK</OPTION>
35 <OPTION VALUE=BLUE>BLUE</OPTION>
36 <OPTION VALUE=RED>RED</OPTION>
37 <OPTION V ALUE=GREEN>GREEN</OPTION>
38 </SELECT>
39 Select a new text color":
40 <SELECT NAME="NewTextColor">
41 <OPTION VALUE=WHITE>WHITE</OPTION>
42 <OPTION VALUE=BLACK>BLACK</OPTION>
43 <OPTION VALUE=BLUE>BLUE</OPTION>
44 <OPTION VALUE=RED>RED</OPTION>
45 <OPTION VALUE=GREEN>GREEN</OPTION>
46 </SELECT>
47 <P><INPUT TYPE=Checkbox NAME=Reset VALUE=TRUE>Check this box to reset
   the colors.<P>
48 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>

```

```

49 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
50 </FORM>
51 </BODY>
52 </HTML>

```

Для отладки функций `setcookie()` в рассмотренных выше сценариях не забудьте включить опцию предупреждения о каждой установке cookie в вашем браузере (рис. 12.1).

Функция `setcookie()` - одна из немногих в PHP, которая может по-разному работать в разных браузерах, так как они реагируют на cookie нестандартно. Для достижения наилучших результатов протестируйте созданный сайт на различных платформах с помощью разных браузеров.

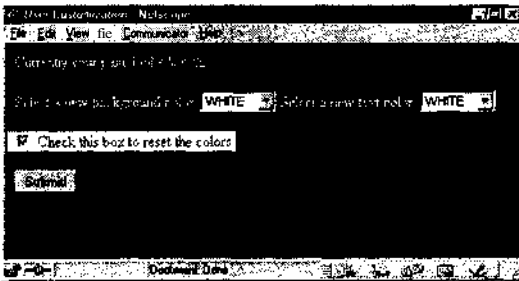


Рис. 12.11 Простая триггерная кнопка, добавленная в HTML-форму, дает пользователю возможность вернуть исходные цвета страницы. Это также удалит существующие cookies

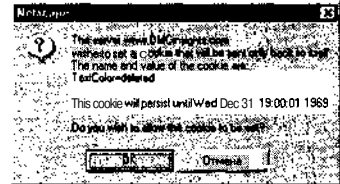


Рис. 12.12 Использование функции `setcookie()` с именем, но без значения, удалит существующую переменную cookie с тем же именем. Задание срока действия, истекшего в прошлом, тоже гарантирует удаление cookie

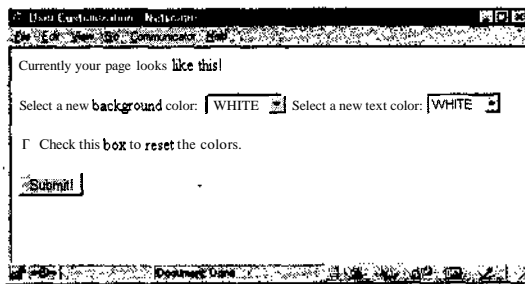


Рис. 12.13 Если пользователь пометил кнопку **Reset** и нажал кнопку **Submit**, PHP удалит cookie (рис. 12.12) и восстановит цвета по умолчанию

Глава 13

Создание Web-приложений

Теперь, когда вы изучили основы программирования на PHP, самое время использовать эти знания для создания профессиональных Web-приложений. В этой главе говорится о функциях и приемах, которые помогут сделать ваши сайты профессиональными, функциональными и удобными для сопровождения.

Использование функций `include` и `require`

До этого момента мы писали сценарии для работы с базами данных, обработки HTML-форм, сохранения cookie и многого другого, но все наши программы состояли только из одного файла. Однако, как только мы начинаем разрабатывать сложные сайты, практика переписывания одних и тех же функций в каждом сценарии становится неэффективной. Поразмыслив, вы наверняка обнаружите, что HTML-дизайн и PHP-функции используются на многих страницах Web-ресурсов. Можно размножить эту общую информацию в каждом отдельном сценарии, но, если вдруг возникнет необходимость внести какие-либо изменения, делать это придется во многих местах. Массу времени удастся сэкономить, создав отдельные файлы для общего кода и включив их в PHP-сценарии с помощью одной из двух инструкций: `include()` и `require()`. Синтаксис при этом будет выглядеть следующим образом:

```
include("file.php");  
require("file.php");
```

Обе команды работают почти одинаково, за исключением одного существенного различия. Независимо от того, в каком месте сценария имеется инструкция `require()`, ею всегда будет вставлен требуемый файл, даже если блок кода, где она задана, в данный момент не работает. Функция `include()`, наоборот,

включит код из **файла**, только если она действительно вызвана. Поэтому необходимо использовать команду `require ()`, когда файл должен быть включен, и `include ()`, если файл в зависимости от обстоятельств может быть включен или нет.

Что делают эти команды? Каждая из них включает указанный файл в основной (чтобы было более понятно, будем называть файл, который содержит строку `include ()` или `require ()`, *родительским файлом*). Любой код внутри файла будет трактоваться как обычный код HTML, если он не находится внутри PHP-скобок в самом включенном файле. Любые переменные, имеющиеся в родительском документе до вызова команд `include ()` или `require ()`, доступны включенному файлу, и любые переменные из включенного файла будут доступны родительскому документу после вызова этих функций.

Включаемые файлы стоит использовать по нескольким причинам. Можно поместить свои собственные функции в общий файл. Удобно также хранить информацию доступа к базе данных в едином конфигурационном файле. Для начала поместим наш HTML-дизайн во включаемые файлы, чтобы он мог быть использован в нескольких страницах.

Выполнение действия

1. Создайте новый PHP-документ в текстовом редакторе.

```
<HTML>
<HEAD>
<TITLE>
```

Хотя перед нами файл с расширением `.php`, в нем только один PHP-раздел, поэтому напомним большую часть кода как стандартный HTML-сценарий во избежание использования многочисленных инструкций `print ()`.

2. Используйте для печати заголовка страницы код, представленный ниже.

```
<?php
print ("{$PageTitle}");
?>
```

Значение переменной `$PageTitle` будет присвоено в родительском документе и затем использовано здесь для заголовка страницы, который выводится на экран в самом верху окна браузера (рис. 13.1).

3. Закончите HTML-заголовок.

```
</TITLE>
</HEAD>
<BODY>
```

4. Создайте таблицу, которая будет управлять разбивкой страницы.

```
<TABLE WIDTH="75%" ALIGN=CENTER BORDER=0>
<TR><TD ALIGN=CENTER><B>Welcome to the Site!</B></TD><TR>
<TR><TD ALIGN=LEFT><P>
```

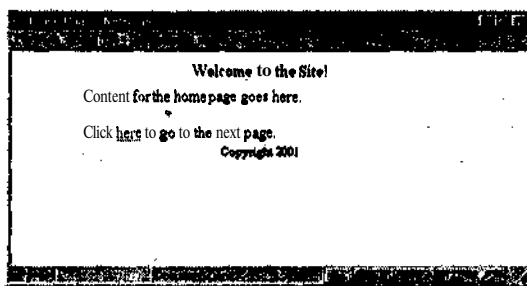


Рис. 13.1 Хотя эта страница предельно проста, она представляет собой пример того, как Web-страницы могут быть разбиты на несколько компонентов, используемых на многих сайтах (см. рис. 13.3)

Сайт выглядит как таблица из трех строк. Верхняя строка будет включать приветствие, хотя можно использовать и графику. Во второй строке поместим собственно содержание страницы, а на третьей показан нижний колонтитул. Мы используем первый включаемый файл для заголовка страницы, а также для верхней и начала второй строки таблицы.

5. Сохраните сценарий как **header.php** (листинг 13.1) и загрузите его на Web-сервер.

Листинг 13.1 ▼ Это первый включаемый файл, который содержит HTML-заголовок, а также начало таблицы. Поскольку каждая страница имеет свой заголовок, используется переменная, печатающая соответствующие значения.

```

1 <HTML>
2 <HEAD>
3 <TITLE>
4 <?php
5   print ("{$PageTitle}");
6 ?>
7 </TITLE>
8 </HEAD>
9 <BODY>
10 <TABLE WIDTH="75%" ALIGN=CENTER BORDER=0>
11 <TRxTD ALIGN=CENTER><B>Welcome to the Site!</B></TD><TR>
12 <TRxTD ALIGN=LEFT><P>
```

Напишем второй включаемый файл для нижнего колонтитула.

6. Создайте новый PHP-документ в текстовом редакторе.
7. Закройте вторую строку таблицы разбивки страницы.

```
<Px/TDx/TR>
```

8. Напишите код для последней, третьей строки таблицы, которая будет содержать информацию об авторском праве.

```
<TRxTD ALIGN=CENTER><SMALL>Copyright 2001</SMALL></TD></TR>
```

9. Закройте таблицу и HTML-страницу.

```
</TABLE>
</BODY>
</HTML>
```

10. Сохраните сценарий как **footer.php** (листинг 13.2) и загрузите его на сервер. Как только два включаемых файла готовы, можно приступить к созданию родительских страниц.

Листинг 13.2 Это второй включаемый файл, который завершит создание таблицы и закончит HTML-страницу. Здесь также представлена информация об авторском праве.

```
1  <P></TD></TR>
2  <TR><TD ALIGN=CENTER><SMALL>Copyright 2001</SMALL></TD></TR>
3  </TABLE>
4  </BODY>
5  </HTML>
```

11. Создайте новый PHP-документ в текстовом редакторе.

```
<?php
```

12. Присвойте имя страницы переменной \$PageTitle.

```
$PageTitle = "Home Page";
```

Значение переменной \$PageTitle используется для создания заголовка страницы, который будет размещен в верхней части окна браузера. Так как заголовки страниц различаются между собой, оформим их как переменную, используемую включаемым файлом.

13. Включите файл заголовка.

```
require ("header.php");
```

В этом случае не имеет значения, какая из двух функций применяется, но мы использовали require () для того, чтобы гарантировать включение. Данная строка поместит весь код файла header.php в эту страницу. Строка должна стоять в сценарии после того, как переменной \$PageTitle присвоено значение, иначе страница не будет иметь заголовка.

14. Создайте содержимое страницы.

```
print ("Content for the home page goes here.\n");
print ("<P>Click <A HREF=\"page2.php\">here</A> to go
-to the next page.\n");
```

Сюда включено только простое сообщение и ссылка на вторую страницу. Вы можете поместить любое содержание, включая то, которое динамически генерируется PHP.

15. Включите файл нижнего колонтитула и закройте страницу.

```
require ("footer.php");
?>
```

16. Сохраните страницу как index.php (листинг 13.3) и загрузите ее на сервер.

Листинг 13.3 т Как только вы создали два включаемых файла, команда `require ()` помещает их в родительский файл для создания всей страницы «на лету».

```

1 <?php
2 $PageTitle = "Home Page";
3 require ("header.php");
4 print ("Content for the home page goes here.\n");
5 print ("<P>Click <A HREF=\"page2.php\">here</A> to go to the next
   page.\n");
6 require ("footer.php");
7 ?>
```

А теперь напишем вторую страницу, где будут использоваться включаемые файлы, содержащие тот же дизайн сайта.

17. Создайте новый PHP-документ в текстовом редакторе.

```

<?php
$PageTitle = "Second Page";
require ("header.php");
print ("Content for the second page goes here.\n");
require ("footer.php");
?>
```

Чтобы отличать эту страницу от файла `index.php`, мы присвоили переменной `$PageTitle` новое значение, а затем изменили инструкцию `print ()`.

18. Сохраните сценарий как `page2.php` (листинг 13.4), загрузите его на сервер и протестируйте вместе с файлом `index.php` в браузере (рис. 13.1-13.3).

Листинг 13.4 т Это еще один родительский файл, во многом идентичный сценарию, данному в листинге 13.3. Независимо от количества страниц на вашем сайте всеми ими может быть использован этот базовый шаблон.

```

1 <?php
2 $PageTitle = "Second Page";
3 require ("header.php");
4 print ("Content for the second page goes here.\n");
5 require ("footer.php");
6 ?>
```



Желательно использовать расширение `.php`, а не `.inc`, которое некоторые программисты применяют для включаемых файлов. Хотя для родительского файла PHP это не имеет значения, ловкий пользователь иногда может посмотреть исходный текст файла `.inc`, а вкладываемого файла `.php` - нет (рис. 13.4 и 13.5).

Определение даты и времени

Мы уже использовали функцию `date ()` в нескольких примерах, но она заслуживает более детального рассмотрения. Функция `date ()` возвращает информацию о дате и времени в формате, продиктованном ее аргументами. Однако просто удивительно, как по-разному это можно использовать:

```
date("formatting");
```

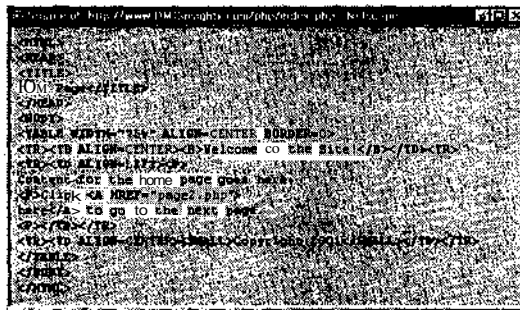



Рис. 13.2 ▽ Ничто в исходном тексте страницы не указывает на то, что она была собрана «на лету». Вы можете увидеть, что два включаемых файла **использованы**, как будто они являются частью самого родительского файла

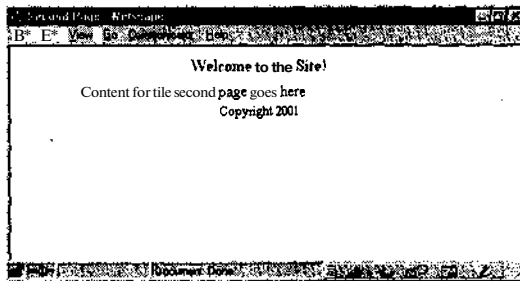


Рис. 13.3 Т. Этой страницей используются те же файлы для верхнего и нижнего колонтитулов, что и представленные на рис. 13.1, а содержание и заголовок - другие

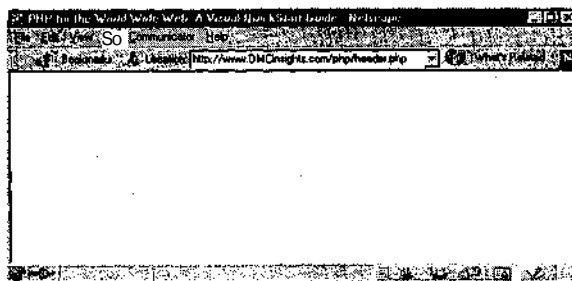


Рис. 13.4 Если для включаемых файлов использовать расширение `.php`, пользователи не смогут просмотреть их напрямую: сервер попытается исполнить код PHP и выведет на экран пустую страницу. В других случаях браузер может вывести на экран начало HTML-кода. В целях безопасности всегда используйте расширение `.png`

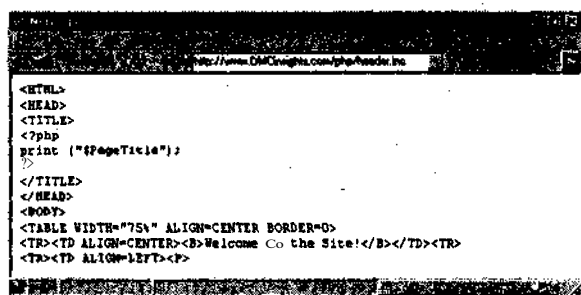


Рис. 13.5 ▼ Если для вкладываемых файлов используется расширение .inc (а это достаточно распространенный вариант), пользователь иногда может увидеть исходный текст страницы напрямую. Если во включаемом файле хранятся пароли и информация о доступе в базу данных, могут возникнуть серьезные неприятности

В главе 6 было показано, что функция `date("A")`; возвращает значения AM и PM. Список форматов даты полностью приведен в табл. С.5 (приложение С). Эти параметры можно комбинировать, например функция `date("l F j, Y")`; возвратит значение Friday January 26, 2001.

Функция `date()` может принимать еще один аргумент, называемый временной меткой. *Временная метка* — это число, обозначающее количество секунд, прошедших с первой секунды 1 января 1970 года — с начала отсчета времени во всех системах Unix. Как говорилось в главе 12, функция `time()` возвращает значение временной метки на текущий момент. Функция `mktime()` может возвращать значение временной метки на конкретное время и дату:

```
mktime(час, минута, секунда, месяц, день, год);
```

Например, строка кода `$Timestamp=mktime(12, 30, 0, 12, 27, 1997)`; присвоит переменной `$Timestamp` количество секунд от точки начала отсчета времени до 12:30 27 декабря 1997 года. Затем это значение может быть отправлено в функцию `date("D", $Timestamp)`; , которая возвратит значение Sat — трехбуквенный формат для обозначения дней недели.

Давайте создадим интерактивный календарь посредством функций `date()` и `mktime()`, которые будут использованы много раз.

Использование функции date

1. Создайте новый PHP-документ в текстовом редакторе.

```
<?php
```

2. Присвойте переменной `$PageTitle` значение, включите файл заголовка.

```
$PageTitle = "Calendar";
require ("header.php");
```

3. Раз уж мы разработали дизайн сайта и создали необходимые включаемые файлы, будем использовать их и дальше. Присвойте переменным `$Month` и `$Year` значения по умолчанию, если они не еще не определены.

```
if ((!$Month) && (!$Year)) {
    $Month = date ("m");
    $Year = date ("Y");
}
```

Когда пользователь обращается к этому ресурсу в первый раз, значения месяца или года не передаются на страницу. PHP использует текущий месяц (например, 1) и год (например, 2001).

4. Создайте временную метку, относящуюся к конкретному месяцу и году.

```
$Timestamp = mktime (0, 0, 0, $Month, 1, $Year);
```

Для получения временной метки мы задаем в функции `mktime()` нули для часов, минут и секунд и единицу - для дней. Месяц и год будут взяты из соответствующих переменных.

5. Определите полное название месяца.

```
$MonthName = date("F", $Timestamp);
```

Если задать значение первого аргумента как `F`, функция `date()` возвратит название месяца (например, January).

6. Создайте таблицу для размещения календаря на экране.

```
print ("<TABLE BORDER=0 CELLPADDING=3 CELLSPACING=0 ALIGN=CENTER>");
```

7. Напечатайте заголовок с указанием месяца и года.

```
print ("<TR BGCOLOR=BLUE><TD COLSPAN=7 ALIGN=CENTER>
<FONT COLOR=WHITE><B>$MonthName $Year</B></FONT></TD></TR>");
```

8. Наберите дни недели.

```
print ("<TR BGCOLOR=BLUE><TD ALIGN=CENTER WIDTH=20xB>
-<FONT COLOR=WHITE>Su</FONT></B></TD><TD ALIGN=CENTER WIDTH=20><B>
-<FONT COLOR=WHITE>M</FONT></B></TD><TD ALIGN=CENTER WIDTH=20><B>
-<FONT COLOR=WHITE>Tu</FONT></B></TD><TD ALIGN=CENTER WIDTH=20><B>
-<FONT COLOR=WHITE>W</FONT></B></TD><TD ALIGN=CENTER WIDTH=20xB>
-<FONT COLOR=WHITE>Th</FONT></B></TD><TD ALIGN=CENTER WIDTH=20><B>
-<FONT COLOR=WHITE>F</FONT></B></TD><TD ALIGN=CENTER WIDTH=20><B>
-<FONT COLOR=WHITE>Sa</FONT></B></TD></TR></n>");
```

Если внутри печатаемого текста нет переменных, а он очень длинный, используйте одинарные кавычки. Тогда можно свободно вставлять новые строки, так что видеть коды PHP и HTML будет легче.

9. Определите первый день недели для этого месяца.

```
• $MonthStart = date("w", $Timestamp);
```

Эта строка присвоит значение дня недели (в цифровой форме от 0 до 6) переменной `$MonthStart` для месяца и года. Так как первый день месяца был использован при задании временной метки, данный код определит этот конкретный день. Таким образом удастся узнать, с какого дня недели начинается месяц: с понедельника, вторника и т.д.

10. Проследите, чтобы значение переменной `$MonthStart` не было нулевым.

```
if ($MonthStart == 0) {
    $MonthStart = 7;
}
```

Если значение переменной `$MonthStart` равно нулю, это означает, что месяц начинается с воскресенья и у нас будут проблемы с использованием данного числа при выполнении сценария. Чтобы избежать проблем, заменим ноль семеркой.

11. Определите последний день месяца.

```
$LastDay = date("d", mktime (0, 0, 0, $Month+1, 0, $Year));
```

Переменной `$LastDay` будет присвоено цифровое значение (например, 31, 28 или 30), равное последнему дню месяца. Это определено использованием нулевого дня и следующего месяца (`$Month+1`) в функции `mktime()`.

12. Задайте начальную дату как первый день месяца.

```
$StartDate = -$MonthStart;
```

Так как календарь будет начинаться с воскресенья, необходимо определить, какое количество дней пропустить до первого дня месяца. Если первый день месяца – вторник, значение `$MonthStart` равно 2, следовательно, значение `$StartDate` равно -2, то есть будут созданы два пустых дня до начала месяца.

13. Напишите цикл, который станет печатать строки календаря (недели).

```
for ($k = 1; $k <= 6; $k++) {
    print ("<TR BGCOLOR=WHITE>");
```

В календаре будет 6 строк, поэтому цикл исполнит этот раздел кода 6 раз.

14. Создайте второй цикл, который будет печатать колонки календаря (дни).

```
for ($i = 1; $i <= 7; $i++) {
```

В календаре будет 7 колонок, одна на каждый день недели.

15. Увеличьте значение переменной `$StartDate` на единицу.

```
$StartDate++;
```

Значение переменной `$StartDate` печатает дату для каждого дня календаря. Поэтому любая итерация цикла увеличивает это значение.

16. Создайте условную конструкцию `for`, которая будет определять, когда печатать строку `$StartDate`.

```
if (($StartDate <= 0) || ($StartDate > $LastDay)) {
    print ("<TD BGCOLOR=GREEN>&nbsp;</TD>");
} elseif (($StartDate >= 1) && ($StartDate <= $LastDay)) {
    print ("<TD ALIGN=CENTER>$StartDate</TD>");
}
```

Если значение переменной `$StartDate` меньше нуля (другими словами, месяц еще не начался) или больше количества дней в месяце (`$LastDate`), на экране монитора будет отображена пустая зеленая область. В том случае если значение переменной `$StartDate` больше нуля, но меньше количества дней в месяце (`$LastDate`), будет напечатана дата.

17. Закройте второй цикл, строку таблицы и первый цикл.

```
}
print ("</TR>\n");
}
```

18. Закончите таблицу.

```
print ("</TABLE>\n");
```

19. А теперь сделаем простую HTML-форму, которая передает страницу обратно в себя.

```
print ("<FORM ACTION=\"calendar.php\" METHOD=GET>\n");
```

20. Создайте два ниспадающих меню: одно для месяца, другое для года.

```
print ("Select a new month to view:\n");
print ("<SELECT NAME=Month>
-<OPTION VALUE=1>January</OPTION>\n
-<OPTION VALUE=2>February</OPTION>\n
-<OPTION VALUE=3>March</OPTION>\n
-<OPTION VALUE=4>April</OPTION>\n
-<OPTION VALUE=5>May</OPTION>\n
-<OPTION VALUE=6>June</OPTION>\n
-<OPTION VALUE=7>July</OPTION>\n
-<OPTION VALUE=8>August</OPTION>\n
-<OPTION VALUE=9>September</OPTION>\n
-<OPTION VALUE=10>October</OPTION>\n
-<OPTION VALUE=11>November</OPTION>\n
-<OPTION VALUE=12>December</OPTION>\n</SELECT>\n");
print ("<SELECT NAME=Year>
-<OPTION VALUE=2001>2001</OPTION>\n
-<OPTION VALUE=2002>2002</OPTION>\n
-<OPTION VALUE=2003>2003</OPTION>\n
</SELECT>\n");
```

Вы можете задать здесь любые годы. Обратите внимание, что два представленных меню сгенерируют переменные `$Month` и `$Year`, как только форма

ГЛАВА 13 ▼ Создание Web-приложений

будет передана. И с этого времени благодаря условной конструкции, начинающейся со строки 5, новые значения будут использоваться вместо значений по умолчанию.

21. Создайте кнопку **Submit** и закройте форму.

```
print { "<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Submit!\">\n");  
print { "</FORM>\n");
```

22. Включите файл нижнего колонтитула и закройте тэг PHP.

```
require ("footer.php");  
?>
```

23. Сохраните сценарий как **calendar.php** (листинг 13.5), загрузите его на сервер и протестируйте в браузере (рис. 13.6 и 13.7).

Листинг 13.5 т В этом достаточно длинном сценарии для изображения ка-

236

ГЛАВА 13 ▼ Создание Web-приложений



Рис. 13.7▼ Можно посмотреть любой месяцлюбого года – календарь всегда точен



Вместо определения последнего дня месяца, как мы сделали на строке 20 (обратившись к нулевой секунде первого дня следующего месяца), можно использовать формат `date ("t ") ;`, который возвращает количество дней в месяце. Для использования этого нового значения придется слегка изменить сценарий.

Использование HTTP-заголовков

Заголовок HTTP (протокола передачи гипертекста) используется для передачи информации между сервером и клиентом (Web-браузером). Обычно эта информация существует в форме HTML, вот почему адреса Web-страниц начинаются с записи `http://`.

HTTP-заголовки – это достаточно сложная тема, которая заслуживает отдельного рассмотрения. HTTP-заголовки могут использоваться в разных целях. В языке PHP доступ к деталям протокола HTTP выполняется через специаль-

```

26     if (($StartDate <= 0) || ($StartDate > $LastDay)) {
27         print ("<TD BGCOLOR=GREEN>&nbsp;</TD>");
28     } elseif (($StartDate >= 1) && ($StartDate <= $LastDay)) {
29         print ("<TD ALIGN=CENTER>$StartDate</TD>");
30     }
31 }
32 print ("</TR>\n");
33 }
34 print ("</TABLE>\n");
35 // Создание формы.
36 print ("<FORM ACTION=\"calendar.php\" METHOD=GET>\n");
37 print ("Select a new month to view:\n");
38 print ("<SELECT NAME=Month>
    <OPTION VALUE=1>January</OPTION>\n
    <OPTION VALUE=2>February</OPTION>\n
    <OPTION VALUE=3>March</OPTION>\n
    <OPTION VALUE=4>April</OPTION>\n
    <OPTION VALUE=5>May</OPTION>\n
    <OPTION VALUE=6>June</OPTION>\n
    <OPTION VALUE=7>July</OPTION>\n
    <OPTION VALUE=8>August</OPTION>\n
    <OPTION VALUE=9>September</OPTION>\n
    <OPTION VALUE=10>October</OPTION>\n
    <OPTION VALUE=11>November</OPTION>\n
    <OPTION VALUE=12>December</OPTION>\n</SELECT>\n");
39 print ("<SELECT NAME=Year>
    <OPTION VALUE=2001>2001</OPTION>\n
    <OPTION VALUE=2002>2002</OPTION>\n
    <OPTION VALUE=2003>2003</OPTION>\n
    </SELECT>\n");
40 print ("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Submit!\">\n");
41 print ("</FORM>\n");
42 require ("footer.php");
43 ?>

```



Рис. 13.6 При первом посещении страницы вы видите календарь текущего месяца. С помощью кнопок, расположенных под календарем, можно посмотреть и другие месяцы (рис. 13.7)



Рис. 13.7 ▼ Можно посмотреть любой месяц любого года – календарь всегда точен



Вместо определения последнего дня месяца, как мы сделали на строке 20 (обратившись к нулевой секунде первого дня следующего месяца), можно использовать формат `date("t")`; , который возвращает количество дней в месяце. Для использования этого нового значения придется слегка изменить сценарий.

Использование HTTP-заголовков

Заголовок HTTP (протокола передачи гипертекста) используется для передачи информации между сервером и клиентом (Web-браузером). Обычно эта информация существует в форме HTML, вот почему адреса Web-страниц начинаются с записи `http://`.

HTTP-заголовки - это достаточно сложная тема, которая заслуживает отдельного рассмотрения. HTTP-заголовки могут использоваться в разных целях. В языке PHP доступ к деталям протокола HTTP выполняется через специальную функцию `header()`. Одна из задач, выполняемых с помощью этой функции, - переадресация пользователя с одной страницы на другую. (Дополнительную информацию по этому вопросу вы можете получить по адресу <http://www.w3.org/Protocols/rfc2616/rfc2616>.)

Для переадресации пользователя применяется следующий код:

```
header("Location:page.php");
```

Функцию заголовка разрешается использовать и для отправки cookie-файлов в целях дублирования функции `setcookie()`, которая иногда выдает разные результаты в разных браузерах:

```
header("Set-cookie:name=value;expires=expiration");
```

Используя функцию `header()`, необходимо понимать следующее: она должна быть вызвана до того, как что-либо было отправлено в Web-браузер, так же как и в случае с функцией `setcookie()`.

Чтобы показать переадресацию, создадим простой сценарий, который будет направлять пользователя при аутентификации на одну страницу, если были введены правильные имя и пароль, и на другую - в противном случае.

Использование функции header

1. Создайте новый PHP-документ в текстовом редакторе:

```
<?php
```

2. Присвойте странице название и включите файл заголовка.

```
$PageTitle = "Login Page";
require ("header.php");
```

3. Создайте условную инструкцию, которая напечатает сообщение, если при аутентификации пользователь допустил ошибки.

```
if ($Message == "Invalid") {
    print ("<B><CENTER><FONT COLOR=RED>The username and password
    -you entered do not match what is on file. Please try again!</FONT>
    -</CENTER></B>\n");
}
```

Если переданные имя и пароль не соответствуют хранящимся в файле, пользователь будет отправлен обратно к этой странице со значением переменной \$Message, равным Invalid. Появится сообщение об ошибке.

4. Создайте HTML-форму для ввода имени пользователя и пароля.

```
print ("<FORM ACTION=\"HandleLogin.php\" METHOD=POST>\n");
print ("Username: <INPUT TYPE=TEXT NAME=UserName><BR>\n");
print ("Password: <INPUT TYPE=PASSWORD NAME=Password><BR>\n");
print ("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Submit!\">\n");
```

5. Включите файл нижнего колонтитула и закройте PHP-страницу.

```
require ("footer.php");
?>
```

6. Сохраните сценарий как login.php (листинг 13.6) и загрузите его на сервер.

Листинг 13.6 т Печатают сообщения об ошибке вполне разумно, так как вы наверняка захотите дать пользователю еще одну попытку, если подлинность его имени и пароля не установлена.

```
1  <?php
2  $PageTitle = "Login Page";
3  require ("header.php");
4  if ($Message == "Invalid") {
5      print ("<B><CENTER><FONT COLOR=RED>The username and password you
      entered do not match what is on file. Please try again!</FONT>
      </CENTER></B>\n");
6  }
7  print ("<FORM ACTION=\"HandleLogin.php\" METHOD=POST>\n");
8  print ("Username: <INPUT TYPE=TEXT NAME=UserName><BR>\n");
```

```
9 print ("Password: <INPUT TYPE=PASSWORD NAME=Password><BR>\n");
10 print ("<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Submit!\">\n");
11 require ("footer.php");
12 ?>
```

Теперь необходимо создать страницу, где будут проверяться введенные имя и пароль пользователя.

7. Создайте новый PHP-документ в текстовом редакторе:

```
<?php
```

8. Создайте условную конструкцию, которая будет проверять соответствие значений переменных `$UserName` и `$Password`.

```
if (($UserName == "Larry") && ($Password == "LarryPass")) {
```

Эта условная инструкция проверяет, соответствуют ли имя пользователя и пароль хранящимся в файле. Вы можете написать сценарий, **который** будет находить в **базе** пароль по имени пользователя. Обычно в целях безопасности эти значения явно не прописываются в сценарии, а извлекаются из базы данных или текстового файла. Однако в нашем примере мы проверяем эти значения прямо в сценарии.

9. Если введенные данные правильны, направим пользователя на главную страницу.

```
header ("Location: index.php?UserName=$UserName");
exit;
```

Это выражение **переправит** пользователя на ресурс `index.php`. Туда также будет отправлено значение имени пользователя.

Инструкция `exit;` «говорит» PHP прекратить выполнение кода этой страницы, так как пользователь уже отправлен на другую.

10. Закончим условную конструкцию и направим пользователя обратно на страницу аутентификации в случае неверного пароля.

```
} else {
    header ("Location: login.php?Message=Invalid");
    exit;
}
```

Если переданные значения не соответствуют хранящимся в файле, пользователю будет предоставлена еще одна возможность зарегистрироваться. Он будет отправлен на страницу `login.php`. Код `?Message=Invalid`, добавленный к URL, заставит сценарий `login.php` напечатать сообщение об ошибке (см. также листинг 13.6).

11. Закройте PHP-страницу.

```
?>
```

12. Сохраните сценарий как `HandleLogin.php` (листинг 13.7) и загрузите его на сервер.

Листинг 13.7 ▼ Этот сценарий будет проверять подлинность имени пользователя и пароля с помощью заранее определенных значений и перенаправлять пользователя соответствующим образом. Вне тэга PHP в этом сценарии не должно быть лишних пробелов, иначе функция `header()` выдаст ошибку.

```

1  <?php
2  if (($UserName == "Larry") && ($Password == "LarryPass")) {
3      header ("Location: index.php?UserName=$UserName");
4      exit;
5  } else {
6      header ("Location: login.php?Message=Invalid");
7      exit;
8  }
9  ?>

```

Модифицируем исходную страницу `index.php`, чтобы на экране появлялось приветственное сообщение.

13. Откройте файл `index.php` в текстовом редакторе (листинг 13.3).

14. Замените строку 4 на следующую:

```
print ("Greetings,$UserName!\n");
```

15. Сохраните сценарий как `index.php` (листинг 13.8), загрузите его на сервер и протестируйте все страницы начиная с `login.php` в Web-браузере (рис. 13.8-13.10).

Листинг 13.8 Мы добавили персонифицированное приветствие, что сделало страницу более динамичной. Значение переменной `$UserName` передается из функции `header()` (листинг 13.7).

```

1  <?php
2  $PageTitle = "Home Page";
3  require ("header.php");
4  print ("Greetings, $UserName!\n");
5  print ("<P>Click <A HREF=\"page2.php\">here</A> to go to the next
6  page.\n");
7  require ("footer.php");
8  ?>

```

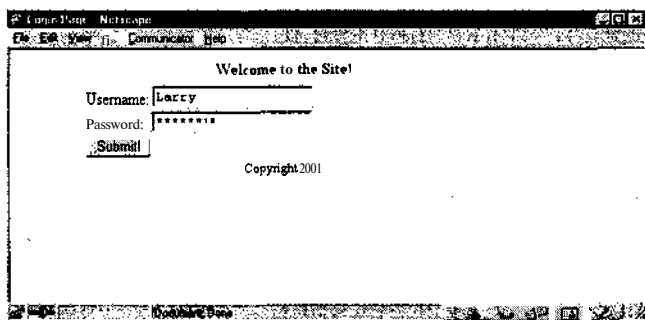


Рис. 13.8 Это простая страница аутентификации для ввода имени пользователя и пароля

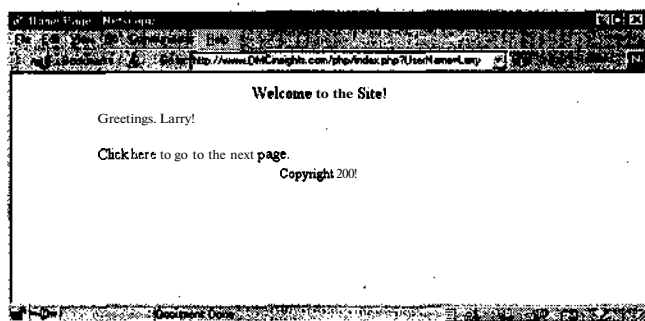


Рис. 13.9 ↑ После успешной аутентификации пользователь направляется на страницу `index.php`, где его приветствуют по имени

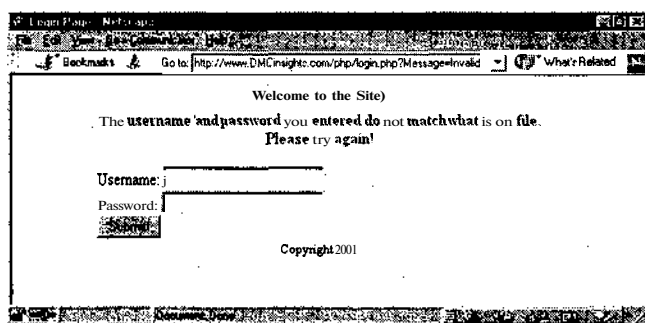


Рис. 13.10 ▼ Если пользователь ввел недействительное имя или пароль, он будет направлен обратно на страницу `login.php` и увидит такое сообщение

Отправка электронной почты

С помощью PHP можно легко отправлять электронную почту:

```
mail ("mailto", "subject", "body");
```

Для отправки сообщений функцией `mail()` используется серверное приложение электронной почты, такое как `sendmail` в среде UNIX. Эта функция может принимать еще один аргумент, который допустимо использовать для добавления дополнительных параметров к сообщению, включая адрес отправителя, приоритет доставки, адреса отправки копии и т.д.

```
mail ("mailto", "subject", "body", "From: fromaddress");
```

Отправка электронной почты с помощью PHP

1. Создайте новый PHP-документ в текстовом редакторе.

```
<?php
```

2. Присвойте странице имя и включите файл заголовка.

```
$PageTitle = "Sending Emails";
require ("header.php");
```

3. Создайте условную конструкцию для обработки переданной формы.

```
if ($BeenSubmitted) {
```

4. Если был передан адрес электронной почты получателя, отправьте сообщение.

```
    if ($MailTo) {
        if (mail($MailTo, $Subject, $Body, "From: $MailFrom")) {
            print ("<B><CENTER><FONT COLOR=BLUE>Your email has been
                -successfully sent!</FONT></CENTER></B>\n");
        } else {
            print ("<B><CENTER><FONT COLOR=RED>Your email was not
                -successfully sent due to a system error!</FONT></CENTER>
                -</B>\n*");
        }
    }
```

Мы поместили функцию mail () в условную конструкцию, чтобы пользователь получил сообщение, если почта успешно отправлена.

5. Закончите условные конструкции.

```
    } else {
        print ("<B><CENTER><FONT COLOR=RED>Please enter the recipient's
            -mail to address!</FONT></CENTER></B>\n");
    }
}
```

6. Закройте PHP-раздел и создайте HTML-форму, в которую вводятся два адреса электронной почты, тема, само письмо и скрытое значение для определения того, была ли форма передана.

```
?>
<FORM ACTION="email.php" METHOD=POST> Recipient's Email Address:
-><INPUT TYPE=TEXT NAME="MailTo" SIZE="50"><BR>
Your Email Address: <INPUT TYPE=TEXT NAME="MailFrom" SIZE="50"xBR>
Email Subject: <INPUT TYPE=TEXT NAME="Subject" SIZE="80"><BR>
Email Body:<TEXTAREA NAME="Body" ROWS="10" COLS="50">
</TEXTAREAXP>
<INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
```

7. Закройте форму и включите файл нижнего колонтитула.

```
</FORM>
<?php
require ("footer.php");
?>
```

8. Сохраните сценарий как `email.php` (листинг 13.9), загрузите его на сервер и протестируйте в браузере (рис. 13.11-13.14).

Листинг 13.9 в PHP для отправки электронной почты предназначена только одна функция - `mail()`. Мы поместили вызов функции `mail()` внутри различных условных конструкций, чтобы почта отправлялась только после указания адреса получателя. .

```

1  <?php
2  $PageTitle = "Sending Emails";
3  require ("header.php");
4  if ($BeenSubmitted) {
5      if ($MailTo) {
6          if (mail($MailTo, $Subject, $Body, "From: $MailFrom")) {
7              print ("<B><CENTER><FONT COLOR=BLUE>Your email has
              been successfully sent!</FONT></CENTER></B>\n");
8          } else {
9              print ("<B><CENTER><FONT COLOR=RED>Your email was not
              successfully sent due to a system error!</FONT></CENTER>
              </B>\n");
10         }
11     } else {
12         print ("<B><CENTER><FONT COLOR=RED>Please enter the recipient's
         mail to address!</FONT></CENTER></B>\n");
13     }
14 }
15 ?>
16 <FORM ACTION="email.php" METHOD=POST>
17 Recipient's Email Address: <INPUT TYPE=TEXT NAME="MailTo"
    SIZE="50"><BR>
18 Your Email Address: <INPUT TYPE=TEXT NAME="MailFrom" SIZE="50"><BR>
19 Email Subject: <INPUT TYPE=TEXT NAME="Subject" SIZE="80"><BR>
20 Email Body: <TEXTAREA NAME="Body" ROWS="10" COLS="50">
21 </TEXTAREA>
22 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
23 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
24 </FORM>
25 <?php
26 require ("footer.php");
27 ?>

```



Можно отправлять сообщения с прикрепленными файлами, но это требует более сложного программирования (с использованием объектов). К счастью, уже разработано много работающих решений, которые доступны для использования. См. также приложение С.

Welcome to the Site!

Recipient's Email Address:
php@DRCinsights.com

Your Email Address:
person@address.com

Email Subject
Testing PHP's Email Capabilities

Email Body:
This is the body of the email which will be sent.
The bod? can go over multiple lines!

Submit

Copyright 2001

Рис. 13.11 Т С ПОМОЩЬЮ PHP эта простая HTML-форма позволяет вам отправлять почту из браузера

Welcome to the Site!

Your e-mail has been successfully sent!

Recipient's Email Address:
|

Your Email Address:
|

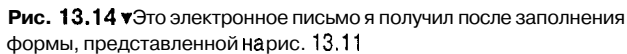
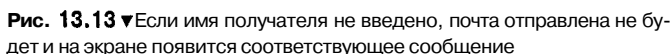
Email Subject
|

Email Body:
|

Submit

Copyright 2001

Рис. 13.12 Т Если почта была отправлена, на экран будет выведено сообщение об этом



14 Глава

Отладка сценариев

В этой главе говорится о некоторых полезных приемах при отладке. К сожалению, каким бы опытным ни был программист, нечаянно сделанная ошибка проявляется, когда ее меньше всего ждешь, поэтому к таким случаям нужно быть всегда готовым.

В данной главе освещаются следующие вопросы: минимизация количества ошибок в коде, сообщения об ошибках, устранение проблем. Хотя очень важно избегать ошибок еще на этапе разработки, еще важнее знать, как поступать с ошибками, которые неизбежно возникают при эксплуатации Web-сайтов.

Распространенные ошибки

Самый распространенный тип ошибок – синтаксические, когда вы забываете, так сказать, расставить все точки над «i». В результате на экране монитора появляются сообщения, подобные тем, что представлены на рис. 14.1. Первым делом в РНР выявляются ошибки именно этого типа, так как перед выполнением кода обязательно проверяется его синтаксис. Чтобы избежать при программировании подобного рода ошибок, необходимо соблюдать следующие правила:

- заканчивать каждую исполняемую строку кода точкой с запятой;
- закрывать кавычки, а также круглые, квадратные и фигурные скобки;
- экранировать с помощью обратного слеша все одинарные и двойные кавычки внутри функции `print()`.

Очень часто РНР и текстовый редактор по-разному интерпретируют строки. Так, в сообщении может содержаться информация о том, что обнаружена

ошибка, например в строке 12, а это не соответствует действительности. Воспринимайте указываемое PHP место ошибки как отправную точку ее поиска.

Ошибки возникают **также**, если вы пытаетесь выполнить некоторое невозможное действие. Эти ошибки появляются, например, тогда, когда функции `setcookie()` или `header()` вызываются после того, как код HTML уже был отправлен в браузер, когда функция вызывается без соответствующих аргументов или когда вы пытаетесь записать данные в файл, не имея на это полномочий. Также ошибки выявляются при попытке исполнить код (рис. 14.2).

Достаточно распространены и логические ошибки - те, что допускает сам программист. Одна из причин подобных проблем - использование некорректного имени **переменной**. Если это происходит, вы получите не сообщение об ошибке (рис. 14.1 и 14.2), а странные или непредсказуемые результаты. Устранять логические ошибки труднее всего. Только тщательная проверка и четкий анализ ситуации помогут в таких случаях.

Чтобы избежать отправки заголовка после того, как браузер уже получил HTML-код или пустую строку, добавим проверку в сценарий `HandleLogin.php` (речь о нем шла в главе 13).

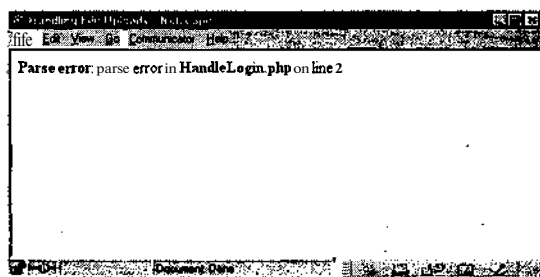


Рис. 14.1 т Наверное, в ваши сценарии уже неоднократно прокрадывались синтаксические ошибки

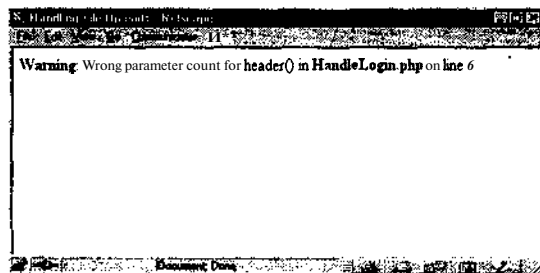


Рис. 14.2 т На экране появляется соответствующее сообщение, если при выполнении сценария возникают ошибки, вызванные некорректным использованием функций (например, неверное написание имени **функции** или использование неправильного количества аргументов при вызове функции)

Предотвращение распространенных ошибок

1. Откройте файл `HandleLogin.php` в текстовом редакторе (листинг 14.1).

Листинг 14.1 т Исходная страница `HandleLogin.php` выглядит неплохо, но она будет еще **лучше**, если использовать функцию `headers_sent()`.

```
1  <?php
2  if (($UserName == "Larry") && ($Password == "LarryPass")) {
3      header ("Location: index.php?UserName=$UserName");
4      exit;
5  } else {
6      header ("Location: login.php?Message=Invalid");
7      exit;
8  }
9  ?>
```

2. Создайте пустую строку до открывающего PHP-тэга. Она сгенерирует сообщение об ошибке (рис. 14.3).

3. После открывающего PHP-тэга добавьте условную конструкцию.

```
if ( headers_sent() ) {
    print ("Cannot process your request due to a system error!\n");
} else {
```

Функция `headers_sent()` возвращает истинное значение, если какой-либо фрагмент кода HTML или пустая строка уже были отправлены в браузер. Если бы это произошло, попытка использовать функцию `header()` инициировала бы отправку в браузер предупреждения пользователю. Вместо этого, если заголовок будет послан, на данной странице будет представлена типовая системная ошибка.

Если заголовок не отправлялся, функция `headers_sent()` возвратит ложное значение, и оставшийся код страницы будет выполнен.

4. Не забудьте закрыть условную конструкцию до закрывающего PHP-тэга!

```
}
```

5. Сохраните сценарий как `HandleLogin.php` (листинг 14.2), загрузите его на сервер в один каталог с `login.php` и протестируйте обе страницы в браузере (рис. 14.4).

Листинг 14.2 т Функция `headers_sent()` помогает избежать распространенной ошибки отправки заголовка или cookie-файла после того, как на браузер уже пришла информация (рис. 14.3).

```
1
2  <?php
3  if ( headers_sent() ) {
4      print ("Cannot process your request due to a system error!\n");
5  } else {
6      if (($UserName == "Larry") && ($Password == "LarryPass")) {
7          header ("Location: index.php?UserName=$UserName");
```

```

8         exit;
9     } else {
10        header ("Location: login.php?Message=Invalid");
11        exit;
12    }
13 }
14 ?>

```



Некоторые текстовые редакторы имеют утилиты, контролирующие количество открывающих и закрывающих скобок и кавычек.

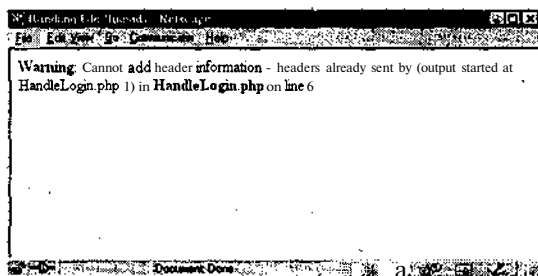


Рис. 14.3 т Попытка отправить cookie-файл или заголовок после того, как Web-браузер уже получил какую-либо информацию, вызывает такую ошибку

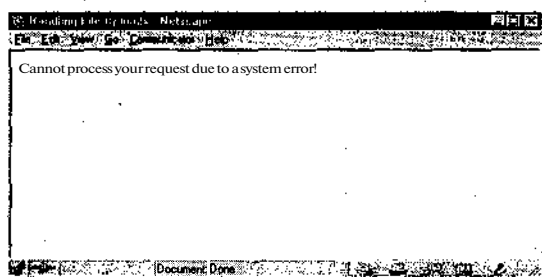


Рис. 14.4 т Используя функцию `headers_sent()`, можно выводить на экран менее озадачивающие пользователя сообщения. Тем самым проблема не решается, но это все же лучше, чем посылать пользователю сообщения, подобные представленным на рис. 14.3

Сообщения о возможных ошибках и их протоколирование

В языке PHP есть хорошая встроенная поддержка для генерации сообщений об ошибках и их обработки. Вы сможете многому научиться с помощью этих средств.

Функция `error_reporting()` указывает, о каких типах ошибок PHP должен информировать. Строка `error_reporting(0)` полностью отключает составление сообщений об ошибках. Ошибки все равно будут, просто вы о них больше не узнаете. И наоборот, строка `error_reporting(E_ALL)` будет выдавать сообщения обо всех произошедших ошибках. (В руководстве по PHP, а также в таблице С.6 (приложение С) представлена информация об уровнях, которые можно задать в отношении сообщений об ошибках.)

Функция `error_reporting()` «говорит» о том, какие ошибки должны сопровождаться сообщениями, а функция `error_log()` «инструктирует» PHP, как протолировать ошибки:

```
error_log("message", "type", "destination");
```

При возникновении проблемы ошибка может быть автоматически записана в файл протокола, либо же вы получите соответствующее сообщение по электронной почте. Разумное использование функции `error_log()` позволяет Web-мастерам и программистам постоянно знать, что происходит на их Web-сайтах. Чтобы в случае ошибки вы получали информацию по электронной почте, добавьте в ваш код следующую строку:

```
error_log("message", "l", "php@DMCinsights.com");
```

Изменим сценарий `email.php` (глава 13) так, чтобы, если не удалось отправить информацию по электронной почте, осуществлялась запись в файл.

Использование функции `error_log()`

1. Откройте файл `email.php` в текстовом редакторе (листинг 14.3).

Листинг 14.3 Это сценарий `email.php`, взятый из главы 13. В настоящее время он не имеет функции протолирования ошибок, которая сделала бы его более полезным.

```
1  <?php
2  $PageTitle = "Sending Emails";
3  require ("header.php");
4  if ($BeenSubmitted) {
5      if ($MailTo) {
6          if (mail($MailTo, $Subject, $Body, "From: $MailFrom")) {
7              print ("<B><CENTER><FONT COLOR=BLUE>Your email has been
8                  successfully sent!</FONT></CENTER></B>\n");
9          } else {
10             print ("<B><CENTER><FONT COLOR=RED>Your email was not
11                 successfully sent due to a system error!</FONT></CENTER>
12                 </B>\n");
13         }
14     } else {
15         print ("<B><CENTER><FONT COLOR=RED>Please enter the recipient's
16             mail to address!</FONT></CENTER></B>\n");
17     }
18 }
```

```

15 ?>
16 <FORM ACTION="email.php" METHOD=POST>
17 Recipient's Email Address: <INPUT TYPE=TEXT NAME="MailTo"
    SIZE="50"><BR>
18 Your Email Address: <INPUT TYPE=TEXT NAME="MailFrom" SIZE="50"><BR>
19 Email Subject: <INPUT TYPE=TEXT NAME="Subject" SIZE="80"×BR>
20 Email Body:<TEXTAREA NAME="Body" ROWS="10" COLS="50">
21 </TEXTAREAxP>
22 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
23 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
24 </FORM>
25 <?php
26 require ("footer.php");
27 ?>

```

2. После строки 9 добавьте запись, данную ниже.

```

error_log ("Unable to send an email to $MailTo from
$MailFrom at " . time(). "\n", 3, "errors.txt");

```

Каждый раз, когда функция `mail()` не может быть выполнена, в файл журнала записывается простое сообщение. Оно указывает, что `$MailFrom` не может послать сообщение `$MailTo` в определенное время (заданное с помощью функции `time()`). Опытный администратор может просмотреть этот файл и либо отследить пользователей, у которых возникли проблемы, либо определить, когда функция `mail()` перестала работать корректно.

3. Сохраните сценарий как `email.php` (листинг 14.4) и загрузите его на Web-сервер.

Создадим пустой документ `errors.txt`, который будет исполнять роль протокола ошибок.

4. Создайте пустой документ в текстовом редакторе.

5. Сохраните документ как `errors.txt` и загрузите его на сервер в один каталог с `email.php`.

6. Задайте такие права доступа к файлу `errors.txt`, чтобы все категории пользователей могли записывать данные в этот файл.

7. Протестируйте `email.php` в браузере (рис. 14.5-14.7).

Листинг 14.4 т В модифицированной версии сценария `email.php` функция `error_log()` используется для протоколирования любых проблем, возникающих при отправке электронной почты. Допустимо использовать функцию `error_log()`, чтобы получать сообщения об ошибке по электронной почте. Однако это становится невозможным, если отправить почту трудно или не удается.

```

' 1 <?php
2 $PageTitle = "Sending Emails";
3 require ("header.php");
4 if ($BeenSubmitted) {
5     if ($MailTo) {
6         if (mail($MailTo, $Subject, $Body, "From: $MailFrom")) {

```

```

7      print ("<B><CENTER><FONT COLOR=BLUE>Your email has been
      successfully sent!</FONT></CENTER></B>\n");
8  } else {
9      print ("<B><CENTER><FONT COLOR=RED>Your email was not
      successfully sent due to a system error!</FONT></CENTER>
      </B>\n");
10     error_log ("Unable to send an email to $MailTo from
      $MailFrom at " . time(). "\n", 3, "errors.txt");
11 }
12 } else
13     print ("<B><CENTER><FONT COLOR=RED>Please enter the recipient's
      mail to address!</FONT></CENTER></B>\n");
14 }
15 }
16 ?>
17 <FORM ACTION="email.php" METHOD=POST>
18 Recipient's Email Address: <INPUT TYPE=TEXT NAME="MailTo"
      SIZE="50"×BR>
19 Your Email Address: <INPUT TYPE=TEXT NAME="MailFrom" SIZE="50"×<BR>
20 Email Subject: <INPUT TYPE=TEXT NAME="Subject" SIZE="80"×<BR>
21 Email Body:<TEXTAREA NAME="Body" ROWS="10" COLS="50">
22 </TEXTAREAxP>
23 <INPUT TYPE=HIDDEN NAME=BeenSubmitted VALUE=TRUE>
24 <INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit!">
25 </FORM>
26 <?php
27 require ("footer.php");
28 ?>

```

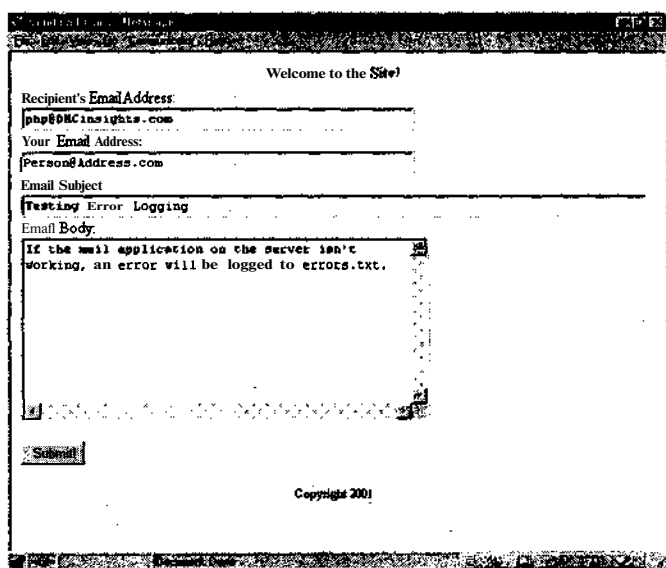


Рис. 14.5т Страница email.php изменена так, чтобы в файл записывались любые проблемы. Пользователь этого не видит

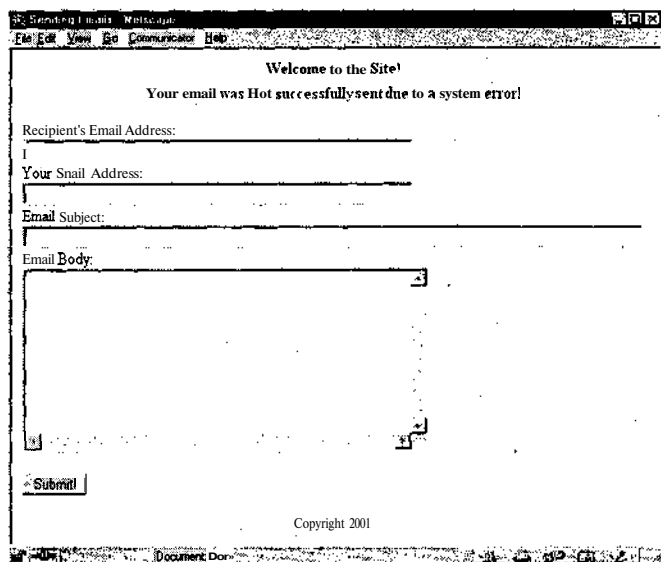


Рис. 14.6 ▼ Все, что видит пользователь при возникновении проблемы, – простое сообщение над формой. Системный администратор может получить более подробную информацию из файла протокола ошибок (рис. 14.7)

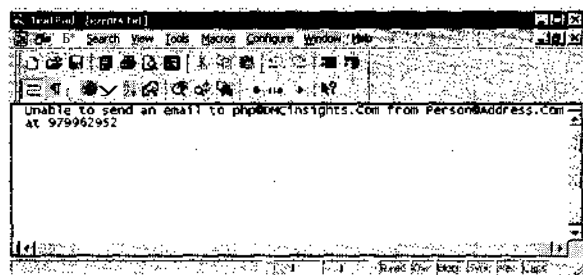


Рис. 14.7 ↑ При создании файла протокола ошибок вы можете задать параметры того, какую информацию хотите в нем видеть. Этот файл записывает значения переменных \$MailTo и \$MailFrom, а также временную метку



Если прикладная программа электронной почты на сервере работает корректно, этот сценарий ничего не запишет в файл протокола. Если вы хотите посмотреть, что делает сценарий, если не сможет отправить сообщение, измените строку 6 на следующую:

```
if (!mail($MailTo,$Subject,$Body,"From:$MailFrom")){
```

Отслеживание ошибок

Труднее всего обнаружить ошибки в логике программы. В этом случае не будет подсказки, с какой строки начинать поиск. При возникновении такого рода

ошибок вы просто видите, что полученные результаты отличаются от ожидаемых. Чтобы найти, какие ошибки были сделаны и где, необходимо провести небольшое «детективное расследование».

В этом вам помогут три приема:

- использование комментариев;
- применение инструкции `print ()`;
- отслеживание переменных.

Комментарии можно использовать не только для документации ваших сценариев, но и для исключения проблематичных строк кода. Если вы видите сообщение о том, что ошибка обнаружена в строке 12, прокомментируйте эту строку. Если в результате этого действия ошибка не исчезла, значит, она в другом месте.

В более сложных сценариях я часто использую инструкцию `print ()`, чтобы на экран выводились сообщения о том, что происходит во время исполнения сценария. Если сценарий состоит из нескольких шагов, порой нелегко определить, на каком из них возникла проблема. С помощью инструкции `print ()` вы можете сузить поиск до одного шага.

Часто причиной некорректной работы сценария является обращение к несуществующей переменной, или к существующей, но с ошибочным именем. Проверить это можно с помощью инструкции `print ()`, которая позволяет отслеживать значения переменных в процессе выполнения сценария. В этом случае вы будете знать наверняка, какие именно переменные вызвали проблему.

Эти приемы я покажу, модифицировав страницу `HandleLogin.php` (см. начало данной главы) и показав, как отладить сценарий, если получены непредсказуемые результаты.

Отладка сценария

1. Откройте сценарий `HandleLogin.php` в текстовом редакторе (см. также листинг 14.2).
2. Удалите первую пустую строку. Теперь на первой строке будет размещен открывающий PHP-тэг.

```
• <?php
```

3. После условной конструкции `headers_sent ()` (строка 2) добавьте инструкцию `print ()`.

```
if ( headers_sent() ) {  
    print ("Headers have been sent. Not attempting to verify.<P>\n");  
    print ("Cannot process your request due to a system error!\n");
```

Если проблема возникла из-за того, что заголовки были уже отправлены, это будет ясно при тестировании.

4. После конструкции `else` (строка 5) добавьте еще три инструкции `print ()`.

```
} else {  
    print ("Headers have not been sent. Attempting to verify.<P>\n");
```

```
print ("UserName is $UserName. <P>");
print ("Password is $Password. <P>");
```

Первая **инструкция** `print ()` связана с расположенной выше *и* указывает, что проблема не зависит от функции `header ()`.

Две последующие инструкции `print ()` указывают, какие значения были получены для переменных `$UserName` и `$Password`. Вы имеете возможность **отследить** значения переменных по всему сценарию, распечатывая их в ключевых моментах.

- После второй условной конструкции `if` (строка 9) добавьте еще одну инструкцию `print` `O` и закомментируйте строку с функцией `header ()`.

```
if (($UserName == "Larry") && ($Password == "LarryPass")) {
    print ("<P>Match!");
    // header ("Location: index.php?UserName=$UserName");
    exit;
} else {
```

Напечатав слово "Match!", вы будете знать, проверены ли имя пользователя и пароль, что позволит исключить возможную причину ошибки.

Так как вы отлаживаете сценарий, то можете закомментировать вызов функции `header ()`, поставив в начале строки два обратных следа. Закончив отладку, удалите эти знаки, и команда снова будет работать.

- После строки с `else` (строка 13) добавьте еще одну инструкцию `print ()`, после чего закомментируйте строку с функцией `header ()`.

```
    print ("<P>Not a Match!");
    // header ("Location: login.php?Message=Invalid");
    exit;
}
?>
```

- Сохраните сценарий как **HandleLogin.php** (листинг 14.3), загрузите его на сервер в один каталог с **login.php** и протестируйте обе страницы в браузере (рис. 14.8-14.10).

Листинг 14.5 † Можно модифицировать страницу **HandleLogin.php** для отладки, если она странно работает и выдает непредсказуемые результаты. Использование инструкций `print()`, отслеживание переменных и подробное комментирование строк кода поможет быстро решить любую проблему.

```
1 <?php
2 if ( headers_sent() ) {
3     print ("Headers have been sent. Not attempting to verify.<P>\n");
4     print ("Cannot process your request due to a system error!\n");
5 } else {
```

```
6   print ("Headers have not been sent. Attempting to verify.<P>\n");
7   print ("UserName is $UserName. <P>");
8   print ("Password is $Password. <P>");
9   if (($UserName == "Larry") && ($Password == "LarryPass")) {
10      print ("<P>Match!");
11  // header ("Location: index.php?UserName=$UserName");
12      exit;
13  } else {
14      print ("<P>Not a Match!");
15  // header ("Location: login.php?Message=Invalid");
16      exit;
17  }
18 }
19 ?>
```

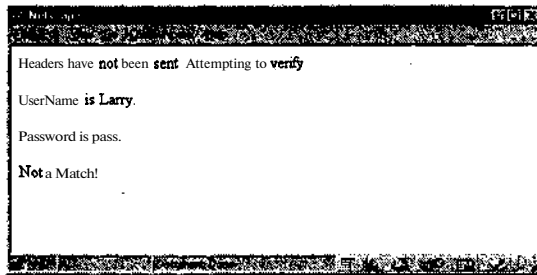


Рис. 14.8т При отладке проблемного сценария невероятно значима возможность печати сообщений с подробным описанием выполнения программы. Здесь вы можете определить, что проблема вызвана не заголовками или отсылкой к неправильным переменным

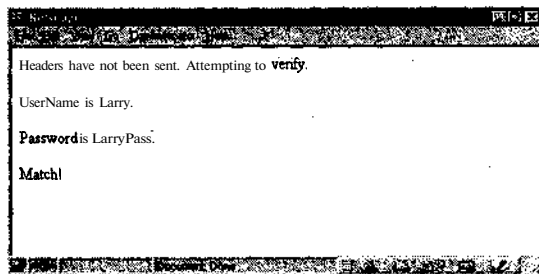


Рис. 14.9т Все события, которые происходят на странице `HandleLogin.php`, подробно описаны. Если возникают какие-либо проблемы, их поиск сужается до строк функции `header()`

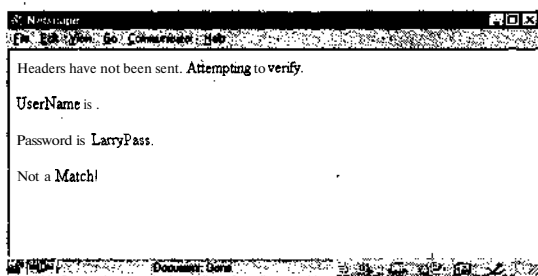


Рис. 14.10 — Вполне обычная картина. Причина несоответствия — обращение к той переменной. Обратите внимание на то, что переменная `UserName` не имеет значения. Из этого вытекает следующее утверждение: существует несоответствие между тем, что используется сценариями `login.php` и `HandleLogin.php` для обращения к этому значению

Использование инструкции `die`

Последний тип ошибок — всевозможные нештатные ситуации. Они возникают, когда какое-либо событие на сервере происходит не так, как обычно. К счастью, нештатные ситуации более редки, чем ошибки программирования. Например, если база данных MySQL не работает, то все связанные с ней функции генерируют ошибки. Или; как мы видели в примере со страницей `email.php`, если серверное приложение электронной почты не функционирует (на сервере UNIX), это тоже создаст проблему. Хотя системные ошибки не зависят от программиста, их необходимо иметь в виду при написании кода.

Дальнейшее распространение ошибок можно остановить с помощью инструкции `die`. Эта инструкция заставляет PHP прекратить исполнение сценария, как только происходит первая ошибка и дальнейшая работа невозможна. Она также может отправить в браузер сообщение об ошибке или вызвать функцию. Например,

```
$Link=mysql_connect($Host,$User,$Password) or die
→("Couldn't connect to database.");
```

Если по какой-либо причине не удастся установить связь с базой данных, сценарий прекратит свое исполнение и напечатает сообщение «Couldn't connect to database».

```
$Link=mysql_connect($Host,$User,$Password) or die (mysql_error());
```

В этом примере, если код PHP не может установить связь с базой данных, инструкция `die` вызовет функцию `mysql_error()`. Эта функция напечатает реальную ошибку, сгенерированную базой данных MySQL.

Оба примера работают: если код PHP определит, что первая часть условной конструкции `or` (значение, возвращаемое функцией `mysql_connect`) имеет ложное значение, то будет инициирована вторая часть условной конструкции (инструкция `die`) с целью узнать значение всего логического оператора. Другими

словами, строка «говорит»: если установить связь с сервером БД невозможно, делай это. Если связь с базой устанавливается, значит, это то, что требовалось, и вторая часть логического оператора `or` не проверяется и не выполняется.

Добавим инструкцию `die` в базовый сценарий, который устанавливает связь с MySQL, а затем пытается выбрать базу данных.

Выполнение действия

1. Создайте новый PHP-документ в текстовом редакторе.
2. Начните со стандартного HTML-документа.

```
<HTML>
<HEAD>
<TITLE>Die!</TITLE></HEAD>
<BODY>
```

3. Откройте PHP-раздел и задайте переменные для доступа в базу данных.

```
<?php
// Установка значений переменных для доступа к базе данных.
$Host = "localhost";
$User = "username";
$Password = "password";
$DBName = "database";
```

4. Отключите отчеты об ошибках.

```
error_reporting(0);
```

Так как все ошибки будут обслуживаться инструкцией `die`, мы отключаем заданный по умолчанию метод информирования об ошибках во избежание избыточности.

5. Подсоединитесь к базе MySQL.

```
$Link = mysql_connect ($Host, $User, $Password) or die("Couldn't
connect to the database!");
```

Эта строка кода «говорит» PHP попытаться установить связь с MySQL. Если по какой-либо причине сделать этого не удастся, вы увидите сообщение «Couldn't connect to database», а исполнение сценария приостановится. Очень важно сразу же остановить исполнение сценария: если не удастся подсоединиться к базе данных, следующие две строки кода, зависящие именно от этой операции, также вызовут сообщения об ошибках. (В действительности сообщения об ошибке не появятся, так как мы отключили функцию `error_reporting`, но программа работать не будет).

6. Выберите базу данных.

```
mysql_select_db ($DBName, $Link) or die(mysql_error());
```

Если выбрать базу данных, определенную в аргументе `$DBName`, не удалось, то сценарий прекратит исполнение и будет распечатано сообщение об ошибке, сгенерированное MySQL.

7. Отключите связь с MySQL и закройте PHP-раздел.

```
mysql_close ($Link);
?>
```

8. Создайте простое сообщение и закройте **HTML-страницу**.

```
Testing the die statement!
</BODY>
</HTML>
```

Я добавил эту текстовую строку, чтобы страница имела какое-либо содержание. Страницу легко модифицировать, чтобы она отправляла в базу данных и запросы.

9. Сохраните сценарий как `die.php` (листинг 14.6), загрузите его на сервер и протестируйте в браузере (рис. 14.11-14.13).

Листинг 14.6 Для сценариев, состоящих из нескольких строк кода, каждая из которых зависит от успешного исполнения предыдущей строки, инструкция `die` бесценна для прекращения дальнейшего распространения ошибок и их быстрого обнаружения. В связи с тем что инструкция `die` обрабатывает ошибки, встроенная в PHP обработка ошибок была выключена (сравните рис. 14.11 и 14.13).

```
1- <HTML>
2 <HEAD>
3 <TITLE>Die!</TITLE></HEAD>
4 <BODY>
5 <?php
6 // Установка переменных для доступа к базе данных.
7 $Host = "localhost";
8 $User = "username";
9 $Password = "password";
10 $DBName = "database";
11 error_reporting(0);
12 $Link = mysql_connect ($Host, $User, $Password) or die("Couldn't
   connect to thedatabase!");
13 mysql_select_db ($DBName, $Link) or die(mysql_error());
14 mysql_close ($Link);
15 ?>
16 Testing the die statement!
17 </BODY>
18 </HTML>
```



Вы можете предотвратить создание сообщений об ошибках с помощью размещения символа коммерческое «а» перед вызовом любой функции. Символ `@` называют оператором управления ошибками, он подавляет любые сообщения об ошибках, вызванные функцией, перед которой стоит. Символ используется следующим образом:

```
$Link=@mysql_connect ($Host, $User, $Password);
```

Обратите внимание на отсутствие пробела между знаком `@` и именем функции, а также на то, что подавление сообщений не меняет результат, выдаваемый функцией. В отличие от инструкции `die` символ `@` не дает вам возможности распечатать сообщение об ошибке или вызвать функцию.

Инструкцию `die` можно использовать и для вызова ваших собственных функций. Например, если вы создали функцию для печати сообщений об ошибках в определенном формате, можно воспользоваться данной инструкцией:

```
$Link=@mysql_connect ($Host, $User, $Password) or  
-die(print_message("Couldn't connect."));
```

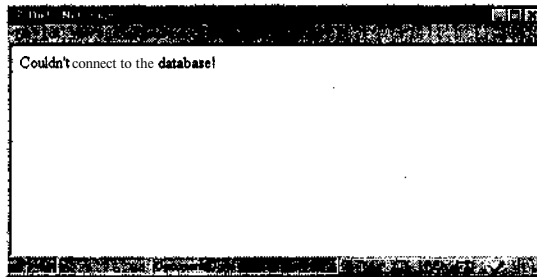


Рис. 14.11 Отключив сообщения об ошибках и используя инструкцию `die`, пользователь получает более вразумительные оповещения о проблемах. Сценарий также прекратит исполнение страницы, так как не завершен обязательный первый шаг

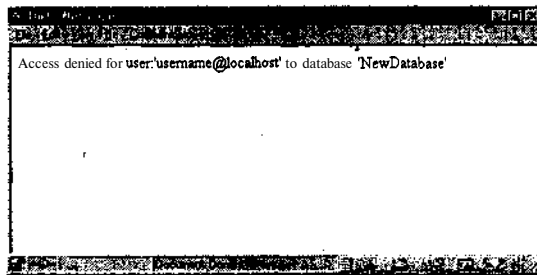


Рис. 14.12 Здесь инструкция `die` вызывает функцию `mysql_error()`, которая распечатывает сообщение об ошибке, сгенерированное базой данных MySQL

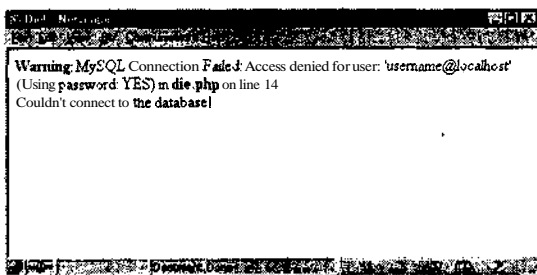


Рис. 14.13 Если сообщения об ошибках не отключены, конечный пользователь получит и сообщение PHP, и то, что сгенерировано инструкцией `die`. См. рис. 14.11, где отчеты об ошибках отключены

Приложение

Установка и конфигурация

Многим пользователям PHP, особенно тем, кто только изучает этот язык, никогда не придется устанавливать PHP на сервер и конфигурировать его. Впрочем, сделать это не очень сложно, тем более что рассматриваемый язык является свободно распространяемым. При этом вы сами выбираете версию PHP и устанавливаете именно те дополнительные библиотеки, которые нужны для вашего приложения.

Перед установкой необходимо ответить на два главных вопроса. Во-первых, под какой операционной системой будет работать сервер? Во-вторых, какое серверное приложение вы собираетесь использовать? В этом разделе говорится о том, как установить PHP на серверы UNIX (например, Linux) и Windows. Что касается самого Web-сервера, самое распространенное приложение для UNIX-серверов - Apache, так что, скорее всего, вы будете использовать его. На Windows-машинах иногда применяют сервер IIS.

Когда эти вопросы решены, необходимо продумать следующее: какие базы данных поддерживать, что использовать для создания графики, форматы PDF или Shockwave, как обеспечить совместимость со стандартом XML и т.д. Все эти вопросы должны быть решены до начала установки PHP.

В данной главе мы рассмотрим основы инсталляции на компьютер, работающий под Linux, а затем установку под Windows 2000. Изложенного материала вполне достаточно, если не возникнут какие-либо осложнения.

Установка на сервер Linux

Популярность операционной системы Linux, особенно при ее использовании в качестве Web-сервера, значительно возросла за последние несколько лет. Немаловажную роль в этом сыграла стабильность системы и тот факт, что она

распространяется свободно. Однако настроить операционную систему Linux не так просто, хотя и намного легче, чем раньше.

Для установки PHP на Linux-сервер необходимо иметь доступ с правами администратора. Однако сервер может и не стоять у вас на столе - подобную работу допустимо выполнить и дистанционно, если сервер Linux доступен по сети.

Установка Apache и PHP

1. Загрузите последние стабильные версии Apache и PHP в общий каталог, например /usr/local/ (рис. А1).

Apache можно найти на сайте <http://www.Apache.org>.

2. Если файлы имеют расширение .gz, распакуйте их с помощью этих строк (нажмите клавишу **Enter** после печати каждой строки).

```
gunzip php-4.0.0pl1.tar.gz
gunzip apache_1.3.14.tar.gz
```

Здесь указаны версии PHP и Apache, установленные для показа работы. Вы должны обязательно указывать те версии, которые устанавливаете (например, php-3.0.3pl1).

3. Распакуйте файлы.

```
tar -xvf php-4.0.4pl1.tar
tar -xvf apache_1.3.14.tar
```

Необходимо выполнить эти две команды, и вы получите исходные файлы, распакованные во множество каталогов. После исполнения каждой команды на экране будет появляться список файлов, извлекаемых из архива (рис. А.2).

4. Войдите в только что созданный каталог Apache и запустите программу конфигурации (рис. А.3).

```
cd ../apache_1.3.14
./configure-prefix=/www
```

Команда `cd` позволяет менять папки. Зайдя в каталог Apache, вы сможете сконфигурировать его.

5. Войдите в каталог PHP (рис. А.4) и запустите программу конфигурации.

```
cd ../php-4.0.4pl1
./configure-with-apache=../apache_1.3.14-enable-track-vars
```

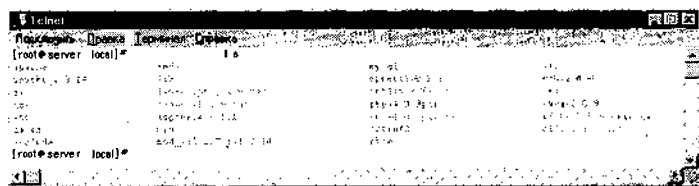


Рис. А.1 Команда `ls` выводит на экран все файлы текущего каталога. PHP и Apache находятся в этой папке вместе с другими библиотеками и приложениями

Рис. А.3 После установки и конфигурирования Apache появится такое сообщение. Оно информирует о том, что все работает правильно

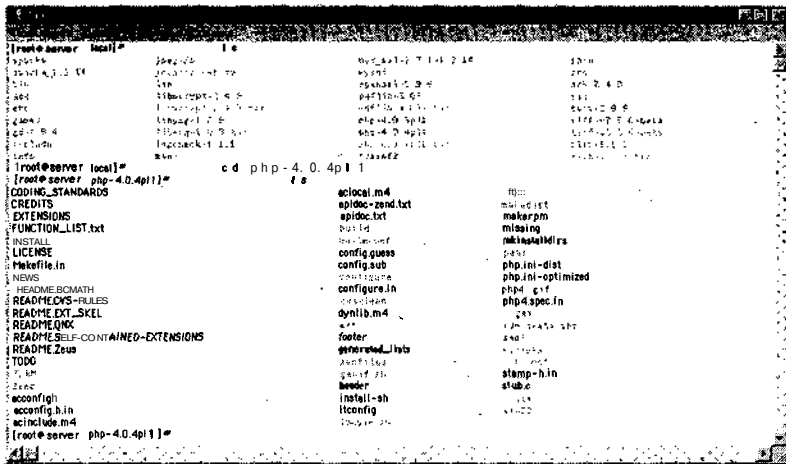


Рис. А.4 т Все распакованные файлы и каталоги теперь находятся в каталоге PHP

Конфигурационные параметры выбираются в зависимости от того, какие возможности PHP вы хотели бы использовать. Каждая конфигурационная команда требует указания на каталог, где можно найти конкретный элемент. Сейчас мы рассматриваем случай, когда PHP должен работать с Apache, и указываем, где находится папка Apache. Опция `-enable-track-vars` необходима для правильной работы PHP с HTML-формами.

6. Создайте исполняемый файл и установите РНР (рис. А.5).

```
make
make install
```

С помощью этих двух строк кода вы устанавливаете сконфигурированный PNR так, чтобы с ним мог работать сервер Apache.

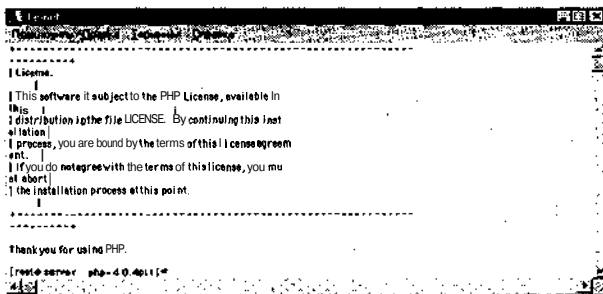


Рис. А.5 После успешной установки и конфигурирования РНР вы видите сообщение о лицензировании и стандартную благодарность за то, что выбрали этот язык

7. Вернитесь в каталог Apache, сконфигурируйте его, затем создайте исполняемый файл и установите его.

```
cd ../apache_1.3.14./configure-activate-module=rc/modules/php.4/libphp4.a
make
make install
```

Теперь, когда каталог Apache установлен и сконфигурирован, он вполне готов к работе.

8. Скопируйте файл `php.ini` в новый каталог.

```
cp/usr/local/php-4.0.4pl1/php.ini-dist /usr/local/lib/php.ini
```

Файл `php.ini` определяет работу PHP. В пакете он имеет имя `php.ini-dist`. Чтобы сервер Apache мог работать с этим файлом, его следует скопировать в соответствующий каталог и переименовать.

9. Запустите Apache,

```
bin/apachectl start
```

Эта строка кода в каталоге Apache запустит Web-сервер.

10. Проверьте, работают ли PHP и Apache, зайдя на нужный ресурс в браузере (рис. А.6).

Если сервер функционирует в режиме on-line, вы можете посетить сайт, например <http://www.DMCinsights.com>. Если машина работает в режиме off-line, используйте адрес <http://localhost/> или IP-адрес сервера.

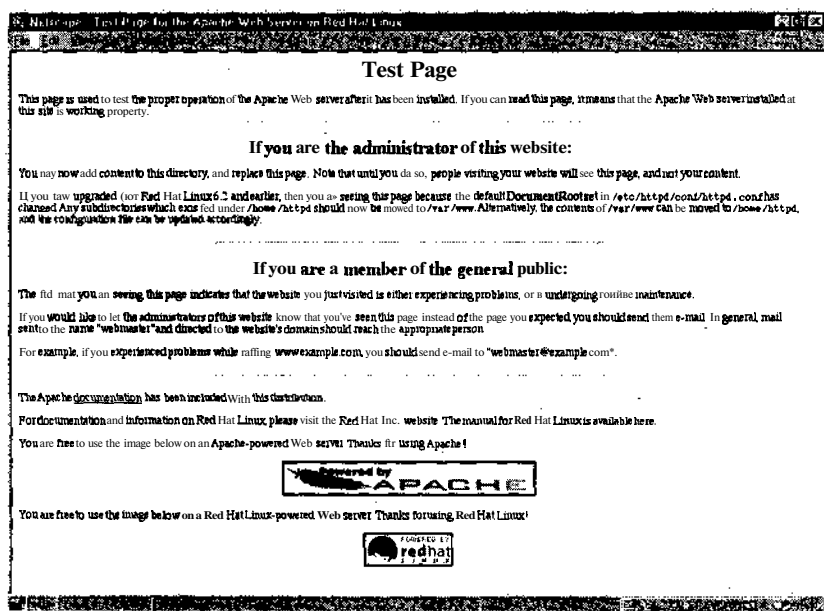


Рис. А.6 Эта страница используется по умолчанию для индикации того, что Apache работает корректно

Вы можете протестировать установку PHP с помощью файла `test.php` (см. главу 1).

Так как операционная система Linux может работать на устаревшем оборудовании (у меня есть одна Linux-машина с 16 Мб памяти, жестким диском в 1 Гб и процессором частотой 90 МГц!), создание Web-сервера из старой Windows-машины - вполне разумная вещь и хорошая возможность поучиться.

Установка на сервер Windows

Установить PHP и Apache на Windows-машину во многих отношениях легче, чем на Linux-сервер, и этому во многом способствует графический интерфейс. У вас может и не быть полных прав администратора сервера, а после установки удастся тестировать свои программы на этой же машине.

PHP можно установить под Windows 95, 98, ME, NT или 2000, главное, загрузить файлы именно для вашей версии Windows, имеющиеся на сайтах Apache и PHP. Разработаны также специальные скрипты, которые упрощают установку PHP.

Установка Apache и PHP

1. Загрузите самые последние, стабильные версии PHP и Apache на компьютер (рис. А.7).

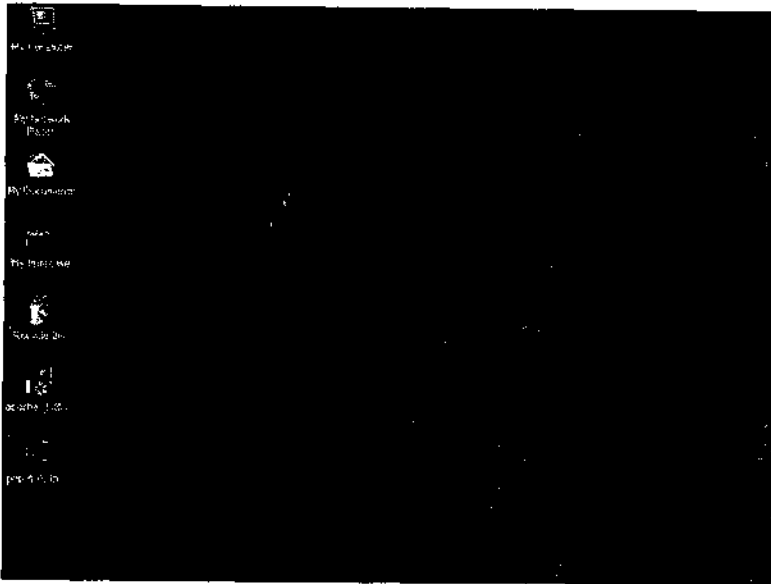


Рис. А.7. Я загрузил необходимые файлы для инсталляции под Windows и поместил их на Рабочий стол

2. Распакуйте PHP в соответствующий каталог (например, C:/php). Для этого вам потребуется программа-архиватор.
3. Установите Apache, запустив программу инсталляции (рис. А.8).

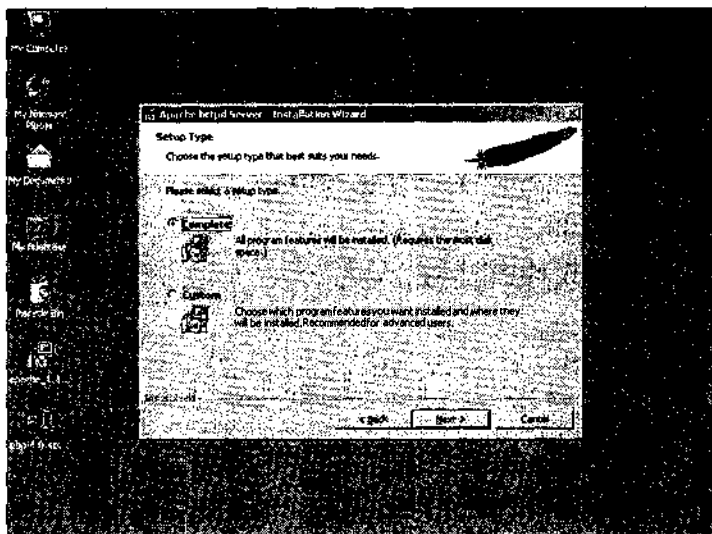


Рис. А.8 т Двойной щелчок мыши по пиктограмме Apache запускает процесс установки

4. Скопируйте файл `php.ini-dist` из каталога PHP (рис. А.9) в соответствующий системный каталог (например, C:/windows). Сохраните файл как `php.ini`.

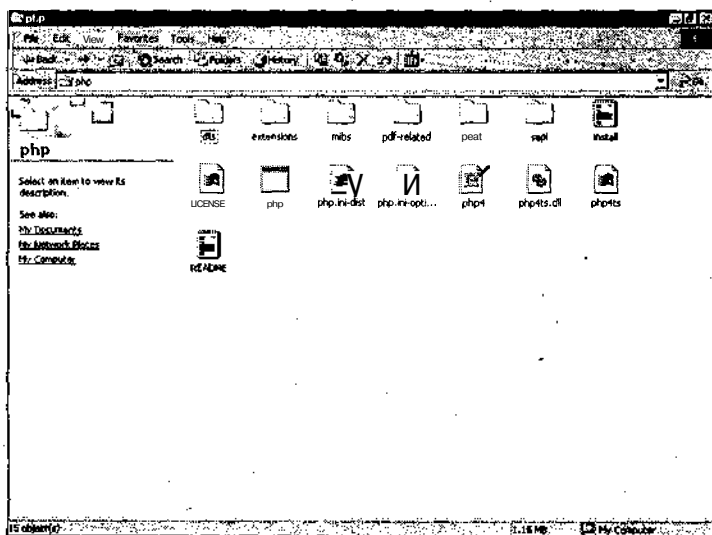


Рис. А.9 т Распакованные PHP-файлы были помещены в каталог C:\php

5. Запустите сервер Apache.

Легкий способ сделать это - воспользоваться меню **Start ► Programs ► Apache Web Server ► Start Apache**.

6. Протестируйте с помощью браузера, работают ли PHP и Apache.

Самый легкий способ проверить работоспособность сервера Apache - напечатать адрес <http://localhost/> в окне браузера. Протестировать PHP можно с помощью файла test.php.



PHP можно установить на Windows-машину и с другими Web-серверами, например IIS.

Конфигурация

Лучше всего сконфигурировать PHP во время установки, но можно менять параметры и после этого. Файл `php.ini`, который вы скопировали в соответствующий каталог во время установки, содержит набор рабочих параметров PHP (рис. АЛО и А.11). Вы можете отредактировать эти параметры и перезапустить серверное приложение. При возникновении каких-либо проблем восстановить исходный файл `php.ini` можно из файла `php.ini-dist`, который после установки являлся точной копией первого.

Чтобы PHP поддерживал базы данных, мгновенно генерировал графику, использовал функцию `mysql()` и т.д., необходимо загрузить дополнительные

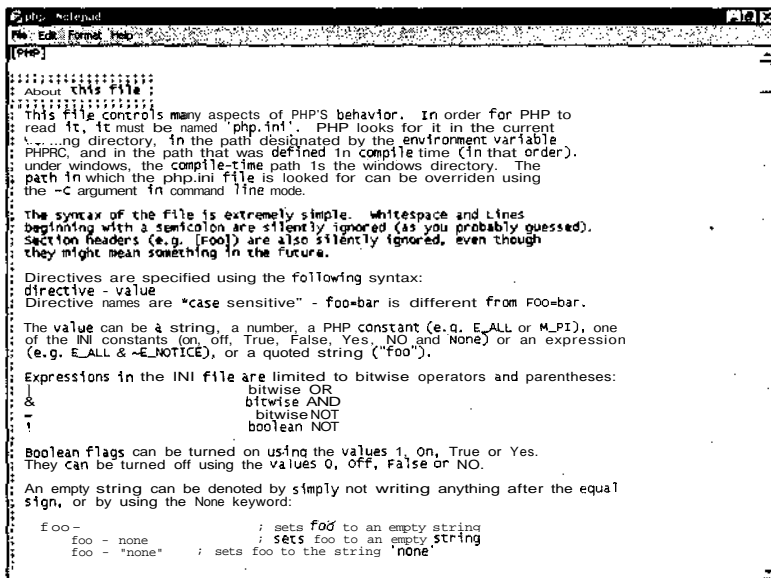


Рис. А.10: Документ `php.ini` - самый важный файл для определения того, как работает PHP

```

; thought of
register_argc_argv = On ; This directive tells PHP whether to declare the
; variables (that would contain
; the GET information). If you
; don't use these variables, you
; should turn it off for
; Increased performance
post_max_size = 8M ; Maximum size of POST data that PHP will accept.
; This directive is deprecated. Use
; variables_order instead.
; Magic quotes
magic_quotes_gpc = On ; magic quotes for incoming GET/POST/Cookie data
magic_quotes_runtime = off ; magic quotes for runtime-generated data, e.g. data from
; SQL, from exec(), etc.
magic_quotes_sybase = off ; use Sybase-style magic quotes (escape ' with ''
; instead of \')
; automatically add files before or after any PHP document
auto_prepend_file =
auto_append_file =
; As of 4.0.4, PHP always outputs a character encoding by default. In
; the Content-type: header. To disable sending of the charset, simply
; set it to be empty.
; PHP's built-in default is text/html
default_mimetype = "text/html"
default_charset = "iso-8859-1"
; Paths and directories
; include_path ; UNIX: "/pat h1: /pat h2" windows: "\\pat h1; \pat h2"
; the root of the php pages, used
doc_root =
; the directory under which php
user_dir =
; opens the script using /-username, used only if nonempty
extension_dir = ; directory in which the loadable
; extensions (modules) reside
enable_dl = On ; whether or not to enable the d1C
function.

```

Рис. А. 11 ▼ Файл `php.ini` содержит много страниц с параметрами, а также инструкции относительно того, как их изменить

библиотеки и пакеты, а затем указать их **расположение** программе установки, или же поправить файл `php.ini` позже. В файле `php.ini` содержатся очень подробные комментарии обо всех возможных опциях. В руководстве по PHP также представлена подробная информация на эту тему.

В приложение

Безопасность

О безопасности аппаратного и системного программного обеспечения, а также ваших собственных программ надо думать постоянно. Конечно, не нужно считать каждый сайт потенциальным объектом для несанкционированных действий со стороны, но оставлять свои проекты без защиты тоже не стоит. О некоторых аспектах безопасного программирования уже упоминалось в некоторых главах книги и в приложении А. Здесь коротко говорится о некоторых дополнительных шагах, которые вы можете предпринять в целях защиты программ, а также дается информация о ряде интересных ресурсов Internet, посвященных безопасности.

Криптография и SSL

Криптография - это процесс изменения (шифрования) формата данных, чтобы их было труднее прочитать постороннему. Некоторые криптографические системы, такие как PGP, свободно доступны на сайте <http://www.pgp.com> (рис. В.1). В этой программе используются открытые и секретные ключи для шифрования и дешифрования информации. Другие криптографические системы, такие как встроенная в PHP функция `сгурт()`, шифруют данные, но не дешифруют их. Подробнее о функции `сгурт()` можно узнать в разделе «Строки» пособия по PHP.

Необходимого уровня криптографической защиты вашего сайта можно достичь, загрузив библиотеку `mcсгурт` с <http://mcrypt.hellug.gr/> и сконфигурировав PHP во время установки для поддержки `mcсгурт`. Использование этой библиотеки позволит вам применять функции `mcсгурт()`, которые могут шифровать и дешифровать информацию. Подробнее о `mcсгурт()` вы также можете узнать на вышеупомянутом сайте и в пособии по PHP.

Криптография - только часть решения по безопасности, так как она может быть использована только тогда, когда данные были получены сервером. На

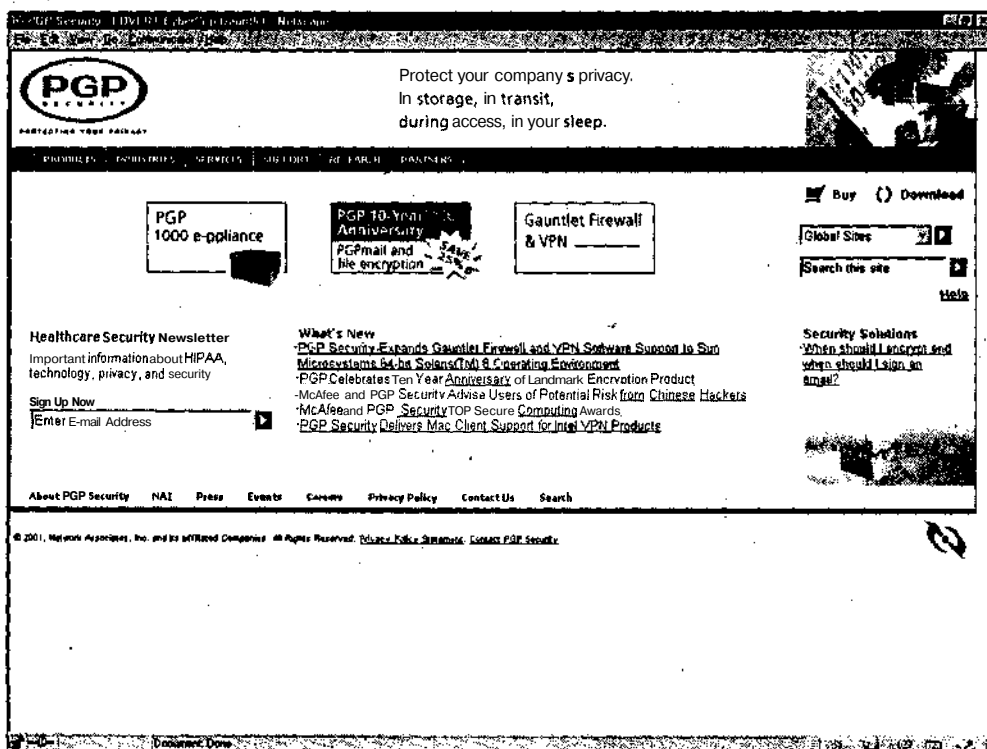


Рис. В.1 Имеются бесплатная и коммерческая версии PGP, позволяющие вам посылать и принимать зашифрованные данные

созданном сайте также допустимо использовать преимущества протокола SSL. SSL, протокол безопасных соединений, – это способ защищенного обмена информацией между клиентом (Web-браузером) и сервером. Применение протокола SSL (префикс `https://` в URL) обязательно для приложений электронной торговли. По SSL можно также посылать cookie, задав соответствующие параметры при использовании функции `setcookie()`. Узнайте у вашего провайдера или администратора сервера, поддерживает ли ваша машина протокол SSL.



Используемые в PHP-приложении пароли должны быть всегда зашифрованы. Если ваш сервер не поддерживает функцию `mcrypt()`, используйте `crypt()` для шифрования пароля, введенного во время аутентификации, затем проверяйте его на соответствие хранящемуся в базе зашифрованному паролю.

Написание безопасного PHP-кода

Хотя при использовании PHP нет таких проблем с безопасностью, как при обращении к CGI-скриптам или ASP, они все же существуют. Есть несколько вещей, которые надо постоянно помнить при программировании.

Во-первых, следует размещать файлы с критичной информацией, такой как пароли, вне корневого каталога Web-документов. Каждое серверное приложение использует для Web-документов **корневой** каталог по умолчанию. Файлы, находящиеся в этом каталоге, могут быть доступны через URL из любого браузера. Файлы, хранящиеся выше корневого каталога Web-сервера, просмотреть уже не так просто. Однако их можно спокойно использовать в PHP с помощью включения кода в скрипт, например:

```
require ("../secure.php");
```

Эта строка позволит использовать файл `secure.php`, хранящийся в каталоге, который находится на один уровень выше текущего каталога (рис. В.2). Однако этот файл недоступен напрямую из сети Internet, так как расположен вне корневого каталога Web-документов.

Вторая рекомендация касается передаваемых пользователем данных из формы HTML. При передаче информации, которая не должна быть перехвачена, необходимо всегда использовать метод POST, так как метод GET добавит передаваемые данные к URL, сделав их видимыми в окне браузера.

Кроме того, необходимо осторожно относиться к передаваемым данным, поскольку таким образом злонамеренный пользователь может разрушить систему. Не очень честные, но хитрые люди могут послать через HTML-форму JavaScript или исполняемый код на ваш сайт. Этот код может переслать им критичную информацию, внести изменения в базу данных и т.п. Предотвратить такого рода посягательства нетрудно, если проверять все входящие данные с помощью регулярных выражений (см. главу 8).

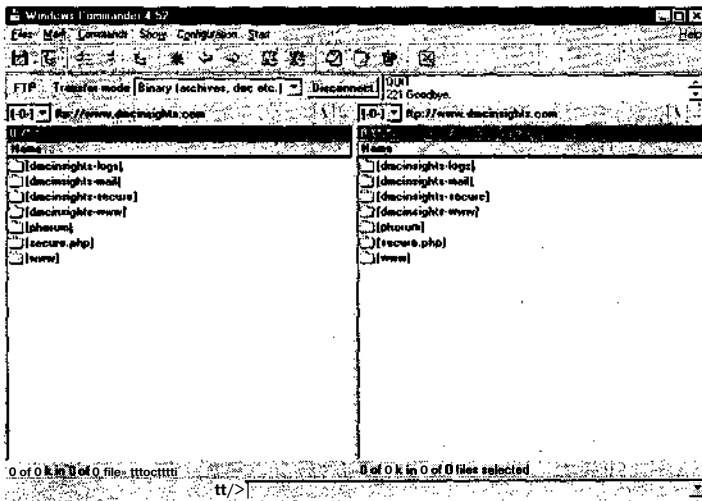


Рис. В.2 Этот каталог находится выше корневого каталога Web-документов (`www` или `dmcinsights-www`) и поэтому недоступен пользователю через браузер. Я могу, однако, использовать его для хранения особо чувствительных документов, таких как `secure.php`

В приложении С даны ссылки на сайты, где можно найти статьи по этой теме, а также примеры регулярных выражений.

Ресурсы по вопросам безопасности

Если вы собираетесь серьезно заниматься Web-программированием, вам стоит узнать о Web-безопасности гораздо больше, чем было изложено в данном приложении.

Существуют сотни Web-сайтов, где можно найти информацию по вопросам безопасности. Самые лучшие ресурсы в этом отношении, по-моему, следующие:

- <http://www.cert.org>;
- <http://www.security-focus.com>;
- <http://www.packetstorm.security.com>;
- <http://www.w3.org/Security/Faq/www-security-faq.html>.

Общим вопросам безопасности посвящено множество книг, есть и издания, с помощью которых вы сможете создать безопасный Web-сервер Windows NT или Linux.

Обязательно прочитайте раздел руководства PHP, посвященный безопасности. Просмотрите соответствующий раздел в документации по базе данных, которую вы используете на сервере. Например, в руководстве по MySQL, даются конкретные советы в отношении совместного использования PHP и MySQL.

С Приложение

Ресурсы PHP

Основная задача этой книги - дать начинающим программистам на PHP хорошую основу для дальнейшего получения знаний. В связи с этим некоторые темы были опущены или освещались не в полном объеме. В данном приложении перечислены некоторые полезные PHP-ресурсы сети Internet и даны советы о том, где найти дополнительную информацию по базам данных и другим дополнительным темам.

Руководство по PHP

Прежде чем начать работать с языком, каждый программист по PHP **должен** приобрести соответствующее руководство. Вы найдете его на официальном PHP-сайте по адресу <http://www.php.net/docs.php> (рис. С.1) или на других сайтах. Руководство почти на десяти языках можно получить в любом из следующих форматов: PDF, HTML, простой текст, для карманных компьютеров Palm. На официальном Web-сайте также имеется аннотированная версия **руководства** (<http://www.php.net/manual/net>), куда пользователи добавляют собственные полезные советы и комментарии, призванные помочь разрешать некоторые проблемы при использовании PHP.

Пособие начинается с описания вопросов установки и безопасности. Затем говорится о базовых конструкциях языка и некоторых возможностях PHP. Основная часть пособия посвящена описанию встроенных функций PHP, разбитых по темам. Однако начинающему программисту не всегда понятен формат описания этих функций. Часто вы можете увидеть такие строки кода:

```
array file (string filename [, int use_include_path])  
int mysql_close ([int link_identifier])  
void exit(void)  
double round (double val)
```

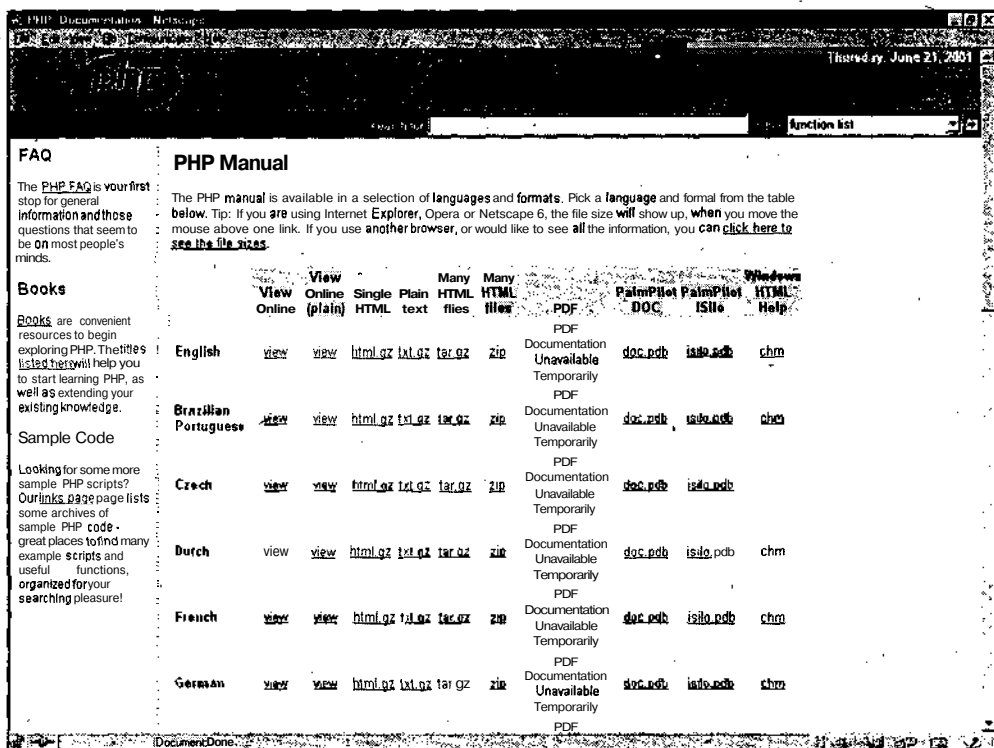


Рис. С. 1 У На сайте PHP представлено много разных версий пособия по этому языку, которые можно свободно брать. Я бы порекомендовал загрузить одну из них на ваш компьютер и использовать ее, не загружая понапрасну сеть

Очень важно понять, что первое слово каждой строки обозначает тип значения, которое возвращает функция. Функция `file()` возвращает массив, `mysql_close()` - целое число, функция `exit()` не возвращает ничего, `round()` - число с плавающей запятой (двойной точности). В скобках указаны аргументы (и типы аргументов), которые принимает функция. Необязательные аргументы взяты в квадратные скобки. Например, функция `exit()` не принимает никаких аргументов, `round()` обязательно требует число с плавающей запятой, а для `mysql_close()` вполне допустимо указывать необязательный аргумент `link_identifier`.

Web-сайты и сетевые конференции

Я упомяну только несколько Web-сайтов, где можно найти самую разную информацию по программированию. Посетите их и выберите то, что вам больше понравится. На большинстве ресурсов также содержатся ссылки на другие сайты, посвященные PHP.

Первый и самый очевидный выбор - это официальный сайт PHP (адрес <http://www.php.net>).

Во-вторых, вам необходимо посетить сайт создателей четвертой версии PHP Zend.com (<http://www.zend.com>). Там вы можете найти массу полезных утилит, а также получить много ценной информации.

Информация по конкретным темам представлена на сайте PHPBuilder (<http://www.phpbuilder.com>). Здесь содержатся десятки статей, объясняющих, как с помощью PHP выполнять конкретные задачи (рис. С.2). Кроме того, на сайте работают форумы по поддержке пользователей и имеется библиотека с образцами кодов.

PHPstart4all (<http://php.start4all.com>) - хороший ресурс для тех, кто только начинает использовать PHP. Он содержит ссылки на PHP-сайты, учебные материалы по PHP, книги по этому языку, PHP-проекты с открытым исходным текстом, библиотеки сценариев.

Сайт PHP Resource Index (<http://php.resourceindex.com>) также предлагает интерактивные учебные материалы, библиотеки кода и сценариев (рис. С.3).

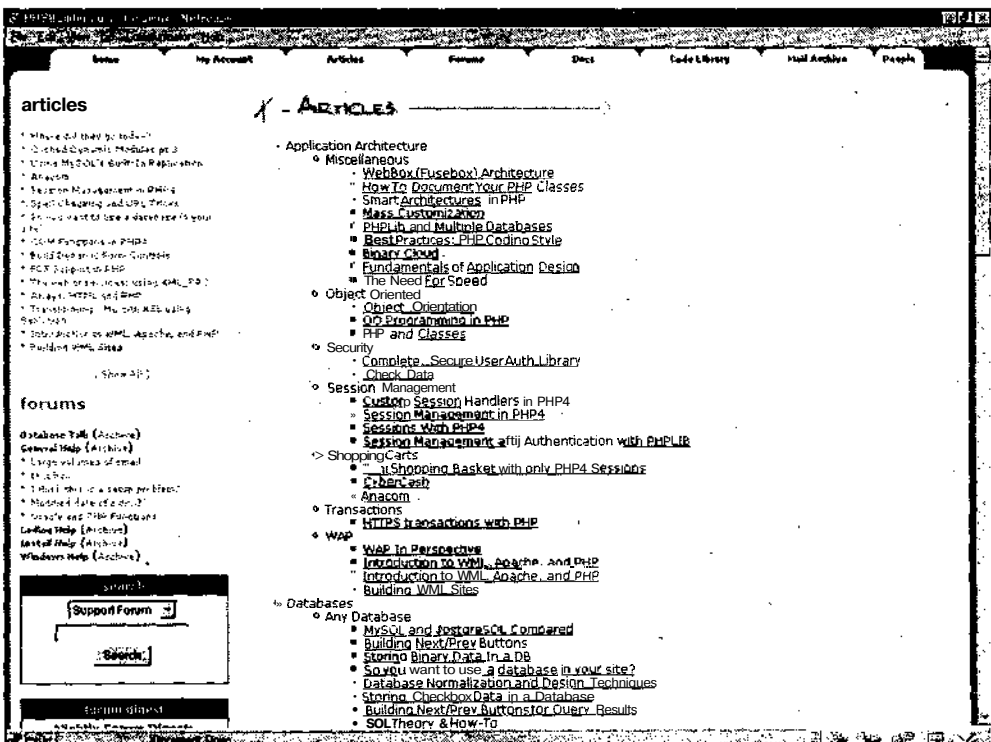


Рис. С.2 На сайте PHPBuilder представлен большой список статей, посвященных отдельным аспектам PHP начиная с базовых (например, документирование кода) и заканчивая более сложными (например, объектно-ориентированное программирование)

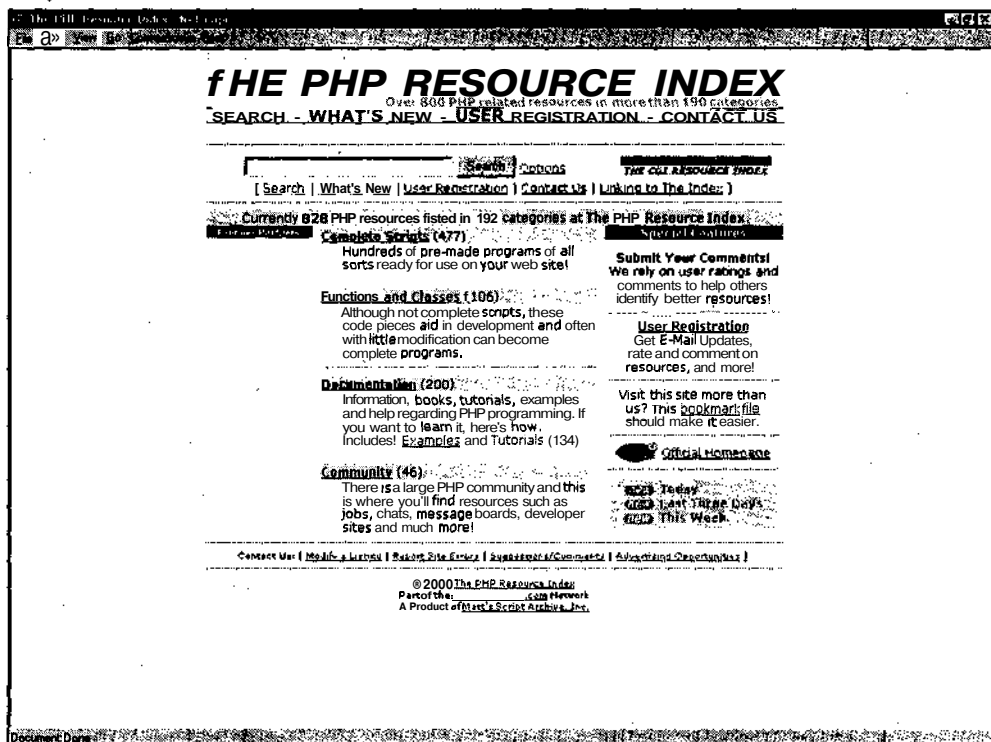


Рис. С.3 т На сайте PHP Resource Index и других подобных ресурсах имеются огромные библиотеки образцов кода и сценариев, которые можно использовать в работе

Библиотеки кода можно найти на следующих сайтах:

- WeberDev (<http://www.weberdev.com/maincat.php3?categoryID=106&category=PHP>);
- HotScripts (<http://www.hotscripts.com/PHP/>).

Последний Web-ресурс, о котором стоит упомянуть здесь, - это PHP Coding Standard (http://utvikler.start.no/code/php_coding_standard.html). Стандарт - это документ, дающий рекомендации в отношении правильного формата и синтаксиса имен переменных, управляющих структур и т.д. при программировании на PHP. Хотя вовсе не обязательно следовать всем правилам, есть отличные и хорошо продуманные рекомендации, которые помогут сократить количество ошибок в ваших программах.

Если у вас есть доступ к сетевым конференциям, вы можете использовать их для представления своих идей широкой публике, а также найти там ответы на интересующие вас вопросы. Самая большая англоязычная сетевая конференция - alt.php. Доступ на alt.php можно получить через вашего провайдера или через платную службу Usenet. Существуют сетевые конференции и на других языках.¹

¹ Русскоязычные программисты могут начать с ресурса <http://phpclub.unet.ru> и далее по ссылкам. - Прим. науч. ред.

Ресурсы по базам данных

В зависимости от того, какую СУБД вы используете, вам понадобятся соответствующие ресурсы для работы с ней. Наибольшее распространение для работы с PHP получила база MySQL, за ней идет PostgreSQL. Однако в PHP поддерживается большинство стандартных баз данных.

Узнать больше об использовании MySQL удастся на официальном Web-сайте MySQL (<http://www.mysql.com>, рис. С.4). Вы можете загрузить оттуда пособие по изучению этой БД. Там представлены также книги по MySQL, такие как самоучитель «MySQL за 21 день» Марка Маслаковски (Mark Maslakowski) и Тони Батчера (Tony Butcher), «MySQL и mSQL» Ренди Джей Ярпера (Randy Jay Yarger), Джорджа Риса (George Reese) и Тима Кинга (Tim King), выпущенных в издательстве O'Reilly.

Информацию по использованию баз данных PostgreSQL можно получить на сайте (<http://www.postgresql.org>). Одна из популярных книг по этой базе данных - «PostgreSQL: введение и концепции» - написана Брюсом Момджияном (Bruce Momjian).

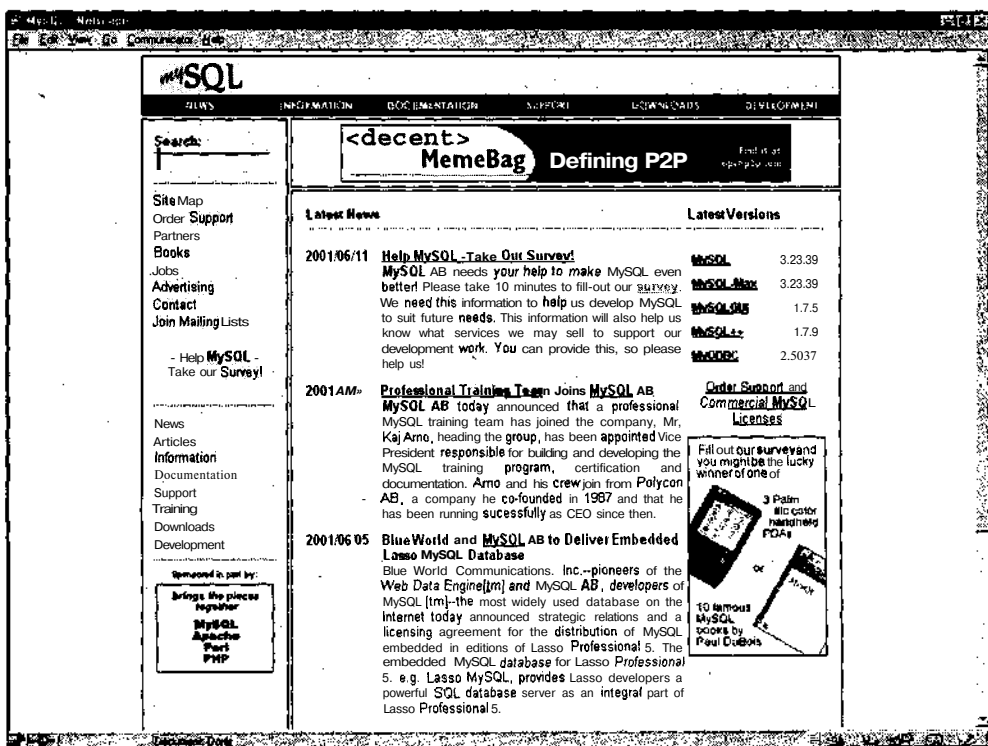


Рис. С.4 На Web-сайте MySQL вы найдете большое количество необходимой документации и загружаемые файлы MySQL

Если вы хотите получить общее представление о разработке баз данных, вас могла бы заинтересовать книга Майкла Хернандеса (Michael J. Hernandez) «Разработка баз данных для простых смертных». В подобных изданиях не рассматриваются конкретные вопросы использования баз MySQL или PostgreSQL, однако книги дают понимание того, что лежит в основе создания и использования баз данных.

Сложные темы

Хотя настоящей книги достаточно, чтобы вы начали работать с языком PHP самостоятельно, есть несколько тем, которые вы, вероятно, захотите изучить более глубоко.

Одна из них - создание объектов и классов с помощью PHP. Когда вы наберетесь опыта и постройте солидную библиотеку кодов, создание и использование объектов повысит скорость вашего программирования и снизит количество ошибок. Знакомство с объектами в PHP упростит изучение объектно-ориентированных языков, таких как Java (несмотря на то что в PHP используются объекты, он не является объектно-ориентированным языком). Вы можете найти хорошие интерактивные учебные материалы по объектам в PHP на сайте Zend.com (<http://www.zend.com/zend/tut/class-intro.php>, рис. С.5).

Две другие темы, которые я порекомендовал бы изучить глубже, - применение функций для возвращения множественных значений, а также использование ссылок на переменные - подробно освещены в пособии по PHP. Первая обсуждается в разделе «Функции» и объясняет, как использовать массив для возвращения нескольких значений из одной функции. Вторую можно найти в разделе «Переменные», где подробно описано, как создавать ссылки на имена переменных, как присваивать им значения и разыменовывать их обратно. Эти темы более сложны, но зато, овладев соответствующими навыками, вы сможете считать себя профессиональным программистом.

Вопрос о правах доступа к файлу вкратце обсуждался в главе 10. Если вы работаете на UNIX-сервере и хотите знать больше, обратитесь либо к пособию по UNIX (вызывается с помощью команды `man`), либо к прекрасной книге Элизабет Кастро (Elizabeth Castro) «PERL и CGI для Всемирной паутины». В приложении Кастро описывает не только права доступа и проблемы безопасности в среде UNIX, но и другие вопросы, связанные с UNIX. Если вы работаете на Windows-сервере, то можете найти информацию о правах доступа к файлам на сайте компании Microsoft (<http://www.microsoft.com>) или в справочных файлах Windows.

На протяжении всей книги я рекомендовал вам использовать уже существующий код для расширения возможностей регулярных выражений и другой функциональности PHP. На сайтах, упомянутых выше, есть примеры кода, которые

можно скопировать и использовать в работе. Другие программисты уже разработали огромное количество шаблонов для регулярных выражений и готовых сценариев. Не стоит изобретать в тысячный раз то, как с помощью PHP отправить электронную почту с прикрепленными файлами или другие стандартные операции. Вы можете использовать данный код непосредственно или доработать его для себя - в любом случае это позволит вам сэкономить уйму времени и научит разным способам достижения цели.

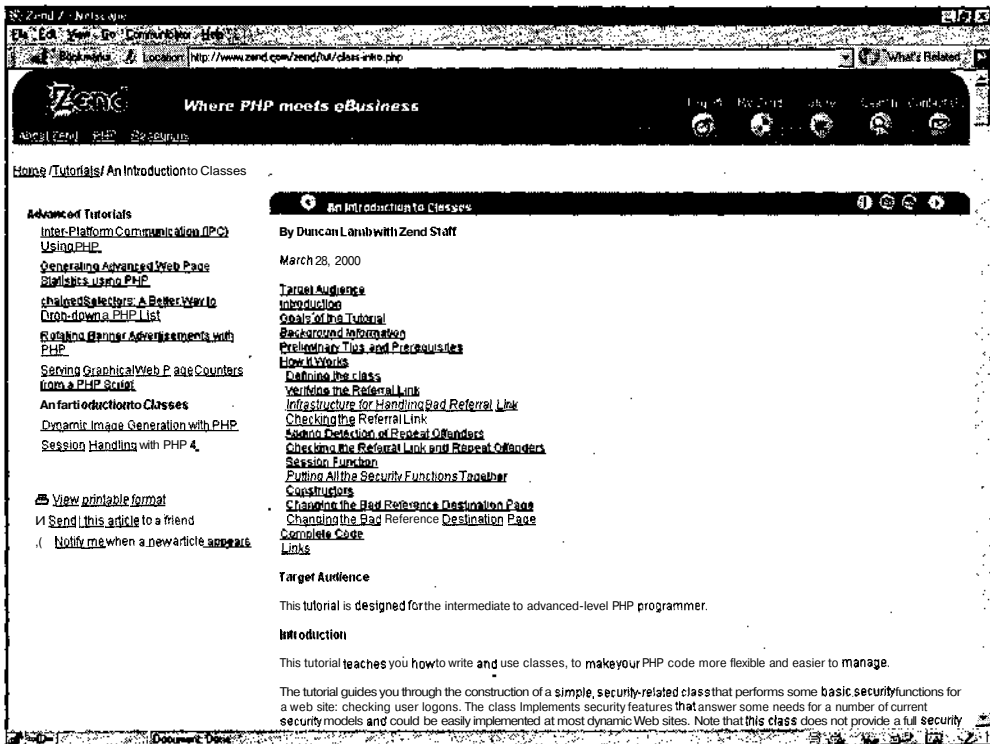


Рис. С.5 На сайте Zend.com представлен очень подробный и простой учебник по объектно-ориентированному программированию в PHP

Таблицы

В последних главах книги я упоминал о нескольких таблицах с полезной информацией. Они перед вами, при этом обратите внимание на таблицу, в которой представлен перечень операторов в порядке их приоритета.

Таблица C.1 т Это неполный список операторов, расположенных в порядке их приоритета (от высшего к низшему). Так, умножение имеет более высокий приоритет, чем сложение

Приоритет операторов	
	! ++ --
	* / %
	+ - .
	< <= > >=
	== !=
	&&
	= += -= *= /= .= %=
	and
	xor
	or

Таблица C.2 т Специальные символы для регулярных выражений

Символ	Соответствия
.	любой символ
^a	«a» в начале строки
a\$	«a» в конце строки
a+	по крайней мере одна «a»
a?	ноль или одна «a»
\n	новая строка
\t	табуляция
\	экранировать
(ab)	«a» и «b» вместе
a b	«a» или «b»
a{2}	«aa»
a{1,}	«a», «aa», «aaa» и т.д.
a{1,3}	«a», «aa», «aaa»
[a-z]	любая строчная буква
[A-Z]	любая прописная буква
[0-9]	любая цифра

Таблица C.3 т Не забывайте, что в PHP уже определено несколько классов символов, использующихся при создании шаблонов регулярных выражений. Здесь представлены только основные

Предопределенные классы для регулярных выражений	
Класс	Соответствия
[[:alpha:]]	любая буква
[[:digit:]]	любая цифра
[[:alnum:]]	любая буква или цифра
[[:space:]]	любой пробел
[[:upper:]]	любая прописная буква
[[:lower:]]	любая строчная буква
[[:punct:]]	любой знак пунктуации

Таблица С.4 т Используемый при открытии файла параметр определяет, что можно делать с этим файлом - записывать в него данные, читать его и т.д.

Режимы файла	
Режим	Позволяет
r	Только чтение файла
w	Только запись данных в файл (создает файл, если он не существует, в противном случае усекает его размер до нуля перед записью)
a	Добавление новых данных в конец файла (создает файл, если его нет)
r+	Чтение файла и запись в него данных
w+	Чтение файла и запись в него данных (создает файл, если он не существует, в противном случае усекает его размер до нуля перед записью)
a+	Чтение файла и добавление новых данных в конец файла (создает файл, если его нет)

Таблица С.5 т Различные форматы даты - это большой список, который лично я никак не могу запомнить. Держите эту таблицу **рядом**, когда используете функцию `date()`

Варианты формата даты	
Символ	Формат
a	am или pm
A	AM или PM
d	день месяца из двух цифр: от 01 до 31
D	день недели, сокращенная форма: Sun, Mon и т.д.
p	месяц, полная форма: January
g	время дня в 12-ти часовом формате: от 1 до 12
G	время дня в 24-х часовом формате: от 0 до 23
h	время дня в 12-ти часовом формате: от 01 до 12
h	время дня в 24-х часовом формате: от 00 до 23
i	минуты: от 00 до 59
j	день месяца: от 1 до 31
1 (строчная L)	день недели, полная форма: Sunday
m	месяц из двух цифр: от 01 до 12
n	месяц, сокращенная форма: Jan
n	месяц (цифрами): от 1 до 12
s	секунды; от 00 до 59
s	английский двухбуквенный суффикс порядкового числительного: th, nd, rd и т.д.
t	количество дней в данном месяце: от 28 до 31
p	секунды с точки отсчета времени
w	день недели как одна цифра: от 0 (воскресенье) до 6 (суббота)
y	год из двух цифр: 01
Y	год из четырех цифр: 2001
z	день года: от 0 до 365

Таблица С.6 т Эти опции используются и при разработке сайта, и при его эксплуатации

Уровни сообщений об ошибках		
Число	Константа	Значение
1	E_ERROR	Фатальная ошибка этапа выполнения
2	E_WARNING	Нефатальное предупреждение этапа выполнения
4	E_PARSE	Ошибка синтаксического анализатора
8	E_NOTICE	Сообщение этапа выполнения
16	E_CORE_ERROR	Фатальная ошибка при запуске PHP. Только PHP 4.0
32	E_CORE_WARNING	Предупреждение (нефатальная ошибка) при запуске PHP. Только PHP 4.0
64	E_COMPILE_ERROR	Фатальная ошибка компилятора. Только PHP 4.0
128	E_COMPILE_WARNING	Предупреждение компилятора. Только PHP 4.0
256	E_USER_ERROR	Сгенерированная пользователем ошибка. Только PHP 4.0
512	E_USER_WARNING	Сгенерированное пользователем предупреждение. Только PHP 4.0
1024	E_USER_NOTICE	Сгенерированное пользователем сообщение. Только PHP 4.0
	E_ALL	Все вышеперечисленное

Предметный указатель

Б

База данных 195

В

Выражение логическое 86

З

Знак 35

— автодекремент 55

! Логическое Нет 87

!= не равно 84

комментарий 29

\$ доллар 33

\$ доллар, метасимвол 132

% остаток от деления 107

&& Логическое И 87

() в выражении 57

() в функции 24

() скобки, метасимвол 137

* «звездочка», метасимвол 132

* умножение чисел 51

+ плюс, метасимвол 132

+ сложение чисел 51

++ автоинкремент 55

- вычитание чисел 52

. конкатенация строк 65

. точка, метасимвол 132

/ деление чисел 52

< меньше 84

<= меньше или равно 84

= присвоение значения 36

== равенство логическое 84

> больше 84

>= больше или равно 84

? вопрос, метасимвол 132

? использование в URL 46

@ коммерческое «а» 135

@ перед функцией 258

[] квадратные скобки 110

[] скобки, метасимвол 138

^ «крышечка», метасимвол 132

_ подчеркивание 33

{ } фигурные скобки 80

| вертикальная черта,
метасимвол 133

|| Логическое Или 87

допустимый в URL 67

кавычки 24

новая строка 35

обратный слеш, метасимвол 132

печать новой строки 27

экранирование 25, 35

Значение

null 149

истина (true) 80

логическое 80

ложь (false) 80

И

Индекс 74

К

Каталог 180

Комментарий 28

HTML 31

Л

Лексема 74

Литерал 131

М

Массив 35, 109

Метасимвол 132

класс 138

О

Оператор

AND, Логическое И 87

break 95

- die 256
- for 106
- global 157
- if 79
- if-else 89
- if-elseif 91
- if-elseif-else 91
- if-then 79
- if-then-else 89
- NOT, Логическое Нет 87
- OR, Логическое Или 87
- require() 224
- return 152
- switch 94
- while 101
- XOR, логическое Или Нет 87
- блок 80
- больше 84
- больше или равно 84
- взятие по модулю 107
- логический 86
- меньше 84
- меньше или равно 84
- неравенства 84
- приоритет 56
- присваивания 36
- равенства 84

П

- Переменная 32
- локальная 157
- массив 35, 109
- область действия 157
- получение из экранной формы 38
- предопределенная 36
- строка 34
- тип 34
- число 34

Р

- Регулярные выражения 131

Ф

- Файл
 - запись 168
 - открытие 167

- права доступа 166
- режим при открытии 168
- Форма 38
- Функция
 - abs() 59
 - addslashes() 71
 - array_merge() 114
 - arsort() 119
 - asort() 119
 - ceil() 59
 - closedir() 189
 - copy() 185
 - crypt() 72
 - date() 228
 - decrypt() 72
 - die 256
 - each() 116
 - echo() 24
 - encrypt() 72
 - ereg() 133
 - ereg_replace() 140
 - ereg() 133
 - ereg_replace() 140
 - error_log() 249
 - error_reporting() 249
 - explode() 121
 - file() 173
 - filesize() 188
 - floor() 59
 - fopen() 167
 - fwrite() 168
 - header() 236
 - headers_sent() 247
 - implode() 121
 - include() 224
 - is_writeable() 172
 - join() 125
 - krsort() 119
 - ksort() 119
 - mail() 240
 - md5() 73, 75
 - mkdir() 180
 - mktime() 230
 - opendir() 189
 - phpinfo() 20
 - print() 21, 23

`printf()` 24, 53
`rand()` 59
`readdir()` 189
`rename()` 188
`round()` 58
`rsort()` 118
`setcookie()` 212
`shuffle()` 119
`sort()` 118
`sprintf()` 55
`srand()` 59
`stripslashes()` 71
`strlen()` 75
`strtok()` 74
`substr()` 75
`trim()` 62
`unlink()` 185
`urldecode()` 71
`urlencode()` 67
аргумент 148
возврат значения 152
вызов 145
доступ к БД 197
неопределенный аргумент 149
порядок аргументов 161
создание 145

Ц

Цикл

`for` 106
`while` 101
счетчик 106

Ч

Число

с плавающей запятой 34
случайное 59
целое 34

Ш

Шаблон 131

Э

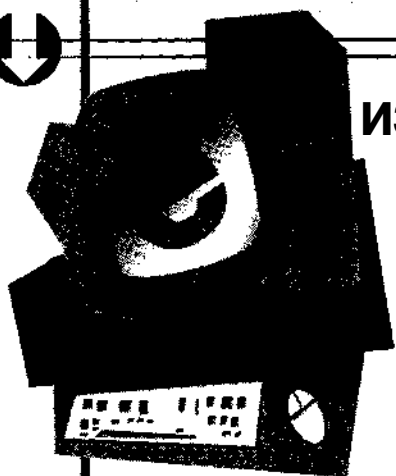
Экранирование 35

С

Cookie 211

время жизни 218
создание и чтение 212
удаление 220

www.dmkpress.ru



ИЗДАТЕЛЬСТВО DMK Press

ПРЕДОСТАВЛЯЕТ ВАМ

возможность приобрести интересующие Вас книги, посвященные компьютерным технологиям и радиоэлектронике, самым быстрым и удобным способом. Для этого Вам достаточно всего лишь посетить Internet-магазин «ДМК Пресс» по адресу **www.dmkpress.ru**. Вашему вниманию будет представлен полный перечень книг по программированию, компьютерному дизайну, проектированию, ремонту радиоаппаратуры, выпущенных в нашем и других издательствах. В Internet-магазине Вы сможете приобрести любые издания, не отходя от домашнего компьютера: оформите заказ, воспользовавшись готовым бланком, и мы доставим Вам книги в самый короткий срок по почте или с курьером.

Internet-магазин на www.dmkpress.ru.

- экономит Ваше время, позволяя заказать любые книги в любом количестве, не выходя из дома;
- избавляет Вас от лишних расходов: мы предлагаем компьютерную и радиотехническую литературу по ценам значительно ниже, чем в магазинах;
- дает возможность легко и быстро оформить заказ на книги — как новинки, так и издания прошлых лет, пользующиеся постоянным спросом.

Если Вы живете в Москве, то доставка с курьером позволит Вам увидеть книгу перед покупкой. При этом Вам не придется пользоваться кредитными картами или оплачивать почтовые услуги.





НОВИНКА

Эффективное использование ПК

Автор: Зелинский С. Э.
Формат: 70x100 ¹/₁₆
Объем: 592 с.
ISBN: 5-94074-064-2



Издание содержит информационный и справочный материал по темам, связанным с повседневной работой на ПК. Здесь можно найти ответы на вопросы, актуальные для пользователей, компьютеры которых оснащены процессорами от x386 до Pentium III, а в качестве операционной системы установлены MS-DOS, Windows 3.x, Windows 95/98 или Windows ME.

В книге рассказывается о настройке и условиях эффективной работы различных устройств современного компьютера, описываются русифицированные версии операционных систем. Вы научитесь работать с текстовым редактором, программами электронных таблиц и подготовки презентаций из пакета Microsoft Office 2000, графическими программами Paint и Imaging, Веб-браузером Internet Explorer 5 и входящим в него почтовым клиентом Outlook Express 5, а также программами архивации и шифрования данных.

Книга адресована тем, кто хочет самостоятельно освоить ПК, научиться устранять возможные проблемы с его устройствами, выполнять эффективную настройку системы и овладеть основами работы с популярными офисными приложениями, утилитами и программами Internet.

Цена указана
без стоимости
доставки

Издательство «ДМК Пресс»
высылает книги почтой
наложенным платежом

Заказы присылайте по адресу:

Оптовые закупки:

E-mail:

Интернет-магазин:

107014, Москва, а/я 468

тел. (095) 962-1703

369-7528

sale@dmk.ru,
dmk@home.relline.ru

<http://www.dmkpress.ru>

Ларри Ульман

Основы программирования на РНР

Главный редактор	<i>Захаров И. М.</i>
Перевод с английского	<i>Макаров М. В.</i>
Выпускающий редактор	<i>Петроградская А. В.</i>
Технический редактор	<i>Прока С. В.</i>
Верстка	<i>Белова И. Е.</i>
Графика	<i>Шаклунов А. К.</i>
Дизайн обложки	<i>Панкусова Е. Н.</i>

ИД №01903 от 30.05.2000

Подписано в печать 30.06.2001. Формат 70×100¹/₁₆.

Гарнитура «Баскервиль». Печать офсетная.

Усл. печ. л. 18. Тираж 3000.

Зак. № 1727

Издательство «ДМК Пресс», 105023, Москва, пл. Журавлева, д. 2/8.

Электронные адреса: www.dmkpress.ru, info@dmk.ru

Отпечатано в Раменской типографии

104100, Московская обл., г. Раменское, Сафоновский пр., 1