SAMS
Освой самостоятельно



Рассмотрен PHP 5.0!

PHP

10 минут на урок

Крис Ньюман

SAMS
Освой
самостоятельно

PHP

10 минут на урок

SAMS Teach Yourself

PHP

Chris Newman

in 10 Minutes

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

SAMS Освой самостоятельно

PHP

Крис Ньюман

10 минут на урок

ВИЛЬЯМС
Издательский дом "Вильямс"
Москва • Санкт-Петербург • Киев
2006

ББК 32.973.26-018.2.75
Н93
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция *С.В. Бойко*
По общим вопросам обращайтесь в Издательский дом "Вильямс"
по адресу:
info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Ньюман, Крис.

Н93 Освой самостоятельно PHP. 10 минут на урок. : Пер. с англ. — М. : Издательский дом "Вильямс", 2006. — 272 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0937-6 (рус.)

Эта книга поможет вам в кратчайшие сроки освоить PHP — самый популярный язык программирования для Web. Начиная с простых языковых конструкций, автор урок за уроком рассматривает все более сложные темы, такие как операции для работы с числами и строками, регулярные выражения, обработка дат и времени, создание пользовательских функций, обработка форм, реализация защищенных паролем страниц, данные cookies и сеансы, работа с базами данных, настройка PHP, отладка и обработка ошибок.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Sams Publishing.

Authorized translation from the English language edition published by Sams Publishing, Copyright © 2005.

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 is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2006.

ISBN 5-8459-0937-6 (рус.)
ISBN 0-672-32762-7 (англ.)

© Издательский дом "Вильямс", 2006
© by Sams Publishing, 2005

Оглавление

Введение 14
Урок 1. Знакомство с PHP 19
Урок 2. Переменные 27
Урок 3. Управление порядком выполнения 35
Урок 4. Функции 45
Урок 5. Работа с числами 53
Урок 6. Обработка строк 61
Урок 7. Работа с массивами 71
Урок 8. Регулярные выражения 81
Урок 9. Работа с временем и датой 91
Урок 10. Использование классов 99
Урок 11. Обработка HTML-форм 107
Урок 12. Динамическая генерация HTML-кода для форм 117
Урок 13. Проверка форм 127
Урок 14. Данные cookies и сеансы 135
Урок 15. Аутентификация пользователя 143
Урок 16. Взаимодействие с Web-сервером 153
Урок 17. Работа с файловой системой 161
Урок 18. Выполнение программ на Web-сервере 171
Урок 19. Использование базы данных MySQL 179
Урок 20. Абстрагирование от базы данных 189
Урок 21. Выполнение PHP-сценариев в командной строке 199
Урок 22. Обработка ошибок 207
Урок 23. Настройка PHP 215
Урок 24. Безопасность при использовании PHP 225
Урок 25. Использование PEAR 233
Приложение А. Установка PHP 241
Предметный указатель 247

Содержание

Введение 14
Урок 1. Знакомство с PHP 19
Основы PHP 19
Написание серверных сценариев 20
Дескрипторы PHP 21
Первый сценарий 22
Команда `echo` 23
Комментарии 25
Резюме 26
Урок 2. Переменные 27
Разберемся в переменных 27
Выбор имени для переменной 28
Выражения 29
Переменные в строках 29
Типы данных 31
Приведение типа 32
Переменная переменной 33
Резюме 33
Урок 3. Управление порядком выполнения 35
Условный оператор 35
Операторы сравнения 37
Логические операторы 37
Множественное условное ветвление 38
Выражение `switch` 40
Циклы 41
Цикл `while` 41
Цикл `do` 42
Цикл `for` 42
Вложенные условия и циклы 43
Обрыв выполнения цикла 43
Резюме 43
Урок 4. Функции 45
Применение функций 45
Определение функции 46
Аргументы и возвращаемые значения 47
Удачный и неудачный результат выполнения 48
Необязательные аргументы 49
Область видимости переменных 50
Использование библиотечных файлов 51

Подключение библиотечных файлов 51
Резюме 52
Урок 5. Работа с числами 53
Арифметика 53
Арифметические операторы 53
Операторы инкремента и декремента 54
Комбинированные операторы 55
Приоритет операторов 55
Числовые типы данных 56
Что такое `NULL`? 57
Функции для работы с числами 57
Округление чисел 57
Сравнение 58
Случайные числа 59
Математические функции 59
Резюме 60
Урок 6. Обработка строк 61
Анатомия строки 61
Выделение специальных символов обратной косой чертой 61
Конкатенация 62
Сравнение строк 63
Форматирование строк 64
Возможности `printf` 64
Форматирующие коды 65
Возможности `sprintf` 67
Строковые функции 67
Переключение регистра 68
Разбивка строки 68
Резюме 70
Урок 7. Работа с массивами 71
Что такое массив 71
Создание массива и доступ к нему 71
Вывод содержимого массива 72
Прохождение по массиву 73
Ассоциативный массив 74
Функции для работы с массивами 75
Сортировка 75
Перетасовка массива случайным образом 76
Функции для работы с множествами 77
Внутри массива 77
Сериализация 78
Многомерный массив 79
Доступ к двухмерному массиву 79

Определение многомерного массива 79
Резюме 80
Урок 8. Регулярные выражения 81
Введение в регулярные выражения 81
Типы регулярных выражений 81
Возможности ereg 82
Соответствие набору символов 82
Общие классы символов 83
Проверка положения 84
Поиск с помощью символов подстановки 85
Повторяющиеся шаблоны 85
Несколько практических примеров 86
Разбивка строки на компоненты 88
Поиск и замена 89
Резюме 90
Урок 9. Работа с временем и датой 91
Форматы дат 91
Выбор формата данных 91
Формат временной метки Unix 92
Работа с временной меткой 93
Форматирование даты 93
Перевод в формат временной метки 94
Перевод в формат временной метки 96
Получение информации из значения временной метки 96
Резюме 97
Урок 10. Использование классов 99
Объектно-ориентированное программирование на PHP 99
Что такое класс 100
Когда использовать классы 100
На что похожи классы 101
Создание и использование объектов 101
Методы и свойства 102
Использование классов сторонних разработчиков 103
Резюме 105
Урок 11. Обработка HTML-форм 107
Отправка данных из формы в PHP 107
Дескриптор <FORM> 107
Дескриптор <INPUT> 108
Дескриптор <TEXTAREA> 110
Дескриптор <SELECT> 110
Соберем все вместе 110
Обработка форм с помощью PHP 112

Доступ к значениям формы 112
Элемент скрытого ввода 114
Сценарий для отправки электронной почты 115
Функция mail 115
Резюме 116
Урок 12. Динамическая генерация HTML-кода для форм 117
Установка значений по умолчанию 117
Стандартные значения для ввода 117
Установка атрибута CHECKED для типа CHECKBOX 118
Установка переключателя 119
Установка стандартного значения для раскрывающегося списка 121
Создание элементов формы 122
Создание динамической группы переключателей 122
Создание динамического раскрывающегося списка 123
Элемент множественного выбора 124
Резюме 126
Урок 13. Проверка форм 127
Требование нужных полей 127
Вывод ошибок проверки 128
Обязательные правила для данных 130
Подсветка полей с ошибками 131
Резюме 133
Урок 14. Данные cookies и сеансы 135
Файлы Cookies 135
Составляющие cookies 135
Доступ к значениям cookies 137
Создание данных cookies с помощью PHP 138
Удаление cookies 139
Сеансы 139
Открытие сеанса 140
Использование переменных сеанса 141
Резюме 141
Урок 15. Аутентификация пользователя 143
Типы аутентификации 143
Базовая HTTP-аутентификация 143
Аутентификация с помощью сеансов 145
Создание системы аутентификации 146
Как работает система 146
Аутентификация пользователя 147
Шифрование паролей 149
Анализ удобства использования 150
Резюме 151

Урок 16. Взаимодействие с Web-сервером 153
HTTP-заголовки 153
Отправка специальных заголовков 153
Заголовки перенаправления 154
Проверка факта отправки заголовков 154
Вывод HTTP-заголовков 156
Изменение настроек кэширования 156
Переменные окружения сервера 158
Информация о сценарии 158
Информация о пользователе 159
Информация о сервере 159
Резюме 160
Урок 17. Работа с файловой системой 161
Управление файлами 161
Права доступа к файлам 161
Получение информации о файле 162
Перемещение и копирование файлов 164
Работа с именами файлов 164
Чтение и запись файлов 165
Простые методы для чтения и записи файлов 165
Низкоуровневый доступ к файлу 165
Произвольный доступ к файлу 167
Запись с помощью указателя файла 168
Работа с файлами данных 168
Работа с URL 169
Работа с каталогами 169
Резюме 170
Урок 18. Выполнение программ на Web-сервере 171
Выполнение локальных программ 171
Функция passthru 171
Применение апострофа 172
Создание командной строки 173
Окружение сервера 174
Определение платформы сервера 174
Переменные окружения 174
Часовые пояса 175
Анализ безопасности 176
Разделение обратной косой чертой команд оболочки 177
Резюме 178
Урок 19. Использование базы данных MySQL 179
Использование MySQL 179
Соединение с базой данных MySQL 180
Выполнение SQL-запросов 181
Команды, которые изменяют базу данных 181

Извлечение полученных данных 182
Получение всей строки из данных 184
Отладка SQL 185
Ошибки SQL 185
Ошибки соединения 186
Резюме 187
Урок 20. Абстрагирование от базы данных 189
Класс PEAR DB 189
Установка класса DB 190
Названия источников данных 191
Использование класса DB 192
Выполнение запросов 193
Извлечение выбранных данных 193
Сокращение вызовов 195
Совместимость между базами данных 195
Режим совместимости 196
Работа с кавычками 197
Последовательности 197
Ограничение запроса 198
Резюме 198
Урок 21. Выполнение PHP-сценариев в командной строке 199
Среда командной строки 199
Отличие между исполняемыми файлами CLI и CGI 199
Написание сценариев PHP для оболочки Linux/Unix 200
PHP-сценарии командной строки для Windows 202
Вставка PHP-кода 202
Написание сценариев для командной строки 202
Режим вывода символов 203
Аргументы командной строки 203
Потоки ввода-вывода 204
Создание настольных приложений 206
Резюме 206
Урок 22. Обработка ошибок 207
Система уведомления об ошибках 207
Изменение уровня ошибок 207
Специальный обработчик ошибок 209
Генерирование пользовательской ошибки 211
Сохранение ошибок 212
Подавление ошибок и предупреждений 213
Оператор подавления ошибки 213
Предотвращение показа ошибок 214
Резюме 214

Урок 23. Настройка PHP 215
Настройки конфигурации 215
Возможности php.ini 215
Замена файла php.ini 217
Настройка для отдельного каталога 217
Динамическая настройка 218
Директивы настройки 218
Настройка PHP-окружения 218
Настройка расширений PHP 223
Настройка безопасности системы 223
Подгружаемые модули 223
Загрузка расширения по требованию 223
Загрузка модулей при старте 224
Резюме 224
Урок 24. Безопасность при использовании PHP 225
Безопасный режим 225
Ограничения, налагаемые безопасным режимом 225
Активизация безопасного режима 227
Другие возможности безопасности 228
Сокрытие PHP 228
Безопасность файловой системы 229
Контроль доступа к функциям 230
Безопасность базы данных 230
Резюме 232
Урок 25. Использование PEAR 233
Введение в PEAR 233
Библиотека кодов PEAR 233
Распространение и сборка пакетов 234
Стандарты кодирования PEAR 234
Важнейшие классы PHP 235
Оперативная поддержка по PEAR 235
Использование PEAR 235
Поиск пакета PEAR 235
Использование инсталлятора PEAR 236
Как сделать собственный вклад в проект PEAR 238
Резюме 239
Приложение А. Установка PHP 241
Установка на Linux/Unix 241
Сборка Apache из исходных кодов 241
Компиляция и установка PHP 243
Установка на платформе Windows 245
Установка Apache 245
Установка PHP 246
Решение проблем 246
Предметный указатель 247

Об авторе

Крис Ньюман (Chris Newman) работает программистом-консультантом и специализируется на разработке Web-ориентированых приложений с использованием баз данных для клиентов по всему миру.

Крис — выпускник Кильского университета. Проживает в Сток-он-Тренте (Stoke-on-Trent), Англия. В 1999 году основал фирму Lightwood Consultancy, Ltd. для работы в области Internet-технологий. Lightwood предоставляет услуги Web-хостинга под маркой DataSnake. Кроме того, компания первой включила поддержку SQLite в PHP как стандартную возможность для всех пользователей.

Подробности о компании Lightwood Consultancy Ltd. можно найти по адресу: `www.lightwood.net`, а с Крисом можно связаться по адресу: `chris@lightwood.net`.

Введение

Эта книга о PHP — самом популярном языке сценариев для Web. Пособие ориентировано на занятых людей, поэтому проработка одного урока займет не больше 10 минут. Так что, если вы хотели изучить PHP, а раньше не было возможности, приступайте прямо сейчас.

Для кого предназначена эта книга

Книга предназначена для тех, кто хочет изучить PHP; при этом не требуется опыт программирования или написания сценариев. Если вы не делали этого раньше, PHP может стать вашим первым языком программирования.

Те, кто уже имеют опыт написания программ, но никогда не программировали приложения для Web, смогут по этой книге изучить PHP и научиться разрабатывать Internet-приложения.

В этой книге нет информации об HTML. Знание основ HTML облегчит вам понимание примеров, но не является обязательным.

Как организована эта книга

Книга состоит из пяти частей.

Часть I. Основы PHP

В первой части дается описание основных конструкций языка PHP.

- **Урок 1. Знакомство с PHP.** В этом уроке рассказывается, для чего предназначен PHP, и приводится несколько примеров, которые демонстрируют его применение внутри Web-страницы.
- **Урок 2. Переменные.** В этом уроке объясняется, как присвоить значение переменной, и приводится несколько простых примеров.

- **Урок 3. Управление порядком выполнения.** В этом уроке рассматриваются условные операторы и циклы, с помощью которых можно управлять порядком выполнения PHP-сценария.
- **Урок 4. Функции.** В этом уроке показывается, как реализовать модульность, чтобы вынести дублирующиеся части кода в функцию.

Часть II. Работа с данными

В этой части описаны типы данных, с которыми может работать PHP.

- **Урок 5. Работа с числами.** В этом уроке даются подробные примеры операций с числами, которые можно выполнять в PHP.
- **Урок 6. Обработка строк.** В этом уроке рассматривается набор функций для обработки строк в PHP.
- **Урок 7. Работа с массивами.** В этом уроке поясняется, как работать с массивами, и рассматриваются функции, которые позволяют выполнять действия над ними.
- **Урок 8. Регулярные выражения.** В этом уроке рассматриваются регулярные выражения, которые позволяют выполнять сложные операции со строками.
- **Урок 9. Работа с временем и датой.** В этом уроке показывается, как обрабатывать время и дату в PHP.
- **Урок 10. Использование классов.** В этом уроке демонстрируются возможности объектно-ориентированного PHP и приводится пример использования класса в сценарии.

Часть III. Web-среда

В этой части показывается, как с помощью PHP выполнять разработку в Web-окружении.

- **Урок 11. Обработка HTML-форм.** В этом уроке описывается, как использовать PHP для обработки данных, отправленных пользователем из HTML-формы.
- **Урок 12. Динамическая генерация HTML-кода для форм.** В этом уроке рассматриваются способы создания HTML-компонент "на лету" с помощью PHP.

- **Урок 13. Проверка форм.** В этом уроке рассматриваются способы проверки формы на корректность данных, отправленных из HTML-форм.
- **Урок 14. Данные cookies и сеансы.** В этом уроке показывается, как передавать данные между Web-страницами с помощью сеансов и как отправить данные cookies браузеру пользователя.
- **Урок 15. Аутентификация пользователя.** В этом уроке рассматриваются способы защиты пользовательских страниц с помощью пароля.
- **Урок 16. Взаимодействие с Web-сервером.** В этом уроке рассматривается механизм взаимодействия PHP с Web-сервером.

Часть IV. Использование служб с помощью PHP

В четвертой части рассматриваются варианты работы PHP с внешними программами и службами.

- **Урок 17. Работа с файловой системой.** В этом уроке рассматриваются функции PHP для работы с файловой системой.
- **Урок 18. Выполнение программ на Web-сервере.** В этом уроке рассматриваются функции PHP для выполнения программ на Web-сервере.
- **Урок 19. Использование базы данных MySQL.** В этом уроке показывается, как сохранять и извлекать данные из базы данных MySQL.
- **Урок 20. Абстрагирование от базы данных.** В этом уроке показывается, как реализовать обобщенный интерфейс для доступа к базе данных с помощью уровня абстракции, чтобы сделать сценарий более переносимым.
- **Урок 21. Выполнение PHP-сценариев в командной строке.** В этом уроке показано, как использовать PHP для написания мощных сценариев оболочки.
- **Урок 22. Обработка ошибок.** В этом уроке обсуждается несколько способов нахождения и исправления ошибок сценария.

Часть V. Настройка и расширение PHP

В последней части книги обсуждаются вопросы администрирования PHP.

- **Урок 23. Настройка PHP.** В этом уроке описаны популярные конфигурационные опции, которые устанавливаются во время работы и могут изменять поведение PHP.
- **Урок 24. Безопасность PHP.** В этом уроке обсуждаются проблемы, связанные с безопасностью, и использование Safe Mode (безопасный режим) на общем Web-сервере.
- **Урок 25. Использование PEAR.** В этом уроке рассказывается о библиотеке свободно распространяемых классов — PEAR.

На какую версию PHP рассчитана эта книга

Во время написания этого пособия текущей версией PHP была PHP 5.0.3. Кроме специально оговоренных случаев, все примеры работают под PHP 4.1.0 и выше.

Соглашения, принятые в этой книге

В этой книге используются различные шрифты для кода и обычного текста. Важная информация выделяется особым образом.

Для кода в тексте и вывода на экран используется моноширинный шрифт.

```
Вот так будет выглядеть моноширинный текст,
который вы увидите на экране.
```

Места для подстановки переменных и выражений отображены *моноширинным курсивом*. Нужно заменять места для подстановки на значения, которые они представляют.



Пиктограмма Примечание указывает на интересную информацию по данной теме.



В тексте с пиктограммой Совет дается рекомендация о том, как лучше решить конкретную проблему.



Пиктограмма Внимание предупреждает о потенциальной опасности и поможет избежать проблем.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@williamspublishing.com

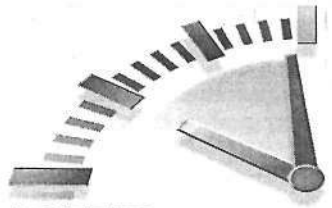
WWW: http://www.williamspublishing.com

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

Урок 1



Знакомство с PHP

В этом уроке вы узнаете, для чего применяется PHP и что он может делать.

Основы PHP

Эта книга дает отличный шанс разобраться в PHP; наверное, поэтому вы ее и купили. PHP — очень популярен, и не зря. Даже если никто не говорил вам о нем, вы точно посещали Web-сайты, которые используют PHP. В этом уроке рассказывается о том, что такое PHP, как он работает и что может делать.

PHP — язык программирования, созданный для разработки динамических Web-страниц. Его интерпретатор размещается на Web-сервере и обрабатывает инструкции, которые находит внутри Web-страницы, перед тем как отправить готовый HTML-код браузеру пользователя. Некоторые элементы страницы генерируются на лету, поэтому страница может меняться при каждой загрузке. Например, можно показывать текущее время в верхней части каждой Web-страницы сайта. О том, как это сделать, будет рассказано ниже в этом уроке.

Название PHP образовалось от рекурсивного акронима — *PHP: Hypertext Preprocessor* (обработчик гипертекста). Изначально он назывался PHP/FI, где "FI" значит *Forms Interpreter* (интерпретатор форм). Наиболее мощная возможность PHP — легкость обработки данных, отправленных пользователем из HTML-форм. Кроме того, PHP может обращаться к базам данных и генерировать страницу на основе SQL-запроса. Например, можно ввести поисковую фразу в поле формы на Web-странице, обратиться к базе данных с этим запросом и создать страницу с найденными



Пиктограмма Внимание предупреждает о потенциальной опасности и поможет избежать проблем.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать *название книги и ее авторов, а также ваш обратный адрес*. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@williamspublishing.com

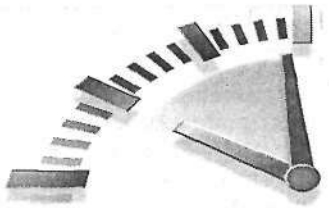
WWW: http://www.williamspublishing.com

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

Урок 1



Знакомство с PHP

В этом уроке вы узнаете, для чего применяется PHP и что он может делать.

Основы PHP

Эта книга дает отличный шанс разобраться в PHP; наверное, поэтому вы ее и купили. PHP — очень популярен, и не зря. Даже если никто не говорил вам о нем, вы точно посещали Web-сайты, которые используют PHP. В этом уроке рассказывается о том, что такое PHP, как он работает и что может делать.

PHP — *язык программирования*, созданный для разработки динамических Web-страниц. Его интерпретатор размещается на Web-сервере и обрабатывает инструкции, которые находит внутри Web-страницы, перед тем как отправить готовый HTML-код браузеру пользователя. Некоторые элементы страницы генерируются на лету, поэтому страница может меняться при каждой загрузке. Например, можно показывать текущее время в верхней части каждой Web-страницы сайта. О том, как это сделать, будет рассказано ниже в этом уроке.

Название PHP образовалось от рекурсивного акронима — *PHP: Hypertext Preprocessor* (обработчик гипертекста). Изначально он назывался PHP/FI, где "FI" значит *Forms Interpreter* (интерпретатор форм). Наиболее мощная возможность PHP — легкость обработки данных, отправленных пользователем из HTML-форм. Кроме того, PHP может обращаться к базам данных и генерировать страницу на основе SQL-запроса. Например, можно ввести поисковую фразу в поле формы на Web-странице, обратиться к базе данных с этим запросом и создать страницу с найденными

результатами. Такой механизм часто встречается на сайтах электронных магазинов или на поисковых серверах.

PHP — очень гибкий язык с нестрогой типизацией, что заметно облегчает его изучение, даже если вы не программировали раньше. Пользователи, которые уже знают другие языки программирования, найдут много сходств. PHP очень похож на смесь C, Perl, и Java. Тот, кто знает один из этих языков, сможет легко адаптировать наработанный стиль программирования к PHP.

Написание серверных сценариев

На начальной стадии изучения важно разобраться, где находится PHP в общей структуре Web-среды. Это позволит понять, что PHP может делать и чего не может.

Модуль PHP устанавливается на Web-сервере и дает указание, чтобы файлы с определенным расширением интерпретировались как код PHP. Весь PHP-код в сценарии выполняется на сервере и генерирует вывод, который отправляется на пользовательский браузер.



Расширение файла

Обычно сервер настроен так, что файлы *имя*.php интерпретируется как PHP, а файлы *имя*.html просто посылаются Web-браузеру пользователя, без предварительной обработки с помощью PHP.

Интерпретатор PHP вызывается только один раз во время загрузки страницы. Это происходит при переходе по ссылке, отправлении формы или когда набирается адрес страницы в окне браузера. После загрузки страницы PHP больше не выполняет действий до тех пор, пока браузер не запросит новую страницу.

Так как PHP работает на стороне сервера, его нельзя применять для проверки корректности ввода данных на стороне пользователя. То есть нельзя реализовать проверку отдельного поля формы на соответствие определенным критериям при переходе к другому полю. Проверка корректности на стороне пользователя реализуется с помощью языка JavaScript, поддерживаемого Web-браузером. В связке с PHP он помогает реализовать требуемую функциональность.

Красота PHP в том, что он не привязан к конкретному браузеру. Сценарий работает одинаково с любым браузером. При написании серверного сценария отпадает необходимость заботиться о поддержке JavaScript или о совместимости с более ранними версиями браузеров. От браузера требуется только поддержка HTML, который генерирует сценарий.

Дескрипторы PHP

Рассмотрим часть кода из реальной странички под управлением PHP, которая показывает текущую дату:

```
Сегодня <?php echo date('j F Y');?>
```

Дескриптор <?php указывает на то, что дальше следует PHP-код, а не HTML, и он заканчивается закрывающим дескриптором ?>. В этом примере команда date создает форматированный вывод текущей даты в виде: день, месяц и год, а команда echo выводит сгенерированную дату на экран.



Разделитель выражений

Точка с запятой используется для разделения команд PHP. В предыдущем примере была только одна инструкция, и точку с запятой можно опустить. Но хорошим стилем в PHP считается всегда ее ставить. Это указывает на то, что команда завершена.

В этой книге весь PHP-код заключен в дескрипторы <?php ... ?>. Но иногда можно встретить другой стиль дескрипторов: <? (короткий дескриптор) , <% (дескриптор в стиле ASP) или <SCRIPT LANGUAGE="php"> (дескриптор SCRIPT). В стандартной конфигурации только полный <?php и SCRIPT дескрипторы будут работать всюду. Остальные вариации включаются с помощью опций настройки PHP. Работа с конфигурационным файлом php.ini рассматривается в уроке 23, "Настройка PHP".

Все, что не включено в PHP-дескрипторы, передается браузеру в том же виде, как появляется в сценарии. Поэтому, в предыдущем примере текст Сегодня размещается перед сгенерированной датой.



Стандартные дескрипторы PHP

Хорошим стилем считается всегда использовать дескриптор `<?php`, потому что такой вариант будет работать на любой системе без дополнительной настройки сервера. Те, кто предпочитают использовать `<?` для сокращения, при переносе кода на другой сервер должны убедиться, что он поддерживает короткий стиль дескрипторов.

Первый сценарий

Перед тем как продолжить, нужно убедиться в том, что вы сможете создавать и запускать сценарии, которые приведены в этой книге. Это можно сделать даже на рабочем компьютере. Инструкция по установке PHP находится в приложении А, "Установка PHP". Кроме того, большинство компаний, предоставляющих Web-хостинг, включают PHP в пакет услуг, так что, возможно, у вас уже есть доступ к необходимой Web-среде.

Теперь создадим файл `time.php` и наберем в нем пример из листинга 1.1. Файл нужно разместить в области, доступной для Web-сервера с поддержкой PHP. Ниже приводится небольшая вариация предыдущего примера с датой.

Листинг 1.1. Вывод системного времени и даты

```
Текущее время
<?php echo date('H:i:s');?>
и дата
<?php echo date('j F Y');?>
```

В браузере введите URL файла с кодом. На экране появится текущая дата и время в соответствии с системными временем Web-сервера.



Локальный запуск PHP

Если запустить PHP на персональном компьютере, PHP-код в сценарии выполнится только в случае доступа через Web-сервер, на котором установлен модуль PHP. Если просто запустить сценарий непосредственно в браузере, например с помощью двойного щелчка мышью или перетаскиванием файла в окно браузера, он будет трактоваться как обычный HTML-код.



Размещение Web-документов

Если использовать установленный по умолчанию Web-сервер Apache, файл `time.php` нужно поместить в каталог `C:\Program Files\ApacheGroup\Apache\htdocs`, тогда правильный URL-адрес вашего файла будет `http://localhost/time.php`.

Сценарий из листинга 1.1 генерирует неудобный вывод — нет пробела между временем и союзом *и*. Любой перевод строки в сценарии, находящийся внутри дескрипторов PHP, не отображается в сгенерированном HTML коде.

Если задействовать опцию браузера View Source (Просмотр в виде HTML), то вывод будет таким:

```
Текущее время
15:33:09и дата
13 October 2004
```

Можно вставить символ пробела после `?>`. Тогда эта строка будет содержать не PHP-элемент, и вывод корректно разделится пробелом.

Команда echo

Вместо использования PHP для вставки маленьких элементов внутри Web-страницы, можно разместить сценарий внутри PHP-дескрипторов и генерировать страницу с помощью набора PHP-инструкций. Команда `echo` передает нужный вывод на браузер. В листинге 1.1 команда `echo` показывает результат выполнения команды `date`, которая возвращает строку, содержащую отформатированную текущую дату. В листинге 1.2 для достижения аналогичного

результата используется набор команд echo в одном блоке PHP-кода.

Листинг 1.2. Применение echo для отправки вывода на браузер

```
<?php
echo "Текущее время ";
echo date('H:i:s');
echo " и дата ";
echo date('j F Y');
?>
```

Все нединамические текстовые элементы, предназначенные для вывода, заключены в кавычки. Текстовые строки можно заключать как в двойные (листинг 1.2), так и в одинарные кавычки. Существенная разница между ними показывается в уроке 2, "Переменные". Следующие выражения полностью эквивалентны:

```
echo "Текущее время ";
echo 'Текущее время ';
```

Отметим, что символы пробела перед закрывающей кавычкой используются для того, чтобы отделить вывод функции date от окружающего текста. Кроме того, вывод листинга 1.2 немного отличается от вывода в листинге 1.1. Чтобы заметить эту разницу, нужно задействовать опцию View Source. Сценарий из листинга 1.2 непосредственно генерирует следующий вывод:

```
Текущее время 15:59:50 и дата 13 October 2004
```

В этом выводе нет символов перевода строки. Но вывод в браузере будет полностью идентичен выводу листинга 1.1. Дело в том, что HTML не делает различия между разделительными символами. Несколько пробелов подряд, перевод новой строки или табуляция всегда отображаются как одиночный пробел.

Символ перевода строки внутри PHP-кода не выводится на экран. Перевод строки используется для форматирования кода в удобочитаемый вид. Несколько коротких команд могут помещаться в одной строке, а длинная команда — занимать несколько строк. Именно поэтому для завершения команды ставится точка с запятой.

Листинг 1.3 аналогичен листингу 1.2, но он отформатирован так, что разобраться в нем очень сложно.

Листинг 1.3. Плохо отформатированный сценарий, выводящий дату и время

```
<?php echo "Текущее время "; echo date('H:i:s'); echo
" и дата "
; echo date(
'j F Y'
);
?>
```



Использование перевода строки

Для того чтобы отправить браузеру символ перевода строки, нужно использовать последовательность символов \n. Другие последовательности символов рассматриваются в уроке 6, "Обработка строк".

Комментарии

Удобочитаемость кода можно повысить при помощи комментариев. *Комментарий* — это часть свободного текста, которую можно разместить в любом месте сценария и которая полностью игнорируется PHP. Различные стили комментариев показаны в табл. 1.1.

Таблица 1.1. Стили комментариев в PHP

Комментарий	*Описание*
// или #	Однострочные комментарии. Все содержимое до конца строки игнорируется.
/* ... */	Одно- или многострочный комментарий. Все содержимое между символами /* и */ игнорируется.

Несмотря на обилие комментариев, листинг 1.4 генерирует одинаковый вывод с листингами 1.1, 1.2 и 1.3. Поскольку комментарии игнорируются интерпретатором PHP, вывод будет состоять только из времени и даты.

Листинг 1.4. Использование комментариев в сценарии

```
<?php
/* time.php
Этот сценарий выводит на экран браузера
```

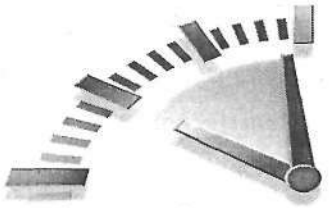
```
текущее время и дату
*/
echo "Текущее время ";
echo date('H:i:s'); // Часы, минуты, секунды
echo " и дата ";
echo date('j F Y'); // Название дня, месяца и текущий год
?>
```

В начале листинга 1.4 находится заголовочный блок комментариев с названием файла и коротким описанием сценария, а внутренние комментарии описывают вывод команды `date`.

Резюме

В этом уроке вы узнали, как работает PHP в Web-окружении и увидели как выглядит простой сценарий PHP. В следующем уроке вы научитесь использовать переменные.

Урок 2



Переменные

В этом уроке вы узнаете, как присвоить значение переменной в PHP, и задействуете это на практике.

Разберемся в переменных

Переменные — это ячейки, в которых можно сохранять информацию. Они являются фундаментальными конструкциями любой программы.

Возьмем переменную с именем `number`, в которой находится значение `5`, и переменную `name`, содержащую значение `Крис`. В следующем примере определяются эти переменные и присваиваются соответствующие значения:

```
$number = 5;
$name = "Крис";
```

Перед именем переменной в PHP ставится знак доллара. Объявить переменную очень просто. Для этого нужно поставить переменную и знак равенства слева, а присваиваемое значение — справа.



Объявление переменных

В отличие от некоторых языков программирования, PHP не требует объявления переменной перед присваиванием значения. Поэтому можно сразу присвоить значение переменной и использовать ее.

В PHP переменные можно подставлять всюду, где ставятся фиксированные значения. В следующем примере команда `echo` выводит на экран значение, которое находится

в переменной. Это аналогично вставке фрагмента фиксированного текста вместо переменной:

```
$name = "Крис";
echo "Привет, ";
echo $name;
```

На выходе получим:

```
Привет, Крис
```

Выбор имени для переменной

Для переменных следует подбирать осмысленные имена. Это позволяет быстро разобраться в сценарии спустя несколько месяцев после его написания.

Не называйте переменные `$a`, `$b` и т.п. Через некоторое время будет тяжело разобраться, что значит каждая буква. Хорошее имя переменной прямо указывает на значение, которое она содержит (например, `$price`, `$name`).



Зависимость от регистра

Имена переменных в PHP зависят от регистра. Например, переменные `$name` и `$Name` — разные и могут содержать различные значения в одном сценарии.

Имена переменных могут содержать только буквы латинского алфавита, цифры и символ подчеркивания. Имя переменной должно начинаться с буквы или подчеркивания. В табл. 2.1 приводится несколько примеров корректных и некорректных имен переменных.

Таблица 2.1. Примеры корректных и некорректных имен переменных

Корректные имена переменных	*Некорректные имена переменных*
`$percent`	`$pct%`
`$first_name`	`$first-name`
`$line_2`	`$2nd_line`



Использование подчеркиваний

Чтобы удобнее назвать переменную, содержащую несколько слов в имени, используют подчеркивания. Например, `$first_name` и `$date_of_birth` более удобочитаемы за счет подчеркиваний.

Другой способ разделения: написание первой буквы каждого слова в верхнем регистре. Например: `$FirstName` и `$DateOfBirth`. Используя этот стиль, нужно помнить, что регистр имеет значение.

Выражения

Переменным можно присваивать не только фиксированные значения но и *выражения*. *Выражение* — это два или больше значений, объединенных при помощи *операторов* для получения результата. Несмотря на простоту следующего примера, в тексте ниже поясняется, как он работает:

```
$sum = 16 + 30;
echo $sum;
```

Значение справа от знака равенства присваивается переменной `$sum`. Значения `16` и `30` соединяются оператором суммирования — знаком плюс (`+`). Результатом выражения будет сумма двух значений. Как и ожидалось, на выходе получим `46`.

Теперь вместо фиксированных значений, перейдем к переменным, чтобы обобщить пример выше:

```
$a = 16;
$b = 30;
$sum = $a + $b;
echo $sum;
```

Значения `$a` и `$b` суммируются, и снова на выходе получаем `46`.

Переменные в строках

Выше было сказано, что строки нужно заключать в кавычки и что существует большая разница между одинарными и двойными кавычками. В строках, заключенных в двойные кавычки, знак доллара внутри строки указывает на необходимость подставить значение соответствующей

переменной. С другой стороны, если в строке, заключенной в одинарные кавычки, встречается знак доллара — он интерпретируется как обычный символ и никаких подстановок делать не нужно.

Примеры ниже помогут разобраться в этом различии. В этом примере значение переменной $name подставляется в строку:

```
$name = "Крис";
echo "Привет, $name";
```

На выходе получим: Привет Крис.

В этом примере знак доллара интерпретируется как символ, и никаких подстановок не происходит:

```
$name = 'Крис';
echo 'Привет, $name';
```

На выходе получим: Привет, $name.

Иногда PHP нужно указать, где начинается и заканчивается переменная. Для этого применяются фигурные скобки ({}). Например, нужно указать единицы измерения — килограммы или граммы. Тогда выражение будет следующим:

```
echo "Полный вес {$weight}kg";
```

Без фигурных скобок вокруг $weight PHP будет искать переменную $weightkg, которой вообще может не существовать в сценарии.

Аналогичные операции выполняются с помощью оператора *конкатенации* (соединения) — знак точки, который объединяет несколько строк в одну. Ниже показывается этот способ:

```
echo 'Полный вес ' . $weight . 'kg';
```

Здесь три значения — две фиксированные строки и переменная $weight — соединяются вместе. Порядок соединения соответствует порядку появления в выражении. Пробел после слова вес отделяет его от значения переменной $weight.

Если в $weight находится значение 99, на выходе получим:

```
Полный вес 99kg
```


Типы данных

Каждая переменная, в зависимости от значения, которое она содержит, имеет определенный тип. Основные типы данных PHP показаны в табл. 2.2.

Таблица 2.2. Типы данных PHP

Тип данных	*Описание*
boolean (булев)	Значение истинности, может быть TRUE (истина) или FALSE (ложь)
integer (целый)	Численное значение, может быть положительным или отрицательным целым числом
float (одинарной точности с плавающей точкой) или double (двойной точности с плавающей точкой)	Число с плавающей точкой. Может быть любое десятичное число
string (строчный)	Буквенно-цифровое значение. Может содержать любое количество ASCII-символов

В момент присваивания значения переменной сразу устанавливается и ее тип. PHP определяет тип переменной автоматически, на основе присваиваемого значения. Функция gettype позволяет узнать тип данных, который PHP установит для переменной.

Вывод следующего кода показывает, что тип десятичного числа double:

```
$value = 7.2;
echo gettype($value);
```

Функцию gettype дополняет функция settype. Она дает возможность менять тип переменной. Если значение переменной несовместимо с новым типом, оно изменится на ближайшее возможное.

В следующем примере строчный тип меняется на целый:

```
$value = "22ое Января 2005";
settype($value, "integer");
echo $value;
```

В этом примере строка начинается с числа, но в целом не является им. Во время преобразования PHP выбирает

все до первого нецифрового символа. Остальная часть отбрасывается, так что на выходе получим число 22.



Анализ типа данных

Обычно функции settype и gettype практически не используются. Очень редко возникает необходимость манипулировать типами переменной. Как упоминалось выше, PHP автоматически устанавливает тип данных для каждой переменной.

Приведение типа

Если ожидается определенный тип значения, PHP выполняет неявное изменение типа данных. Такой механизм называется *приведением типа*. Например, оператор суммы должен находиться между двумя числами. Поэтому перед выполнением операции строки конвертируются в integer или double. Сумма следующего выражения дает целый результат:

```
echo 100 + "10 дюймов";
```

В этом примере суммируются 100 и 10 — на экране получим 110.

Аналогичная ситуация повторяется, когда над числовыми данными выполняется строковая операция. Если выполнить строковую операцию над числовым значением, число сначала конвертируется в строку. Фактически это и произошло, когда мы применяли оператор конкатенации в примере выше. Значение $weight, которое выводилось, было числовым.

Результатом строковой операции всегда будет строка, даже если она выглядит как число. В следующем примере на выходе получим 69, но функция gettype показывает что в переменной $number находится строчное значение:

```
$number = 6 . 9;
echo $number;
echo gettype(6 . 9);
```

Множество мощных функций и операторов для работы с числами и строками в PHP будет показано в уроке 5, "Работа с числами", и в уроке 6, "Обработка строк".

Переменная переменной

В PHP есть механизм, который позволяет использовать значение, сохраненное в одной переменной, как имя для другой. На первый взгляд не очень понятно, но пример ниже наглядно демонстрирует этот механизм:

```
$my_age = 21;
$varname = "my_age";
echo "Значение $varname равно ${$varname}";
```

На выходе получим:

```
Значение my_age равно 21
```

Так как строка заключена в двойные кавычки, знак доллара сигнализирует о том, что нужно подставить значение переменной в строку. Конструкция ${$varname} указывает на то, что значение переменной, имя которой содержится в $varname, нужно подставить в строку. Такой механизм называется *переменная переменной*.

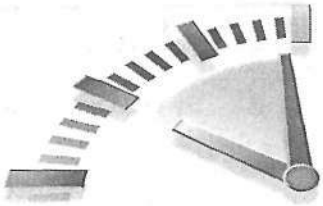
Фигурные скобки вокруг $varname используются для того чтобы получить имя переменной. Такая запись нужна для строк с двойными кавычками, в других случаях фигурные скобки можно опустить. В следующем примере на выходе получим то же, что и выше, но уже с использованием оператора конкатенации:

```
echo 'Значение ' . $varname . ' равно ' . $$varname;
```

Резюме

В этом уроке вы узнали, как работают переменные в PHP. В следующем уроке вы научитесь использовать условные операторы и циклы для управления порядком выполнения сценария.

Урок 3



Управление порядком выполнения

В этой главе вы научитесь управлять порядком выполнения сценария с помощью циклов и условных операторов.

В этой главе рассматриваются два типа структур для управления сценарием: условные операторы, которые позволяют выполнять определенную часть кода при выполнении заданного условия, и циклы, которые позволяют выполнять блок кода несколько раз.

Условный оператор

Условный оператор в PHP состоит из ключевого слова `if` и условия в круглых скобках. В следующем примере проводится проверка того, что `$number` меньше `10`, и только в этом случае оператор `echo` выводит на экран сообщение:

```
$number = 5;
if ($number < 10) {
    echo "$number меньше десяти";
}
```

Условие `$number < 10` удовлетворяется только в случае, если значение слева от символа `<` меньше значения справа. Если условие истинно, — выполняется код в фигурных скобках. Иначе сценарий перейдет к следующему выражению после закрывающей фигурной скобки.



Булевы значения

Любое условное выражение возвращает булево значение. Оператор `if` использует значения `TRUE` и `FALSE`, чтобы определить, нужно ли выполнять следующий блок. Все нулевые значения рассматриваются как `FALSE`, а все ненулевые — как `TRUE`.

Предыдущее выражение возвращает `TRUE`, потому что `5` меньше `10`. В результате выполняется код в фигурных скобках, и генерируется соответствующий вывод. Если заменить значение переменной `$number` на 10 или выше и перезапустить сценарий, условие не выполнится и на выходе будет пусто.

Фигурные скобки используются для группирования блока кода. В условном выражении они окружают код, который выполняется, если условное выражение истинное.



Различные скобки

В сценариях PHP встречаются три типа скобок: круглые (`()`), фигурные (`{}`) и квадратные (`[]`).

Необязательно ставить фигурные скобки после условного оператора `if`. Если они опущены, а условие истинно, выполняется первое после оператора `if` выражение. Все последующие выражения выполняются независимо от истинности условия оператора `if`.



Фигурные скобки и отступы

Несмотря на то что PHP не требует отступов, лучше поставить несколько пробелов перед выражениями внутри фигурных скобок. Это визуально отделяет блок кода от других выражений.

Даже если условный оператор или цикл используется для одного выражения, лучше ставить фигурные скобки. Это повышает читабельность кода, когда используется несколько вложенных конструкций.

Операторы сравнения

В PHP существует много операторов для сравнения двух значений. Операторы сравнения PHP показаны в табл. 3.1.

Таблица 3.1. Операторы сравнения PHP

Оператор	*Описание*
==	Равно
===	Строгое равенство (равенство типа и значения)
!=	Не равно
!==	Нестрогое равенство
<	Меньше чем
<=	Меньше или равно
>	Больше чем
>=	Больше или равно



= или ==?

Нужно осторожно использовать символ двойного равенства (==). Одиночный символ = всегда значит присваивание, и условие будет истинным, если присваиваемое значение истинно. Кроме того, нужно помнить, что `TRUE` — любое ненулевое значение. В большинстве случаев для сравнения лучше использовать ==.

Логические операторы

Чтобы проверить одновременное выполнение ряда условий, можно комбинировать несколько критериев в одном условном выражении. В примере ниже проверяется, находится ли `$number` между `5` и `10`:

```
$number = 8;
if ($number >= 5 and $number <= 10) {
    echo "$number не меньше пяти и не больше десяти";
}
```

Ключевое слово `and` — это *логический оператор*, который принимает истинное значение только в случае, когда выражения слева и справа истинны. То есть `$number` должен быть больше или равен `5` и меньше или равен `10`.

Таблица 3.2. Логические операторы в PHP

Оператор	*Название*	*Описание*
`! a`	NOT (отрицание)	Истина, если *a* ложь
`a && b`	AND (И)	Истина, если и *a* и *b* истинны
`a || b`	OR (ИЛИ)	Истина, если *a* или *b* истинны
`a and b`	AND (И)	Истина, если и *a* и *b* истинны
`a xor b`	XOR (исключающее ИЛИ)	Истина, если *a* или *b* истинны, но не одновременно
`a or b`	OR (ИЛИ)	Истина, если *a* или *b* истинны

Есть два способа выполнить логическое И и ИЛИ в PHP. Разница между `and` и `&&` (а также между `or` и `||`) в приоритете при выполнении вычислений. В табл. 3.2 операторы перечислены в порядке снижения приоритета. Следующие выражения очень похожи, но выполняют различные операции:

```
a or b and c
a || b and c
```

В первом условии `and` имеет высший приоритет и выполняется первым. В целом условие будет истинным если *a* истинно или *b* и *c* истинны. В следующем условии `||` имеет высший приоритет. Поэтому, чтобы получить истинное значение, *c* должно быть истинным, а также *a* или *b*.



Операторные символы

Логические операторы И и ИЛИ состоят из двойных символов `&&` и `||`, соответственно. В одиночном варианте эти символы имеют другое значение. Это будет рассмотрено в уроке 5, "Работа с числами".

Множественное условное ветвление

С помощью оператора `else` в выражении `if` можно задать альтернативное действие, если условие не удовлетворяется. В следующем примере устанавливается, больше или меньше десяти значение переменной `$number`:


```
$number = 16;
if ($number < 10) {
    echo "$number меньше десяти";
}
else {
    echo "$number больше десяти";
}
```

Оператор `else` позволяет реализовать альтернативное поведение для условного оператора. Чтобы добавить несколько ответвлений, используется ключевое слово `elseif`. Этот оператор проверяет следующее условие, если предыдущее не сработало.

В следующем примере функция `date("H")` возвращает текущий час — число от 0 до 23. На экране получим соответствующее приветствие:

```
$hour = date("H");
if ($hour < 12) {
    echo "Доброе утро";
}
elseif ($hour < 17) {
    echo "Добрый день";
}
else {
    echo "Добрый вечер";
}
```

На выходе получим `Доброе утро`, если время на сервере за полночь и не больше 11:59. `Добрый день`, если — за полдень и не больше пяти вечера. `Добрый вечер` — от пяти вечера и больше.

Как видно, в `elseif` проверяется, чтобы `$hour` не превышало 17. Но нет проверки на полный диапазон между 12 и 17. Так сделано потому, что `if` выполняет проверку на меньше 12, т.е. программа не дойдет до блока `elseif`, если `$hour` меньше 12.

Если все условия нарушаются, выполнится код в блоке `else`. Для значений `$hour` 17 и выше в условиях `if` и `elseif` получим ложное значение.



`elseif` в сравнении с `else if`

В PHP можно написать `elseif` как два слова: `else if`. Интерпретатор PHP обрабатывают их по-разному, но поведение будет одинаковым.

Выражение switch

Конструкция if позволяет разместить произвольное количество операторов elseif, но такая структура очень запутанная. Выражение switch имеет более компактный вид и является альтернативой условному выражению с множеством ветвлений. В примере ниже выражение switch проверяет значение $name на принадлежность к одному из списков:

```
switch ($name) {
    case "Дима":
    case "Таня":
        echo "Привет, $name, ты мой друг";
        break;
    case "Адольф":
    case "Саддам":
        echo "Ты мне не друг, $name";
        break;
    default:
        echo "Я тебя не знаю, $name";
}
```

Каждое case-выражение задает значение, для которого будет выполняться соответствующий код. Если присвоить имя переменной $name и запустить сценарий, на экране браузера появится приветствие другу, если ваше имя Дима или Таня. Но если вы Адольф или Саддам, вас опознают как недруга. С другим именем программа не узнает вас.

В операторе switch может быть любое количество case-выражений. Если тестируемое значение (в нашем случае переменная $name) соответствует одному из них, весь PHP-код после выражения case будет выполняться до первой команды break.



Остановка выполнения

Оператор break играет существенную роль в операторе switch. Когда выполняется условие оператора case, PHP-код будет выполняться дальше, включая следующее выражение case, которое будет проверять другое значение. В отдельных случаях такое поведение может понадобиться, но обычно ожидается другое. Поэтому нужно ставить оператор break после каждого выражения case.

При других значениях $name выполняется блок кода выражения default. Как и else, default является необязательным и выполняется, только если выше не обнаружено ничего подходящего.

Циклы

В PHP есть три типа циклов. Они позволяют выполнять блок кода определенное количество раз, но имеют некоторые различия.

Цикл while

После ключевого слова while следует условие в круглых скобках, а за ним — блок кода (тело цикла). Тело цикла будет выполняться до тех пор, пока условие в скобках будет истинным. Если условие станет ложным, цикл завершится, и тело цикла перестанет выполняться.



Бесконечный цикл

Условие цикла должно изменяться в теле таким образом, чтобы в определенный момент нарушить его. В противном случае цикл может выполняться бесконечно.

Ниже приводится пример, где в цикле while подсчитывается квадрат чисел от 1 до 10:

```
$count = 1;
while ($count <= 10) {
    $square = $count * $count;
    echo "Квадрат $count равен $square <br>";
    $count++;
}
```

Вначале переменная-счетчик $count инициализируется единицей. В теле цикла while подсчитывается квадрат числа, и увеличивается значение переменной $count. Оператор ++ (инкремент) увеличивает значение переменной, за которой он стоит, на единицу.

Цикл будет повторяться до тех пор, пока выполняется условие $count <= 10, поэтому на экране получим квадрат всех чисел от 1 до 10.

Цикл do

Цикл do отличается от while тем, что условие находится после тела цикла. Из-за этого отличия цикл do выполняется как минимум один раз, даже если условие изначально ложно.

Следующий пример цикла do выводит числа и их квадраты от одного до десяти:

```
$count = 1;
do {
    $square = $count * $count;
    echo "Квадрат $count равен $square <br>";
    $count++;
} while ($count <= 10);
```

Цикл for

Конструкция for позволяет создавать циклы в более компактной форме. В следующем примере на выходе получим то же, что и в двух предыдущих:

```
for ($count = 1; $count <= 10; $count++) {
    $square = $count * $count;
    echo "Квадрат $count равен $square <br>";
}
```

Как видно, конструкция for позволяет использовать значительно меньше кода, чтобы выполнять действия аналогичные while и do.

Выражение for состоит из трех частей, разделенных точкой с запятой:

- Первая часть выражения вычисляется только один раз перед началом выполнения цикла. В предыдущем примере в этой части инициализирована переменная $count.
- Вторая часть содержит условие. Если условие истинно, выполняется тело цикла. Как и в случае цикла while, при нарушении условия, тело цикла больше не выполняется.
- Третья часть выражения вычисляется каждый раз в конце каждого витка (итерации) цикла. В предыдущем примере, $count увеличивается на единицу после вывода очередной строки.

Вложенные условия и циклы

До этого рассматривались только простые примеры циклов и условий. Но применение управляющих конструкций этим не ограничивается. Можно вкладывать циклы и условные операторы друг в друга и получать сложные правила для управления порядком выполнения сценария.



Помните про отступы

Чем сложнее выражения используются в сценарии, тем важнее делать отступы в коде. Отступы позволяют понять, к какой структуре принадлежит соответствующий блок кода.

Обрыв выполнения цикла

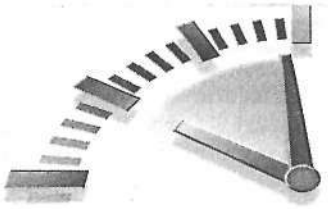
Выше оператор break использовался в выражении switch. Кроме того, оператор break позволяет завершить выполнение цикла и продолжить выполнение сценария.

Ключевое слово continue позволяет завершить текущую итерацию цикла. Но, в отличие от break, сценарий переходит на начало цикла и продолжает выполнение, пока не нарушится условие цикла.

Резюме

В этом уроке вы узнали, как управлять порядком выполнения сценария с помощью условных операторов и циклов. В следующем уроке вы узнаете о том, как создавать повторно используемые функции из блоков PHP-кода.

Урок 4



Функции

В этой главе вы узнаете, как вынести часто используемую часть кода в отдельную функцию.

Применение функций

Функция используется для того, чтобы заменить часть кода на процедуру, которую можно вызвать одной инструкцией. В PHP есть много готовых функций для выполнения различных задач. Некоторые встроены в PHP, а более специализированные доступны, только если установить соответствующее расширение при установке PHP.

Электронный справочник по PHP (`www.php.net`) содержит много полезной информации. Кроме отличной документации в нем есть комментарии и советы пользователей. При желании можно самому прокомментировать определенный раздел.



Электронный справочник

Для того чтобы быстро получить справку по функции, нужно набрать в браузере адрес: `www.php.net/`*`имя_функции`*.

Раньше рассматривалось применение функции `date` для того, чтобы сгенерировать форматированную текущую дату. Рассмотрим внимательно пример из урока 1, "Знакомство с PHP". Он выглядит так:

```
echo date('j F Y');
```

В электронном справочнике по PHP приводится следующий прототип этой функции:

```
string date (string format [, int timestamp])
```

Это значит что функции date передается строчный аргумент format и необязательный целый аргумент timestamp. Функция возвращает строчный тип. В этом примере передается j F Y как форматирующий аргумент, но нет аргумента timestamp. Команда echo выводит результат действия функции.



Прототипы

Каждая функция имеет заданный прототип. Он определяет, сколько и какого типа аргументы нужно передать, и что будет на выходе. Необязательные параметры заключаются в квадратные скобки ([]).

Определение функции

Кроме встроенных функций, PHP позволяет задавать собственные. Такой механизм имеет ряд преимуществ. Сокращается количество кода, который нужно набирать, повышается гибкость и понятность сценария. Если нужно изменить поведение функции, исправление вносится в месте определения функции, а не во всех местах, где используется этот код.



Модульный код

Объединение задач в отдельные функции позволяет повысить *модульность* кода. При увеличении размера сценария, его легче изменять.

В следующем примере показывается, как задать и использовать функцию в PHP:

```
function add_tax($amount) {
    $total = $amount * 1.09;
    return $total;
}
$price = 16.00;
echo "Цена без налога: $price <br>";
echo "Цена с налогом: ";
echo add_tax($price);
```

Ключевое слово function определяет функцию add_tax, которая выполняет код в фигурных скобках (*тело функции*). Тело функции обязательно нужно заключать в фигурные скобки. Определение функции add_tax требует одного аргумента $amount, который заключен в круглые скобки. Переменная $amount содержит переданный аргумент и доступна в теле функции.

В первой строке кода производится обычное перемножение $amount на 1.09. Это эквивалентно добавлению 9% к начальному значению. Результат сохраняется в переменной $total. За ключевым словом return ставится значение, которое функция возвращает как результат. После запуска этого сценария на выходе получим:

```
Цена без налога: 16
Цена с налогом: 17.44
```

Этот пример показывает, как использовать функцию в разных местах Web-страницы. Например, нужно вывести список товаров, которые есть на складе. Для этого вызывается функция, чтобы показать цену с учетом налога. Если процент налога изменяется, достаточно поменять коэффициент в функции add_tax, и все цены автоматически изменятся на необходимую величину.

Аргументы и возвращаемые значения

При вызове функции в круглых скобках указывается список аргументов. Если аргументов больше одного, нужно разделять их запятыми. В некоторых функциях аргументов вообще может не быть, но круглые скобки все равно ставятся после имени функции.

С помощью встроенной функции phpinfo можно получить информацию о конфигурации и модулях PHP. Эта функция не требует аргументов, поэтому ее можно вызывать так:

```
<?php phpinfo();?>
```

Если создать такой сценарий и зайти на соответствующую страницу, на экране появится системная информация и настройки PHP.

Удачный и неудачный результат выполнения

Функция phpinfo генерирует собственный вывод, поэтому не нужно ставить echo. Правда, из-за этого нельзя присвоить генерируемый вывод переменной. Кроме того, phpinfo всегда возвращает 1.



Возвращение True или False

Функции, которые не возвращают результат, используют определенный код, чтобы известить об успешности выполнения операции. Обычно нулевое значение (FALSE) сигнализирует о неудаче, а ненулевое (TRUE) свидетельствует об успешном завершении операции.

В следующем примере с помощью функции mail производится попытка отправить сообщение по электронной почте из PHP-сценария. Первые три аргумента задают адрес получателя, тему и тело сообщения. Оператор if использует возвращаемое значение mail, чтобы определить успешность отправления письма:

```
if (mail("chris@lightwood.net",
    "Привет", "Это тестовое письмо")) {
    echo "Письмо успешно отправлено";
}
else {
    echo "Письмо нельзя отправить";
}
```

Если Web-сервер не настроен для отправки писем или при отправке письма случилась ошибка, функция mail возвращает ноль, что сигнализирует о проблеме. Ненулевое значение свидетельствует об удачном отправлении письма.



Возвращаемые значения

Несмотря на то что необходимость проверить возвращаемое значение возникает не всегда, нужно знать, что каждая функция PHP возвращает некоторое значение.

Необязательные аргументы

Функция mail использует несколько аргументов: адресат, тема и сообщение. Это обязательные параметры. Прототип функции mail задает еще четвертый необязательный параметр. В нем можно задать дополнительные заголовки письма.

Вызвав mail с двумя аргументами, получим предупреждение. Например, сценарий содержащий строку

```
mail("chris@lightwood.net", "Привет");
```

сгенерирует примерно такое предупреждение:

```
Warning: mail() expects at least 3 parameters, 2 given in
/home/chris/mail.php on line 3
```

А эти примеры mail корректны:

```
mail("chris@lightwood.net", "Привет", "Это тестовое
письмо");
mail("chris@lightwood.net", "Привет", "Это тестовое
письмо",
"Cc: editor@samspublishing.com");
```

Для того чтобы задать несколько аргументов в функции, нужно в ее определении перечислить все имена переменных, разделив их запятой. Чтобы задать необязательный аргумент, нужно присвоить ему значение в этом списке, как при обычном присваивании.

Следующий пример является вариацией функции add_tax с двумя аргументами — чистой стоимостью и процентом налога. По умолчанию $rate имеет значение 10 и является необязательным аргументом:

```
function add_tax_rate($amount, $rate=10) {
    $total = $amount * (1 + ($rate / 100));
    return($total);
}
```

При использовании этой функции оба приведенных ниже вызова, корректны:

```
add_tax_rate(16);
add_tax_rate(16, 9);
```

В первом примере используется значение по умолчанию 10%, а во втором задается значение 9%. Этот вариант работает так же, как и первый пример функции add_tax.



Необязательные аргументы

Все обязательные значения должны идти первыми, а необязательные аргументы должны располагаться в конце списка аргументов. Иначе PHP не сможет распознать, какие аргументы передаются в функцию.

Область видимости переменных

Значения нужно передавать в функцию через аргументы, т.к. существует *область видимости переменных*. Это правила, которые определяют, в каких областях сценария можно получить доступ к переменной.

Основное правило гласит, что переменные, определенные в главном теле сценария, недоступны внутри функции. Аналогично, внутренние переменные функции не видны в главной части сценария.



Области видимости

Переменные, доступные внутри функции, называются *локальными*. Это значит, что они доступны только внутри функции. Нелокальные переменные называются *глобальными*.

Локальные и глобальные переменные могут иметь одинаковые имена и содержать различные значения. Несмотря на это лучше избегать одинаковых имен, чтобы не путаться и сделать сценарий более удобочитаемым.

Когда вызывается функция `add_tax`, вычисляется значение переменной `$total` и возвращается как результат. Однако даже после вызова `add_tax` локальная переменная `$total` не определена за пределами этой функции.

В примере ниже сделана попытка вывести значение глобальной переменной внутри функции:

```
function display_value() {
    echo $value;
}
$value = 125;
display_value();
```

Этот сценарий не выводит ничего, потому что `$value` не определено в локальной области.

Чтобы получить доступ к глобальным переменным внутри функции, используется ключевое слово `global` в начале тела функции. Таким образом изменяется область видимости переменной. Теперь ее можно изменять и извлекать значение. В коде ниже приводится пример:

```
function change_value() {
    global $value;
    echo "До: $value <br>";
    $value = $value * 2;
}
$value = 100;
display_value();
echo "После: $value <br>";
```

Значение `$value` теперь изменяется внутри функции, и на выходе получаем следующее:

```
До: 100
После: 200
```

Использование библиотечных файлов

Если создать удобную функцию, может возникнуть ситуация, что она понадобится в другом сценарии. Не стоит ее копировать туда, лучше создать специальный библиотечный файл. Теперь функция будет храниться и изменяться в одном месте.

Перед тем как продолжить, нужно создать библиотечный файл с именем `tax.php` и поместить туда функции `add_tax` и `add_tax_rate`. Другого PHP-кода в нем быть не должно.



Использование библиотечных файлов

PHP-код в библиотечных файлах нужно заключать внутрь `<?php` тегов, как и в обычном сценарии. В противном случае при подключении к сценарию содержимое будет отображаться как HTML-код.

Подключение библиотечных файлов

Чтобы связать внешний библиотечный файл с другим сценарием, нужно использовать ключевое слово `include`.

В примере ниже подключается файл tax.php, и теперь функцию add_tax можно вызвать из сценария:

```
include "tax.php";
$price = 95;
echo "Цена без налога: $price <br>";
echo "Цена с налогом: ";
echo add_tax($price);
```



Опция include_path

По умолчанию оператор include ищет запрашиваемые файлы в текущем и нескольких системных каталогах. Чтобы подключить файл из другого места, нужно указать путь к нему.

Чтобы не указывать полный путь, в операторе include используется опция include_path. Как это сделать, рассказывается в уроке 23, "Настройка PHP".

Чтобы подключить библиотечный файл только один раз, используется ключевое слово include_once. Если попытаться определить функцию с одинаковым именем, возникнет ошибка. Оператор include_once позволяет избежать этого, если файл уже подключен в другом библиотечном файле. Очень удобно иметь всего один библиотечный файл, который подключает несколько других, содержащих набор нужных функций, вместо одного огромного библиотечного файла.



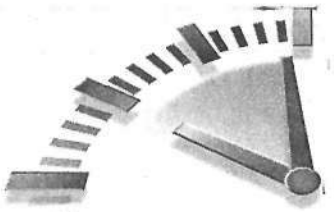
require

Инструкции require и require_once очень похожи на include и include_once, но ведут себя поразному. В случае ошибки include выводит сообщение об ошибке, но сценарий продолжает выполняться. При неудаче в операторе require выполнение сценария прекращается.

Резюме

В этом уроке вы научились использовать функции, чтобы повысить модульность кода. В следующем уроке вы узнаете о том, как работать с числовыми данными в PHP.

Урок 5



Работа с числами

В этом уроке вы узнаете, какие операции с числами можно выполнять в PHP.

Арифметика

В PHP есть все базовые арифметические операции. Тому, кто раньше не программировал, некоторые обозначение могут быть непонятны, поэтому сделаем быстрый обзор основных арифметических правил в PHP.

Арифметические операторы

Операция суммирования выполняется с помощью символа плюс (+). В этом примере вычисляется сумма 6 и 12, а результат выводится на экран:

```
echo 6 + 12;
```

Операция вычитания выполняется с помощью знака минус (-). Иногда он используется как дефис. В следующем примере из 24 вычитается 5:

```
echo 24 - 5;
```

В некоторых случаях знак минус используется для того, чтобы сделать число отрицательным (например, -20).

Чтобы перемножить два числа, используется символ звездочка (*). В следующем примере перемножаются 4 и 9:

```
echo 4 * 9;
```

Деление выполняется с помощью косой черты (/). В примере ниже 48 делится на 12:

```
echo 48 / 12;
```



Деление

Если два целых числа делятся без остатка, на выходе получим целое. Иначе будет `double`. Дробный результат не округляется до целого.

Чтобы найти остаток от деления, используется символ процента (%). В примере ниже на выходе получим 3 — остаток от деления 21 на 6:

```
echo 21 % 6;
```



Остаток от деления

Оператор остатка от деления можно использовать для того, чтобы определить четность числа с помощью операции `$number % 2`. Ноль (`0`) получим для всех четных и единицу — для всех нечетных (потому что любое нечетное число при делении на 2 дает остаток 1).

Операторы инкремента и декремента

Чтобы увеличить или уменьшить число на единицу, нужно использовать, соответственно, оператор *инкремента* — два плюса (++) или оператор *декремента* — два минуса (--). В следующем примере оба выражения прибавляют единицу к переменной `$number`:

```
$number++;
++$number;
```

Положение оператора определяет момент, когда будет производиться увеличение на единицу.

В выражении ниже единица вычитается из `$countdown` до вывода результата на экран:

```
echo --$countdown;
```

В следующем примере, наоборот, вывод на экран опережает операцию декремента:

```
echo $countdown--;
```

Операторы инкремента и декремента часто используются в циклах. Следующий пример — типичный случай цикла

`for`, где используется счетчик для того, чтобы выполнить тело цикла десять раз:

```
for ($count=1; $count<=10; $count++) {
    echo "Счетчик = $count<br>";
}
```

В данном случае при каждом повторении цикла выводится содержимое переменной `$count`.

Комбинированные операторы

Комбинированные операторы позволяют сократить код, когда арифметическая операция выполняется над существующей переменной. В следующем примере используется комбинированное суммирование для того, чтобы увеличить на шесть значение переменной `$count`:

```
$count += 6;
```

Эта операция заменяет такую последовательность: взять начальное значение переменной `$count`, прибавить к нему шесть, и результат присвоить `$count`. Поэтому комбинированный оператор эквивалентен следующему коду:

```
$count = $count + 6;
```

Все основные арифметические операции имеют соответствующие комбинированные операторы и перечислены в табл. 5.1.

Таблица 5.1. Комбинированные операторы

Оператор	*Эквивалент*
`$a += $b`	`$a = $a + $b;`
`$a -= $b`	`$a = $a - $b;`
`$a *= $b`	`$a = $a * $b;`
`$a /= $b`	`$a = $a / $b;`
`$a %= $b`	`$a = $a % $b;`

Приоритет операторов

Правила приоритетности операторов определяют порядок вычислений. Вот пример неоднозначного выражения:

```
echo 3 * 4 + 5;
```

Непонятно, то ли 3 умножается на 4 и потом к результату прибавляется 5, что дает 17; то ли суммируются 4 и 5, а потом результат умножается на 3, что дает 27. Если запустить этот сценарий, в результате получим 17.

Это произошло из-за того, что умножение имеет больший приоритет, чем суммирование. Поэтому, когда эти операторы встречаются в одном выражении, умножение выполняется первым, при этом используются значения, которые непосредственно стоят вокруг оператора умножения.

Если нужно выполнить суммирование раньше, можно использовать круглые скобки:

```
echo 3 * (4 + 5);
```

В этом случае получим 27.

Порядок выполнения арифметических операций отвечает порядку, известному со школьной скамьи: скобки, экспонента, умножение/деление и суммирование/вычитание.

Полный список приоритета операций в PHP, включая множество операторов, которые здесь не упоминаются, можно найти в электронном справочнике по адресу: `www.php.net/manual/en/language.operators.php`.

Числовые типы данных

Выше было показано, что PHP устанавливает тип данных каждому значению. Поэтому для чисел тип данных будет целый или с плавающей точкой.

Чтобы узнать, какой тип установлен для числа, можно использовать функцию `is_float` или `is_int`. Аналогично, функция `is_numeric` позволяет узнать, является ли значение произвольным числом.

В следующем примере проверяется, принадлежит ли значение переменной `$number` к целому типу:

```
$number = "28";
if (is_int($number)) {
    echo "$number целое";
}
else {
    echo "$number не целое";
}
```

При инициализации переменной справа стоит строка. Поэтому, несмотря на то, что она содержит номер, условие будет ложным.

Хоть и переменная `$number` в предыдущем примере — строка, PHP достаточно гибок, чтобы использовать это значение в числовых операциях. В следующем примере показано, как инкрементируется строка, содержащая число, и в результате получается целое:

```
$number = "6";
$number++;
echo "$number имеет тип " . gettype($number);
```

Что такое NULL?

Значение `NULL` — это тип данных и одновременно значение, которое указывает на отсутствие значения. Оно не имеет числового значения, но в операциях сравнения соответствует нулю:

```
$number = 0;
$empty = NULL;
if ($number == $empty) {
    echo "Значения эквивалентны";
}
```



Сравнение типов

Чтобы убедиться в том, что оба значения и их типы совпадают, нужно использовать оператор сравнения тройное равенство (`===`).

Функции для работы с числами

Рассмотрим функции для работы с числами в PHP.

Округление чисел

В PHP есть три функции для того, чтобы округлить десятичное число до целого.

Функции `ceil` или `floor` округляют число, соответственно, вверх или вниз до ближайшего целого. Например, `ceil(1.3)` возвращает 2, а `floor(6.8)` возвращает 6.



Округление отрицательных чисел

Нужно помнить, как округляются отрицательные числа. В результате операции `floor(-1.1)` получим `-2` как следующее меньшее целое число, а не `-1`. Аналогично, в результате `ceil(-2.5)` получим `-2`.

Чтобы округлить до ближайшего целого, используется функция `round`. Дробная часть меньше 0.5 округляется вниз, а 0.5 или выше округляется вверх. Например, `round(1.3)` возвращает 1, а `round(1.5)` возвращает 2.

Функции `round` можно передать необязательный параметр точности. В примере ниже значение округляется до двух цифр после запятой:

```
$score = 0.535;
echo round($score, 2);
```

На выходе получим `0.54`. Из-за третьей после точки цифры 5 округление дает большее число, чем изначальное.

Кроме того, функция `round` с отрицательной точностью округляет целое до определенной цифры, как в примере ниже:

```
$distance = 2834;
echo round($distance, -2);
```

Сравнение

Чтобы найти наименьшее или наибольшее число из группы элементов, используются функции `min` и `max`, соответственно. На вход функциям нужно передать два или больше аргументов, а на выходе будет, соответственно, наименьшее или наибольшее число в списке.

Выражение ниже выводит большую из переменных `$a` и `$b`:

```
echo max($a, $b);
```

Нет ограничения на количество сравниваемых значений. В примере ниже из большого списка значений выбирается минимальное:

```
echo min(6, 10, 23, 3, 88, 102, 5, 44);
```

На выходе получим 3.

Случайные числа

Функция `rand` используется для генерации случайного целого с помощью встроенного генератора случайных чисел. На вход функции `rand` можно передать два необязательных аргумента, которые задают диапазон, в котором нужно генерировать случайные числа.



Граница случайного

Константа `RAND_MAX` содержит значение наибольшего случайного числа, которое можно сгенерировать с помощью используемой системы. Это значение может отличаться для различных платформ.

В следующем выражении генерируется случайное число от одного до десяти и выводится на экран:

```
echo rand(1, 10);
```

Можно вставить эту строку в сценарий и запускать его несколько раз. Тогда значение на экране изменяется каждый раз при запуске.

На самом деле не существует случайных чисел, сгенерированных с помощью компьютера. На практике числа выбираются из очень длинной последовательности, которая имитирует случайную. Чтобы в этой последовательности всегда начинать со случайной позиции, нужно сбросить генератор случайных чисел с помощью функции `srand`, которая не требует никаких аргументов.



Алгоритмы случайных чисел

В PHP есть другой генератор случайных чисел — Мерсена-Твистера (Mersenne-Twister), который генерирует более качественную последовательность случайных чисел, чем `rand`. Этот алгоритм реализован в функциях `mt_rand` и `mt_srand`.

Математические функции

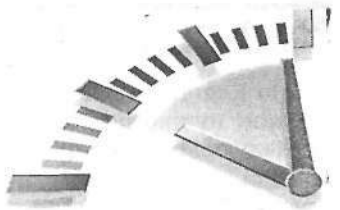
В PHP есть много математических функций. Это и тригонометрия, и логарифмы, и перевод числа из одной систе-

мы счисления в другую. Обычно такие функции не применяются для написания Web-сценариев. Поэтому они не рассматриваются в этой книге. Подробную информацию по этим функциям для математических нужд можно найти в электронном справочнике по адресу: `www.php.net/manual/en/ref.math.php`.

Резюме

В этом уроке вы узнали о том, как работать с числами. В следующем уроке вы узнаете все о работе со строками в PHP.

Урок 6



Обработка строк

В этом уроке вы узнаете о мощных функциях для обработки строк, которые есть в языке PHP.

Анатомия строки

Строка — это набор символов, которые рассматриваются как один объект. В PHP строки заключаются в кавычки. Поэтому для того, чтобы определить строку, нужно присвоить переменной строчку, заключенную в одиночные или двойные кавычки.

Два выражения ниже идентичны. Оба создают переменную `$phrase`, которая содержит строку справа:

```
$phrase = "Пусть всегда будет мир!";
$phrase = 'Пусть всегда будет мир!';
```



Символы кавычек

В PHP кавычки не указывают направления. Один и тот же символ указывает на начало и завершение строки. Для строк нельзя использовать символ апострофа (`). Вместо этого используется одиночная кавычка (').

Выделение специальных символов обратной косой чертой

В одиночных кавычках можно использовать двойные, и наоборот. Например, две строки ниже — корректные:

```
$phrase = "'Привет', - сказал Вася.";
$phrase = '"Хорошо", - сказал я.';
```

Но для того, чтобы использовать ту же кавычку внутри строки, нужно поставить перед ней обратную косую черту. В следующем примере показывается этот механизм:

```
$phrase = '\'Привет\', - сказал Вася.';
$phrase = "\"Хорошо\", - сказал я.";
```

Если в примерах выше не выделить кавычку обратной косой чертой, PHP обнаружит несоответствие кавычек и выдаст ошибку.

Какой стиль кавычек использовать, зависит от личных предпочтений и сказывается на аккуратности кода. Но, как показано в уроке 2, "Переменные", в строках, заключенных в двойные кавычки, заменяются знак доллара и имя переменной на соответствующее значение, а в одиночных кавычках такая структура трактуется как обычный текст. Для того чтобы вывести знак доллара в строке, заключенной в двойные кавычки, достаточно поставить передним обратную косую черту. Например, две строки ниже полностью эквивалентны:

```
$offer = 'Сэкономьте $10 на первом заказе';
$offer = "Сэкономьте \$10 на первом заказе";
```

Если не выделить обратной косой чертой символ доллара во второй строке, PHP попытается найти переменную $10, которая к тому же имеет некорректное имя.

Символ обратной косой черты можно использовать в строке с двойными кавычками для вставки специальных символов. Трехзначное число после обратной косой черты соответствует ASCII-символу в восьмеричном формате.

Чтобы вставить общие непечатные ASCII-символы, используются стандартные комбинации символов с обратной косой чертой. Символ новой строки это \n, табуляция — \t и т.п. Получить полный список можно с помощью команды man ascii (на Unix-подобной системе) или на сайте — www.ascii.cl.

Конкатенация

Выше показано, как с помощью оператора конкатенации (точки) можно объединять строки в одну. Комбинированная версия этого оператора (.=) позволяет добавить значение к окончанию данной строки.

В примере ниже поэтапно создается строка и выводится на экран:


```
$phrase = "Я хочу ";
$phrase .= "изменить ";
$phrase .= "мир ";
$phrase .= "к лучшему";
echo $phrase;
```

Как и ожидалось, появится полное предложение. Пробелы после слов хочу, изменить и мир сделаны для того, чтобы они не слились в одно слово.

Сравнение строк

Строки можно сравнивать с помощью стандартных операторов сравнения. Чтобы убедиться в равности двух строк, используется знак двойного равенства (==):

```
if ($password == "пароль") {
    echo "Вы угадали пароль!";
}
```

При сравнении строк оператор сравнения выполняет зависимую от регистра проверку. Поэтому проверка не сработает, если в переменной $password находится, к примеру, слово ПаРоль.

Другие операторы сравнения — <, <=, > и >= — выполняют сравнение на основе ASCII-значения каждого символа в строке. Проверка ниже разделяет людей на две группы, в зависимости от начальной буквы фамилии: первая от A до M и вторая от N до Z:

```
if ($last_name < "N") {
    echo "Вы в первой группе 1";
}
else {
    echo "Вы во второй группе 2";
}
```



ASCII-значения

Так как сравнение строк производится на основе соответствующих ASCII-значений, все символы в нижнем регистре имеют большее значение, чем эквивалентные в верхнем. Символы a-z лежат в диапазоне 97–122, а символы A-Z занимают диапазон 65–90.

Форматирование строк

В PHP легко создавать форматированные строки с помощью функций printf и sprintf. Эти функции похожи на аналогичные в языке C, но в PHP есть некоторые отличия.

Возможности printf

Функция printf используется для вывода форматированной строки. В самом простом случае на вход printf передается строка, и он выводит ее так же, как и echo:

```
printf("Привет, мир");
```

Сила printf заключается в том, что она позволяет подставлять вместо специальных комбинаций — *спецификаторов формата* нужные значения. Чтобы задать спецификатор, используется знак процента (%) и символ, который задает формат заменяемого значения.

В следующем примере используется спецификатор формата %f, для числа с плавающей точкой:

```
$price = 5.99;
printf("Цена: %f", $price);
```

Второй параметр printf заменяет спецификатор %f, и на выходе получаем:

```
Цена: 5.99
```

Функция printf не имеет ограничений на количество заменяющих значений. Главное, чтобы количество спецификаторов формата и заменяемых значений совпадало. В примере ниже показывается этот механизм и подставляется строчное значение:

```
$item = "данной единицы товара";
$price = 5.99;
printf("Цена %s равна %f", $item, $price);
```

В табл. 6.1 показаны форматирующие символы, которые применяются в функции printf для разных типов значений.

Пусть вместо спецификатора %f используется %d, чтобы вывести значение $price:

```
$price = 5.99;
printf("Как десятичное число, цена равна %d", $price);
```


Таблица 6.1. Форматирующие символы printf

Символ	*Значение*
b	Двоичное (основа 2) число
c	ASCII-символ с численным значением аргумента
d	Целое (основа 10) со знаком
e	Число в инженерной нотации (например, 2.6e+3)
u	Десятичное целое без знака
f	Число с плавающей точкой
o	Восьмеричное (основа 8) число
s	Строка
x	Шестнадцатеричное (основа 16) число в нижнем регистре
X	Шестнадцатеричное (основа 16) число в верхнем регистре

В этом случае PHP рассматривает значение аргумента как целое, и от числа берется только целая часть. На выходе получим:

```
Как десятичное число, цена равна 5
```



Десятичный

Формат %d представляет десятичное целое. *Десятичное* — говорит о том, что используется десятичная система счисления (т.е. по основанию 10), а не то, что в числе есть десятичная точка.

Существуют различные форматы для отображения чисел по основанию 16 (*шестнадцатеричный*, %x), 8 (*восьмеричный*, %o) и 2 (*двоичный*, %b).

Форматирующие коды

Спецификаторы формата также устанавливают необязательные параметры: отступ, выравнивание, ширину и точность выводимого значения. Это позволяет выполнять довольно сложное форматирование.

Спецификатор ширины указывает, сколько символов отводится в строке на вывод форматируемого значения. Он

ставится между знаком процента и спецификатором типа. Пример ниже позволяет убедиться, что выводимое имя занимает ровно десять знакомест:

```
$name1 = "Юра";
$name2 = "Петя";
$name3 = "Роман";
echo "<PRE>";
printf("%10s \n", $name1);
printf("%10s \n", $name2);
printf("%10s \n", $name3);
echo "</PRE>";
```



Отступ

В этом примере используется дескриптор <PRE>, чтобы вывести на экран пробелы перед именем. Потому что без него браузер рассматривает несколько пробелов как один.

Строчные отступы не используются для генерации динамических Web-страниц, но они очень удобны, когда нужно сгенерировать обычный текст. Например, содержимое электронного письма.

Запустив этот пример, можно убедиться что перед всеми именами есть отступ слева на соответствующее число символов. В результате получаем список имен с выравниванием по правому краю.

По умолчанию выравнивание идет справа на заданную ширину. Но это можно изменить с помощью знака минус перед значением ширины. Выравнивание по левому краю в предыдущем примере будет выполняться, если задать спецификатор формата в виде `%-10s`. Но, по сути, такой формат генерирует такой же вид, как простой `%s`. Строка получит отступ в десять символов справа.

Символ пробела для отступа можно заменить на любой. Для этого нужно поставить одиночную кавычку и нужный символ перед значением ширины. В следующем примере показывается, как число из пяти цифр заполняется нулями:

```
$order = 201;
printf("Номер заказа: %'05d", $order);
```

На выходе получим:

```
Номер заказа: 00201
```


Спецификатор точности используется для чисел с плавающей точкой, чтобы задать количество символов после точки, и наиболее часто применяется для денежных значений. Таким образом, даже имея целое значение, всегда получим две цифры для копеечных значений.

Спецификатор точности следует через точку после необязательного спецификатора ширины. В примере ниже `%.2f` используется для денег и задает два знака после точки, а необязательный спецификатор ширины не используется:

```
$price = 6;
printf("Цена: %.2f", $price);
```

Ниже показан отформатированный вывод:

```
Цена: 6.00
```



Ширина чисел с плавающей точкой

Для чисел с плавающей точкой спецификатор ширины задает количество символов перед точкой. Например, `%6.2f` состоит из девяти символов вместе с точкой и двумя числами после нее.

Возможности sprintf

Функция `sprintf` используется для того, чтобы присвоить форматированную строку переменной. Синтаксис полностью аналогичен `printf`, но вместо вывода на экран, функция возвращает строку как результат.

Например, для того, чтобы присвоить значение цены новой переменной, нужно сделать следующее:

```
$new_price = sprintf("%.2f", $price);
```

Функции `printf` и `sprintf` имеют одинаковые спецификаторы формата.

Строковые функции

Рассмотрим доступные в PHP функции для работы со строками. Полный список можно найти в электронном справочнике по адресу `www.php.net/manual/en/ref.strings.php`.

Переключение регистра

Можно переключать регистр строки из строчного в заглавный, и наоборот, с помощью функций strtoupper или strtolower соответственно.

Пример ниже показывает этот механизм на строке со словами в разном регистре:

```
$phrase = "I love PHP";
echo strtoupper($phrase) . "<br>";
echo strtolower($phrase) . "<br>";
```

На выходе получим:

```
I LOVE PHP
i love php
```

Для перевода первой буквы в верхний регистр используется функция ucfirst:

```
$phrase = "welcome to the jungle";
echo $ucfirst($phrase);
```

Для перевода первой буквы каждого слова в верхний регистр (полезно в случае ФИО) используется функция ucwords:

```
$phrase = "green bay packers";
echo ucwords($phrase);
```

Функции ucfirst и ucwords изменяют только первый символ и не следят за регистром остальных. Поэтому, если вся строка находится в верхнем регистре, может выйти не то, что ожидается. Чтобы добиться нужного эффекта, необходимо скомбинировать их с функцией strtolower, как показывается в примере ниже:

```
$name = "CHRIS NEWMAN";
echo ucwords(strtolower($name));
```

Разбивка строки

Функция substr позволяет извлечь часть строки. Для этого нужно задать точку отсчета и необходимую длину. В следующем примере показано, как это работает:

```
$phrase = "Я люблю PHP";
echo substr($phrase, 4, 5);
```

Функция substr возвращает часть переменной $phrase начиная с четвертой позиции, длиной в пять символов. Отметим, что отсчет позиции начинается с нуля, а не с единицы. Поэтому на выходе получим блю P.

Если не указать длину, возвращается подстрока от заданной позиции до конца строки. Следующий пример выводит люблю PHP для переменной $phrase:

```
echo substr($phrase, 2);
```

Если стартовая позиция — отрицательное число, то функция substr начинает отсчет с конца строки. В примере ниже на экран выводится три последних символа строки, в данном случае — PHP:

```
echo substr($phrase, -3);
```

Для того чтобы узнать длину строки, используется функция strlen:

```
echo strlen($phrase);
```

Чтобы найти позицию символа или строки в другой строке, используется функция strpos. Первый аргумент содержит "где искать", а второй "что искать".

В следующем примере выводится позиция символа @ в электронном адресе:

```
$email = "chris@lightwood.net";
echo strpos($email, "@");
```



Позиция в строке

Нужно помнить, что позиция символа в строке отсчитывается с левого края и начинается с нуля. Позиция 1 соответствует второму символу в строке. Когда strpos находит искомую последовательность в начале строки, на выходе получим ноль. Но когда искомой комбинации не обнаружено, — на выходе получим FALSE.

Чтобы определить эту разницу, нужно проверять тип. Например, условие strpos($a, $b) === 0 будет истинным только в том случае, когда $b содержится в $a, начиная с нулевой позиции.

Функция strstr извлекает часть строки от совпадающей позиции и до конца строки. Эта функция создана для удобства, чтобы не использовать комбинацию strpos и substr.

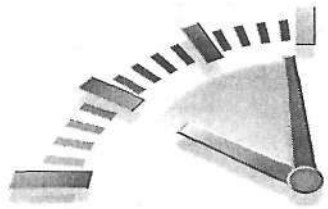
Два следующих выражения полностью эквивалентны:

```
$domain = strstr($email, "@");
$domain = strstr($email, strpos($email, "@"));
```

Резюме

В этом уроке вы научились работать со строками с помощью PHP. В следующем уроке вы узнаете о том, что такое массивы и какие операции с ними можно выполнять.

Урок 7



Работа с массивами

В этом уроке вы узнаете о том, как использовать массивы для хранения и извлечения пронумерованных данных.

Что такое массив

Массив — это тип переменной, которая позволяет сохранять и пронумеровать набор значений. Массивы очень полезны для хранения данных, которые имеют что-то общее или логически группируются в множество.

Создание массива и доступ к нему

Предположим, нужно хранить среднюю температуру каждого месяца в году. Если использовать обычные однозначные переменные, также называемые *скалярными*, нужно 12 отдельных переменных — `$temp_jan`, `$temp_feb` и т.д. Массив позволяет использовать одно имя переменной для группирования значений, а *индекс* (или *ключ*) позволяет найти соответствующее значение для каждого месяца.

В следующем примере создается массив `$temps` с 12-ю значениями, которые соответствуют всем месяцам, с января по декабрь:

```
$temps = array(38, 40, 49, 60, 70, 79,
               84, 83, 76, 65, 54, 42);
```

Созданный массив `$temps` содержит 12 значений и проиндексирован от 0 до 11. Для того чтобы обратиться к значению массива, после имени нужно указать соответствующий индекс. Например, покажем температуру в марте:

```
echo $temps[2];
```



Индексные номера

Так как по умолчанию массив начинается с нуля, индекс для марта (третьего месяца) равен двум.

Квадратные скобки также используются для того, чтобы присвоить значение отдельному элементу массива. Чтобы установить новое значение ноябрю, используется следующая конструкция:

```
$temps[10] = 56;
```



Функция `array`

Функция `array` позволяет быстро создать массив из списка значений. Это намного удобнее, чем по отдельности добавлять каждый элемент в массив.

Если при инициализации элемента массива не указать в квадратных скобках индекс, значение индекса устанавливается автоматически. Оно будет на единицу больше самого большого индекса в массиве. В примере ниже пустой массив `$temps` последовательно заполняется значениями, что полностью идентично примеру выше:

```
$temps[] = 38;
$temps[] = 40;
$temps[] = 49;
...
```

Здесь элементу `$temps[0]` присваивается значение `38`, элементу `$temps[1]` — `40` и так далее. Если нужно присвоить значение элементу `$temps[0]`, желательно убедиться в том, что массив не инициализирован раньше. Для этого можно инициализировать массив пустым значением. В примере ниже показывается эта конструкция:

```
$temps = array();
```

Вывод содержимого массива

В PHP есть полезная функция `print_r`. Она позволяет рекурсивно вывести все значения массива. В сценарии ниже определяется массив значений температур, и после этого выводится его содержимое на экран:


```
$temps = array(38, 40, 49, 60, 70, 79,
               84, 83, 76, 65, 54, 42);
print "<PRE>";
print_r($temps);
print "</PRE>";
```

Результат функции `print_r` нужно выводить между дескрипторами `<PRE>`, т.к. он форматируется пробелами и переводами строки. На экране получим:

```
Array
(
    [0] => 38
    [1] => 40
    [2] => 49
    [3] => 60
    [4] => 70
    [5] => 79
    [6] => 84
    [7] => 83
    [8] => 76
    [9] => 65
    [10] => 54
    [11] => 42
)
```



`print_r`

Функция `print_r` очень помогает при программировании сценариев, даже если не использовать ее при выводе на реальной Web-странице. Если неизвестно, какие значения находятся в массиве, можно просто задействовать `print_r` и сразу получить ответ.

Прохождение по массиву

Можно легко повторить способ, которым функция `print_r` проходит по всем значениям массива, чтобы выполнить нужное действие над всеми элементами массива.

С помощью цикла `while` можно найти все индексы и соответствующие им значения в массиве, как это делается в `print_r`:

```
while (list($key, $value) = each($temps)) {
    echo "Ключ $key имеет значение $val <br>";
}
```

Все индексы массива сохраняются в переменной $key, а значение — в $value.

В PHP для прохождения по массиву есть более удобная конструкция foreach. Что предпочесть — while или foreach, — зависит от личных предпочтений и требований к читабельности кода.

Ниже эквивалентный пример для цикла foreach:

```
foreach($temps as $key => $value) {
    ...
}
```



Циклы

Можно заметить, что в примере с $temps для итерации по массиву можно воспользоваться циклом for, ведя отсчет по индексам от 0 до 11. Но, как будет показано дальше, такой прием не всегда сработает, потому что ключи массива не всегда составляют последовательность и даже могут быть строчными.

Ассоциативный массив

Выше приводился пример массива с числовыми ключами. *Ассоциативный* массив позволяет использовать для индексов более наглядные текстовые ключи.

Чтобы присвоить значение массиву для ассоциативного ключа и иметь доступ через него, достаточно в квадратных скобках написать ключевое имя в кавычках, как в примере ниже:

```
$temps["jan"] = 38;
echo $temps["jan"];
```

Чтобы определить ассоциативный массив с помощью функции array, нужно, кроме значения, указать ключ. Для этого используется символ =>, который показывает связь между ключом и значением:

```
$temps = array("jan" => 38, "feb" => 40, "mar" => 49,
               "apr" => 60, "may" => 70, "jun" => 79,
               "jul" => 84, "aug" => 83, "sep" => 76,
               "oct" => 65, "nov" => 54, "dec" => 42);
```

Элемент в ассоциативном массиве находится в том порядке, в котором его определили (про сортировку массивов рассказывается ниже в этом уроке). Прохождение по ассоциативному массиву также происходит в том порядке, в котором определялись элементы.

Можно вызвать print_r для массива, чтобы убедиться в этом. Ниже несколько строк вывода:

```
Array
(
    [jan] => 38
    [feb] => 40
    [mar] => 49
...
```

Функции для работы с массивами

Выше уже рассматривалась функция array, которая используется для создания массива. Теперь рассмотрим другие функции PHP для работы с массивами.

В PHP есть очень много функций для работы с массивами. Поэтому здесь приводятся только часто используемые. Так что, если возникнет необходимость выполнить сложные действия с массивом, нужно обратиться к электронному справочнику по адресу www.php.net/ref.array.

Сортировка

Для сортировки значений в массиве используется функция sort или одна из ее производных, как в примере ниже:

```
sort($temps);
```



Функции сортировки

Функции sort передается массив, который она сортирует. Готовый массив не возвращается, а возвращаемое значение свидетельствует об удачном или неудачном выполнении операции.

Сортировка массива $temps с помощью функции sort расставит все значения в порядке возрастания. Так, нулевому элементу будет соответствовать наименьшее значение. Поэтому невозможно узнать, какое значение соответствует какому месяцу.

Для того чтобы сберечь ключевые значения, используется функция asort. Она меняет порядок при прохождении по массиву, но индексы останутся теми же. После сортировки $temps индекс 0 содержит среднюю температуру января, но при прохождении по массиву элементы извлекаются в порядке сортировки.

Выведем с помощью ассоциативного массива $temps название месяца и соответствующую среднюю температуру — от самой холодной до самой теплой:

```
$temps = array("jan" => 38, "feb" => 40, "mar" => 49,
               "apr" => 60, "may" => 70, "jun" => 79,
               "jul" => 84, "aug" => 83, "sep" => 76,
               "oct" => 65, "nov" => 54, "dec" => 42);
asort($temps);
foreach($temps as $month => $temp) {
    print "$month: $temp <br>\n";
}
```

Массивы сортируются не только по значениям, но и по ключам. Для этого используется функция ksort. Для ассоциативного массива функция ksort выстроит все элементы в алфавитном порядке по названиям месяца в ключах. Таким образом, после прохождения по отсортированному массиву первым получим элемент $temps["apr"], дальше — $temps["aug"] и так далее.

Чтобы изменить порядок сортировки этих функций, вместо sort используется функция rsort. Функция asort заменяется на arsort, а ksort — на krsort. Чтобы изменить порядок элементов в массиве без сортировки, используется array_reverse.

Перетасовка массива случайным образом

Кроме сортировки массива в определенном порядке, в PHP есть функции для быстрой перетасовки элементов в случайном порядке.

Функция shuffle работает так же, как и остальные функции сортировки. На вход подается один аргумент — массив, и производится перетасовка элементов в случайном порядке. Как и в sort, ассоциативные ключи теряются, и перемешанные значения индексируются числами.

Функции для работы с множествами

Если рассматривать массив как множество значений, можно выполнять операции над множествами с помощью функций обработки массивов.

Чтобы объединить значения из разных массивов (операция объединения), используется функция array_merge с двумя и более аргументами, как в примере ниже:

```
$union = array_merge($array1, $array2, $array3, ...);
```

Функция возвращает массив, который состоит из всех элементов перечисленных массивов. В этом примере массив $union содержит все элементы массива $array1, за ними следуют элементы массива $array2 и так далее.

Чтобы изъять повторяющиеся значения из массива, нужно использовать функцию array_unique. Из двух ключей, указывающих на одинаковое значение, останется только один.

С помощью функции array_intersect находится пересечение двух массивов.

Следующий пример создает новый массив $intersect, содержащий элементы массива $array1, которые есть и в массиве $array2:

```
$intersect = array_intersect($array1, $array2);
```

Функция array_diff позволяет найти разницу между двумя массивами. В примере ниже массив $diff состоит из элементов массива $array1, которых нет в массиве $array2:

```
$diff = array_diff($array1, $array2);
```

Внутри массива

Функция count возвращает количество элементов в массиве. На вход нужно передать один аргумент — массив. Например, в следующем выражении показывается, что в массиве $temps находится 12 значений:

```
echo count($temps);
```

Чтобы определить, находится ли значение внутри массива, не проходя по всем значениям, используется функция in_array или array_search. Первый элемент — искомое значение, второй — массив, где происходит поиск:

```
if (in_array("PHP", $languages)) {
    ...
}
```

Эти функции отличаются возвращаемыми значениями. Если значение находится внутри массива, функция array_search возвращает соответствующий ключ, а in_array только булево значение.



Нужное в искомом

Иногда можно запутаться из-за того, что аргументы *искомое* и *где искать* для функций in_array и array_search размещаются в обратном порядке к строковым функциям, таким как strpos и strstr.

Чтобы проверить наличие отдельного ключа в массиве, используется функция array_key_exists. В следующем примере проводится поиск ключа "декабрь" ("dec") в массиве $temps:

```
if (array_key_exists("dec", $temps)) {
    ...
}
```

Сериализация

Функция serialize создает текстовое представление информации, которая хранится в массиве. Этот мощный механизм позволяет легко сохранять содержимое массивов PHP в базу данных или файл.

В уроках 17, "Работа с файловой системой", и 19, "Использование базы данных MySQL", рассматривается специфика сохранения информации в файловой системе и базе данных. Теперь рассмотрим, как работает сериализация массивов.

Функция serialize возвращает строку, которая содержит ключи и значения массива в структурированном формате. После этого ее можно декодировать с помощью функции unserialize, чтобы получить исходный массив.

Сериализированный массив $temps выглядит так:

```
a:12:{s:3:"jan";i:38;s:3:"feb";i:40;s:3:"mar";i:49;
s:3:"apr";i:60; s:3:"may";i:70;s:3:"jun";
i:79;s:3:"jul";i:84;s:3:"aug";i:83;s:3:"sep";
si:76;s:3:"oct";i:65;s:3:"nov";i:54;s:3:"dec";i:42;}
```


Не нужно разбираться в содержимом этой строки. Можно просто подать ее на вход функции unserialize, и она сама восстановит исходный массив.

Многомерный массив

Часто очень полезно использовать двух- или даже многомерные массивы. PHP позволяет создавать такие структуры.

Доступ к двухмерному массиву

Двухмерный массив — это массив массивов. Например, нужно хранить в массиве среднемесячные значения температур по годам. Для этого удобно использовать два ключа — отдельно для месяца и для года. Тогда для вывода февральской температуры в 1995 году нужно использовать:

```
echo $temps[1995]["feb"];
```

Так как $temps массив массивов, $temps[1995] — это массив температур, проиндексированный по месяцам, и для обращения к его элементу нужно ввести имя в квадратных скобках.

Определение многомерного массива

Чтобы определить многомерный массив, достаточно помнить, что это массив, который содержит массивы.

Инициализировать значение можно с помощью прямого обращения к отдельному элементу, как показано ниже:

```
$temps[1995]["feb"] = 41;
```

Кроме того, можно определить многомерный массив, вкладывая функции array в соответствующие места. В следующем примере определяется несколько первых месяцев на три года (полный массив будет намного больше, чем этот):

```
$temps = array (
    1995 => array ("jan" => 36, "feb" => 42, "mar" => 51),
    1996 => array ("jan" => 37, "feb" => 42, "mar" => 49),
    1997 => array ("jan" => 34, "feb" => 40, "mar" => 50)
);
```

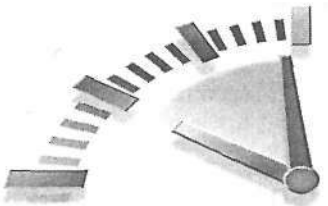
Функция print_r позволяет выводить все уровни, из которых состоит массив, в форматированном виде. Каждый отступ указывает на следующий уровень иерархии. Ниже приводится вывод трехмерного массива $temps.

```
Array
(
    [1995] => Array
    (
        [jan] => 36
        [feb] => 42
        [mar] => 51
    )
    [1996] => Array
    (
        [jan] => 37
        [feb] => 42
        [mar] => 49
    )
    [1997] => Array
    (
        [jan] => 34
        [feb] => 40
        [mar] => 50
    )
)
```

Резюме

В этом уроке вы узнали о том, как создавать массивы данных и работать с ними. В следующем уроке рассматривается, как с помощью регулярных выражений выполнять сложный поиск и обработку строк.

Урок 8



Регулярные выражения

В этой главе вы научитесь выполнять сложные операции со строками при помощи регулярных выражений. Вы узнаете, как с помощью регулярных выражений можно реализовать проверку строк и сложный поиск с заменой.

Введение в регулярные выражения

С помощью регулярных выражений можно написать правила, которые позволяют идентифицировать определенный формат строки. Довольно сложные правила описываются с помощью нескольких символов. Поэтому поначалу они могут показаться немного сложными.

В самом простом виде регулярное выражение содержит строку, которую нужно найти. В более сложном виде задается детальный шаблон символов и строка разбивается на части на основе этого шаблона.

Типы регулярных выражений

В PHP поддерживается два типа регулярных выражений: основанные на расширенном POSIX-синтаксисе, которые рассматриваются в этом уроке, и "перловские" — Perl-Compatible Regular Expression (PCRE).

Оба типа позволяют выполнять одинаковые функции, используя различный синтаксис. Поэтому не нужно знать оба варианта. Тому, кто уже знает Perl, лучше освоить функциональность PCRE вместо POSIX-синтаксиса.

Подробная документация по PCRE находится по адресу: www.php.net/manual/en/ref.pcre.php.

Возможности ereg

Функция ereg в PHP позволяет проверить строку на соответствие регулярному выражению. В примере ниже с помощью простого регулярного выражения находится подстрока PHP в переменной $phrase:

```
$phrase = "Я люблю PHP";
if (ereg("PHP", $phrase)) {
    echo "Выражение найдено";
}
```

Если запустить этот сценарий, можно убедиться, что выражение действительно обнаружило соответствие с переменной $phrase.

Регулярные выражения зависят от регистра, поэтому, если выражение в нижнем регистре, пример не сработает. Чтобы выполнить безразличное к регистру сравнение, нужно использовать eregi:

```
if (eregi("php", $phrase)) {
    echo "Выражение найдено";
}
```



Производительность

Вышеприведенные примеры можно реализовать с помощью менее сложных функций со строками из урока 6, "Обработка строк", таких как strstr. Сценарий выполняется быстрее, если использовать строковые функции вместо ereg для простых проверок.

Соответствие набору символов

Кроме поиска точной последовательности символов в строке, с помощью регулярных выражений можно искать по списку символов, если заключить их в квадратные скобки. Для этого нужно перечислить все символы, и при обнаружении одного из них, устанавливается соответствие.

Ниже приводится выражение, эквивалентное ранее приведенному с функцией eregi:


```
if (ereg("[Pp][Hh][Pp]", $phrase)) {
    echo " Выражение найдено ";
}
```

В этом выражении проверяется наличие букв P, H и P в заданном порядке следования. При этом не учитывается регистр.

С помощью символа "дефис" можно задать диапазон символов. Для этого нужно поставить его между двумя буквами или цифрами. Например, [A-Z] соответствует любой букве латинского алфавита, а [0-4] соответствует любой цифре от нуля до четырех.

Следующее выражение истинно только в том случае, если $phrase содержит одну из букв в верхнем регистре:

```
if (ereg("[A-Z]", $phrase)) ...
```

С помощью символа ^ можно указать, каких символов не должно быть в искомом выражении. В примере ниже истинное значение возвращается, только если $phrase содержит как минимум один нецифровой символ:

```
if (ereg("[^0-9]", $phrase)) ...
```

Общие классы символов

Можно использовать различные наборы символов в регулярных выражениях. Чтобы проверить наличие буквенных символов, нужно следующее регулярное выражение:

```
[A-Za-z0-9]
```

Аналогичный по работе класс символов можно записать в более наглядном виде:

```
[[:alnum:]]
```

Символы [: и :] свидетельствуют о том, что выражение содержит класс символов. Существующие наборы классов перечислены в табл. 8.1.

Таблица 8.1. Классы символов для регулярных выражений

Название класса	*Описание*
alnum	Все буквы латинского алфавита и цифры, A–Z, a–z, и 0–9
alpha	Все буквы A–Z и a–z
digit	Все цифры 0–9

Окончание табл. 8.1

Название класса	*Описание*
lower	Все буквы в нижнем регистре a–z
print	Все печатные символы, включая пробел
punct	Все символы пунктуации, кроме пробела и класса alnum
space	Все разделительные символы, включая табуляцию и перевод строки
upper	Все буквы в верхнем регистре A–Z

Проверка положения

Все выражения, рассмотренные выше, проверяют наличие шаблона в произвольном месте строки. Но регулярные выражения позволяют также указывать позицию, где ожидается определенный шаблон.

Символ ^, когда не является частью класса переменных, указывает на начало строки, а $ — на окончание. С помощью этих символов проверяется, находится ли символ, соответственно, в начале или в конце переменной $phrase:

```
if (ereg("^[a-z]", $phrase)) ...
if (ereg("[a-z]$", $phrase)) ...
```

Чтобы убедиться в том, что вся строка соответствует регулярному выражению, можно поместить его между символами ^ и $. Например в следующем условии проверяется, что $number содержит только одну цифру.

```
if (ereg("^[[:digit:]]$", $number) ...
```



Знак доллара

Если с помощью выражения нужно найти символ $, необходимо разделить его обратной косой чертой \$, тогда он не трактуется как конец строки.

Если выражение находится в двойных кавычках, нужно использовать \\$. Иначе знак $ сначала будет интерпретироваться как переменная в строке, а после первой обратной косой черты — как конец строки.

Поиск с помощью символов подстановки

Символ "точка" (.) в регулярном выражении является символом подстановки и соответствует произвольному символу. Например, следующее выражение найдет соответствие с любым словом из четырех символов с двумя буквами *o* посередине:

```
if (ereg("^.oo.$", $word)) ...
```

Символы ^ и $ указывают на начало и конец строки, а каждая точка может быть произвольным символом. Этот шаблон соответствует словам *book* или *tool*, но не *buck* или *stool*.



Символы подстановки

Регулярное выражение с одной точкой соответствует строке, содержащей хотя бы один символ. Нужно использовать символы ^ и $ чтобы наложить ограничение на длину строки.

Повторяющиеся шаблоны

Выше приводились шаблоны для отдельного символа или класса символов в строке. Показывались способы указать произвольный символ, чтобы задать широкий спектр шаблонов в регулярных выражениях. Есть также специальные символы, позволяющие указать, сколько раз отдельный шаблон повторяется в строке.

Символ (*) указывает на то, что последовательность встречается ноль или больше раз, а плюс (+) — как минимум один раз.

Два примера ниже с символами * и + очень похожи. Оба проверяют наличие буквенно-цифровых символов произвольной длины. Первое условие также соответствует пустой строке, потому что звездочка допускает ноль и больше совпадений [[:alnum:]]:

```
if (ereg("^[[:alnum:]]*$", $phrase)) ...
if (ereg("^[[:alnum:]]+$", $phrase)) ...
```

Нужно использовать круглые скобки, чтобы выделить последовательность искомых символов, которые нужно повторить. Например, условие ниже соответствует строке с чередующейся последовательностью букв и цифр:

```
if (ereg("^([[:alpha:]][[:digit:]])+$", $string)) ...
```

Символ плюс указывает на то, что последовательность буква/цифра должна встречаться не менее одного раза. Чтобы задать фиксированное количество повторений, нужно указать номер в фигурных скобках. Можно задать одно или два числа через запятую для указания диапазона, как показывается в примере ниже:

```
if (ereg("^([[:alpha:]][[:digit:]]){2,3}$", $string)) ...
```

Это выражение соответствует четырем или шести символам с чередующимися буквами и цифрами. Но единичная комбинация буквы и цифры или более трех повторений не соответствуют этой комбинации.

Знак вопроса (?) указывает на то, что комбинация должна встретиться не больше одного раза. Его поведение аналогично действию комбинации {0,1}.

Несколько практических примеров

В большинстве случаев регулярные выражения используются для того, чтобы проверить корректность данных, введенных пользователем. Ниже приводится несколько практических примеров использования регулярных выражений.

Почтовый индекс

Имеется почтовый индекс заказчика, который находится в переменной $zip, и нужно убедиться, что он введен в правильном формате. Почтовый индекс Соединенных Штатов состоит из пяти цифр и опционально сопровождается дефисом с четырьмя цифрами после него. Выражение ниже проверяет почтовый индекс на соответствие этому формату:

```
if (ereg("^[[:digit:]]{5}(-[[:digit:]]{4})?$", $zip)) ...
```

Первая часть регулярного выражения проверяет, что $zip начинается с пяти цифр. Вторая часть выражения заключена в круглые скобки, и после них ставится знак вопроса, который указывает на то, что выражение в скобках необязательное. Во второй части выполняется проверка на дефис и четыре цифры за ним.

Даже если второй части выражения нет, знак $ указывает на окончание строки. Таким образом, чтобы условие выполнилось, после указанных выражений не должно быть никаких символов. Поэтому такое выражение вернет истинное значение, если на вход подать почтовый индекс в виде 90210 или 90210-1234.

Телефонные номера

Предположим, нужно убедиться в том, что телефонный номер задан в формате (555)555-5555. В нем нет необязательных частей. Но круглые скобки имеют специальное значение для регулярных выражений. Поэтому их нужно разделить обратной косой чертой.

Следующее выражение проверяет соответствие этому формату:

```
if (ereg("^\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$",
                $telephone)) ...
```

Email адрес

Нужно учитывать много различных параметров при проверке электронных адресов. В самом простом случае почтовый адрес для домена .com выглядит как somename@somedomain.com.

Но может быть много вариаций. Например, домены верхнего уровня могут быть из двух символов, как в .ca, или даже из четырех, как .info. Некоторые специфические домены стран состоят из двух частей, как .co.uk или .com.au.

Как видно, регулярное выражение для электронного адреса должно быть довольно общим. Несмотря на это некоторые общие допущения в формате электронных адресов, позволяют создать правила, которые отсеивают много некорректных адресов.

Адрес состоит из двух важных частей, которые разделяются символом @. Символы слева от знака @, составляют название почтового ящика адресата. Они могут быть буквенно-цифровыми и включать несколько других символов.

Предположим, название ящика адресата может быть произвольной длины и состоять из любых символов, кроме символа @. Перед тем как перечислить приемлемые символы, нужно решить, например, включать ли одиночную ка-

вычку? Обычно достаточно убедиться в том, что в электронном адресе только один символ @ и все, что до него, является корректным названием ящика.

В регулярном выражении имя домена должно состоять из двух и более частей, разделенных точкой. Кроме того, можно задать, что последняя часть может быть не меньше двух и не больше четырех символов в длину. Это справедливо для всех доменов верхнего уровня, которые используются в данный момент.

В части домена набор допустимых символов более бедный, чем в названии почтового ящика. Допускается использование буквенно-цифровых символов в нижнем регистре и знака дефис.

Учитывая все приведенные выше условия, получаем следующее регулярное выражение для проверки электронного адреса:

```
if (ereg("^[^@]+@([a-z0-9\-]+\.)+[a-z]{2,4}$", $email)) ...
```

Это выражение разбивается на такие части: вначале любое количество символов, после которых следует символ @. А сразу за ним одна или более частей, состоящих из букв в нижнем регистре, цифр и дефиса. Каждая часть заканчивается точкой, а финальная может быть не меньше двух и не больше четырех букв в длину.



Когда остановиться?

Это выражение можно улучшить. Например, имя домена не может начинаться с дефиса и должно быть не больше 63 символов в длину. Но для проверки электронных адресов этого выражения вполне достаточно.

Разбивка строки на компоненты

Раньше круглые скобки использовались для группировки частей, чтобы выделить повторяющийся шаблон. Кроме того, круглые скобки используются для выделения части выражения. С помощью ereg шаблон разбивается на части, в зависимости от круглых скобок.

Третий необязательный параметр ereg сохраняет все части шаблона, которые выделяются круглыми скобками в регулярном выражении.

Задействуем выражение с электронным адресом. В примере ниже используется три набора круглых скобок: чтобы отделить имя почтового ящика, имя домена (отдельно от расширения) и расширение домена:

```
$email = "chris@lightwood.net";
if (ereg("^([^@]+)@([a-z\-]+\.)+([a-z]{2,4})$",
$email, $match)) {
    echo "Почтовый ящик: " . $match[1] . "<br>";
    echo "Имя домена: " . $match[2] . "<br>";
    echo "Тип домена: " . $match[3] . "<br>";
}
else {
    echo "Электронный адрес некорректен";
}
```

Если запустить сценарий в браузере, получим следующий вывод:

```
Почтовый ящик: chris
Имя домена: lightwood.
Тип домена: net
```

Отметим, что первый искомый шаблон (почтовый ящик) содержится в элементе $match[1]. Массив, как обычно, начинается с нулевого элемента. Дело в том, что элемент $match[0] содержит полное найденное выражение.

Поиск и замена

С помощью функции ereg_replace выполняется поиск и замена в строке с помощью регулярных выражений. На вход подается три аргумента: искомое регулярное выражение, строка, в которой производится замена, и строка-заменитель. На выходе получаем модифицированную строку.



str_replace

Для простых замен, которые не требуют регулярных выражений, лучше использовать функцию str_replace вместо ereg_replace. Функция str_replace намного эффективней, потому что PHP не пытается искать регулярное выражение.

Например, для того, чтобы удалить телефонный номер перед выводом на экран, можно использовать следующее:

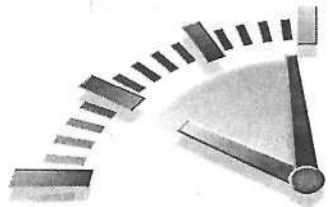
```
echo ereg_replace( "\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$", "(XXX)XXX-XXXX", $string);
```

Аналогично тому, как eregi используется вместо ereg для независимого от регистра поиска, eregi_replace выполняет независимый от регистра поиск и замену.

Резюме

В этом уроке вы научились работать с простыми регулярными выражениями. Подробнее о них можно прочитать в книге Бена Форта *Освой самостоятельно регулярные выражения за 10 минут* (ИД "Вильямс", 2005 г.). В следующем уроке вы узнаете, как манипулировать значениями даты и времени в PHP.

Урок 9



Работа с временем и датой

В этом уроке вы научитесь сохранять, показывать и изменять значение даты и времени в PHP.

Форматы дат

В PHP нет специального типа данных для хранения значений даты. Поэтому сначала нужно определиться с тем, как лучше хранить эти значения.

Выбор формата данных

Часто встречаются даты, записанные в структурированном виде. Например, 05/03/1974 или 2001-12-31. Такой формат не подходит для обработки даты. Но второй вариант удобней первого. В нем данные следуют в порядке важности — сначала год, потом месяц и затем день. Поэтому сравнить две даты можно с помощью стандартных операторов отношения в PHP.

Строка 2002-01-01 больше, чем 2001-12-31. Но так как операции сравнения быстрее работают с числами, лучше хранить эти данные в числовом виде. Например, 20020201; тогда формат будет YYYYMMDD. Его можно расширить для того, чтобы хранить время. Тогда формат примет следующий вид: YYYYMMDDHHMMSS. Как видим, все элементы расположены в порядке уменьшения веса.

К сожалению, такой формат не позволяет выполнять арифметические операции. Конечно, можно добавить один к 20040501, но если следующий календарный день отно-

сится к другому месяцу, получим бессмысленный результат. Например, добавив единицу к 20030531, получим бессмысленную дату — 32 мая.

Формат временной метки Unix

В формате *временной метки Unix* дата задается в целочисленном представлении. Его значение отсчитывается в секундах от полночи 1 января 1970 года.



Начало эпохи Unix

Нулевая временная метка соответствует 12 часам ночи 1 января 1970 года по гринвичскому часовому поясу (GMT). Эта дата также называется началом *эпохи Unix*.

В данный момент дата и время состоят из десяти цифр в формате временной метки. Функция time позволяет определить текущее значение временной метки:

```
echo time();
```

Формат временной метки Unix очень удобен для расчетов, потому что время задается в секундах. Например, чтобы увеличить время на час, нужно прибавить 3600 секунд к значению временной метки. Если же прибавить 86 400, время изменится на один день. Потому что час состоит из 3600 секунд, а день — из 86 400.

Но у этого формата есть один недостаток — нельзя работать с датами до 1970 года. Правда, на некоторых системах поддерживаются отрицательные значения временной метки, и можно обрабатывать даты до эпохи Unix. Но это работает не всюду, и лучше его не использовать.

Временная метка хорошо подходит для работы с текущим временем, но не всегда — для дней рождения или важных исторических дат. Поэтому перед использованием нужно убедиться в том, что диапазон охватывается форматом временной метки.



Ограничение временной метки

Максимальное значение временной метки Unix зависит от архитектуры системы. Большинство платформ использует 32-битовое целое, чтобы хранить значение временной метки. Поэтому максимальное значение будет 3:14, 19 января 2038 года.

Работа с временной меткой

Иногда лучше использовать собственный формат даты, но в большинстве случаев идеально подойдет формат временной метки. Рассмотрим, как с ним работать в PHP.

Форматирование даты

В уроке 1, "Знакомство с PHP", используется функция date. Она показывает текущую дату с форматированием, которое задается в аргументе:

```
echo date("j F Y H:i:s");
```

На выходе получим:

```
12 November 2004 10:23:55
```

Второй необязательный аргумент для date задает значение даты, которую нужно отобразить, в формате временной метки. Вот пример вывода даты с десятизначным значением временной метки:

```
echo date("j F Y H:I:s", 1000000000);
```

Коды форматирования для функции date перечислены в табл. 9.1.

Таблица 9.1. Коды форматирования для date

Код	*Описание*
a	am или pm в нижнем регистре
A	AM или PM в верхнем регистре
d	Двухзначный формат дня месяца, 01–31
D	Трехбуквенное название дня недели, Mon–Sun
F	Полное название месяца, January–December

Окончание табл. 9.1

Код	*Описание*
g	12-часовый формат без нуля спереди для значения с одной цифрой, 1–12
G	24-часовый формат без нуля спереди для значения с одной цифрой, 0–23
h	12-часовый формат с нулем спереди для значения с одной цифрой, 01–12
H	24-часовый формат с нулем спереди для значения с одной цифрой, 00–23
I	Минуты с нулем спереди, 00–59
j	День месяца без нуля спереди, 1–31
l	Полное название дня недели, Monday–Sunday
m	Номер месяца с нулем спереди, 01–12
M	Трехбуквенное название месяца, Jan–Dec
n	Номер месяца без нуля спереди, 1–12
s	Секунды с нулем спереди, 00–59
S	Порядковый суффикс для дней месяца: st, nd, rd или th
w	Номер дня недели, 0–6, где 0 — воскресенье
W	Номер недели в году, 0–53
y	Двухзначковый год
Y	Четырехзначковый год
z	День года, 0–365

Перевод в формат временной метки

Для того чтобы узнать значение временной метки для нужной даты, в PHP есть готовая функция mktime, которая рассчитывает его на основе заданной даты и времени.

Порядок аргументов следующий: час, минуты, секунды, месяц, день и год. В примере ниже переменной $timestamp присваивается значение временной метки, которое соответствует 25 декабрю 2001 года в 8 часов утра:

```
$timestamp = mktime(8, 0, 0, 12, 25, 2001);
```

Формат временной метки Unix начинается с 1 января 1970 года в полночь по GMT. Функция mktime возвращает значение временной метки на основе часового пояса, который установлен на Web-сервере. Например, mktime с одинаковыми аргументами для Техаса вернет значение временной метки на 3600 больше, чем для компьютера в Нью-Йорке.



Безопасное дневное время

Если формат временной метки используется для хранения даты, то три первых аргумента mktime влияют на значение только в полночь, когда время может изменить дату.

Например, переводя часы на час назад и добавив при этом к значению временной метки 86 400 секунд, получим прибавку в два дня. Чтобы этого избежать, лучше использовать полдень вместо полночи.



Среднее время по гринвичскому меридиану

Функция gmmktime возвращает время в формате временной метки относительно GMT (Лондонское время, где используется безопасное дневное время).

Функция mktime — нестрогая и позволяет вводить бессмысленные аргументы. Например, несуществующий день месяца. Если попытаться узнать значение временной метки для 29 февраля в невисокосном году, на выходе получим значение для 1 марта:

```
echo date("d/m/Y", mktime(12, 0, 0, 2, 29, 2003));
```

Это позволяет выполнять различные арифметические операции с датой и временем. В примере ниже рассчитывается и выводится дата и время для 37 часов после полудня 30 декабря 2001 года:

```
$time = mktime(12 + 37, 0, 0, 12, 30, 2001);
echo date("d/m/Y H:i:s", $time);
```

Просто прибавляя нужное значение к одному из аргументов в mktime, можно смещать искомое значение временной метки. На выходе получим следующий вид для даты и времени:

```
01/01/2002 01:00:00
```

В этой дате корректно сместились все значения: день, месяц, год и время. Здесь учитывается то, что декабрь последний месяц в году, и количество дней в нем.

Перевод в формат временной метки

Значение даты в формате DD-MM-YYYY легко перевести в формат временной метки. Для этого нужно выделить значения между дефисами. Функции explode передается символ-разделитель и строка, которую нужно разбить на части. На выходе получим массив выделенных значений.

В примере ниже значение даты разбивается на компоненты, и рассчитывается значение временной метки:

```
$date = "03-05-1974";
$parts = explode("/", $date);
$timestamp = mktime(12, 0, 0,
                    $parts[1], $parts[0], $parts[2]);
```

Функция strtotime позволяет автоматически получать значение временной метки из различных форматов представления даты. Вот примеры для различных форматов даты:

```
$timestamp = strtotime("3 May 04");
$timestamp = strtotime("3rd May 2004");
$timestamp = strtotime("May 3, 2004");
$timestamp = strtotime("3-may-04");
$timestamp = strtotime("2004-05-03");
$timestamp = strtotime("05/03/2004");
```

Нужно помнить, что в последнем примере формат имеет вид MM/DD/YYYY, а не DD/MM/YYYY. Полный список форматов, которые понимает функция strtotime, находится по адресу: www.gnu.org/software/tar/manual/html_chapter/tar_7.html.

Получение информации из значения временной метки

Можно использовать функцию date, чтобы узнать часть или всю информацию о дате на основе значения временной метки. Но в PHP для этих целей есть специальная функция getdate, которая возвращает различные параметры на основе значения временной метки.

Функции getdate передается значение временной метки, а на выходе получаем ассоциированный массив с ключами, которые перечислены в табл. 9.2.

Таблица 9.2. Значения ключей, возвращаемых функцией getdate

Ключ	*Описание*
seconds	Секунды, 0–59
minutes	Минуты, 0–59
hours	Часы, 0–23
mday	День месяца, 1–31
wday	День недели, 0–6, где 0 — воскресенье
yday	День года, 0–365
mon	Номер месяца, 1–12
year	Четырехзначковый год
weekday	Полное название дня недели, Sunday–Saturday
month	Полное название месяца, January–December

В примере ниже функция getdate определяет, будний или выходной заданный день:

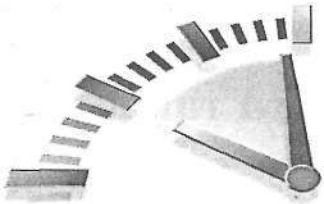
```
$now = getdate();
switch ($now["wday"]) {
    case 0: // Воскресенье
    case 6: // Суббота
    echo "Это выходной";
    break;
    default: echo "Это рабочий день";
}
```

Нужно помнить, что getdate без аргумента возвращает массив с индексами из табл. 9.2 для текущего времени.

Резюме

В этом уроке вы узнали, как работать со значениями даты и времени в PHP. В следующем уроке вы узнаете о классах в PHP и о том, как использовать библиотеки классов от сторонних разработчиков.

Урок 10



Использование классов

В этом уроке изложены основы объектно-ориентированного программирования (ООП) в PHP. Вы увидите, как определяются классы и как работать с классами от сторонних разработчиков.

Объектно-ориентированное программирование на PHP

При необходимости для PHP можно использовать объектно-ориентированный подход. В PHP 5 поддержка ООП существенно улучшена.

Тот, кто работал с языками C++ или Java, возможно, захотят использовать ООП в PHP. А тот, кто до этого пользовался процедурными языками, возможно, не захочет использовать объекты вообще. Как видим, есть много путей, чтобы решить одну и ту же задачу.

Для новичков в программировании и PHP, не имеет значения, какой подход использовать. Известно, что концепция ООП легче усваивается теми, кто еще не программировал в процедурном стиле. Но это не значит что методы ООП можно описать в десятиминутном уроке этой книги!

Главная задача урока — дать основные понятие о том, как создаются классы в PHP. Тогда, если возникнет потребность использовать объекты, можно написать сценарий в стиле ООП. Но главное то, что можно использовать свободно распространяемые библиотеки классов от сторонних разработчиков. Они доступны на сайте `www.phpclasses.org`, а также в проекте PEAR. О нем подробнее рассказывается в уроке 25, "Использование PEAR".

Что такое класс

Класс — это шаблон структуры, который задает объект. В нем могут содержаться *функции*, также называемые *методами класса*. И *переменные*, также называемые *свойствами класса*, или *атрибутами*.

Каждый класс состоит из набора PHP-выражений, задающих порядок выполнения задачи или набора задач, которые часто встречаются. Класс может содержать *закрытые* методы, которые используются внутри класса для выполнения внутренних функций класса, и *открытые* методы, через которые происходит взаимодействие с классом.

В хорошо спроектированном классе внутренняя реализация скрыта. Взаимодействие с ним происходит только через открытые методы. Они предоставляют простой интерфейс для работы с функциональностью класса. Если связать сложные блоки кода в класс, сценарий, который его использует, не будет ничего "знать" о том, как выполняется отдельная операция. Необходим только список всех открытых методов.

Сейчас есть много готовых классов от сторонних разработчиков. Поэтому не нужно тратить время на то, чтобы реализовать функциональность, которая уже свободно доступна.

Когда использовать классы

Нужно помнить, что нет реальных преимуществ использования классов по сравнению с библиотекой функций, которая находится в отдельном файле. ООП не является лучшим подходом к программированию, это просто другой способ мышления. Методика программирования определяется только личными предпочтениями.

Преимущество ООП проявляется в больших проектах, потому что позволяет легко наращивать нужную функциональность. В ООП класс может наследовать свойства других классов и расширять их. Таким образом, однажды разработанная реализация может легко изменяться для конкретного случая. Этот механизм называется *наследованием* и является ключевым свойством ООП.

Те, кто захотят подробнее узнать об ООП, могут обратиться к книге Антони Синтеса *Освой самостоятельно объ-*

ектно-ориентированное программирование за 21 день (ИД "Вильямс", 2002 г.).

На что похожи классы

Классы — это набор различных функций и переменных. Примерно так они и выглядят в PHP. Определение класса очень напоминает определение функции. Оно начинается с ключевого слова `class` и идентификатора, после которого следует тело класса в парных фигурных скобках ({}).

Ниже приводится простой пример класса, чтобы показать, как он выглядит. Этот класс содержит только одно свойство — `myValue` и один метод — `myMethod` (который ничего не делает):

```
class myClass {
    var $myValue;
    function myMethod() {
        return 0;
    }
}
```

Тот, кто уже знаком с ООП и хочет поскорее использовать его в PHP, может обратиться к электронному справочнику по адресу: `www.php.net/manual/en/language.oop5.php`.

Создание и использование объектов

Чтобы создать экземпляр объекта определенного класса, используется ключевое слово `new`, как в примере ниже:

```
$myObject = new myClass;
```

Здесь `myClass` нужно определить в сценарии, обычно в подключаемом файле, а `$myObject` становится объектом класса `myClass`.



Множество объектов

Один и тот же класс можно использовать несколько раз в одном сценарии. Для этого нужно создавать объекты класса с новыми именами.

Методы и свойства

К методам и свойствам, которые определяются в myClass, можно обратиться из объекта $myObject. Ниже приводится общий пример:

```
$myObject->myValue = "555-1234";
$myObject->myMethod();
```

Символ стрелки (->) состоит из дефиса и символа "больше чем". Он указывает на метод или свойство заданного объекта. Чтобы обратиться к текущему объекту внутри тела класса, используется специальное имя $this.

Пример ниже создает myClass с методом, который обращается к одному из свойств объекта:

```
class myClass {
    var $myValue = "Юлия";
    function myMethod() {
        echo "myValue равно " . $this->myValue . "<br>";
    }
}
$myObject = new myClass;
$myObject->myMethod();
$myObject->myValue = "Геннадий";
$myObject->myMethod();
```

В этом примере есть два отдельных вызова метода myMethod. В первый раз он выводит значение по умолчанию свойства myValue, которое устанавливается в определении класса. Второй вызов происходит после присваивания атрибуту нового значения. Класс использует $this, чтобы обратиться к собственному свойству. Ему не нужно знать, что в сценарии имя объекта $myObject.

В классе можно создать специальный метод, именуемый *конструктором*. Тогда при создании объекта в круглых скобках после названия класса можно указать аргументы, и эти значения поступят на вход конструктору. Такой механизм позволяет инициализировать свойства объекта при создании. Выглядит это так:

```
$myObject = new myClass($var1, $var2);
```


Использование классов сторонних разработчиков

Лучший способ изучить работу с классами — использовать их. Рассмотрим популярную библиотеку, разработанную Мануэлем Лемосом (Manuel Lemos). Она позволяет выполнять проверку электронного адреса. Сначала нужно загрузить этот класс по адресу: www.phpclasses.org/browse/file/28.html и сохранить в файле email_validation.php.

Класс Мануэля, кроме проверки почтового адреса, проверяет существование домена. После этого он соединяется с удаленным почтовым сервером, чтобы убедиться в существовании почтового ящика.



Поиск домена

Чтобы использовать этот пример на Web-сервере под управлением Windows, нужно загрузить дополнительный файл getmxrr.php, который содержит поиск доменов для PHP. Он находится по адресу: www.phpclasses.org/browse/file/2080.html.

В сценарии email_validation.php определяется класс email_validation_class. Первое, что нужно сделать, — создать экземпляр объекта, который выполняет проверку с именем $validator:

```
$validator = new email_validation_class;
```

Можно задать нужные свойства для нового класса. Некоторые нужны для корректной работы класса, а остальные позволяют изменять стандартное поведение.

Для каждого объекта нужно установить свойства. Это название ящика и имя домена реального почтового адреса, от имени которого выполняется проверка. Для этих свойств нельзя задать стандартные значения, их нужно установить самому, как показано ниже:

```
$validator->localuser = "chris";
$validator->localhost = "lightwood.net";
```

Необязательное свойство timeout задает максимальное количество секунд на ожидание соединения с удаленным почтовым сервером. Установка свойства debug позволяет увидеть на экране процесс соединения с удаленным серве-

ром и узнать, какие запросы выполняет сценарий во время проверки. Выражение ниже устанавливает время ожидания 10 секунд и включает режим отладки:

```
$validator->timeout = 10;
$validator->debug = TRUE;
```

Полный список свойств настройки для объекта проверки приводится в табл. Таблица 10.1.

Таблица 10.1. Свойства класса email_validation_class

Свойство	*Описание*
timeout	Задает максимальное количество секунд ожидания соединения с удаленным почтовым сервером
data_timeout	Задает максимальное количество секунд для обмена данными с почтовым сервером. Если задан ноль, то используется значение timeout
localuser	Задает пользовательскую часть адреса электронной почты, от имени которого выполняется проверка
localhost	Задает имя домена адреса электронной почты, от имени которого выполняется проверка
debug	Активизирует режим вывода на экран всего процесса соединения с удаленным сервером
html_debug	Задает опцию вывода сообщений из режима отладки в виде HTML

Методы в классе email_validation_class в основном закрытые, к ним нельзя обращаться напрямую. Но внутренний код составляет завершенный набор функций. Рассмотрев содержимое email_validation.php, можно увидеть определение функций, таких как Tokenize, GetLine и VerifyResultLines. Но они совершенно бесполезны за пределами класса, в котором определены.

В объекте есть только один открытый метод под названием ValidateEmailBox, который и выполняет нужную проверку. При вызове он устанавливает в виде строки почтовый адрес, который нужно проверить. Ниже показан пример вызова ValidateEmailBox:

```
$email = "chris@datasnake.co.uk";
if ($validator->ValidateEmailBox($email)) {
    echo "$email является корректным почтовым адресом";
}
```



```
else {
    echo "$email не прошел проверки";
}
```

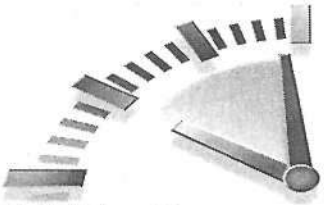
Метод ValidateEmailBox возвращает значение, которое показывает результат проверки. Если включить режим отладки с помощью атрибута debug, кроме вывода из сценария, можно увидеть примерно следующее:

```
Resolving host name "mail.datasnake.co.uk"...
Connecting to host address "217.158.68.125"...
Connected.
S 220 mail.datasnake.co.uk ESMTP
C HELO lightwood.net
S 250 mail.datasnake.co.uk
C MAIL FROM: <chris@lightwood.net>
S 250 ok
C RCPT TO: <chris@datasnake.co.uk>
S 250 ok
C DATA
S 354 go ahead
This host states that the address is valid.
Disconnected.
```

Резюме

В этом уроке вы узнали о реализации ООП в PHP и увидели, как использовать классы в собственных сценариях. В следующем уроке вы узнаете о том, как PHP взаимодействует с HTML-формами.

Урок 11



Обработка HTML-форм

Причина популярности PHP в том, что он позволяет легко обрабатывать данные, полученные из HTML-форм. В этом уроке вы узнаете, как получить доступ ко всем типам данных из HTML-форм.

Отправка данных из формы в PHP

Обычно формы используются для того, чтобы пользователь мог отправить данные на Web-сервер. Рассмотрим подробнее, как это происходит.

Дескриптор <FORM>

Дескриптор <FORM> задает область, в которой размещаются элементы для ввода текста и другие управляющие элементы, которые отправляют введенные значения на определенный URL.

Атрибут ACTION дескриптора <FORM> позволяет задать адрес сценария, куда передаются значения из формы. URL можно задать относительно текущей страницы или в полной форме, начав с http://.

Атрибут METHOD определяет метод пересылки данных из формы. Есть всего два метода передачи — GET и POST. На первый взгляд они практически не отличаются. Данные, отправленные методом GET, добавляются в конец URL, а данные, оправленные методом POST, не показываются пользователю.



Метод GET

URL, в котором передаются данные методом GET, встречается очень часто. Обычно пользователь даже не догадывается о том, что происходит. Часто этот механизм используется в поисковых системах на Web-страницах. Если после ввода запроса происходит переход по ссылке и в конце адреса добавляется префикс `?search=yourword`, это указывает на то, что используется метод GET.

В большинстве случаев лучше использовать метод POST. Кроме того что в адресе не появляется длинного префикса, метод POST не ограничивает объем передаваемых данных. А для метода GET это значение определяется максимальной длинной URL, с которым может работать пользовательский браузер (в Internet Explorer это 2048 символов) и версией HTTP-протокола Web-сервера (HTTP/1.0 позволяет не меньше 256 символов, а HTTP/1.1 — не меньше 2048).

Дескриптор <INPUT>

Дескриптор <INPUT> позволяет задать несколько типов ввода для HTML-формы. Атрибут TYPE задает тип ввода, а значение TEXT соответствует простому элементу текстового ввода.

Текстовое поле для ввода почтовых адресов задается следующим образом:

```
<INPUT TYPE="TEXT" NAME="name" SIZE="30" VALUE="">
```

В этом HTML-коде, атрибут VALUE — пустой, потому что не нужно задавать стандартное значение для этого поля. В данном случае его вообще можно не писать.



Длина поля

Размер поля не влияет на то, как PHP обрабатывает принятые значения. Для текстового поля ввода установлен размер в 30 символов, но не указывается атрибут MAXLENGTH, так что пользователи с очень длинными именами смогут их ввести.

Тип ввода CHECKBOX задает *флажок*, который может находится только в двух состояниях — установлен/сброшен. Флажок используется для значений истина/ложь. Создадим элемент, который спрашивает пользователя о разрешении связаться с ним:

```
<INPUT TYPE="CHECKBOX" NAME="may_contact" VALUE="Y"
CHECKED>
```

В этом случае после загрузки страницы атрибут CHECKED автоматически установит флажок.

Тип RADIO (*переключатель*) напоминает обычный флажок, но вместо значений истина/ложь, переключатель имеет несколько значений, и выбрать можно только одно.

В примере ниже задается элемент "переключатель", который проверяет пол:

```
<INPUT TYPE="radio" NAME="gender" VALUE="m"> Мужчина
<INPUT TYPE="radio" NAME="gender" VALUE="f"> Женщина
```



Название переключателя

Атрибут NAME объединяет переключатели. Можно выбрать только один вариант для группы переключателей. Кроме того, можно разместить несколько групп переключателей на странице. В этом примере оба варианта имеют одинаковое имя gender.

Чтобы выбрать один из вариантов по умолчанию, достаточно указать в нем атрибут CHECKED. Например, если сайт ориентирован на женскую аудиторию, можно отметить по умолчанию соответствующую опцию:

```
<INPUT TYPE="radio" NAME="gender" VALUE="m"> Мужчина
<INPUT TYPE="radio" NAME="gender" VALUE="f" CHECKED> Женщина
```

Рассмотрим последний на сегодня тип элемента ввода — SUBMIT (*кнопка подачи формы*). Кнопка подачи формы отправляет содержимое формы по адресу, указанному в атрибуте формы ACTION. Надпись на кнопке задается в атрибуте VALUE. В примере ниже создается кнопка подачи формы с надписью "Отправить комментарий":

```
<INPUT TYPE=SUBMIT VALUE="Отправить комментарий">
```

Для кнопки подачи формы можно указать атрибут NAME, но это редко используется. Ниже в этом уроке рассказывается, как это повлияет на значения, которые получает PHP.

Дескриптор <TEXTAREA>

Дескриптор <TEXTAREA> создает элемент ввода текста с несколькими строками. Во многих отношениях он напоминает тип TEXT дескриптора <INPUT>, но отличается способом задания в HTML-коде.

Стандартное значение дескриптора <TEXTAREA> может занимать несколько строк, поэтому вместо атрибута VALUE оно размещается между парой дескрипторов:

```
<TEXTAREA ROWS=4 COLS=50 NAME="comments">
Введите ваш комментарий здесь</TEXTAREA>
```

PHP не интересует, из какого элемента получено значение, различие будет только в HTML-коде.

Дескриптор <SELECT>

Напоследок рассмотрим элемент формы <SELECT>, который называется *раскрывающимся списком*.

Чаще всего раскрывающийся список используется для того, чтобы выбрать один элемент из заданного списка значений. В примере ниже создается раскрывающийся список. В нем перечислены источники, из которых посетители могли узнать о сайте:

```
<SELECT NAME="referrer">
<OPTION VALUE="search">Поисковый сервер</OPTION>
<OPTION VALUE="tv">Телевизионная реклама</OPTION>
<OPTION VALUE="billboard">Доска объявлений</OPTION>
<OPTION SELECTED VALUE="other">Другое</OPTION>
</SELECT>
```

В данном случае атрибут SELECTED устанавливает стандартный элемент "Другое", даже если он установлен в верху списка. Если нет элемента с атрибутом SELECTED, первый элемент выбирается по умолчанию.

Соберем все вместе

Если собрать все элементы формы вместе, добавить несколько текстовых надписей и немного форматирования, можно создать простую форму для отправки комментариев, как показано в листинге 11.1.

Листинг 11.1. Web-форма для отправки пользовательских комментариев

```
<FORM ACTION="send_comments.php" METHOD=POST>
<TABLE>
<TR>
  <TD>Ваше имя:</TD>
  <TD><INPUT TYPE="TEXT" NAME="name" SIZE=30></TD>
</TR>
<TR>
  <TD>Ваш почтовый адрес:</TD>
  <TD><INPUT TYPE="TEXT" NAME="email" SIZE=30></TD>
</TR>
<TR>
  <TD>Ваш пол:</TD>
  <TD><INPUT TYPE="RADIO" NAME="gender" VALUE="m"> Мужчина
      <INPUT TYPE="RADIO" NAME="gender" VALUE="f"> Женщина
  </TD>
</TR>
<TR>
  <TD>Как вы нас нашли?</TD>
  <TD>
    <SELECT NAME="referrer">
    <OPTION VALUE="search">Поисковый сервер</OPTION>
    <OPTION VALUE="tv">Телевизионная реклама</OPTION>
    <OPTION VALUE="billboard">Доска объявлений</OPTION>
    <OPTION SELECTED VALUE="other">Другое</OPTION>
    </SELECT>
  </TD>
</TR>
<TR>
  <TD>Можно вам написать?</TD>
  <TD><INPUT TYPE="CHECKBOX" NAME="may_contact"
             VALUE="Y" CHECKED></TD>
</TR>
<TR>
  <TD>Комментарии</TD>
  <TD><TEXTAREA ROWS=4 COLS=50
      NAME="comments">Введите ваш комментарий здесь
     </TEXTAREA></TD>
</TR>
</TABLE>

<INPUT TYPE="SUBMIT" VALUE="Отправить комментарий">
</FORM>
```

Обработка форм с помощью PHP

Теперь рассмотрим, как обработать элементы формы с помощью PHP после отправки пользователем данных.

Доступ к значениям формы

Доступ к значениям формы в PHP можно получить с помощью специальных массивов. Массивы $_GET и $_POST содержат значения, отправленные, соответственно, методами GET и POST. Гибридный массив $_REQUEST содержит значения из обоих массивов, а также значения массива $_COOKIE, который рассматривается в уроке 14, "Данные cookies и сеансы".



Суперглобальность

Системные массивы, имя которых начинается с подчеркивания, называются *суперглобальными*. К ним можно обратиться из любой точки сценария PHP, независимо от области видимости. Например, не нужно объявлять массив $_POST как глобальный для того, чтобы использовать его в функции.

Доступ к значениям элементов формы интуитивно понятен. Элементу формы с определенным именем соответствует элемент массива $_GET или $_POST с соответствующим ключом. А значение этого элемента соответствует введенному пользователем значению для этого элемента.

Например, почтовый адрес из странички comments.html содержится в элементе $_POST["email"], а текст комментария — в $_POST["comments"].

Для типов CHECKBOX и RADIO атрибут VALUE определяет значение, которое получит PHP. Если установить флажок may_contact, элемент $_POST["may_contact"] будет содержать значение Y. Если же этот элемент оставить не установленным, он вообще не появится в соответствующем массиве. Поэтому нужно использовать функцию isset, чтобы узнать о том, установлен ли флажок.



Стандартное значение флажка

Если флажок установлен, но в HTML-коде для него не задан атрибут VALUE, в PHP передается значение on.

Группа переключателей gender создает элемент $_POST["gender"], который содержит значение m или f, в зависимости от выбранного значения. Если ни один из переключателей не отмечен, как и в случае с флажком, соответствующий элемент не создается вообще.

С помощью функции print_r легко увидеть все значения, полученные из формы. Для этого на ее вход подается массив $_POST:

```
echo "<PRE>";
print_r($_POST);
echo "</PRE>";
```

Это очень полезный прием отладки, который позволяет увидеть все данные, которые сценарий получает из формы. Если создать файл send_comments.php и поместить в него код, приведенный выше, на экране появятся значения каждого элемента формы. Вот пример вывода:

```
Array
(
    [name] => Крис Ньюман
    [email] => chris@lightwood.net
    [gender] => m
    [referrer] => search
    [may_contact] => Y
    [comments] => Это просто мой любимый сайт
)
```

Даже значение кнопки подачи формы можно увидеть в PHP. Для этого нужно задать имя и отправить форму, щелкнув на нужной кнопке. Следующая форма содержит две кнопки с разными именами, а PHP позволяет определить, какая из них отправила форму:

```
<FORM ACTION="button.php" METHOD=POST>
<INPUT TYPE="SUBMIT" NAME="button1" VALUE="Кнопка 1">
<INPUT TYPE="SUBMIT" NAME="button2" VALUE="Кнопка 2">
</FORM>
```

В button.php можно использовать следующую проверку для того, чтобы определить, на какой кнопке сделан щелчок:

```
if (isset($_POST["button1"])) {
    echo "Вы щелкнули на кнопке 1";
```

```
}
elseif (isset($_POST["button2"])) {
    echo "Вы щелкнули на кнопке 2";
}
else {
    echo "Я не знаю на какой кнопке вы щелкнули!";
}
```

Атрибут VALUE кнопки подачи формы задает текст на кнопке, а также передается в PHP, когда щелкают на этой кнопке.



Кнопка submit

Новые браузеры отправляют данные формы, если нажать клавишу <Enter>, когда фокус находится на элементе ввода. Даже если в форме всего один элемент подачи формы, его значение передается в PHP, только если щелкнуть на этой кнопке.

Элемент скрытого ввода

Есть еще один тип дескриптора <INPUT>, который используется для передачи данных между сценариями и при этом не отображается на Web-странице.

Тип HIDDEN имеет атрибуты NAME и VALUE, которые заменяют это значение. Он используется, чтобы подменить эти значения.

Следующий элемент скрытого ввода передается в PHP-сценарий при отправке формы, и элемент $_POST["secret"] содержит это значение:

```
<INPUT TYPE="HIDDEN" NAME="secret" VALUE="Это секрет">
```

Но будьте осторожны. Тип HIDDEN не подходит для передачи секретных паролей или другой важной информации. Несмотря на то что он не отображается на Web-странице, просмотр HTML-кода позволяет узнать эти значения.

Сценарий для отправки электронной почты

Чтобы реализовать максимум удобств, реализуем отправку пользовательских комментариев на почтовый ящик владельца Web-сайта. Посмотрим, как объединить все части, чтобы сделать сценарий-обработчик, который будет выполнять эту функцию.

Функция mail

Функция mail отправляет сообщения с помощью системного почтового сервиса. На системах Linux/Unix для отправки почты используется утилита sendmail. На серверах под управлением Windows для этого используется протокол SMTP. Для корректной работы нужно задать имя почтового сервера в файле php.ini. В уроке 23, "Настройка PHP", вопрос настройки рассматривается более подробно.

Для функции mail нужно задать три аргумента: почтовый адрес получателя, тему и текст сообщения. Четвертый необязательный аргумент задает дополнительные заголовки письма. Это позволяет указать специальные параметры From: или Cc:.

Сценарий send_comments.php в листинге Листинг 11.2 принимает данные из формы комментариев и отправляет их владельцу Web-сайта на почтовый ящик.

Сценарий проходит по всем значениям массива $_POST и создает строку $body для текста письма. Отметим, что символы \n используются для разделения строчек, потому что письмо передается обычным текстом, без HTML-форматирования.

Листинг 11.2. send_comments.php

```
<?php
$body = "Этот комментарий отправлен с помощью Web-
сайта\n\n";
foreach($_POST as $field => $value) {
    $body .= sprintf("%s = %s\n", $field, $value);
}
```

```
mail("owner@website.com", "Комментарий отправленный с
помощью Web-сайта", $body,
            'From: "Web-комментарий"
<comments@website.com>');
?>
<H1>Спасибо</H1>
Ваш комментарий отправлен!
```

Письмо, которое получит владелец сайта, будет выглядеть примерно так:

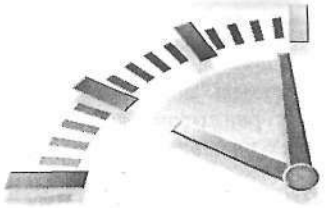
```
Этот комментарий отправлен с помощью Web-сайта
name = Крис Ньюман
email = chris@lightwood.net
gender = m
referrer = search
may_contact = Y
comments = Это просто мой любимый сайт
```

Это письмо отформатировано довольно небрежно, потому что генерируется автоматически. Конечно, при желании его можно оформить намного лучше. Например, можно заменить кодовые значения `gender` и `referrer` на полное описание.

Резюме

В этом уроке вы научились обрабатывать данные, полученные из HTML-форм. В следующем уроке вы узнаете, как динамически генерировать такие элементы HTML, как раскрывающиеся списки и переключатели.

Урок 12



Динамическая генерация HTML-кода для форм

В этом уроке вы узнаете о том, как создавать элементы HTML-форм с помощью PHP. Эти приемы позволяют задавать значения по умолчанию для элементов ввода, а также генерировать раскрывающиеся списки и группы переключателей на основе данных сценария.

Установка значений по умолчанию

Рассмотрим несколько простых примеров того, как установить при помощи PHP стандартные значения для некоторых элементов при загрузке страницы.

Стандартные значения для ввода

Значение по умолчанию для текстового поля ввода задается в атрибуте `VALUE`. Это значение будет отображаться в форме после загрузки страницы, и если его не изменить, оно и передается в PHP-сценарий при отправке формы.

Рассмотрим страницу электронного магазина с корзиной для покупок. У покупателя есть возможность изменить количество единиц товаров в корзине, перед окончательной отправкой заказа. Текущее количество товаров в корзине можно изменить в небольшом текстовом поле. После этого пользователь может щелкнуть на кнопке подачи формы и

обновить содержимое корзины. В листинге 12.1 показано, как реализовать магазин. Он содержит всего один товар и дает возможность выбрать его количество для покупки.

Листинг 12.1. Значение по умолчанию для текстового поля ввода

```
<?php
if(isset($_POST["quantity"])) {
    $quantity = settype($_POST["quantity"], "integer");
}
else {
    $quantity = 1;
}

$item_price = 5.99;
printf("%d x item = $%.2f",
       $quantity, $quantity * $item_price);
?>
<FORM ACTION="buy.php" METHOD=POST>
Обновить количество:
<INPUT NAME="quantity" SIZE=2
 VALUE="<?php echo $quantity;?>">
<INPUT TYPE=SUBMIT VALUE="Изменить количество">
</FORM>
```

При начальной загрузке страницы переменная `$quantity` равна единице, и это значение используется при расчете общей цены. После этого внутри выражения `VALUE` выполняется инструкция PHP, которая выводит значение переменной `$quantity`. Если ввести новое значение, оно будет использоваться вместо предыдущего.

Сценарий нужно сохранить в файле `buy.php`, тогда форма передаст данные в этот же сценарий. Если изменить количество и щелкнуть на кнопке подачи формы, сценарий рассчитает новую итоговую стоимость. Кроме того, после каждой перезагрузки страницы устанавливается новое значение по умолчанию.

Установка атрибута CHECKED для типа CHECKBOX

Атрибут `CHECKED` устанавливает или сбрасывает флажок. С помощью PHP можно выполнить проверку для дескриптора `<INPUT>`, чтобы определить состояние атрибута `CHECKED` для типа `CHECKBOX`:


```
<INPUT TYPE="CHECKBOX"
NAME="mybox" <?php if(условие) echo "CHECKED";?>>
```

Такой вариант выглядит слегка непривычно, потому что два символа > находятся в конце дескриптора — один для того, чтобы закрыть PHP-код, а другой, — чтобы закрыть дескриптор `<INPUT>`. На самом деле положение атрибута `CHECKED` не имеет значения, его можно поставить в другом месте для повышения удобочитаемости:

```
<INPUT <?php if(условие) echo "CHECKED";?>
TYPE="CHECKBOX" NAME="mybox">
```



Закрытие дескрипторов PHP

Все небольшие вставки PHP-кода в HTML нужно обязательно закрывать комбинацией ?>. Если ее пропустить, весь дальнейший HTML-код трактуется как PHP, и на экране получается множество ошибок!

При вставке PHP в HTML-код нужно внимательно следить за пробелами. Если в примере выше удалить пробел после закрытия дескриптора PHP и условие будет истинным, на выходе получим следующий HTML-код:

```
<INPUT CHECKEDTYPE="CHECKBOX" NAME="mybox">
```

Название `CHECKEDTYPE` не является частью дескриптора `<INPUT>`, поэтому браузер вместо `CHECKBOX` выведет стандартный тип `TEXT`! Поэтому лучше всегда оставлять место вокруг динамических элементов в HTML-коде.

Установка переключателя

Атрибут `CHECKED` также используется для группы `RADIO`, чтобы установить элемент по умолчанию. Например, в электронном магазине можно предложить три варианта с различной ценой. Для того чтобы узнать, на каком варианте остановился покупатель, достаточно установить один вариант по умолчанию. При желании покупатель всегда сможет выбрать другой вариант (переключатель сбрасывается только при выборе другого из этой же группы):

```
<INPUT TYPE="RADIO" CHECKED
 NAME="shipping" VALUE="economy"> Экономный
<INPUT TYPE="RADIO" NAME="shipping" VALUE="standard">
Стандартный
<INPUT TYPE="RADIO" NAME="shipping" VALUE="express">
Быстрый
```

Чтобы динамически установить атрибут CHECKED для одного из группы переключателей, нужно создать условие, которое проверит текущее значение переменной $shipping на совпадение с соответствующим элементом. В листинге 12.2 приводится пример.

Листинг 12.2. Установка значения по умолчанию для группы переключателей

```
<?php
if (!isset($shipping)) {
    $shipping = "economy";
}
echo "Вы получите $shipping вариант заказа";
?>
<FORM ACTION="shipping.php" METHOD=POST>
<INPUT TYPE="RADIO" NAME="shipping" VALUE="economy"
 <?php if ($shipping == "economy") echo "CHECKED";?>>
Экономный

<INPUT TYPE="RADIO" NAME="shipping"  VALUE="standard"
 <?php if ($shipping == "standard") echo "CHECKED";?>>
Стандартный

<INPUT TYPE="RADIO" NAME="shipping" VALUE="express"
 <?php if ($shipping == "express") echo "CHECKED";?>>
Быстрый
<INPUT TYPE="SUBMIT" VALUE="Изменить вариант заказа">
</FORM>
```

Такой вариант выглядит не очень компактно даже для небольшой группы переключателей из трех элементов. Ниже в этом уроке показывается, как динамически создать группу переключателей с помощью более изящного механизма.

Установка стандартного значения для раскрывающегося списка

Атрибут SELECTED устанавливает дескриптор <OPTION> как элемент по умолчанию. Если в группе <OPTION> нет элемента с атрибутом SELECTED, первый устанавливается по умолчанию.

Можно использовать PHP для установки атрибута SELECTED, пройдя по всему списку дескрипторов <OPTION>. Но результат будет таким же громоздким, как и для группы RADIO. В листинге 12.3 приводится аналог примера из листинга 12.2 для группы элементов <OPTION>.

Листинг 12.3. Установка стандартного значения для раскрывающегося списка

```
<?php
if (!isset($shipping)) {
    $shipping = "economy";
}
echo " Вы получите $shipping вариант заказа";
?>
<FORM ACTION="shipping.php" METHOD=POST>
<SELECT NAME="shipping">
<OPTION <?php if ($shipping == "economy") echo
"SELECTED";?>
    VALUE="economy">Экономный</OPTION>
<OPTION <?php if ($shipping == "standard") echo
"SELECTED";?>
    VALUE="standard">Стандартный</OPTION>
<OPTION <?php if ($shipping == "express") echo
"SELECTED";?>
    VALUE="express">Быстрый</OPTION>
<INPUT TYPE="SUBMIT" VALUE="Изменить вариант заказа">
</FORM>
```

Как и для группы переключателей, с помощью функции динамической генерации можно создавать большие раскрывающиеся списки и задавать нужный стандартный вариант.

Создание элементов формы

Теперь рассмотрим, как с помощью специальных PHP-функций можно динамически генерировать элементы HTML-форм. Такая модульность позволяет использовать функцию снова и снова, если возникает необходимость повторно задействовать соответствующий элемент.

Создание динамической группы переключателей

Для процедуры, которая генерирует группу переключателей, нужны три параметра: имя группы, список значений и список названий. Можно использовать ассоциативный массив для того, чтобы передавать одновременно значения и их названия в функцию.

Пусть нужно реализовать функциональность, которая с помощью следующего кода генерирует группу переключателей:

```
$options = array("economy" => "Экономный",
                 "standard" => "Стандартный",
                 "express" => "Экспресс");
$default = "economy";
$html = generate_radio_group("shipping", $options,
$default);
```

Как видно, такая функция очень подходит для регулярного применения в HTML-формах. Набор похожих функций значительно облегчает выполнение рутинных задач. Ниже приводится пример реализации функции generate_radio_group:

```
function generate_radio_group($name, $options, $default="") {
    name = htmlentities($name);
    foreach($options as $value => $label) {
        $value = htmlentities($value);
        $html .= "<INPUT TYPE=\"RADIO\" ";
        if ($value == $default) {
            $html .= "CHECKED ";
        }
        $html .= "NAME=\"$name\" VALUE=\"$value\">";
        $html .= $label . "<br>";
    }
    return($html);
}
```


Ядром функции является цикл, который проходит по всем значениям массива $options. Он создает по очереди каждый дескриптор <INPUT> с одинаковым значением атрибута NAME и разными — атрибута VALUE. Поясняющий текст размещается после каждой кнопки. Сразу за ним ставится форматирующий дескриптор
 между каждым переключателем этой группы. При желании можно отформатировать этот список с помощью таблицы или другим подходящим способом.

При очередной итерации сценарий сравнивает текущее значение $value с переданным значением в переменной $default. Если они совпадают, атрибут CHECKED добавляется в генерирующийся HTML. Напомним, что важно ставить пробелы. Поэтому после CHECKED выводится дополнительный пробел в HTML-коде.

Значение $default является необязательным. Функцию generate_radio_group можно вызвать с двумя аргументами. Тогда не устанавливается переключатель по умолчанию.



HTML-объекты

Функция htmlentities позволяет заменить определенные символы в строке на соответствующие HTML-объекты. Значение переменных $name и $value выводится внутри других HTML-дескрипторов и может их нарушить. Применение htmlentities позволяет избежать этого нежелательного поведения.

Создание динамического раскрывающегося списка

Динамический раскрывающийся список реализуется так же, как и группа переключателей. Теперь в каждой итерации цикла создается дескриптор <OPTION> для всех элементов по очереди. Кроме того функция выводит дескрипторы <SELECT> и </SELECT> вокруг списка элементов <OPTION>. Функция generate_menu выглядит так:

```
function generate_menu($name, $options, $default="") {
   $html = "<SELECT NAME=\"$name\">";
   foreach($options as $value => $label) {
      $html .= "<OPTION ";
```

```
        if ($value == $default)
            $html .= "SELECTED ";
            $html .= "VALUE=\"$value\">$label</OPTION>";
        }
    $html .= "</SELECT>";
    return($html);
}
```

Функция возвращает полный HTML-код для раскрывающегося списка с нужными опциями. Можно также реализовать ее без внешних дескрипторов <SELECT> вокруг, это позволяет установить нужный сценарий JavaScript на событие onChange или любое другое.

Элемент множественного выбора

Атрибут MULTIPLE в элементе <SELECT> позволяет пользователю выбрать несколько элементов из списка. Для этого нужно удерживать клавишу <Ctrl> и щелкнуть мышкой на нужных опциях. Чтобы обработать несколько выбранных (отмеченных) элементов в PHP, имя должно быть массивом. При отправке формы массив будет состоять из значений выбранных элементов в порядке их следования.

Например, создадим список с множественным выбором, как в следующем HTML-коде. Выберем и передадим значения в PHP-сценарий с одной инструкцией — print_r. На экране мы увидим, что элемент $_POST["colors"] является массивом всех выбранных значений:

```
<SELECT MULTIPLE NAME="colors[]">
<OPTION VALUE="red">Красный</OPTION>
<OPTION VALUE="white">Белый</OPTION>
<OPTION VALUE="синий">Синий</OPTION>
</SELECT>
```

Если выбрать все три варианта, в массиве $_POST["colors"] будет три элемента с числовыми индексами от 0 до 2 с соответствующими значениями red, white и blue.

Аналогичный принцип подходит для любого типа ввода. Если существует несколько элементов с одинаковым именем, которое заканчивается парой квадратных скобок, PHP создает массив с таким именем и элементами для каждого значения.

Такой вариант очень удобен, если нужно реализовать множественный выбор с помощью набора флажков. Вместо

того чтобы называть по-разному все флажки, можно установить одинаковое имя массива для всех флажков. После отправки формы создается массив элементов, который содержит все выбранные значения.

Окончательный вариант примера в этом уроке использует функцию generate_checkboxes для создания набора флажков с одинаковым именем. Его можно использовать как альтернативу к <SELECT MULTIPLE>. Это еще один способ реализации множественного выбора в HTML-форме. Функция с простым примером использования приводится в листинге 12.4.

Листинг 12.4. Реализация множественного выбора с помощью флажков

```
<?php
function generate_checkboxes($name,
                    $options, $default=array()) {
    if (!is_array($default)) {
        $default = array();
    }
    foreach($options as $value => $label) {
        $html .= "<INPUT TYPE=CHECKBOX ";
        if (in_array($value, $default)) {
            $html .= "CHECKED ";
        }
        $html .= "NAME=\"{$name}[]\" VALUE=\"$value\">";
        $html .= $label . "<br>";
    }
    return($html);
}

$options = array("movies" => "Ходить в кино",
                 "music" => "Слушать музыку",
                 "sport" => "Играть или смотреть спортивные
передачи",
                 "travel" => "Путешествовать");
$html = generate_checkboxes("interests",
                            $options, $interests);
?>
<H1>Пожалуйста, укажите область ваших интересов</H1>
<FORM ACTION="interests.php" METHOD=POST>
<?php print $html;?>
<INPUT TYPE=SUBMIT VALUE="Продолжить">
</FORM>
```

В функции generate_checkboxes аргумент $default является массивом, а не одиночным значением. Это позволяет выбрать несколько вариантов по умолчанию. Массив $default может точно соответствовать набору значений, которые получает PHP из соответствующего HTML-элемента.

Функция in_array позволяет узнать, какие флажки нужно установить. Она проверяет, находится ли данное значение в списке стандартных значений. Если значение $value содержится в массиве $default, флажок устанавливается при загрузке страницы.

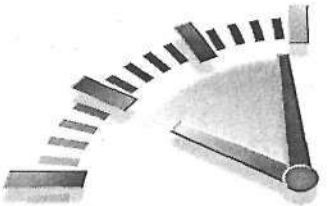
В листинге 12.4 эта функция используется для части Web-странички, где запрашивается область интересов посетителя. Он может выбрать произвольное количество вариантов из списка. После этого сценарий отправляет эти значения самому себе и устанавливает соответствующие флажки при выводе. Тогда пользователь может изменить свой выбор.

В массиве $interests, который формируется в PHP, каждый элемент — это значения ключа из массива $options. Отыскать подпись, соответствующую каждому выбранному варианту, можно, обратившись к соответствующему элементу массива $options.

Резюме

В этом уроке вы научились динамически генерировать HTML-компоненты и освоили несколько приемов для создания динамических элементов ввода. В следующем уроке вы узнаете о том, как проверить значения HTML-форм.

Урок 13



Проверка форм

В этом уроке вы узнаете как реализовать проверку форм в удобном для пользователя виде.

Принцип проверки отправленных пользователем данных очень простой: нужно проверить по очереди все элементы массива $_POST на соответствие набору критериев. Тут возникает другая нетривиальная задача. Нужно дать пользователю возможность с минимальными усилиями исправить все ошибки и снова отправить нужные данные.

Требование нужных полей

Основной принцип проверки форм — требование соответствующего значения для отдельного поля. Если элемент для ввода текста отправляется с пустым значением, элемент в массиве $_POST создается, но содержит пустое значение. Поэтому нельзя использовать функцию isset, чтобы проверить, введено ли значение. Нужно проверить значение элемента, сравнив его с пустой строкой или, как показано ниже, с помощью булева оператора отрицания:

```
if (!$_POST["fieldname"]) { ... }
```

Все поля формы создают соответствующий элемент в $_POST. Поэтому можно воспользоваться простым циклом, чтобы проверить, нет ли пустых значений в массиве:

```
foreach($_POST as $field => $value) {
    if (!$value) {
        $err .= "Необходимо ввести $field<br>";
    }
}
if ($err) {
    echo $err;
    echo "Пожалуйста вернитесь назад и внесите изменения";
```

```
}
else {
    // Продолжение сценария
}
```

Перед тем как написать сообщение о присутствии пустых полей, сценарий создает строку ошибки `$err` для всех пустых полей. После завершения цикла содержимое `$err` выводится на экран, если есть ошибки. Если их нет и `$err` пуста, сценарий продолжает выполняться в выражении `else`.



Предупреждение об ошибках

Всегда выводите все предупреждения об ошибках, связанные с отправленными данными. Нужно дать возможность пользователю исправить все ошибки за один раз.

Такой подход работает, только когда все значения обязательные. Можно улучшить этот механизм, задав список требуемых полей в сценарии. Кроме того, в ассоциативном массиве можно указать надписи для каждого поля для вывода в сообщениях об ошибке:

```
$required = array("first_name" => "Имя",
                  "email" => "Почтовый ящик",
                  "telephone" => "Номер телефона");
foreach($required as $field => $label) {
    if (!$_POST[$field]) {
        $err .= "Необходимо ввести $label<br>";
    }
}
```

Вывод ошибок проверки

Следующая проблема состоит в том, куда направить пользователя в случае ошибки. Раньше рассматривался вариант, когда при обнаружении ошибки пользователю нужно было щелкнуть на кнопке Назад в браузере. А после этого исправить соответствующие ошибки.

Кроме дополнительного действия, пользователю придется снова вводить данные и исправлять ошибки. Примерно такая ситуация возникает в примере с электронным магазином.

Потеря данных при щелчке на кнопке Назад также зависит от настроек кэширования на Web-сервере, в Web-браузере, а также у Internet-провайдера. В большинстве случаев все работает нормально. Но при использовании сеансов PHP автоматически отправляется заголовок `no-cache` на пользовательский браузер. Это приводит к тому, что при использовании кнопки Назад все данные из полей формы обязательно сбрасываются в начальное положение. Про сеансы в PHP рассказывается в уроке 14, "Данные cookies и сеансы".

Поэтому лучше всего хранить форму и сценарий обработки в одном файле и отсылать данные самому себе. В этом случае обнаруженная ошибка выводится на странице формы. А все ранее введенные значения можно автоматически установить по умолчанию.

В листинге 13.1 находится практически готовый пример формы регистрации `register.php`. Имя и адрес электронной почты — обязательные, а номер телефона — необязательный.

Листинг 13.1. Пример формы регистрации с обязательными полями

```
<?php
$required = array("name" => "Ваше имя",
                  "email" => "Адрес электронной почты");
foreach($required as $field => $label) {
    if (!$_POST[$field]) {
        $err .= "Необходимо ввести $label<br>";
    }
}
if ($err) {
    echo $err;
}
?>
<FORM ACTION="register.php" METHOD=POST>
<TABLE BORDER=0>
<TR>
  <TD>Your Name</TD>
  <TD><INPUT TYPE=TEXT SIZE=30 NAME="name"
             VALUE="<?php echo $_POST["name"];?>"></TD>
</TR>
<TR>
  <TD>Адрес электронной почты</TD>
  <TD><INPUT TYPE=TEXT SIZE=30 NAME="email"
             VALUE="<?php echo $_POST["email"];?>"></TD>
</TR>
<TR>
  <TD>Номер телефона</TD>
```

```
  <TD><INPUT TYPE=TEXT SIZE=12 NAME="telephone"
             VALUE="<?php echo
$_POST["telephone"];?>"></TD>
</TR>
</TABLE>
<INPUT TYPE=SUBMIT VALUE="Зарегистрироваться">
</FORM>
<?php
}
else {
    echo "Спасибо за регистрацию";
}
?>
```

Недостаток такого решения в том, что сообщение об ошибке возникает еще до отправки формы. Это исправляется проверкой на существование в сценарии массива $_POST с помощью функции is_array. Кроме того, для проверки $err используется $_POST и, если его нет, форма вообще не выводится на экран.

Условие, которое проверяет $err, охватывает HTML-форму, и, несмотря на то, что дескрипторы PHP закрыты вокруг участков HTML-кода, форма выводится на экран, только если выполняется условие.

После удачного завершения работы с формой, выводится простое сообщение. В этом месте можно выполнить обработку полученных данных. Например, записать их в базу данных. О том, как это сделать, рассказывается в уроке 19, "Использование базы данных MySQL". Или принудительно перенаправить браузер на другую страницу с помощью HTTP-заголовка Location, как показывается ниже:

```
header("Location: newpage.php");
```

Обязательные правила для данных

Иногда кроме проверки того, введены ли данные, нужно проверить корректны ли они, перед тем как их обработать. Например, можно проверить адрес электронной почты или номер телефона на соответствие формату, с помощью правил из урока 8, "Регулярные выражения". Кроме того, можно задать минимальный размер поля. Это позволяет

убедиться, что пользователь не ввел x во всех полях для быстрого перехода к следующей странице.

Все поля могут иметь различные правила проверки, поэтому нельзя воспользоваться циклом. Придется написать отдельное правило для каждого поля. Кроме того, если использовать этот механизм в связке с проверкой на пустое значение поля, нужно сначала убедиться, что значение введено, а затем выполнять дальнейшую проверку. Иначе, кроме ошибки о том, что поле пустое, пользователь получит сообщение о некорректном формате этого поля.

Следующие правила нужно добавить к листингу 13.1 после проверки обязательных полей. Они проверяют корректность форматов почтового адреса и телефонного номера:

```
if ($_POST["email"] &&
    !ereg("^[^@]+@([a-z0-9\-]+\.)+[a-z]{2,4}$",
        $_POST["email"])) {
    $err .= "Неправильный формат адреса электронной
почты<br>";
}

if ($_POST["telephone"] &&
    !ereg("^\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$",
        $_POST["telephone"])) {
    $err .= "Номер телефона нужно указать в формате:
(555)555-5555 <br>";
}
```

Дополнительные правила в случае ошибки добавляют новые сообщения к $err. Остальная часть сценария остается той же.

Подсветка полей с ошибками

Вместо того чтобы заваливать пользователя сообщениями об ошибке, лучше выделить цветом соответствующие поля.

Реализация очень напоминает пример выше. Но вместо добавления каждого сообщения об ошибке к $err нужно задать сообщение об ошибке для всех полей. Для хранения сообщений можно использовать массив. Тогда для проверки формы достаточно посчитать количество элементов в $warnings.

Для каждого поля записывается правило. Если оно нарушается, в массив $warnings добавляется соответствующее сообщение об ошибке:

```
if (!ereg("^[^@]+@([a-z\-]+\.)+[a-z]{2,4}$",
        $_POST["email"])) {
    $warnings["email"] = "Неправильный формат";
}
```

После этого в самой форме можно вывести сообщение вслед за соответствующим полем:

```
<TR>
  <TD>Адрес электронной почты</TD>
  <TD><INPUT TYPE=TEXT SIZE=30 NAME="email"
             VALUE="<?php echo $_POST["email"];?>"></TD>
  <TD><b><?php echo $warnings["email"];?></b></TD>
</TR>
```

В листинге 13.2 приводится обновленная версия файла register.php, в которой используется прием с подсветкой пропущенных или некорректных значений.

Листинг 13.2. Проверка форм с использованием внутренних сообщений

```
<?php
$required = array("name" => "Ваше имя",
                  "email" => "Адрес электронной почты");
foreach($required as $field => $label) {
    if (!$_POST[$field]) {
        $warnings[$field] = "Требуется";
    }
}

if ($_POST["email"] &&
    !ereg("^[^@]+@([a-z\-]+\.)+[a-z]{2,4}$",
$_POST["email"])) {
    $warnings["email"] = "Неправильный формат адреса
электронной почты";
}

if ($_POST["telephone"] &&
    !ereg("^\([[:digit:]]{3}\)[[:digit:]]{3}-
[[:digit:]]{4}$",
        $_POST["telephone"])) {
    $warnings["telephone"] = "Номер телефона нужно указать
в формате:(555)555-5555";
}

if (count($warnings) > 0) {
?>
<FORM ACTION="register.php" METHOD=POST>
<TABLE BORDER=0>
<TR>
  <TD>Ваше имя</TD>
```



```
  <TD><INPUT TYPE=TEXT SIZE=30 NAME="name"
             VALUE="<?php echo $_POST["name"];?>"></TD>
  <TD><?php echo $warnings["name"];?></TD>
</TR>
<TR>
  <TD>Адрес электронной почты</TD>
  <TD><INPUT TYPE=TEXT SIZE=30 NAME="email"
             VALUE="<?php echo $_POST["email"];?>"></TD>
  <TD><?php echo $warnings["email"];?></TD>
</TR>
<TR>
  <TD>Номер телефона</TD>
  <TD><INPUT TYPE=TEXT SIZE=12 NAME="telephone"
             VALUE="<?php echo
$_POST["telephone"];?>"></TD>
  <TD><?php echo $warnings["telephone"];?></TD>
</TR>
</TABLE>
<INPUT TYPE=SUBMIT VALUE="Зарегистрироваться">
</FORM>
<?php
}
else {
    echo "Спасибо за регистрацию";
}
?>
```

В первом цикле присваивается текст предупреждения "Требуется" всем незаполненным полям. Каждое специфическое правило проверки содержит собственное сообщение об ошибке.

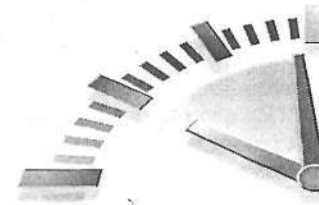
Способ выделения поля можно выбирать самому в зависимости от личной фантазии и умения воплотить ее в виде HTML. Например, проверяя наличие элемента в $warnings для каждого поля, можно изменить стиль поля ввода на затененный цвет фона, как показывается ниже:

```
<INPUT TYPE=TEXT SIZE=30 NAME="email"
<?php if ($warnings["email"]) echo "STYLE=\"shaded\"";?>
VALUE="<?php echo $_POST["email"];?>">
```

Резюме

В этом уроке вы узнали о том, как проверить значения, введенные пользователем в HTML-форму, и как предоставить возможность исправить ошибки. В следующем уроке вы узнаете о данных cookies и сеансах в PHP.

Урок 14



Данные cookies и сеансы

В этом уроке рассматриваются механизмы обмена информацией между страницами Web-сайта без использования форм. Это данные cookies и PHP-сеансы.

Файлы Cookies

Файлы cookies — это маленькие кусочки информации, которые сохраняются в пользовательском браузере. Обычно в них хранится информация, которая используется для идентификации пользователя. Она позволяет индивидуально настроить Web-сайт под каждого посетителя.

Вместо того чтобы каждый раз отправлять данные через форму или как параметр URL, данные cookies автоматически возвращаются в сценарий пользовательским браузером. Даже после окончания работы с сайтом и перехода на другой, эти данные могут хранится до следующего посещения.

Например, если реализовать защищенный доступ к Web-сайту, можно хранить имя пользователя в файлах cookies. Это позволит избежать его повторного набора при очередном посещении. Теперь нужно вводить только пароль. На сайтах сообществ можно записывать данные о последнем посещении в файлы cookies. Тогда можно сделать так, чтобы все новые сообщения на форуме, с момента последнего посещения, выделялись как новые.

Составляющие cookies

Параметр cookie состоит из имени и значения, как обычная переменная PHP. Сервер отправляет инструкцию

создать значение cookie браузеру при помощи специального HTTP-заголовка. Это выполняется до отправки содержимого страницы. Встречая такой заголовок, браузер выполняет соответствующее действие.

HTTP-заголовок, который создает cookie, имеет одинаковый вид, независимо от того, какими средствами он генерируется. Это может быть PHP или другой механизм Web-сервера. Создадим заголовок, который создает cookie под названием email:

```
Set-Cookie: email=chris@lightwood.net
```



HTTP-заголовок

Обычно HTTP-заголовки не отображаются в браузере. Как с помощью PHP посылать различные типы HTTP-заголовков, рассматривается в уроке 16, "Взаимодействие с Web-сервером".

У данных cookies есть определенное время жизни. Некоторые существуют до закрытия окна браузера и хранятся в оперативной памяти компьютера. Другие имеют фиксированное время жизни и теряют актуальность в определенный момент. Они хранятся на жестком диске. Следующий HTTP-заголовок создает cookie email, который заканчивается в конце 2005 года:

```
Set-Cookie: email=chris@lightwood.net;
            expires=Sat, 31-Dec-2005 23:59:59 GMT
```

Если не указать атрибут expires в заголовке Set-Cookie, значение cookie автоматически уничтожается при закрытии окна браузера.

Кроме того, существуют дополнительные атрибуты — имя домена и путь, куда браузер будет возвращать значение cookies. При повторном посещении страницы, для которой установлены cookies, браузер посылает эти значения обратно на Web-сервер.

По умолчанию cookies пересылаются обратно на любую страницу одного и того же домена, с которого они были получены. Если установить значение домена и путь, то cookies посылаются только на заданные поддомены или на отдельные страницы сайта.

Заголовок ниже создает cookie email. Его значение отправляется на любой поддомен lightwood.net, если запрашивается страница из каталога /scripts:

```
Set-Cookie: email=chris@lightwood.net;
domain=.lightwood.net;
            path=/scripts
```



Поддомены

Атрибут cookies domain можно установить либо для домена из которого они отправляются, либо для всех поддоменов домена из которого они отправляются. Для этого нужно использовать формат *.ваш_домен*.com. Это защитный механизм. Он позволяет избежать влияния одного сайта на другие. Например, нельзя установить cookies для домена www.php.net из любых Web-сайтов, которые не размещены на сервере php.net.

Доступ к значениям cookies

Суперглобальный массив $_COOKIE содержит все данные cookies, которые браузер отправляет для сценария. Данные cookies передаются обратно на Web-сервер с помощью HTTP-заголовков. На основе этой информации PHP создает массив $_COOKIE.

Получить доступ к значению cookie можно так же, как и к данным формы. Например, в следующем выражении выводится текущее значение cookie email:

```
echo $_COOKIE["email"];
```

Иногда с данными cookies возникает неразбериха. Тогда можно создать простой сценарий, который покажет все значения на экране. Это позволяет быстро обнаружить проблему. Вот простой пример:

```
echo "<PRE>";
print_r($_COOKIES);
echo "</PRE>";
```

Создание данных cookies с помощью PHP

Выше упоминался способ создания данных cookies с помощью HTTP-заголовков. Но этим способом пользуются мало. Дело в том, что в PHP есть специальная функция, которая выполняет это значительно проще:

```
setcookie("email", "chris@lightwood.net", time() + 3600);
```

Здесь не нужно использовать строгий формат даты. Время жизни для `setcookie` можно задать в формате временной метки Unix. Это позволяет установить нужное время жизни на фиксированный отрезок времени или до определенного момента в будущем.



Время жизни

Время жизни задает последнюю дату, когда нужно передавать обратно данные cookies. Процесс сравнения происходит на локальном компьютере и зависит от настройки системных часов. Поэтому, если они настроены неправильно, этот процесс становится неконтролируемым.

Два следующих параметра используются для того, чтобы задать имя домена и путь для cookies. Иногда нужно установить имя домена и путь, но не задавать время жизни. Для этого нужно установить NULL для третьего аргумента:

```
setcookie("email", "chris@lightwood.net", NULL,
          ".lightwood.net", "/scripts");
```

Последний необязательный аргумент `setcookie` — это флаг, который устанавливает для браузера безопасный режим передачи данных cookies. Для этого используется зашифрованное SSL-соединение с Web-сервером. На это указывает то, что адрес страницы начинаться с `https://`.



Сохранение пароля в файлах cookies

Иногда может возникнуть соблазн сохранять пароль в cookies, чтобы автоматически авторизироваться при очередном посещении. Это небезопасно, даже если установлен флаг `secure`.

Значения cookies хранятся в файле как обычный текст. Их можно увидеть, просмотрев содержимое файла на жестком диске. Многие программы-шпионы пытаются выкрасть значение пароля, который находится в файлах cookies.

Удаление cookies

В PHP нет функции `unsetcookie`, которая бы позволяла удалить данные cookies. Но это можно сделать с помощью функции `setcookie`. Для этого нужно задать пустое значение и время жизни, которое уже прошло.

В примере ниже удаляется значение cookie `email`. Для этого время жизни устанавливается на час назад от текущего момента:

```
setcookie("email", "", time() - 3600);
```



Изменение значений cookies

Чтобы удалить или установить новое значение данных cookies, нужно аргументы `domain`, `path` и `ssl-only` оставить такими же, как и при первом создании.

Сеансы

Сеансы очень напоминают cookies, потому что тоже позволяют передавать значения между страницами Web-сайта. Но вместо того, чтобы хранить все значения на каждом Web-браузере, Web-сервер хранит все данные у себя. Браузер получает только одно значение cookie для идентификации. По этому идентификатору PHP определяет, какие значения относятся к данному пользователю.

Так как браузер и Web-сервер обмениваются только одним значением, сеансы более эффективны, чем cookies, для хранения больших массивов данных.

Открытие сеанса

Для того чтобы инициализировать новый сеанс в PHP-сценарии, используется функция session_start. Можно также задать необязательный аргумент названия сеанса, но обычно это не используется. Все сценарии на сайте, с которого открывается сеанс, имеют доступ к одинаковому набору переменных сеанса.

Вызов функции session_start создает новый сеанс:

```
session_start();
```

Суперглобальный массив $_SESSION хранит все значения переменных сеанса. В отличие от других суперглобальных массивов, которые рассматривались выше, $_SESSION позволяет напрямую присвоить значение. После этого оно становится доступным во всех сценариях, которые используют этот сеанс.

Рассмотрим сценарий в листинге 14.1, где устанавливаются две переменные сеанса — счетчик посещений страницы и дата последнего визита в формате временной метки.

Листинг 14.1. Использование переменных сеанса для подсчета количества обращений к странице

```
<?php
session_start();
if ($_SESSION["last_visit"]) {
    echo "Дата последнего посещения: ";
    echo date("j F Y, H:i:s", $_SESSION["last_visit"]);
    echo "<br>";
    echo "Всего посещений: ".$_SESSION["num_visits"];
}
else {
    echo "Вы впервые на этой странице";
}
$_SESSION["last_visit"] = time();
$_SESSION["num_visits"]++;
?>
```

Каждый раз при загрузке страницы показываются новые установленные значения. Даже если зайти на другие сайты, а потом вернуться, эти значения сохраняются. Но если закрыть окно браузера и после этого снова зайти, значения сбрасываются.

Использование переменных сеанса

Переменные сеанса, в отличие от данных cookies, позволяют использовать данные различных типов PHP. Данные cookies всегда хранятся в текстовом виде, а переменные сеанса могут принимать любые значения, которые можно установить для обычной переменной PHP.

Например, чтобы хранить список значений в файлах cookies, нужно создать массив и передать его функции serialize. А в переменных сеанса можно просто создать массив и хранить эти данные в сеансе.

В листинге 14.2 используется массив для хранения значений, полученных из формы. Это довольно тривиальный пример, но он наглядно демонстрирует гибкость применения переменных сеанса.

Листинг 14.2. Использование массива в переменных сеанса

```
<?php
session_start();
if (isset($_POST["word"])) {
    $_SESSION["words"][] = $_POST["word"];
}

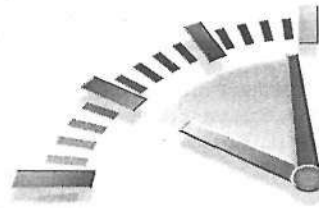
if (is_array($_SESSION["words"])) {
    foreach($_SESSION["words"] as $word) {
        echo $word . "<br>";
    }
}

?>
<FORM ACTION="list.php" METHOD=POST>
Введите слово: <INPUT SIZE="10" NAME="word">
<INPUT TYPE=SUBMIT VALUE="Добавить слово в список">
</FORM>
```

Резюме

В этом уроке вы узнали, как установить значения cookies из PHP и использовать сеансы PHP для хранения значений. В следующем уроке вы узнаете, как создать систему аутентификации пользователя на основе этих механизмов.

Урок 15



Аутентификация пользователя

В этой главе вы узнаете, как реализовать аутентификацию пользователя для того, чтобы защитить отдельные страницы с помощью пароля.

Типы аутентификации

Рассмотрим, как проходит процесс аутентификации с пользовательской стороны. В общем случае нужно запросить ввести имя пользователя, иногда адрес электронной почты, и пароль.

Есть два оптимальных варианта аутентификации пользователя на Web-сайте: с помощью базовой HTTP-аутентификации и с помощью механизма, основанного на сеансах. Ниже рассказывается об отличиях этих методов.

Базовая HTTP-аутентификация

Базовую HTTP-аутентификацию можно реализовать с помощью Web-сервера без участия PHP-сценария. Пример ниже работает для Web-сервера Apache. Для других Web-серверов нужно обратиться к соответствующей документации.

Обычно такая аутентификация устанавливается для каталога, но можно указать и конкретный файл. Файл `.htaccess` позволяет выполнить специальную настройку папки Web-сайта. Кроме того, в нем можно задать необходимость авторизации перед входом. Обычно набор директив настройки выглядит так:

```
AuthType Basic
AuthName "Защищенный Web-сайт"
```

```
AuthUserFile /home/yourname/htpasswd
require valid-user
```

AuthUserFile указывает место, где находится файл с паролем, который генерируется утилитой htpasswd. Чтобы создать новый файл с паролем, нужно выполнить следующую инструкцию в командной строке:

```
$ htpasswd -c /home/yourname/htpasswd chris
New password:
Re-type new password:
```



Файл паролей

Опция -c указывается только при создании нового файла. Программа htpasswd не спрашивает о необходимости перезаписать существующий файл. Утилита htpasswd без опции -c просто добавляет пользователя в существующий файл.

Нужно ввести новый пароль дважды, после чего программа добавит новую запись в указанный файл паролей. Запись состоит из имени пользователя и зашифрованного пароля, который разделяется двоеточием. Но на самом деле не нужно напрямую работать с этим файлом. Обычно файл с паролями выглядит примерно так:

```
chris:XNiv7qSUTFPU6
damon:ZxxE2PTEXeVNU
shelley:SVzAEtxMLEAls
vanessa:cX/t1Pv2oQfrY
```

При доступе к файлу в защищенном каталоге в браузере появится окошко с запросом имени пользователя и пароля. Запрошенная страница загрузится только после ввода корректных данных.

Инструкция require valid-user дает указание Web-серверу показывать страницу только авторизированному пользователю. С помощью инструкции require user можно разрешить доступ только для определенных пользователей:

```
require user chris damon shelley
```

Базовая HTTP-аутентификация позволяет установить доступ к отдельной части сайта для группы пользователей. С помощью инструкции require group можно разрешить доступ для одной или больше группы пользователей.

Ниже приводится пример файла для групп, который обычно имеет имя htgroups. Он разделяет пользователей в файле паролей на две группы:

```
boys: chris damon
girls: shelley vanessa
```

Чтобы разрешить доступ только для группы boys, используется следующий файл .htaccess:

```
AuthType Basic
AuthName "Только для парней"
AuthUserFile /home/yourname/htpasswd
AuthGroupFile /home/yourname/htgroup
require group boys
```

Несмотря на то что базовая HTTP-аутентификация довольно гибкая в настройке, она имеет некоторые недостатки. Во-первых, нельзя изменить внешний вид окошка аутентификации. Если нужно изменить процесс в целом, этот метод не подходит. Во-вторых, файл с паролем хранится в файловой системе Web-сервера. Это затрудняет его изменение с помощью сценария. Подробнее о проблемах с чтением и изменением файлов рассказывается в уроке 17, "Работа с файловой системой".



Дополнительные модули сервера Apache

Некоторые модули от сторонних разработчиков для Web-сервера Apache, например mod_auth_mysql и mod_auth_sqlite, позволяют использовать базовую HTTP-аутентификацию в более удобном виде. Вся информация о паролях хранится в базе данных. Но они установлены не на всех Web-серверах.

Аутентификация с помощью сеансов

Полностью управляемый процесс аутентификации пользователя легко реализовать с помощью PHP-сеансов.

Проще говоря, при аутентификации пользователя в сеансе записывается специальная информация. С ее помощью при попытке доступа к защищенным страницам легко идентифицировать пользователя. При авторизации пользователь вводит имя и пароль в соответствующие поля формы. Теперь вид формы и процесс аутентификации можно настроить под конкретные требования.

Существенное отличие от базовой HTTP-аутентификации состоит в том, что инструкции проверки сеанса пользователя находятся в самом сценарии, а не в конфигурационном файле.



Защита HTML

Если на Web-сайте есть файлы с обычным HTML, без инструкций PHP, и нужно защитить их от несанкционированного доступа, нужно установить для них расширение `.php` и добавить необходимый PHP-код.

Создание системы аутентификации

В оставшейся части урока рассказывается способ реализации аутентификации с помощью сеансов PHP.

Как работает система

Нужно разработать две компоненты. Одна — отвечает за процесс авторизации. Она проверяет имя пользователя и пароль. Вторая компонента представляет собой блок кода, который нужно добавить в начало всех файлов. Этот блок проверяет переменные сеанса и правильность аутентификации перед тем как, продолжить процесс.



Форма аутентификации

Нужно всегда использовать метод `POST` для формы аутентификации. Если отправить имя пользователя и пароль с помощью метода `GET`, эти значения появятся в URL на следующей странице. И тогда их смогут увидеть другие!

Лучше всего выделить блок проверки сеанса в отдельный подключаемый файл `auth.inc`. Тогда для того, чтобы защитить страницу, достаточно добавить такую инструкцию в сценарий:

```
include "auth.inc";
```


Можно использовать всего одну переменную сеанса для имени авторизированного пользователя. Если переменная содержит имя пользователя, значит, он прошел аутентификацию. Тогда для выхода достаточно удалить переменную сеанса. Это довольно безопасный метод. Потому что конфликтующий сеанс можно создать, только работая на том же Web-сервере и под тем же именем домена. Тогда файл `auth.inc` принимает следующий вид:

```
session_start();
if (!isset($_SESSION["auth_username"])) {
    echo "Вам нужно пройти аутентификацию, чтобы получить
доступ к этой странице";
    exit;
}
```

Здесь просто выводится сообщение об ошибке, и завершается выполнение сценария, если пользователь не авторизировался. Ниже показывается, как повысить удобство этого механизма, но сначала нужно создать саму процедуру авторизации.

Аутентификация пользователя

Ядро формы аутентификации составляют два поля — `username` и `password`, а также кнопка подачи формы. Этих компонент достаточно для работы формы. Ее можно отформатировать с помощью обычной таблицы. В листинге 15.1 приводится пример простой формы.

Листинг 15.1. Простая форма аутентификации

```
<FORM ACTION="login.php" METHOD="POST">
<TABLE BORDER=0>
<TR>
  <TD>Имя пользователя:</TD>
  <TD><INPUT TYPE="TEXT" SIZE=10 NAME="username"></TD>
</TR>
<TR>
  <TD>Пароль:</TD>
  <TD><INPUT TYPE="PASSWORD" SIZE=10 NAME="password"></TD>
</TR>
</TABLE>
<INPUT TYPE=SUBMIT VALUE="Авторизироваться">
</FORM>
```



Поле пароля

Тип PASSWORD дескриптора <INPUT> работает так же, как и тип TEXT. Но при вводе символы отображаются звездочками. Для типа PASSWORD есть одно ограничение: нельзя задать стандартное значение с помощью атрибута VALUE.

Сценарий обработки формы login.php ищет полученные значения имени и пароля в списке пользователей. В большинстве случаев достаточно проверить таблицу с пользователями. Про доступ к базам данных из PHP рассказывается в уроках 19, "Использование базы данных MySQL", и 20, "Абстрагирование от базы данных". А сейчас можно просто создать ассоциативный массив с пользователями, которые имеют доступ к сайту. В листинге 15.2 показано, как это сделать.

Листинг 15.2. Сценарий обработки данных аутентификации

```
<?php
session_start();
$passwords = array("chris" => "letmein",
                   "damon" => "thisisme",
                   "shelley" => "mypassword",
                   "vanessa" => "opensesame");

if (!$_POST["username"] or !$_POST["password"]) {
    echo "Нужно ввести имя пользователя и пароль";
    exit;
}
if ($_POST["password"] == $passwords[$_POST["username"]]) {
    echo "Аутентификация прошла успешно";
    $_SESSION["auth_username"] = $_POST["username"];
}
else {
    echo "Неправильные данные аутентификации";
}
?>
```

Сначала создается ассоциативный массив. Ключ соответствует имени пользователя, а пароль — значению. Сперва сценарий проверяет наличие значений пароля и имени пользователя.

После этого проверяется значение пароля для ключа с полученным именем пользователя. Если пароли совпадают, пользователь авторизируется. При этом инициализируется переменная сеанса auth_username. В противном случае выводится сообщение об ошибке.

Если проверка в auth.inc прошла успешно, пользователь получит доступ к защищенной странице.

Шифрование паролей

В приведенном выше примере пароли сохраняются в виде текста. Это небезопасно, потому что каждый, кто может просмотреть исходный код сценария, увидит пароль любого пользователя.



Длинный нос

Даже если сервер сверхзащищенный, можно подсмотреть пароль, просто заглянув через плечо. Поэтому желательно не показывать незашифрованные пароли на экране.

Функция PHP crypt предоставляет простой и эффективный алгоритм одностороннего шифрования. Аналогичный механизм использует программа htpasswd и даже система паролей Unix. Чтобы зашифровать пароль, нужно подать его на вход crypt вместе с переменной $salt, которая содержит другую строку, на которой основывается шифрование:

```
$crypt_password = crypt($password, $salt);
```

Зашифрованную строку нельзя расшифровать. Но функция crypt возвращает тот же результат для того же пароля и строки salt. Таким образом, можно хранить зашифрованную версию пароля и сравнивать ее с зашифрованной версией пароля, который вводит пользователь.



Основа

Если при вызове функции crypt опустить строку salt, последняя будет выбрана случайным образом. В результате для двух одинаковых вызовов этой функции на выходе получатся различные результаты. Для строки salt достаточно двух символов. Именно столько используется при шифровке файла паролей Unix и в утилите htpasswd.

Процесс кодирования с помощью crypt может отличаться для различных серверов. Это зависит от библиотек шифрования, которые используются в системе. Поэтому зашифрованные пароли могут быть несовместимыми между разными системами.

Новая версия login.php приводится в листинге 15.3. Но нужно помнить, что зашифрованные пароли могут не сработать в другой системе.

Листинг 15.3. Сценарий аутентификации с зашифрованными паролями

```
<?php
session_start();
$passwords = array("chris" => "ZXsDiRf.VBlWQ",
                   "damon" => "bQLXBRzdBci7M",
                   "shelley" => "KkTH39mVsoclc",
                   "vanessa" => "69SvRIB9QVukk");
if (!$_POST["username"] or !$_POST["password"]) {
    echo "Вам нужно ввести имя пользователя и пароль";
    exit;
}
$salt = substr($passwords[$_POST["username"]], 0, 2);
if (crypt($_POST["password"], $salt)
== $passwords[$_POST["username"]]) {
    echo "Аутентификация успешно пройдена";
    $_SESSION["auth_username"] = $_POST["username"];
}
else {
    echo "Неправильные данные аутентификации";
}
?>
```

Строка salt находится в первых двух символах зашифрованной строки. Это позволяет присвоить эти два символа $salt и передать на вход функции crypt. В остальном процесс аутентификации полностью идентичен использованию обычных текстовых паролей.

Анализ удобства использования

Реализованный механизм еще очень сырой. В результате ошибки выводится сообщение, и сценарий завершается. И даже если аутентификация успешно пройдена, сценарий завершается, и нужно снова зайти на ту же страницу, чтобы продолжить работу.

В идеале при попытке доступа к защищенной странице сценарий должен выводить форму авторизации. После удачного прохождения аутентификации пользователя перебрасывает на страницу, куда он хотел изначально попасть.

Для того чтобы это реализовать, код в auth.inc должен "знать" адрес и параметры, с которыми пользователь обращался к странице. Эти данные хранятся в переменной $_SERVER["REQUEST_URI"]. После этого auth.inc выводит форму авторизации вместо того, чтобы выносить ее на отдельную страницу.

Следующий элемент ввода HIDDEN нужно добавить в форму аутентификации. Он позволит обработчику аутентификации узнать, из какого сценария происходит вызов. Теперь обработчик сможет перенаправить пользователя на изначально запрашиваемую страницу:

```
<INPUT TYPE="HIDDEN" NAME="destination"
VALUE="<?php print $_SERVER["REQUEST_URI"];?>">
```

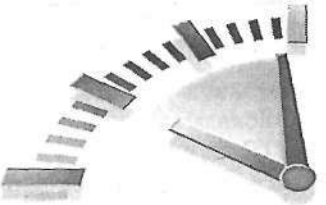
Вместо сообщения об удачной аутентификации нужно перенаправить пользователя на запрашиваемую страницу с помощью следующего выражения:

```
header("Location: $_POST["destination"]);
```

Резюме

В этом уроке вы узнали, как защитить страницу с помощью различных методов аутентификации. В следующем уроке вы увидите, как с помощью PHP получить доступ к другим возможностям HTTP.

Урок 16



Взаимодействие с Web-сервером

В этом уроке рассматриваются способы взаимодействия PHP с Web-сервером.

HTTP-заголовки

Перед тем как получить нужную страницу с Web-сервера, происходит обмен HTTP-сообщениями. Web-браузер посылает набор инструкций с запросом определенной страницы, а сервер дает ответ, в котором содержится информации об успешности запроса. Кроме того, идет обмен множеством другой информации. Эта информация не выводится непосредственно на страницу.

Вот пример HTTP-заголовков Web-сервера с поддержкой PHP, которые приходят в ответ на запрос браузера вместе с обычной Web-страницей:

```
HTTP/1.1 200 OK
Date: Tue, 14 Dec 2004 21:17:28 GMT
Server: Apache/1.3.29 (Unix) mod_gzip/1.3.26.1a PHP/4.3.9
        mod_ssl/2.8.16 OpenSSL/0.9.7c
X-Powered-By: PHP/4.3.9
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

Отправка специальных заголовков

С помощью функции `header` можно отправлять различные HTTP-заголовки. Сначала отправим заголовок, который не выполняет никаких действий. Все заголовки, начинающиеся с `X`, игнорируются браузером и содержат допол-

нительную информацию. Например, заголовок X-Powered-By показывает, что на сервере установлен PHP. Следующий HTTP-заголовок выводит имя автора:

```
header("X-PHP-Author: Chris Newman <chris@lightwood.net>");
```

Конечно, это не приносит никакой пользы, кроме того, что тешит тщеславие автора. Обычный пользовательский браузер даже не заметит этого заголовка!

Выше рассказывалось, что данные cookies можно посылать пользовательскому браузеру с помощью функции setcookie. Также известно, что при вызове этой функции отправляется HTTP-заголовок Set-Cookie. Следующие PHP-выражения эквивалентны:

```
setcookie("mycookie", "somevalue");
header("Set-Cookie: mycookie=somevalue");
```

Заголовки перенаправления

На практике чаще всего используется заголовок Location. Он дает браузеру указание перейти на другой URL. Этот заголовок позволяет управлять порядком прохождения по страницам в зависимости от событий в сценарии. Следующая инструкция перенаправляет пользовательский браузер на другую страницу:

```
header("Location: anotherpage.php");
```

В заголовке Location можно использовать относительные и абсолютные URL. Это позволяет перенаправить пользователя даже на другой домен, как в примере ниже:

```
header("Location: http://www.имя_домена.com/newpage.php");
```

После отправки заголовка Location нужно сразу прервать выполнение сценария командой exit. Тогда браузеру больше не отправится другая информация.

Проверка факта отправки заголовков

Перед тем как начать работать с незаголовочной частью страницы, PHP проверяет, отправлены ли необходимые заголовки на пользовательский браузер. После этого начинается непосредственная работа со страницей. Все HTTP-заголовки нужно передать до отправки содержимого страницы.

Если заголовки уже отправлены, то при попытке повторить это еще раз, PHP выдаст ошибку:

```
Warning: Cannot modify header information - headers already
sent by (output started at /home/chris/public_html/
header.php:4)in /home/chris/public_html/header.php on line 5
```

В случае отправки заголовка Location не нужно больше выводить другой информации, потому что браузер переходит непосредственно на новый адрес. Несмотря на это, нужно осторожно обращаться с HTML-выводом, а также с пробелами. Перевод строки перед дескриптором <?php блокирует отправку дополнительных HTTP-заголовков.

В PHP есть специальная функция headers_sent, которая позволяет проверить, отправлены ли заголовки HTTP. Функция возвращает TRUE, если заголовки отправлены. А FALSE получим, если еще можно отправить дополнительные заголовки.

Условие ниже проверяет, можно ли отправить дополнительные заголовки, и только после этого посылает заголовок перенаправления:

```
if (!headers_sent()) {
    header("Location: newpage.php");
}
```

Конечно, в случае неудачи нужно, чтобы сценарий продолжил работу.

Два необязательных параметра функции headers_sent позволяют узнать название сценария и номер строки, в которой произошла отправка заголовков. Это полезно, если заголовки отправлены по ошибке и нужно узнать, где это произошло.

Сценарий в листинге 16.1 пытается отправить заголовок Location. В случае неудачи предлагается альтернативный путь попадания на нужную страницу. Можно выполнить его на собственном сервере. Для этого в сценарии перед первым дескриптором <?php нужно поставить пробел. Это вызовет преждевременную отправку заголовков.

Листинг 16.1. Сценарий с проверкой возможности отправки заголовков

```
<?php
$destination = "http://www.lightwood.net/";
if (!headers_sent($filename, $line)) {
    header("Location: $location");
```

```
}
else {
    echo "Заголовки уже отправлены в строке $line файла
$filename <br>";
    echo "<A HREF=\"$destination\">Нажмите чтобы
продолжить</A>";
}
?>
```

Вывод HTTP-заголовков

Функция `headers_list` позволяет узнать список заголовков, которые уже отправлены или будут отправлены. Эта функция доступна в PHP 5 и выше. Она возвращает массив с заголовками.

Чтобы обработать этот массив, можно воспользоваться циклом. Но обычно нужно просто посмотреть список заголовков. Для этого идеально подойдет функция `print_r`:

```
print_r(headers_list());
```

Этот код нужно окружить дескрипторами <PRE>, чтобы получить результат в удобочитаемом виде:

```
Array
(
    [0] => X-Powered-By: PHP/5.0.2
    [1] => Set-Cookie: mycookie=somevalue
    [2] => Content-type: text/html
)
```

Изменение настроек кэширования

С помощью HTTP-заголовков можно изменить режим кэширования для конкретной Web-страницы. Этот механизм управляет местом, откуда будет загружаться страница при повторном доступе. Страница может полностью обновляться или загружаться с кэша пользовательского браузера или Internet-провайдера. Таким образом уменьшается нагрузка на Internet-канал.

Заголовок `Cache-Control` задает механизм кэширования для страницы. Основные управляющие значения этого заголовка приведены в табл. 16.1.

Зачастую кэширование убирается для того, чтобы страница обновлялась при повторном посещении.

Таблица 16.1. Основные установки для настройки кэширования

Значение	*Описание*
`public`	Можно сохранять в любом кэше.
`private`	Можно сохранять в кэше браузера, но не в общих системах кэширования.
`no-cache`	Нельзя кэшировать в любом виде между Web-сервером и браузером.

Обычно системы кэширования сами распознают, где происходит частое изменение содержимого. Например, для динамических страниц с использованием PHP. Но для полной уверенности в обновлении содержимого, можно использовать этот вспомогательный механизм.

Для того чтобы избежать кэширования в любом виде, используются следующие заголовки, которые запрещают все виды кэширования:

```
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: ". gmdate("D, d M Y H:i:s") . " GMT");
```

Здесь используется несколько различных типов заголовков. Информацию про редко используемые установки `Cache-Control` можно найти по адресу: `www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9`.

Заголовок `Expires` указывает браузеру на дату актуальности документа. В данном случае документ всегда будет устаревшим и нуждается в обновлении при очередном посещении.

Заголовок `Last-Modified` указывает на дату последнего изменения документа. Если использовать функцию `date`, в заголовке будет передаваться текущая дата. Поэтому браузер будет считать, что страница только что изменилась, и запросит новую копию документа.



Контроль кэширования в сеансах

При открытии сеанса PHP вместе с другими заголовками автоматически отправляется заголовок `no-cache`. Можно изменить механизм кэширования с помощью функции `session_cache_limiter`, используя для аргумента одно из значений, приведенных в табл. 16.1.

Переменные окружения сервера

Рассмотрим, какую информацию может получить PHP от Web-сервера.

Суперглобальный массив $_SERVER содержит элементы, в которых хранится информация об окружении Web-сервера во время текущего запроса. Чтобы увидеть полный список в рамках сценария, нужно выполнить следующую инструкцию:

```
print_r($_SERVER);
```

Примеры в этом разделе подходят для большинства Web-серверов. Но некоторые могут не поддерживать все приведенные значения или использовать другие имена. Чтобы узнать доступные значения, достаточно выполнить выражение выше.

Информация о сценарии

Имя сценария содержится в элементе $_SERVER["SCRIPT_NAME"]. Эта переменная позволяет создавать формы, которые отправляют содержимое самому себе. Тогда при смене имени файла не нужно менять параметр ACTION в сценарии. Ниже приводится пример:

```
<FORM ACTION="<?php print $_SERVER["SCRIPT_NAME"];?>"
      METHOD=POST>
```

Элемент REQUEST_URI похож на SCRIPT_NAME, но содержит полный URI запрашиваемой страницы. Он состоит из полного пути к сценарию включая знак вопроса и значения строки запроса, если они есть. Строка запроса не включается в элемент SCRIPT_NAME. В чистом виде она содержится в элементе $_SERVER["QUERY_STRING"].

Узнать имя домена, на котором выполняется сценарий, можно в переменной $_SERVER["HTTP_HOST"]. Web-сервер может работать с несколькими именами доменов; это позволяет определить, какому из них посылается запрос.

Информация о пользователе

В элементе HTTP_USER_AGENT содержится информация о версии браузера и операционной системе пользователя. Она может выглядеть примерно так:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.5)
            Gecko/20041107 Firefox/1.0
Lynx/2.8.5dev.7 libwww-FM/2.14 SSL-MM/1.4.1 OpenSSL/0.9.7a
```

Три примера выше соответствуют браузерам Internet Explorer, Mozilla Firefox и Lynx. Отметим, что Internet Explorer и Firefox идентифицируют себя как Mozilla-браузеры. Поэтому, чтобы точно определить, какой из них используется, нужно проанализировать информацию дальше.

Условие ниже позволяет генерировать различный вывод для Internet Explorer и Firefox:

```
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
    echo "Вы используете Internet Explorer";
}
elseif (strstr($_SERVER["HTTP_USER_AGENT"], "Firefox")) {
    echo "Вы используете Firefox";
}
else {
    echo "Вы пользуетесь другим браузером";
}
```

Такой механизм может понадобиться из-за различных реализаций DHTML и JavaScript в этих браузерах.

В элементе REMOTE_ADDR содержится IP-адрес, с которого происходит обращение к странице. Это может быть адрес пользователя или прокси-сервера Internet-провайдера. Иногда этот параметр используется для реализации безопасной системы аутентификации.

Если в REMOTE_ADDR находится значение прокси-сервера, то в элементе HTTP_X_FORWARDED_FOR находится IP-адрес пользовательского компьютера.

Имя пользователя, который авторизовался через базовую HTTP-аутентификацию, находится в элементе $_SESSION["REMOTE_USER"].

Информация о сервере

Другие элементы $_SERVER позволяют получить различную информацию о настройках Web-сервера.

Например, `$_SERVER["SERVER_NAME"]` соответствует директиве Apache `ServerName`. Это главное имя данного Web-сервера. Оно может не совпадать с именем домена, куда выполняется запрос, и даже со значением `$_SERVER["HTTP_HOST"]`. Аналогично, в `$_SESSION["SERVER_ADMIN"]` содержится адрес электронной почты Web-мастера, установленный с помощью инструкции `ServerAdmin`.

Элементы `SERVER_ADDR` и `SERVER_PORT` содержат IP-адрес и номер порта, на котором работает Web-сервер. Элемент `$_SERVER["REQUEST_METHOD"]` содержит информацию о методе (`GET` или `POST`) передачи информации в сценарий.

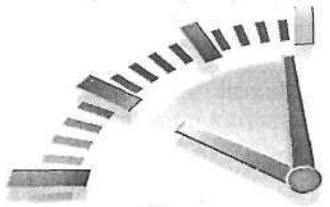
И напоследок: в переменной `$_SERVER["SERVER_SOFTWARE"]` хранится информация о программном обеспечении, которое используется на Web-сервере. Оно совпадает с тем, что отправляется в заголовке `Server`, который рассматривался в начале урока. Выглядит это примерно так:

```
Apache/1.3.29 (Unix) mod_gzip/1.3.26.1a PHP/4.3.9
mod_ssl/2.8.16 OpenSSL/0.9.7c
```

Резюме

В этом уроке вы узнали, как организовать взаимодействие с Web-сервером. В следующем уроке вы узнаете, как работать с файловой системой при помощи PHP.

Урок 17



Работа с файловой системой

У этом уроке вы узнаете, как работать с файловой системой Web-сервера при помощи PHP, а также научитесь читать и изменять содержимое файлов с помощью сценария.

Управление файлами

Рассмотрим, как PHP позволяет работать с файлами, которые находятся на жестком диске Web-сервера.

Права доступа к файлам

Перед началом работы с файловой системой из PHP рассмотрим установку прав доступа для файловой системы. В этой части рассматривается система прав для операционных систем Unix/Linux. Но те же положения справедливы и для других платформ.

Обычно Web-сервер запускается с правами пользователя `apache` или `nobody`. Владельцем Web-документов и сценариев PHP будет пользователь, который их создает. Поэтому если Web-сервер не имеет прав доступа на файл, он не сможет с ним работать.

С помощью команды `chmod` можно установить доступ на чтение для определенного файла всем пользователям. Следующая команда устанавливает флаг чтения (`r`) для всех пользователей, кроме текущего (`o`):

```
$ chmod o+r filename
```

Можно установить глобальные права на чтение/изменение файла с помощью chmod, как показывается ниже:

```
$ chmod o+rw filename
```

С помощью команды man chmod можно получить подробную справку и несколько примеров использования.

Если на файл установлены глобальные права, Web-сервер сможет получить к нему доступ. Но и другие пользователи на этой системе также получат доступ к нему. Иногда лучше установить владельцем файла пользователя apache, не предоставляя глобальный доступ к нему. Суперпользователь может сделать это с помощью команды chown:

```
# chown apache filename
```

Функция PHP chmod также позволяет менять права файла. На вход нужно подать два аргумента: имя файла и режим. Аргумент "режим" может быть строкой в виде o+rw или числом в виде 0644. Для числового формата нужно ставить ноль спереди.



Использование chmod

Те, кто уже пользовались командой chmod, знают, что можно задать режим доступа в трехзначном формате. При работе в системной оболочке предполагается, что на вход chmod подается восьмеричное число, даже если спереди нет нуля. Например, chmod 644 — то же, что и chmod 0644. Но в PHP всегда нужно ставить нуль впереди аргумента прав.

Получение информации о файле

PHP предоставляет широкий набор функций, которые позволяют получить различную информацию о файле в системе. Самая простая и часто используемая — file_exists. Она позволяет узнать, существует ли файл с заданным в аргументе именем. На выходе функция возвращает true, если существует объект в файловой системе с заданным именем. Это может быть каталог или специальный тип файла. Кроме того, в аргументе можно указать путь, как в примере ниже:


```
if (file_exists("/home/chris/myfile")) {
    echo "Файл существует";
}
else {
    echo "Файл не существует";
}
```

Есть целый набор функций, которые позволяют узнать атрибуты файла. Они приводятся в табл. 17.1.

Таблица 17.1. Функции для проверки атрибутов файла

Функция	*Описание*
is_executable	Проверяет, установлен ли у файла флаг выполнения
is_readable	Проверяет, установлен ли у файла флаг чтения
is_writeable	Проверяет, установлен ли у файла флаг записи
is_link	Проверяет, является ли данный файл символической ссылкой
is_file	Проверяет, является ли объект настоящим файлом, а не ссылкой

Другие функции позволяют узнать информацию непосредственно о самом файле. Они приводятся в табл. 17.2.

Таблица 17.2. Функции для поиска информации о файле

Функция	*Описание*
fileatime	Возвращает значение последнего доступа к файлу в формате временной метки Unix
filectime	Возвращает время создания файла или его последнего изменения
filemtime	Возвращает время последнего изменения файла
fileowner	Возвращает идентификатор владельца файла
filegroup	Возвращает идентификатор группы файла
fileinode	Возвращает номер дескриптора файла
fileperms	Возвращает права доступа на файл в восьмеричном виде (например, 0644)
filesize	Возвращает размер файла в байтах
filetype	Возвращает тип файла (возможны варианты: fifo, char, dir, block, link или file)

Перемещение и копирование файлов

Предположим, что есть права на копирование, перенос или удаление файла. Эти операции выполняются, соответственно, с помощью функций: `copy`, `rename` и `unlink`.

Функции `copy` и `rename` требуют два аргумента: источник и место копирования, а функция `unlink` только имя файла.



Путь файла

Нужно осторожно выполнять операции с файлами в PHP, особенно удаление. Всегда нужно знать, в каком каталоге выполняется операция, или задавать полный путь к нужному файлу.

Работа с именами файлов

Функции `basename` и `dirname` позволяют легко узнать имя файла и полный путь. Эти функции позволяют быстро определить имя файла, если оно задается с путем, или путь к файлу, если нужно создать другие файлы в том же месте, где и исходный.

Функция `basename` возвращает все символы от последней обратной черты до конца строки. А функция `dirname` возвращает часть строки до последней обратной черты.

Функция `realpath` требует аргумент "путь" и возвращает абсолютный путь. Таким образом, все символические ссылки изменяются на реальное место вфайловой системе. А все ссылки на текущий или родительский каталог в виде `.` или `..` удаляются.

Функция `tempnam` позволяет легко создать временной файл. Она автоматически генерирует уникальное имя для временного файла. На вход нужно подать два аргумента: имя каталога и префикс для названия файла. Префикс является обязательным аргументом, но может быть пустой строкой. Следующее выражение создает временный файл в `/tmp` без префикса:

```
$filename = tempnam("/tmp", "");
```


Чтение и запись файлов

Теперь рассмотрим, как в PHP читать и изменять файлы.

Простые методы для чтения и записи файлов

В PHP есть специальные высокоуровневые функции. Они позволяют одной командой прочитать содержимое файла или записать туда данные.

Функция `file_get_contents` позволяет записать содержимое файла в переменную. Ей нужно передать имя файла с абсолютным или относительным путем. В примере ниже содержимое файла `file.txt` читается и записывается в переменную `$data`:

```
$data = file_get_contents("file.txt");
```

Другой необязательный булев аргумент можно установить в `true`, чтобы искать файл в дополнительных путях подключения. Как настроить пути для подключения, рассказывается в уроке 23, "Настройка PHP".

Функция `file_put_contents` позволяет записать содержимое переменной в локальный файл. На вход подается два аргумента: имя файла и данные для записи. В следующем примере содержимое переменной `$data` записывается в `file.txt`:

```
file_put_contents("file.txt", $data);
```

Низкоуровневый доступ к файлу

Функции `file_get_contents` и `file_put_contents` являются высокоуровневыми. Они выполняют последовательность операций, которые можно реализовать с помощью низкоуровневых функций PHP. В большинстве случаев нужно просто извлечь данные из файла или записать в него информацию. Но PHP также предоставляет другой гибкий способ для взаимодействия с файловой системой.

Доступ к файлу осуществляется с помощью обработчика файла, который создается функцией `fopen`. Функция `fopen` требует два аргумента: имя файла и режим доступа. Все режимы приводятся в табл. 17.3.

Таблица 17.3. Режимы доступа функции fopen

Режим	*Описание*
r	Открыть только для чтения с начала файла
r+	Открыть для чтения/записи с начала файла
w	Открыть только для чтения. Перезаписать старый файл, если он существует
w+	Открыть для записи и чтения Перезаписать старый файл, если он существует
a	Открыть только для чтения и добавить новые данные в конец файла
a+	Открыть для чтения и записи в конце файла

Обработчик файла указывает на позицию в файле. Из табл. 17.3 можно заметить, что при открытии с помощью fopen обработчик всегда указывает на начало или конец файла. При записи или чтении с помощью обработчика его позиция постоянно изменяется и следующее действие начинается уже с новой точки.

Рассмотрим пример, где содержимое файла читается по нескольку байт за один раз. Функция fopen с аргументом режима r создает обработчик файла только для чтения, который указывает на начало файла.

Функция fread считывает фиксированное количество байт от текущей позиция обработчика файла. На вход нужно подать обработчик файла и количество байт для чтения. Комбинируя цикл и эту функцию, можно прочитать все содержимое файла:

```
$fp = fopen("file.txt", "r");
while ($chunk = fread($fp, 100)) {
    echo $chunk;
}
```

Это компактное условие позволяет проверить успешное выполнение fread при каждой итерации цикла. Если данных для чтения больше нет, fread возвращает FALSE. Этот механизм не позволяет установить, было ли прочитано за один раз меньшее количество байт.

Функция fgets является альтернативой к fread. Она отличается тем, что за один проход считывается одна строка. Аргумент размера для fgets является необязательным, начиная с PHP 4.3. Таким образом, данные читаются до первого знака перевода строки или до того, как будет превышен заданный максимальный размер.

В примере ниже используется fgets в цикле и предполагается, что в файле нет строк длиннее 100 символов:

```
$fp = fopen("file.txt", "r");
while ($line = fgets($fp, 100)) {
    echo $line;
}
```



Обрезка строк

Каждая строка, прочитанная с помощью fgets заканчивается символом перевода строки. Если нужно исключить его из конечного результата, следует использовать функцию rtrim. Она удаляет разделительные символы в конце строки, включая перевод строки и табуляцию.

Завершив работу с обработчиком файла, нужно освободить соответствующие ресурсы с помощью функции fclose:

```
fclose($fp);
```

Произвольный доступ к файлу

Указатель файла может двигаться не только последовательно, но и перемещаться на любую позицию, пока открыт обработчик файла.

Чтобы узнать текущую позицию указателя файла, можно воспользоваться функцией ftell. На выходе получим количество байт от начала файла:

```
$filepos = ftell($fp);
```

Чтобы переместить указатель файла в определенное место, используется функция fseek. В выражении ниже указатель файла смещается на 100 байт от начала файла с помощью обработчика файла $fp:

```
fseek($fp, 100);
```

Обычно нужно перевести указатель файла в его начало. Это можно сделать с помощью fseek, но лучше использовать специальную функцию rewind:

```
rewind($fp);
```

Запись с помощью указателя файла

Функции fgets и fread дополняются операциями записи fputs и fwrite. Эти функции идентичны, а символ перевода строки не отличается от других символов при записи в файл.

В примере ниже открывается файл и в него записывается значение текущего времени:

```
$fp = fopen("time.txt", "w");
fwrite($fp, "Дата изменения ".date("H:i:s"));
fclose($fp);
```

Нужно помнить, что пользователь apache должен иметь право на запись в каталог, чтобы создать новый файл.

Открыв новый файл time.txt, можно увидеть, что там записано текущее время.

Работа с файлами данных

Часто операции с файловой системой используются для того, чтобы извлечь данные из файла с определенной структурой и обработать их в сценарии. В самом простом варианте значения разделяются запятой (CSV-формат).

Чтобы прочитать отдельную строчку, можно использовать функцию explode для разделения по запятым. Но такой подход не сработает, если элементы данных в файле CSV содержат запятые. Обычно при экспорте данных из электронной таблицы столбцы могут содержать запятые, которые находятся в кавычках. Поэтому нужны более сложные правила для корректной обработки данных.

К счастью, в PHP есть функция fgetcsv. Ее работа напоминает fgets, но на выходе получим массив разделенных запятой значений. Аргумент размера fgetcsv является необязательным, начиная с PHP 5.

Часто первая строка CSV-файла содержит названия столбцов. Если такое случается, нужно удалить эту строку перед началом обработки данных. В следующем примере читается содержимое разделенного запятой файла и выводятся все записи на экран с помощью print_r:

```
$fp = fopen("data.csv", "r");
while ($record = fgetcsv($fp, 1000)) {
    echo $chunk;
}
```


Чтение CSV-файлов



Функция fgetcsv требует аргумент длины, как и fgets. В предыдущем примере, выбрано произвольное значение 1000. Но нужно точно знать, что самое большое значение меньше аргумента длины.

Есть специальный механизм записи в CSV-файл без кодирования в этот формат. На вход функции fputcsv передается обработчик файла с массивом аргументов, которые она записывает в отформатированном виде в файл.

Третий и четвертый аргументы fputcsv позволяют задать альтернативный разделитель и ограничивающий символы, соответственно. По умолчанию это запятая и символ двойной кавычки.

Работа с URL

В PHP есть мощный механизм, который позволяет работать с удаленными файлами так же, как с локальными. Таким образом, можно открыть обработчик файла или использовать высокоуровневые функции работы с файлами. При этом путь задается в виде URL. Это позволяет прочитать содержимое Web-страницы из сценария PHP.

Оба выражения ниже правильные:

```
$page =
file_get_contents("http://www.samspublishing.com/");
$fp = fopen("http://www.samspublishing.com/", "r");
```

Несмотря на это, нельзя записывать информацию по HTTP-адресу с помощью file_put_contents или fputs.

Работа с каталогами

Так же, как fopen создает обработчик для файла, с помощью функции opendir можно создать обработчик для каталога, чтобы прочитать его содержимое.

Три функции работают с обработчиком каталога: readdir, rewinddir и closedir. На вход им нужно передать только обработчик каталога.

При очередном вызове readdir на выходе получим следующий файл в заданном каталоге. Порядок, в котором они извлекаются, зависит от порядка расположения в файловой

системе и не может быть изменен. Специальные обозначения . и .. (текущий и родительский каталоги) также возвращаются этой функцией.

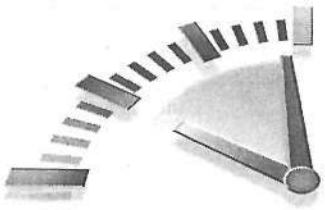
Функция `rewinddir` сбрасывает позицию обработчика каталога в начало списка файлов, а `closedir` закрывает обработчик каталога.

С помощью функции `getcwd` можно получить название текущего рабочего каталога. Она не требует аргументов и возвращает полный путь к текущему каталогу. Рабочий каталог можно изменить с помощью функции `chdir`, а аргумент пути передать в полном или относительном виде.

Резюме

В этом уроке вы научились читать и записывать файлы на жесткий диск Web-сервера. В следующем уроке мы рассмотрим, как выполнять команды сервера из PHP.

Урок 18



Выполнение программ на Web-сервере

В этом уроке вы узнаете, как с помощью PHP выполнить программы на Web-сервере и обработать полученный вывод.

Выполнение локальных программ

PHP позволяет вызывать внешние программы, которые находятся на Web-сервере, несколькими способами.

Функция passthru

Вызвать локальную команду и получить результат на экране можно с помощью функции `passthru`. Инструкция, которая передается в аргументе, выполняется на Web-сервере, и весь полученный вывод отправляется браузеру.

Ниже приводится простой пример, который работает как на Unix/Linux, так и на Windows-системах:

```
passthru("hostname");
```

С помощью команды `hostname` можно узнать название сервера. Команда `hostname` выполняется на локальной системе, и результат выводится на экран.

Второй необязательный аргумент `passthru` позволяет определить код выхода из команды. Так можно узнать результат выполнения операции. Обычно все команды при удачном завершении возвращают ноль. Код возврата может понадобиться, если команда может возвращать разные значения.



Вывод команды

Только стандартный поток вывода отображается в окне браузера. Поэтому для того, чтобы увидеть сообщения об ошибках, нужно перенаправить поток ошибок `stderr` в стандартный поток вывода.
На серверах под управлением Unix это выполняется с помощью вызова функции `passthru("cmd 2>&1")` в оболочке Bourne.

Наиболее часто при ненулевом результате получаем значение 1 для неспецифических ошибок и 127, если такой команды не обнаружено. Другие коды ошибок для каждой отдельной программы нужно искать в документации. В следующем примере вызывается команда hostname, и в зависимости от результата выполняется действие:

```
passthru("hostname", $return);
switch ($return) {
  case 0:    echo "Команда выполнена успешно";
             break;
  case 127:  echo "Команда не обнаружена";
             break;
  default:   echo "Команда не выполнена. Код возврата
$return";
}
```

Применение апострофа

Символ апострофа (`) позволяет быстро выполнить нужную команду на Web-сервере. Строка между апострофами выполняется, а на выходе получаем результат, генерируемый системой.

Ниже приводится аналог примера с passthru для апострофов:

```
echo `hostname`;
```

Апострофы позволяют присвоить результат выполнения команды переменной, как показывается в примере ниже:

```
$hostname = `hostname`;
```

Символ апострофа можно использовать в любом месте сценария PHP. Он приостанавливает выполнение сценария и подставляет полученное значение в сценарий. В примере

ниже показано, как результат выполнения команды на сервере подставляется в условие:

```
if (chop(`hostname`) == "hal9000") {
    echo "Добрый вечер, Дмитрий";
}
```

Результат выполнения hostname заканчивается переводом строки. Это нужно для нормального вывода в командной строке. Функция chop в примере выше удаляет все разделительные знаки для того, чтобы правильно сравнить значения.



Коды возврата

Апострофы не позволяют узнать код возврата. Поэтому вместо них нужно использовать функцию exec. Она работает так же, как и passthru, но результат возвращает в виде строки. Второй необязательный параметр содержит код возврата.

Создание командной строки

Апострофам, функциям passthru и exec команды передаются в виде строки. Кроме того, строку с командами можно построить при помощи переменных и выражений.

В двойных кавычках символ доллара заменяется на соответствующую переменную, но в одинарных кавычках он трактуется как переменная оболочки.

Довольно непривычно выглядит выражение в апострофах, которое исполняет команду, содержащуюся в строке:

```
`$cmd`;
```

В переменной $cmd может быть несколько системных команд. Если не нужно знать результат этих операций, такой вариант корректен.

Нужно помнить про закрывающую точку с запятой. Закрывающий апостроф завершает команду сервера, но не выражение PHP.

Окружение сервера

Теперь рассмотрим, каким образом PHP взаимодействует с окружением локального Web-сервера.

Определение платформы сервера

На каждой системе есть свой набор команд. Поэтому при написании сценария под различные платформы нужно знать, на какой из них происходит работа.

Константа PHP_OS содержит название операционной системы. Обычно нужно определить — Windows- или Unix-подобная система используется на сервере. Дело в том, что все Unix-подобные системы (включая Mac OS) очень похожи друг на друга.

Значение PHP_OS на Web-сервере под управлением Windows может быть Windows, WINNT или WIN32. В будущем к этому списку могут прибавиться новые значения. Поэтому для проверки на Windows достаточно выполнить независимую от регистра проверку первых трех символов строки. В примере ниже показано, как это сделать:

```
if (strtoupper(substr(PHP_OS, 0, 3)) == "WIN") { ... }
```



Darwin

Отметим, что новая версия Mac OS идентифицирует себя как Darwin. Поэтому нужно проверить *начало* строки PHP_OS на совпадение с WIN.

Переменные окружения

Суперглобальный массив $_ENV содержит все значения переменных окружения. *Переменные окружения* — это значения операционной системы. Они доступны для PHP из окружения, в котором работает Web-сервер.

Переменная окружения PATH содержит список путей, в которых нужно искать исполняемые программы. Все места проверяются по очереди, пока программа не будет найдена. В противном случае выводится сообщение об ошибке.

Определить текущее значение пути можно с помощью следующего выражения:


```
echo $_ENV["PATH"];
```

В Unix/Linux-системе он может быть следующим:

```
/bin:/usr/bin:/usr/X11R6/bin:/home/chris/bin
```

Но на платформе Windows он выглядит примерно так:

```
C:\WINDOWS\system32;C:\WINDOWS
```

Нужно помнить, что формат исполняемых путей отличается для различных операционных систем. Unix/Linux-версия использует точку с запятой для разделения позиций и косую черту для путей. В платформе Windows используется точка с запятой и обратная косая черта. Поэтому в PHP есть специальные константы — DIRECTORY_SEPARATOR и PATH_SEPARATOR, которые позволяют определить соответствующие значения.

Во многих случаях изменение значения PATH отличается для разных платформ. Например, если использовать правильное значение константы PATH_SEPARATOR, C:/WINDOWS не существует на платформе Linux. Но к нему можно добавить текущий или относительный каталог.

В следующем примере добавляется каталог bin относительно текущей позиции, к началу системного пути:

```
$newpath = getcwd() . DIRECTORY_SEPARATOR . "bin" .
           PATH_SEPARATOR . $_ENV["PATH"];
putenv("PATH=$newpath");
```

Функции putenv нужно передать аргумент, в котором переменной окружения присваивается новое значение. Это временное изменение. Новое значение хранится только до окончания работы текущего сценария.

Часовые пояса

Переменная окружения TZ содержит настройки часового пояса. Изменяя это значение, можно выводить время в другой точке мира. При этом не нужно знать смещение и выполнять арифметические операции.

Большинство главных городов различных регионов мира имеют легко запоминающиеся значения для TZ (например: Europe/London, US/Pacific). Можно задать значение относительно GMT или другое общее название часового пояса, например GMT-8 или EST.

На большинстве систем названия всех часовых поясов можно найти в /usr/share/zoneinfo.

Сценарий в листинге 18.1 показывает текущее время в различных точках мира.

Листинг 18.1. Использование переменной окружения TZ для изменения часового пояса

```
<?php
$now = time();
$original_tz = $_ENV["TZ"];

echo "Текущее время " . date("H:i:s", $now) . "<br>";

putenv("TZ=US/Pacific");
echo "Сейчас на американском западном побережье " .
          date("H:i:s", $now) . "<br>";
putenv("TZ=Europe/Paris");
echo "Сейчас во Франции " . date("H:i:s", $now) . "<br>";

putenv("TZ=Australia/Sydney");
echo "Сейчас в Сиднее " . date("H:i:s", $now) . "<br>";

putenv("TZ=Asia/Tokyo");
echo "Сейчас в Токио " . date("H:i:s", $now) . "<br>";

putenv("TZ=$original_tz");
?>
```

Отметим, что в начале листинга 18.1 сохраняется значение текущего часового пояса. Это позволяет восстановить его после всех изменений.



Сохранение времени

В начале листинга 18.1 значение временной метки записывается в `$now`. Поэтому все функции `date` получают одинаковое значение. Второй аргумент `date` можно опустить, но во время работы сценария может измениться системное время. Тогда на выходе будет странный результат.

Анализ безопасности

Размещать на сервере сценарии, которые могут выполнять серверные программы, плохое решение с точки зрения безопасности. В уроке 24, "Безопасность PHP", рассказывается, как с помощью безопасного режима наложить ограничение на выполнение локальных программ.

До окончания этого урока будут рассмотрены способы безопасного выполнения программ.

Разделение обратной косой чертой команд оболочки

Рассмотрим сценарий в листинге 18.2. Он создает Web-интерфейс для команды `finger`.

Листинг 18.2 Вызов команды `finger` из Web-формы

```
<FORM ACTION="finger.php" METHOD="POST">
<INPUT NAME="username" SIZE=10>
<INPUT TYPE="SUBMIT" VALUE="Finger username">
</FORM>
<?php
if ($_POST["username"]) {
    $cmd = "finger {$_POST['username']}";
    echo "<PRE>" . `$cmd` . "</PRE>";
}
?>
```

Если запустить этот сценарий в браузере и ввести имя пользователя, результат выполнения `finger` будет выведен на экран.

Но если вместо имени ввести точку с запятой, а за ней — любую команду, например `;ls`, команда `finger` выполнится без аргумента. А вслед за ней выполнится следующая команда. Аналогичный механизм реализуется с помощью других символов, в зависимости от платформы сервера.

Понятно, что это не очень хорошо. Некоторые могут подумать, что нельзя нанести существенный ущерб системе с помощью такого недочета. Но есть программы для взлома, которые могут сделать довольно много, используя этот механизм. Например, с помощью команды `wget` или `lynx` на Web-сервере устанавливается и запускается на выполнение нужная программа. Это может быть `rootkit`, который позволяет находить различные уязвимости на сервере. Или сценарий, который запустит DOS-атаку на сервере и заберет

на себя все ресурсы. Как бы то ни было, но лучше не давать анонимным пользователям такой доступ к серверу.

Функция `escapeshellcmd` позволяет предотвратить появление такой уязвимости. Все символы, кроме одного предназначенного, которые могут обмануть сценарий при выполнении программы, предваряются обратной косой чертой. Таким образом, нежелательные символы становятся аргументом команды.

Чтобы сделать безопасным листинг 18.2, нужно заменить выражение, которое создает `$cmd`, на следующий код:

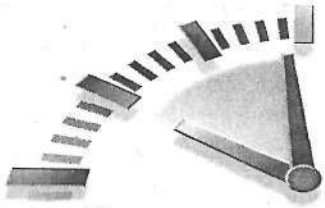
```
$cmd = escapeshellcmd("finger {$_POST['username']}");
```

Теперь, если ввести `;ls` в поле формы, выполнится команда `finger \; ls`. Команда попытается найти в системе пользователей с именем `;` или `ls`.

Резюме

В этом уроке вы узнали, как обезопасить выполнение команд на Web-сервере из PHP и как работать с выводом, который они генерируют. В следующем уроке вы узнаете, как получить доступ к базе данных MySQL из PHP.

Урок 19



Использование базы данных MySQL

В этом уроке вы узнаете, как получить доступ к базе данных MySQL из PHP. PHP так часто используется в связке с MySQL, что довольно сложно встретить его комбинацию с другой СУБД.

Использование MySQL

Для этого урока нужно иметь модуль поддержки базы данных MySQL на Web-сервере, а также базу данных для непосредственной работы с ней. Информацию о том, как установить MySQL, можно найти по адресу: `http://dev.mysql.com/doc/mysql/en/Installing.html`. О том, как включить поддержку MySQL в PHP, рассказывается в уроке 23, "Настройка PHP".



Информация для дальнейшего изучения

Подробнее о MySQL можно прочитать в книге Поля Дюбуа, которая так и называется —*MySQL*. Чтобы получить общее представление о языке SQL, можно обратиться к книге Бена Форты *Освой самостоятельно SQL, 10 минут на урок*.

В PHP 5 добавили новое расширение — `mysqli`, в котором поддерживаются новые возможности, начиная с версии MySQL 4.1 и выше. Кроме того, его можно использовать в объектно-ориентированном стиле. В этой книге рассматривается классическое расширение `mysql`. Оно все еще попу-

лярно на большинстве Web-серверов и поддерживается в PHP 5.

Иногда нужно использовать mysqli вместо mysql. Для этого нужно заменить префикс mysql на mysqli. Но в большинстве случаев они ведут себя одинаково. Подробнее о mysqli можно узнать на странице www.php.net/mysqli.

Соединение с базой данных MySQL

Чтобы соединиться с базой данных MySQL, используется функция mysql_connect. На вход нужно подать три параметра: адрес сервера, имя пользователя и пароль. В большинстве случаев сервер MySQL работает на том же сервере, где и PHP. Поэтому для адреса сервера подходит параметр localhost. Обычно выражение mysql_connect выглядит так:

```
$db = mysql_connect("localhost", "chris", "mypassword");
```



Адрес сервера базы данных

MySQL использует аутентификацию на основе адреса сервера. Поэтому нужно задать адрес, с которым разрешено соединение. Например, MySQL-сервер запущен на *www.домен.com*, а соединение разрешено только для адреса localhost.

Если MySQL-сервер находится на том же сервере, что и PHP, в большинстве случаев подойдет значение localhost.

Функция mysql_connect возвращает идентификатор соединения с базой данных, который присваивается переменной $db в примере выше. Идентификатор используется как аргумент при вызове других функций MySQL.

Отметим, что в параметрах соединения для mysql_connect нет имени базы данных. Просто выбор базы данных является вторым шагом после соединения с MySQL-сервером и выполняется с помощью функции mysql_select_db. Например, в выражении ниже выбирается база данных mydb:

```
mysql_select_db("mydb", $db);
```




Идентификатор соединения

Аргумент $db является необязательным для mysql_select_db и других функций MySQL. Если его опустить, PHP использует последнее соединение с MySQL. Но для понятности кода лучше добавлять идентификатор соединения в функции MySQL.

После вызова mysql_select_db все последующие операции SQL выполняются с выбранной базой данных.

После окончания работы с MySQL в сценарии нужно закрыть соединение и освободить занятые ресурсы с помощью функции mysql_close:

```
mysql_close($db);
```

Выполнение SQL-запросов

Чтобы выполнить SQL-выражение в MySQL, нужно передать на вход функции mysql_query два аргумента: строку запроса и необязательный идентификатор соединения.

В примере ниже выполняется SQL-запрос CREATE TABLE в базе данных MySQL для идентификатора $db:

```
$sql = "CREATE TABLE mytable (col1 INT, col2 VARCHAR(10))";
mysql_query($sql, $conn);
```

Если запустить этот сценарий в браузере и проверить содержимое базы данных MySQL, можно увидеть, что появилась новая таблица mytable.

Все запросы к базе данных выполняются с помощью mysql_query. Не важно, нужно ли внести изменение в данные или получить несколько строк.

Команды, которые изменяют базу данных

Выше в уроке приводился пример с выражением CREATE TABLE. Другие выражения, определяющие структуру базы (DDL-выражения в SQL), выполняются так же: если нет ошибок, сообщение не выводится. Про обработку ошибок рассказывается ниже в этом уроке.

Запросы с DELETE, INSERT или UPDATE (DML-выражения в SQL) позволяют манипулировать данными в строках таблицы. Для того чтобы узнать, сколько строк изменилось, используется функция mysql_affected_rows. Ниже показывается, как это работает для запроса с оператором UPDATE:

```
$sql = "UPDATE mytable SET col2 = 'newvalue' WHERE col1 >
5";
mysql_query($sql, $conn);
echo "Было обновлено " . mysql_affected_rows($conn) . "
строк(а)";
```

Функция mysql_affected_rows требует идентификатор соединения с базой данных и возвращает количество измененных последним запросом строк. Количество затронутых строк в UPDATE может не совпадать с количеством строк, выделенных выражением WHERE. MySQL не обновляет строку, если новое значение совпадает со старым.



Удаление всех строк

Если выполнить запрос DELETE без условия WHERE, функция mysql_affected_rows возвращает нулевое значение, независимо от количества удаленных строк. MySQL просто очищает всю таблицу вместо поочередного удаления всех строк. Поэтому никакого подсчета не происходит.

Извлечение полученных данных

Выражение SELECT возвращает одну или больше строк из базы данных, а PHP предоставляет набор функций для того, чтобы получить доступ к этим данным. Для того чтобы работать с выбранными данными, нужно получить идентификатор ресурса. Его возвращает функция mysql_query:

```
$res = mysql_query($sql, $db);
```

Нельзя непосредственно работать с данными в $res. Для этого нужно передать это значение функциям для извлечения данных.

Можно использовать функцию mysql_result, чтобы обратиться к элементу данных из определенной стро-

столбца в полученном результате. Это очень удобно, если запрос возвращает только одно значение, например результат объединяющих функций.

В приведенном ниже примере выполняется операция SUM над элементами в столбце таблицы, и полученный результат выводится на экран:

```
$sql = "SELECT SUM(col1) FROM mytable";
$res = mysql_query($sql, $conn);
echo mysql_result($res, 0, 0);
```

Функции mysql_result нужно передать три аргумента: идентификатор результата, номер строки и номер столбца. Нумерация столбцов и строк начинается с нуля. Поэтому в примере выше ищется первая строчка в первом столбце по всему множеству значений. Особенность объединяющих функций в том, что они обязательно возвращают одно значение, даже если таблица пуста. Поэтому на выходе всегда получим строку и столбец с результатом. Попытка получить доступ к несуществующей строке или столбцу вызовет сообщение об ошибке.

Функция mysql_num_rows возвращает количество строк, полученных в запросе. Это значение можно использовать в цикле с функцией mysql_result, которая позволяет получить каждую строку в результате. Ниже приводится реализация этого механизма:

```
$sql = "SELECT col1, col2 FROM mytable";
$res = mysql_query($sql, $db);
for ($i=0; $i < mysql_num_rows($res); $i++) {
  echo "col1 = " . mysql_result($res, $i, 0);
  echo ", col2 = " . mysql_result($res, $i, 1) . "<br>";
}
```

В этом примере известно положение столбцов col1 и col2, поэтому можно воспользоваться mysql_result с числовыми значениями, чтобы установить их по очереди.



Название полей

В функции mysql_result можно использовать строку как аргумент номера столбца. В этом случае нужно указать имя столбца. Это очень удобно для запросов SELECT *, где не всегда известен порядок следования столбцов, а также для запросов, где неудобно работать с числовыми значениями столбцов.

Получение всей строки из данных

В PHP есть более удобный механизм, который позволяет сразу извлечь одну строку из полученного результата. С помощью функции mysql_fetch_array создается массив из результата запроса. В этом массиве каждый элемент отвечает столбцу в запросе.

Последовательно вызывая mysql_fetch_array, можно получить все строки запроса. После извлечения всех строк функция возвращает FALSE.

С помощью функции mysql_fetch_array можно создать удобную циклическую структуру, как показывается ниже:

```
$sql = "SELECT col1, col2 FROM mytable";
$res = mysql_query($sql, $conn);
while ($row = mysql_fetch_array($res)) {
    echo "col1 = " . $row["col1"];
    echo ", col2 = " . $row["col2"] . "<br>";
}
```

Все строки данных извлекаются по очереди при очередной итерации цикла. Результирующая строка данных записится в массив и не требует вызова других функций.

Полученный массив, содержит данные с цифровыми и ассоциированными индексами. В этом примере известно, что col1 — первый столбец, поэтому $row["col1"] и $row[0] содержат одинаковое значение.

Такой механизм позволяет получить последовательный доступ ко всем строкам в полученном результате. Кроме того, произвольный доступ реализуется с помощью функции mysql_data_seek. Для этого нужно указать номер строки и перейти к ней, перед вызовом mysql_fetch_array.

Чтобы перейти к десятой строке, используется следующий вариант (нужно помнить, что отсчет начинается с нуля, а не с единицы):

```
mysql_data_seek($res, 9);
```

Если после этого нужно перейти в начало набора данных, нужно найти нулевую строку:

```
mysql_data_seek($res, 0);
```

Если вызвать mysql_data_seek с номером строки, который превышает самый большой из полученных данных, появится сообщение об ошибке. Нужно сначала узнать количество полученных строк с помощью mysql_num_rows, чтобы найти точное значение.



Поиск

Чтобы перейти к последней строке результата, нужно вызвать mysql_data_seek($res, mysql_num_rows($res)-1), потому что номер последней строки на единицу меньше общего количества строк в результате. Это намного проще реализовать с помощью оператора изменения порядка сортировки ORDER BY в SQL-запросе. Теперь вместо последнего нужно выбрать первый элемент.

Отладка SQL

Иногда при работе с MySQL случаются ошибки. Но сообщения об ошибках могут не содержать полную информацию о причине. В следующем разделе рассматривается способ реализации улучшенного механизма уведомления об ошибках. Это позволяет быстро отлаживать слой логики базы данных.

Ошибки SQL

Если происходит ошибка в SQL-выражении, она не выводится сразу. Чтобы определить ошибку, нужно проверить возвращаемое значение mysql_query. Значение NULL возвращается, если происходит ошибка. Это относится как к DDL- так и DML-выражениям, включая SELECT.

Ниже делается попытка выполнить неправильное SQL-выражение (пропущено название таблицы в команде DELETE):

```
$sql = "DELETE FROM";
$res = mysql_query($sql, $db);
if (!$res) {
    echo "Обнаружена ошибка в SQL";
    exit;
}
```

Если нужно найти, почему не удался вызов mysql_query, необходимо использовать функции mysql_error и mysql_errno. Они возвращают, соответственно, сообщение об ошибке MySQL и номер кода ошибки. Необязательный

аргумент идентификатора соединения нужно указывать, если в сценарии открыто несколько MySQL-соединений:

```
if (!$res) {
    echo "Ошибка " . mysql_errno() . " в SQL ";
    echo "<PRE>$sql</PRE>";
    echo mysql_error();
    exit;
}
```



Отладка SQL

При отладке SQL, очень удобно видеть приведший к ошибке запрос. Особенно если в нем используется подстановка переменных. Это легко сделать, если запрос содержится в переменной, аналогично `$sql` в примерах выше. Поэтому лучше не задавать его напрямую в `mysql_query`.

Если в сценарии не обнаружено ошибок SQL, PHP продолжает выполнение. Оно может прерваться, если подставить ложный идентификатор ресурса. При вызове `mysql_result` с неправильным `$res` появится следующее сообщение об ошибке:

```
Warning: mysql_result(): supplied argument is not a valid
MySQL result resource in /home/chris/mysql.php on line 8
```

В этом сообщении нет информации о причине ошибки. Номер строки относится к `mysql_result`, а не `mysql_query`. Чтобы найти причину проблемы, нужно просмотреть сценарий выше.

Ошибки соединения

Если при соединении с базой данных MySQL происходит ошибка, PHP выводит сообщение об этом. Ниже показан пример, где неправильно задан пароль и имя домена.

```
Warning: mysql_connect(): Access denied for user
'root'@'localhost'
(using password: YES) in /home/chris/connect.php on line 3

Warning: mysql_connect(): Unknown MySQL server host
'local-host'
(1) in /home/chris/connect.php on line 3
```


Эти предупреждения генерируется PHP и позволяют выявить причину. Если нужно, можно увидеть реальное сообщение об ошибке и ее код с помощью `mysql_error` и `mysql_errno`.

Например, если запрещен вывод ошибок на экран (подробнее об этом рассказывается в уроке 23). Это удобно при записи информации об ошибке в файл журнала. Можно определить, что соединение не удалось, если идентификатор соединения равен `NULL`.

В следующем коде перед тем, как продолжить работу, выполняется проверка соединения. При необходимости выводится сообщение об ошибке:

```
$db = mysql_connect("localhost", "chris", "mypassword");
if (!$db) {
    echo "При соединении произошла следующая ошибка " .
                mysql_errno() . "<br>";
    echo "Сообщение об ошибке: " . mysql_error();
    exit;
}
```



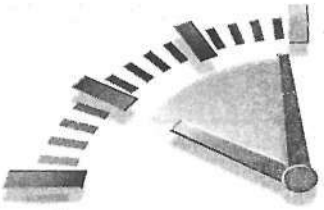
Пароли

Если указать неправильный пароль для соединения, ни PHP, ни `mysql_error` не выводит его на экран.

Резюме

В этом уроке вы узнали, как использовать интерфейс PHP для работы с базой данных MySQL. В следующем уроке вы узнаете, как в PHP реализовать соединение с различными базами данных при помощи уровня абстракции.

Урок 20



Абстрагирование от базы данных

В этом уроке вы узнаете, как обращаться к различным базам данных с помощью одинакового интерфейса. Абстрагирование от базы данных — очень мощная методика. Она позволяет писать сценарии, не зависящие от базы данных. Задать тип базы данных можно в параметрах соединения.

Класс PEAR DB

В PHP есть много готовых решений для уровня абстракции. Но в этом уроке рассматривается класс PEAR DB.

В уроке 25, "Использование PEAR", содержится больше информации о PEAR — библиотеке расширений и приложений для PHP. А также о других полезных классах в нем.

Класс DB реализует абстрагирование от базы данных с помощью расширений для различных баз данных. Все поддерживаемые расширения приводятся в табл. 20.1.

Таблица 20.1. PHP-расширения для баз данных, поддерживаемые классом PEAR DB

Расширение	*База данных*
dbase	dBase (.dbf)
fbsql	FrontBase
ibase	Firebird/Interbase
ifx	Informix
msql	Mini SQL
mssql	Microsoft SQL Server

Окончание табл. 20.1

Расширение	*База данных*
mysql	MySQL
mysqli	MySQL 4.1 и выше
oci8	Oracle 7, 8 и 9
odbc	ODBC
pgsql	PostgreSQL
sqlite	SQLite
Sybase	Sybase



Документация по классу DB

Электронная документация класса PEAR DB находится по адресу: `http://pear.php.net/package/DB`.

Установка класса DB

Чтобы получить список установленных на Web-сервере классов, в том числе и класса DB, нужно выполнить следующую команду:

```
$ pear list
```

Для того чтобы установить класс DB, нужно выполнить следующую команду:

```
$ pear install DB
```

Нужно помнить, что для установки класса PEAR на общем Web-сервере, нужно связаться с системным администратором.

После установки необходимое PHP-расширение автоматически активизируется. Поэтому не нужно устанавливать дополнительные драйверы баз данных, чтобы соединиться с различными типами баз при помощи класса DB.



Для дальнейшего чтения

Подробнее о MySQL можно прочитать в книге Поля Дюбуа, которая так и называется — *MySQL*. Чтобы получить общее представление о языке SQL, можно обратиться к книге Бена Форты *Освой самостоятельно SQL, 10 минут на урок*.

Названия источников данных

Чтобы соединиться с базой данных через класс DB, нужно составить правильное имя источника (DSN). Это строка, в которой задаются параметры соединения. Она напоминает обычные URL, которые используются при доступе на защищенные страницы или при обращении к FTP-серверам.

Следующий DSN используется для соединения с базой данных MySQL, которая работает на localhost:

```
mysql://chris:mypassword@localhost/mydb
```

DSN состоит из таких частей: тип базы данных (mysql), имя пользователя (chris), пароль (mypassword), сервер (localhost) и название базы данных (mydb).

Пример полного синтаксиса DSN приводится ниже, а в табл. 20.2 подробно описан каждый параметр:

```
phptype(dbsyntax)://username:password@protocol+hostspec/
database?option=value
```

Таблица 20.2. Составляющие DSN

Составляющая	*Описание*
phptype	Протокол используемой базы данных (например, mysql или oci8)
dbsyntax	Необязательный параметр, связанный с SQL-синтаксисом. Для ODBC нужно указать тип базы данных (например, access или mssql)
username	Имя пользователя для соединения с базой
password	Пароль для доступа к базе данных
protocol	Протокол соединения (например, tcp или unix)
hostspec	Параметр сервера. Может быть *имя_сервера* или *имя_сервера*:*порт*
database	Имя базы данных
option	Дополнительные параметры соединения, которые разделяются с помощью &

Как показано в примере с MySQL, не все составляющие DSN являются необходимыми. Реальный синтаксис зависит от той информации, которая нужна для базы данных.

Например, строка соединения SQLite, для которой не нужно username, password и hostspec, выглядит так:

```
sqlite:///path/to/dbfile
```

И наоборот, если нужно соединиться с сервером PostgreSQL, который работает на нестандартном порту, нужно задать более сложное выражение:

```
pgsql://username:password@tcp(hostname:port)/dbname
```

Использование класса DB

Чтобы использовать класс DB в сценарии, нужно подключить его с помощью:

```
include "DB.php";
```



Тип базы данных

Тип базы данных в параметре phptype соответствует значению из первого столбца в табл. 20.1.

Чтобы соединиться с базой данных, нужно вызвать метод connect класса DB и передать DSN в качестве аргумента:

```
$db = DB::connect($dsn);
```

Переменная $db принимает значение объекта класса DB. С его помощью можно выполнять различные операции с базой данных.



Объект базы данных

Нельзя создавать новый объект класса DB с помощью оператора new. Для этого нужно воспользоваться DB::connect, который открывает новое соединение с базой данных.

Если не удается соединиться с базой данных, метод возвращает объект класса DB_Error. Его можно проанализировать с помощью методов isError и getMessage. В этом примере выполняется соединение с базой данных и проверяется наличие ошибки:

```
$db = DB::connect($dsn);
if (DB::isError($db)) {
    echo "Ошибка соединения: " . $db->getMessage();
    exit;
}
```


Метод isError возвращает true, если аргумент является объектом класса DB_Error. Это свидетельствует о проблеме при соединении с базой данных. После этого можно вызвать метод getMessage объекта класса DB_Error. Он позволяет получить реальную информацию от сервера базы данных.



Ошибки соединения

Переменной $db в любом случае присваивается объект (независимо от того, удачное ли соединение). Он не принимает значение NULL или FALSE.

Выполнение запросов

Чтобы выполнить SQL-запрос с помощью класса DB, используется метод query. Возвращаемое значение зависит от типа выполняемого запроса. В случае ошибки возвращается объект DB_Error. Он позволяет определить ошибку и проанализировать ее (аналогично анализу ошибки при соединении).

В этом примере выполняется запрос из переменной $sql с проверкой наличия ошибки:

```
$res = $db->query($sql);
if (DB::isError($res)) {
    echo "Ошибка запроса " . $res->getMessage();
    exit;
}
```

Если запрос состоит из команды INSERT, UPDATE или DELETE, при удачном выполнении на выходе получим константу DB_OK. Количество измененных запросом строк можно определить с помощью метода affectedRows из объекта соединения с базой данных:

```
$sql = "UPDATE mytable SET col2 = 'newvalue' WHERE col1 >
5";
$res = $db->query($sql);
echo $db->affectedRows(). " строк было изменено";
```

Извлечение выбранных данных

Выражение SELECT возвращает объект DB_Result, который позволяет получить доступ к записям из результирующего набора данных.

Чтобы узнать количество строк и столбцов в наборе данных, нужно использовать, соответственно, методы numRows и numCols:

```
$sql = "SELECT * FROM mytable";
$res = $db->query($sql);
echo "В результате запроса найдено " . $res->numRows . " строк ".
    "и " . $res->numCols . " столбцов";
```

С помощью метода fetchRow объекта DB_Result можно извлечь строку в виде массива. После каждого вызова fetchRow указатель увеличивается на единицу, и извлекается следующая строка данных. Ниже показано, как в цикле извлечь все строки из результата с помощью fetchRow:

```
$sql = "SELECT col1, col2 FROM mytable";
$res = $db->query($sql);
while ($row = $res->fetchRow()) {
    echo "col1 = " . $row[0] . ", ";
    echo "col2 = " . $row[1] . "<br>";
}
```

В этом примере нумерация элементов $row начинается с нуля. Все выбираемые столбцы перечислены в выражении SELECT. Поэтому точно известен их порядок. А значит, элемент $row[0] содержит значение col1.

Для метода fetchRow можно задать необязательный аргумент, чтобы изменить индексацию массива. По-умолчанию массив индексируется числами. Это соответствует DB_FETCHMODE_ORDERED. Параметр DB_FETCHMODE_ASSOC, создает ассоциативный массив с ключами, соответствующими названиям столбцов.

Ниже приведен аналог предыдущего примера с использованием ассоциативного массива:

```
while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
    echo "col1 = " . $row["col1"] . ", ";
    echo "col2 = " . $row["col2"] . "<br>";
}
```

При необходимости метод fetchRow возвращает объект вместо массива, если передать аргумент DB_FETCHMODE_OBJECT. Вот пример с использованием объектов:

```
while ($row = $res->fetchRow(DB_FETCHMODE_OBJECT)) {
    echo "col1 = " . $row->col1. ", ";
    echo "col2 = " . $row->col2 . "<br>";
}
```




Режимы извлечения

Какой режим использовать, зависит от личного предпочтения. Ассоциативный массив или объект удобнее. Кроме того, они повышают читабельность кода. Но если критична скорость, используйте DB_FETCHMODE_ORDERED.

Сокращение вызовов

Если в результате запроса возвращается одна строка с одним столбцом, например результат объединяющей функции, можно воспользоваться методом getOne. Он позволяет выполнить запрос и сразу получить результат. На вход подается строка запроса, а на выходе получаем результат:

```
$sum = $db->getOne("SELECT sum(col1) FROM mytable");
```

Есть и другие полезные сокращения. Метод getRow позволяет выполнить запрос и получить строку. Метод getAll выполняет запрос и возвращает данные в виде массива. В документации можно найти весь список функций.

Совместимость между базами данных

Уровень абстракции позволяет перенести код управления базой данных на другую платформу. Нужно просто изменить DSN для соединения с базой данных.

Но каждая база данных имеет свои особенности. Поэтому структуру таблиц и SQL-выражения следует выбирать как можно более общими.

Следует использовать только самые простые инструкции языка SQL, которые доступны во всех основных базах данных. Например, SQL с вложенными запросами не поддерживается MySQL 4.0 и ниже. Лучше избегать использования специфических для некоторых баз данных SQL-команд. Например, LIMIT или CREATE SEQUENCE.

Режим совместимости

Класс DB имеет режим совместимости. Он облегчает смену типа базы данных. Директивы настройки для этого режима приводятся в табл. 20.3. Метод setOption позволяет установить нужные параметры, которые комбинируются с помощью логического ИЛИ:

```
$db->setOption('portability',
     DB_PORTABILITY_ERRORS | DB_PORTABILITY_NUMROWS);
```

Таблица 20.3. Константы режима совместимости

Константа	*Режим*
DB_PORTABILITY_ALL	Включить все настройки совместимости
DB_PORTABILITY_NONE	Выключить все настройки совместимости
DB_PORTABILITY_DELETE_COUNT	Включить принудительный режим подсчета для оператора DELETE. Если нет выражения WHERE, добавить в конце WHERE 1=1
DB_PORTABILITY_ERRORS	Включить поддержку одинаковых сообщений об ошибках для различных баз данных
DB_PORTABILITY_LOWERCASE	Принудительный перевод названий таблиц и столбцов в нижний регистр
DB_PORTABILITY_NULL_TO_EMPTY	Перевести извлеченное значение NULL в пустую строку, потому что некоторые базы данных не различают их
DB_PORTABILITY_NUMROWS	Включить поддержку корректной работы метода numRows в Oracle
DB_PORTABILITY_RTRIM	Принудительное отсекание разделительных символов в извлеченных данных

Работа с кавычками

Метод quoteSmart позволяет заключить значения в кавычки для безопасной вставки в таблицу. Строчные значения заключаются в кавычки, и все нужные символы автоматически преобразовываются.

В следующем примере составляется SQL-выражение при помощи quoteSmart, чтобы не перекрывались символы одиночных кавычек:

```
$sql = "INSERT INTO phrases (phrase) " .
       "VALUES ( " . $db->quoteSmart($text) . " )";
```

Ниже показано конечное значение $sql для MySQL-драйвера после выполнения предыдущего выражения:

```
INSERT INTO phrases (phrase)
VALUES ( Одиночная кавычка \' не перекроется!' )
```

Вывод и правило разделения зависят от типа базы данных.

Последовательности

Реализация последовательностей может сильно отличаться для различных баз данных. В MySQL, например, используется атрибут AUTO_INCREMENT для столбца. Для MSSQL это будет IDENTITY. В Oracle CREATE SEQUENCE создает объект, который отслеживает значение последовательности для таблицы.

Класс DB использует свой набор функций для управления последовательностями. Поэтому использование поля с автоматическим увеличением не привязывает к одному типу базы данных.



Последовательности

Если в базе данных нет поддержки последовательностей, класс DB эмулирует их. Для этого создается таблица, в которой содержится значение последовательности. При каждом обращении к последовательности выполняется увеличение.

Метод объекта базы данных — createSequence позволяет создавать последовательность с уникальным идентификатором. Чтобы получить следующее значение в после-

довательности, нужно вызвать метод nextId с этим идентификатором.

В примере ниже создается последовательность с именем order_number и выводится первое значение последовательности:

```
$db->createSequence("order_number");
echo $db->nextId("order_number");
```

Последующие обращения к nextId возвращают увеличенные значения для этой последовательности. Если последовательность больше не нужна, ее можно удалить с помощью метода dropSequence.

Ограничение запроса

Ключевое слово LIMIT базы данных MySQL ограничивает количество строк, которые возвращаются в результате запроса. Это нестандартный оператор SQL. Другие базы данных его не поддерживают.

Класс DB содержит метод limitQuery, который эмулирует LIMIT в SQL-выражениях для обеспечения максимальной совместимости. Этот метод вызывается аналогично query, но требует два дополнительных параметра: начальную строку и количество нужных строк.

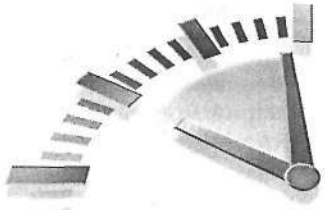
В следующем примере возвращается пять строк из полученного результата, начиная с 11-ой (нумерация строк начинается с нуля):

```
$res = $db->limitQuery("SELECT * FROM mytable", 10, 5);
```

Резюме

В этом уроке вы узнали, как писать переносимые сценарии для различных баз данных с помощью уровня абстракции. В следующем уроке вы научитесь писать и выполнять сценарии для оболочки с помощью PHP.

Урок 21



Выполнение PHP-сценариев в командной строке

PHP разрабатывался для написания динамических Web-страниц. Но из-за большой мощности его стали применять для написания сценариев командной строки и даже для разработки приложений. В этом уроке вы узнаете, как использовать PHP в командной строке и создавать сценарии оболочки.

Среда командной строки

Для того чтобы запускать PHP-сценарии из командной строки, нужно установить интерпретатор PHP на данной системе. Для Web-среды он устанавливается как модуль Apache. Но для использования в интерфейсе командной строки (CLI) его нужно установить как самостоятельное приложение под названием php.

Отличие между исполняемыми файлами CLI и CGI

Начиная с версии 4.2, PHP содержит различные бинарные программы для CGI и CLI. Оба обработчика предоставляют тот же интерпретатор языка. Но версия CLI имеет следующие отличия, удобные для командной строки.

- На вывод не передаются HTTP-заголовки.
- Сообщения об ошибках не содержат HTML-форматирования.

- Значение max_execution_time равно нулю, чтобы сценарий мог выполняться неограниченное время.

Чтобы узнать о том, является программа php CGI- или CLI-версией, нужно запустить ее с параметром -v. Тогда появляется информация о версии. Например, следующий вывод генерирует CLI-версия PHP 5.0.3:

```
PHP 5.0.3 (cli) (built: Dec 15 2004 08:07:57)
Copyright (c) 1997-2004 The PHP Group
Zend Engine v2.0.3, Copyright (c) 1998-2004 Zend
Technologies
```

Значение в круглых скобках после версии PHP показывает используемый вариант серверного интерфейса (SAPI). Его можно определить с помощью функции php_sapi_name.



Версия для Windows

Версия для Windows PHP 4.2 включает два исполняемых файла: CGI-версию с именем php.exe и CLI-версию с именем php-cli.exe. Для PHP 4.3 оба называются php.exe, но находятся, соответственно, в каталогах cgi и cli.

Для PHP 5 и выше php.exe является CLI-версией, а CGI-версия называется php-cgi.exe. Кроме того, есть php-win.exe — исполняемая версия CLI, которая работает без вывода. Это позволяет пользователю не открывать окно с командной строкой.

Написание сценариев PHP для оболочки Linux/Unix

На платформах Linux/Unix *сценарий оболочки* — это текстовый файл, содержащий набор инструкций, которые обрабатываются соответствующим интерпретатором. Самый простой интерпретатор оболочки — Bourne Shell, или sh. Хотя в данный момент более популярна его улучшенная версия — Bourne Again Shell, или bash. Она полностью совместима с sh, но имеет больше полезных возможностей.

Большинство языков оболочек довольно ограниченные и часто требуют вызова дополнительных программ. PHP же предоставляет гибкий и мощный синтаксис, а также луч-

шую производительность, чем большинство стандартных решений.



Расположение PHP

Выполняемый файл php обычно устанавливается в /usr/local/bin или /usr/bin, в зависимости от типа установки. И может находиться вообще в другом месте. Команда which php позволяет его быстро найти.

Все сценарии оболочки начинаются с символов #!, после которых следует полный путь к интерпретатору команд. Для традиционных сценариев оболочки эта строка выглядит так:

```
#!/bin/sh
```

Для сценариев PHP она принимает следующий вид:

```
#!/usr/local/bin/php
```

Для файла нужно установить разрешение на выполнение. Это можно сделать с помощью такой команды:

```
$ chmod u+x myscript.php
```

С помощью следующей команды устанавливается разрешение на выполнение для всех пользователей:

```
$ chmod a+x myscript.php
```

Без флага выполнения сценарий выполняется, если подать его на вход интерпретатора PHP. Для этого вызывается команда php с аргументом имени файла. Две команды ниже — идентичны (параметр -f используется для большей ясности, но не является обязательным):

```
$ php myscript.php
$ php -f myscript.php
```



Название сценария

Нет каких-либо правил для именования сценариев оболочки. Но лучше оставить расширение .php. Тогда понятно, что это сценарий PHP. Сценарии Bourne shell иногда имеют расширение .sh, но чаще в названиях команды вообще не используется расширение.

PHP-сценарии командной строки для Windows

Windows не поддерживает альтернативные интерпретаторы команд. Поэтому, чтобы использовать PHP-сценарий в Windows, нужно передать название файла программе `php.exe`. Параметр `-f` — необязательный, поэтому два таких вызова эквивалентны:

```
> php.exe myscript.php
> php.exe -f myscript.php
```



Сценарии оболочки

Можно создать простой сценарий оболочки, который вызывает `php.exe` с нужным аргументом файла, чтобы запускать сценарий одной командой.

Для этого нужно создать файл `myscript.bat`. В нем нужно разместить команду `php.exe` и название сценария. После этого можно выполнять сценарий вводом слова `myscript` в командной строке.

Вставка PHP-кода

Как и для Web-среды, можно вставлять PHP-код в текст сценария. Любой текст, который находится за пределами дескрипторов `<?php`, передается на вывод.

Но обычно нужно создать сценарий, полностью состоящий из PHP-кода. Поэтому в начале PHP-сценария для оболочки нужно поставить `<?php`. PHP позволяет создавать сценарии, которые выводят только часть элементов в больших статических текстовых файлах.

Написание сценариев для командной строки

В языке PHP есть механизмы, которые помогают писать сценарии для командной строки. Иногда их используют для Web-среды. Ниже приводится их описание.

Режим вывода символов

При генерировании Web-вывода используется дескриптор `<br>`, чтобы разделить строки. При выводе на Web-страницу символа перевода строки, `\n`, разрыв будет только в HTML-версии. При выводе в браузере он незаметен.

В сценариях командной строки используется только текстовый вывод, поэтому нужно использовать перевод строки для форматирования текста. Если сценарий генерирует произвольный вывод, нужно всегда завершать его `\n` после вывода последнего элемента.

Можно получить преимущество от режима с фиксированной шириной символов при выполнении в командной строке, если использовать пробелы для создания столбцов. Функция `printf` позволяет устанавливать ширину и задавать выравнивание символов. Это не дает соответствующего эффекта в HTML-выводе, если не использовать дескриптор `<PRE>`. Подробнее об этом рассказывается в уроке 6, "Обработка строк".

Аргументы командной строки

Можно передать аргументы сценарию оболочки. Для этого они указываются после имени сценария. Количество аргументов хранится в переменной `$argc`, а сами аргументы — в массиве с числовыми индексами `$argv`.



Аргументы

Названия `argc` и `argv` используются исторически. Изначально они применялись в языке C, а сейчас широко применяются во многих языках программирования. В PHP `$argc` создается только по договоренности. Можно вместо этого использовать `count($argv)`, чтобы определить, сколько аргументов передается в сценарий.

Массив `$argv` всегда содержит как минимум один элемент, даже если дополнительные аргументы не передавались в сценарий. Просто `$argv[0]` содержит названия сценария и тогда `$argc` равен `1`.

Для сценария в листинге 21.1 нужно задать два аргумента. В противном случае выводится сообщение об ошиб-

ке, и выполнение завершается. На выходе получаем большее из этих двух аргументов.

Листинг 21.1. Использование параметров командной строки

```
#!/usr/local/bin/php
<?php
if ($argc != 3) {
    echo $argv[0].": Нужно задать точно два параметра\n";
    exit;
}

if ($argv[1] < $argv[2]) {
    echo $argv[1] . " меньше чем ". $argv[2] . "\n";
}
elseif ($argv[1] > $argv[2]) {
    echo $argv[1] . " больше чем ". $argv[2] . "\n";
}
else {
    echo $argv[1] . " равно ". $argv[2] . "\n";
}
?>
```

Начальное условие в листинге 21.1 проверяет `$argc` на равенство 3. Такое число получается из двух параметров командной строки и названия сценария в элементе `$argv[0]`. Кроме того, `$argv[0]` используется при выводе сообщения об ошибке. Такой прием позволяет узнать имя сценария.

Потоки ввода-вывода

Для Web-окружения можно напрямую выводить данные в стандартные потоки ввода, вывода и ошибок. Но это чаще используется в сценариях командной оболочки.

Для доступа к потокам используются функции работы с файловой системой. Нужно просто открыть обработчик файла для соответствующего потока и управлять им таким же образом.

Идентификаторы потока очень напоминают URL (нужно помнить, что PHP позволяет открыть URL с помощью функций доступа к файлам). Они начинаются с `php://`, после которого следует название потока. Следующее выражение открывает для чтения стандартный поток ввода:


```
$fp = fopen("php://stdin", "r");
```

Доступ к потокам часто используется в сценариях командной строки, поэтому в PHP есть удобные сокращения. Константы `STDIN`, `STDOUT` и `STDERR` предоставляют непосредственный доступ к открытым потокам и не требуют вызова `fopen`.

Сценарий в листинге 21.2 использует три стандартных потока. Он читает данные из стандартного ввода и переводит их в верхний регистр с помощью функции `strtoupper`. Если входные данные содержат не только буквы и цифры, выводится сообщение об ошибке в стандартный поток ошибок.

Листинг 21.2. Чтение и запись в стандартные потоки

```
#!/usr/local/bin/php
<?php
while (!feof(STDIN)) {
    $line++;
    $data = trim(fgets(STDIN));

    fputs(STDOUT, strtoupper($data)."\n");
    if (!ereg("^[[:alnum:]]+$", $data)) {
        fputs(STDERR,
            "Ошибка: неправильный номер строки $line\n");
    }
}
?>
```

Если запустить этот сценарий в командной строке, он будет ожидать ввода данных по одной строке за раз и возвращать ее в верхнем регистре. Преимущество стандартного потока ввода в том, что ввод можно перенаправить из другого источника.

Передать содержимое файла `myfile` в сценарий `myscript` и перенаправить вывод в `outfile` можно с помощью следующей команды:

```
$ myscript < myfile > outfile
```

Сценарий из листинга 21.2 переводит все данные файла `outfile` в верхний регистр. Все сообщения об ошибках выводятся на экран, если не перенаправить стандартный поток ошибок.

Все константы и идентификаторы потока, доступные в командной строке PHP, показаны в табл. 21.1.

Таблица 21.1. Доступ к потокам для PHP CLI

Константа	*Идентификатор*	*Поток*
STDIN	php://stdin	Стандартный ввода
STDOUT	php://stdout	Стандартный вывода
STDERR	php://stderr	Стандартный ошибок

Создание настольных приложений

PHP настолько мощный язык, что позволяет создавать полноценные приложения. А поскольку PHP является интерпретирующимся языком, приложения написанные с его использованием, легко переносимы.

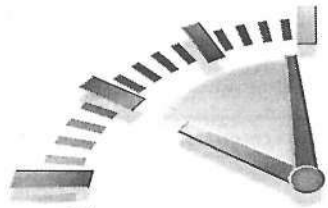
Расширение PHP-GTK реализует интерфейс к оконному набору инструментов GIMP, GTK+. Это позволяет разработчикам PHP создавать полноценные приложения с графическим интерфейсом, которые используют окна, меню, кнопки и поддержку мыши.

Создание сложного приложения с графическим интерфейсом выходит за рамки этой книги. Тем, кто хочет больше узнать о PHP-GTK, нужно обратиться к http://gtk.php.net.

Резюме

В этом уроке вы узнали, как писать PHP-сценарии для командной строки. В следующем уроке вы научитесь обрабатывать ошибки и отлаживать сценарии PHP.

Урок 22



Обработка ошибок

В этом уроке вы узнаете, как эффективно обрабатывать ошибки в сценариях PHP и отлаживать сценарии, которые не работают так, как ожидалось.

Система уведомления об ошибках

Система уведомления об ошибках просто настраивается в PHP и позволяет легко установить требуемый уровень строгости. По-умолчанию, строгий режим отключен. Поэтому сообщения об ошибках появляются только в случае потенциальной опасности неправильной работы сценария.

Изменение уровня ошибок

Функцию error_reporting позволяет изменить уровень уведомления об ошибках. На вход нужно подать одну из констант в табл. 22.1.

Таблица 22.1. Константы для настройки уровня уведомления

Константа	*Описание*
E_ERROR	Сигнализирует о критической ошибке выполнения. Выполнение сценария останавливается
E_WARNING	Устанавливает предупреждения во время выполнения. Не критическая, сценарий продолжает выполняться
E_PARSE	Сигнализирует об ошибках компиляции

Окончание табл. 22.1

Константа	*Описание*
E_NOTICE	Показывает замечания во время выполнения. Может указать на потенциальную ошибку
E_CORE_ERROR	Сигнализирует о критической ошибке, которая генерируется внутри самого PHP
E_CORE_WARNING	Сигнализирует о предупреждении и генерируется внутри самого PHP
E_COMPILE_ERROR	Сигнализирует о критической ошибке Zend engine
E_COMPILE_WARNING	Сигнализирует о предупреждении Zend engine
E_USER_ERROR	Сообщает об ошибках, сгенерированных пользователем при помощи trigger_error
E_USER_WARNING	Сообщает о предупреждениях, сгенерированных пользователем при помощи trigger_error
E_USER_NOTICE	Сообщает о замечаниях, сгенерированных пользователем при помощи trigger_error
E_ALL	Сообщает обо всех ошибках, кроме E_STRICT
E_STRICT	Сообщает обо всех ошибках и предупреждениях, включая предложения PHP по изменению кода для улучшения его совместимости

Эти константы можно объединить с помощью поразрядных операторов и создать необходимый набор битов, который установит нужный уровень уведомления. По умолчанию устанавливается уровень E_ALL & E_NOTICE, который показывает все ошибки и предупреждения, кроме E_STRICT. Последний не перекрывается в E_ALL и E_NOTICE.

Чтобы установить уровень уведомления, который показывает все предупреждения и замечания, нужно выполнить следующую команду:

```
error_reporting(E_ALL);
```


Предупреждения, которые не показываются по умолчанию, не являются критическими и не мешают нормальному выполнению сценария.

Уровень ошибок E_NOTICE может помочь при разработке сценария. Он предупреждает обо всех необъявленных переменных. Кроме того, E_NOTICE не приводит к ошибкам в сценарии. Эти сообщения могут указывать на то, что в названии переменной найдена опечатка и нужно посмотреть раннее набранный код.

Флаг E_NOTICE позволяет проконтролировать стиль кодирования. Например, ключ ассоциативного массива нужно заключать в кавычки. Но небрежный программист может написать `$array[key]`. Нужно помнить, что key в PHP является константой. Если она не определена в сценарии, PHP пытается найти переменную с соответствующим текстовым ключом. Но при включенном E_NOTICE появится сообщение об этой двузначности.

Уровень E_STRICT появился в PHP 5 и очень помогает писать актуальный код. Он уведомляет об использовании упраздненных функций, которые поддерживаются только для обратной совместимости.

Изменить уровень уведомления можно также в файле php.ini или с помощью .htaccess в нужном каталоге с помощью инструкции error_reporting. В уроке 23, "Настройка PHP", описывается, как использовать эти возможности.



Показ ошибок

Настройки log_errors и display_errors определяют место вывода сообщений об ошибках. Это может быть экран или файл журнала.

На рабочем Web-сайте лучше не выводить сообщения об ошибках на экран. Это небезопасно, т.к. злоумышленник может получить информацию о системе.

Специальный обработчик ошибок

PHP позволяет задать специальную функцию, которая вызывается при обнаружении ошибки. Это позволяет заменить стандартное действие вывода на экран или записи в файл журнала.

Функция set_error_handler задает функцию, которая будет специальным обработчиком ошибок. Ей нужно передать два аргумента. Первый, обязательный — имя функции и второй — необязательный, в котором содержится набор битов, задающий необходимый уровень ошибок. Он и будет обрабатываться этой функцией.

Например, для того, чтобы функция myhandler отслеживала все ошибки E_WARNING и E_NOTICE, используется следующая команда:

```
set_error_handler("myhandler", E_WARNING & E_NOTICE);
```

Определенный пользователем обработчик ошибок требует два параметра: код ошибки и строку с сообщением об ошибке. Значения для кода ошибки можно взять из табл. 22.1, чтобы знать, какая ошибка произошла. Также можно добавить три необязательных параметра, которые помогут найти ошибку: название файла, номер строки и контекст, в котором произошла ошибка.

В листинге 22.1 задается специальный обработчик ошибок, который записывает все ошибки в таблицу базы данных MySQL.

Листинг 22.1. Специальный обработчик ошибок

```
<?php
function log_errors($errno, $errstr, $errfile, $errline) {
    $db = mysql_connect("localhost", "loguser",
"logpassword");
    mysql_select_db("test", $db);
    $errstr = mysql_escape_string($errstr);
    $sql = "insert into php_log
                (errno, errstr, errfile, errline)
            values
                ('$errno', '$errstr', '$errfile',
'$errline')";
    $res = mysql_query($sql, $db);
}
set_error_handler("log_errors");
// Присвоение неопределенной переменной вызовет
предупреждение
$a = $b;
?>
```

Таблицу для сбора ошибок можно создать с помощью следующего SQL-выражения:


```
CREATE TABLE php_log (
    error_timestamp timestamp,
    errno int,
    errstr text,
    errfile text,
    errline int
);
```

Нужно помнить, что в этом примере не используется пятый необязательный параметр для контекста. Контекст передается в виде массива. В нем содержится значение каждой переменной, включая локальные и системные суперглобальные, которые определены в сценарии на момент, когда случилась ошибка.

Эта информация может пригодиться при отладке, но ее слишком много, чтобы хранить в файле журнала или таблице. Кроме того, так как значение передается в виде массива, $errcontext нужно сначала передать в функцию serialize, чтобы сохранить в таблице или текстовом файле.



Данные контекста

Обычно нужна только небольшая часть данных контекста, которые передаются функции-обработчику. Поэтому функция может извлечь нужную часть, а остальное отбросить.

Генерирование пользовательской ошибки

Можно воспользоваться функцией trigger_error, чтобы при необходимости сгенерировать ошибку. Если задан специальный обработчик ошибок, он и будет обрабатывать пользовательскую ошибку.

Иначе ее обработкой займется стандартный обработчик ошибок PHP. Нужно передать строку с ошибкой в trigger_error. Также можно добавить необязательную константу типа ошибки. Если ее не задать, будет использована E_USER_NOTICE.

Например, для генерирования пользовательского замечания можно воспользоваться следующим кодом:

```
trigger_error("Случилась какая-то ошибка");
```

На экране появится ошибка похожая на эту:

```
Notice: Случилась какая-то ошибка in /home/chris/error.php
on line 3
```

Чтобы сгенерировать критическую ошибку с аналогичным текстом, используется следующее выражение:

```
trigger_error("Случилась какая-то ошибка", E_USER_ERROR);
```



Типы ошибок

Для пользовательских ошибок нужно использовать E_USER_ERROR и E_USER_WARNING, а не E_ERROR и E_WARNING. Тип E_USER_ERROR тоже трактуется как критическая ошибка и завершает выполнение сценария сразу после обнаружения.

Сохранение ошибок

С помощью функции error_log можно удовлетворить все простые потребности по хранению ошибок. Эта функция позволяет записывать ошибки в файл журнала Web-сервера или любой другой, отправлять их по электронной почте или пересылать на удаленный сервис отладки.

Функции error_log передаются следующие аргументы:

```
error_log($message, $message_type, $destination,
        $extra_headers);
```

Только аргумент message является обязательным. Стандартным действием является запись сообщения в файл журнала PHP. В большинстве случаев это будет файл журнала Web-сервера.

Аргумент message_type задает тип отправки сообщения. Возможные значения перечислены в табл. 22.2.

Таблица 22.2. Значения message_type для error_log

Значение	*Описание*
0	Сообщение записывается в стандартный файл журнала Web-сервера
1	Сообщение отправляется по электронной почте. В параметре destination содержится ящик адресата, а extra_headers позволяет задать дополнительные заголовки письма

Окончание табл. 22.2

Значение	*Описание*
2	Сообщение отправляется на удаленный сервис отладки. Параметр destination содержит адрес сервера. Отметим, что удаленная отладка не поддерживается, начиная с PHP 4 и выше
3	Сообщение добавляется в конец локального файла. Параметр destination содержит имя файла и путь

Например, для отправки сообщения об ошибке по электронной почте используется следующее выражение:

```
error_log("В сценарии произошла ошибка", 1,
          "chris@lightwood.net",
          "From: PHP-сценарий
ошибки<errors@yoursite.com>");
```

Подавление ошибок и предупреждений

PHP позволяет подавлять сообщения об ошибках в сценарии. Можно полностью отключить показ ошибок или выбрать отдельные команды, для которых не будут показываться ошибки.

Оператор подавления ошибки

Для того чтобы подавить вывод ошибки в отдельном операторе, используется символ @. При этом можно выводить специальное сообщение об ошибке.

Например, если при соединении с базой данных MySQL произошел сбой, PHP генерирует собственную ошибку. Но может быть также и альтернативная проверка соединения. Например, проверка идентификатора соединения с базой данных. В следующем примере используется символ @, который блокирует вывод ошибок PHP. Таким образом, в случае ошибки появится только специальное сообщение:

```
$db = @ mysql_connect("localhost", "username", "password");
if (!$db) {
    echo "Не удалось соединиться с базой данных";
    exit;
}
```

Можно поместить символ @ перед любым выражением в PHP. Он подавляет все сообщения об ошибках, которые произошли во время выполнения выражения.

В этом примере подавляется выражение, которое выполняет соединение с базой данных. Общее правило таково: если что-то в PHP может иметь значение, перед ним следует поставить символ @. Нельзя размещать символ @ перед управляющими конструкциями языка. Например, перед определением функции или условным выражением.



Анализ ошибок

Оператор @ не скрывает ошибок, которые обнаружены на стадии анализа кода сценария.

Следующее выражение — правильное. Но кажется, что сообщение об ошибке относится к mysql_connect:

```
@ $db = mysql_connect("localhost", "username", "password");
```



Подавление ошибок

Если в сценарии используется set_error_handler для установки специального обработчика ошибок, оператор @ не оказывает никакого эффекта.

Предотвращение показа ошибок

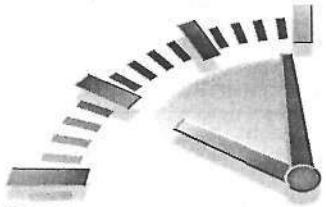
Если директиве настройки display_errors из файла php.ini установить значение Off, это предотвращает вывод всех сообщений об ошибках на экран. Подробнее о работе с файлом php.ini рассказывается в уроке 23.

При отключении вывода ошибок на экран, нужно активизировать директиву log_errors. Все ошибки будут сохраняться в файл журнала. Иначе будет невозможно узнать о потенциальных проблемах на сайте. Не нужно выключать display_errors на стадии разработки Web-сайта.

Резюме

В этом уроке вы узнали, как обнаружить и обработать ошибку в сценарии PHP. В следующем уроке вы научитесь настраивать PHP для конкретных потребностей.

Урок 23



Настройка PHP

В этом уроке вы научитесь изменять глобальные настройки PHP во время выполнения и при помощи файла php.ini, а также для отдельных каталогов при помощи .htaccess.

Настройки конфигурации

PHP позволяет настраивать множество различных аспектов поведения с помощью набора директив настройки. Эти инструкции могут быть глобальными для всего сервера или локальными, в рамках одного сценария.

Возможности php.ini

Файл настройки PHP называется php.ini. Его размещение определяется во время компиляции. Обычно это /usr/local/lib/php.ini на Linux/Unix серверах и C:\WINDOWS\php.ini для Windows-систем.

В файле php.ini находится список директив и их значений, которые разделяются знаком равенства. Обычно php.ini, который поставляется с PHP, очень хорошо документирован и содержит множество комментариев. Комментирующие строки начинаются точкой с запятой. Кроме того, файл разделяется на секции при помощи заголовков в квадратных скобках. Они также игнорируются компилятором.

В листинге 23.1 содержится часть оригинального php.ini из PHP 5, в которой содержатся настройки ведения журнала. Как видите, чтобы внести изменения, документация зачастую не нужна.

Листинг 23.1. Часть файла php.ini

```
; Print out errors (as a part of the output). For
; production web sites,
; you're strongly encouraged to turn this feature off,
; and use error logging
; instead (see below). Keeping display_errors enabled
; on a production web site
; may reveal security information to end users, such as
; file paths on your Web
; server, your database schema or other information.
display_errors = On

; Even when display_errors is on, errors that occur
; during PHP's startup
; sequence are not displayed. It's strongly recommended
; to keep
; display_startup_errors off, except for when debugging.
display_startup_errors = Off

; Log errors into a log file (server-specific log, stderr,
; or error_log (below))
; As stated above, you're strongly advised to use error
; logging in place of
; error displaying on production web sites.
log_errors = Off

; Set maximum length of log_errors. In error_log
information
; about the source is
; added. The default is 1024 and 0 allows to not apply any
;maximum length at all
log_errors_max_len = 1024
```



True или False

Булевым значениям в php.ini соответствуют on или yes для true и off, no или none для false. Эти значения не зависят от регистра.

Если PHP работает как модуль сервера, php.ini перечитывается перед запуском сервера. Все последующие изменения в конфигурационном файле не влияют на работу сервера до очередного перезапуска.

Если PHP работает как CGI, настройки php.ini загружаются во время каждого выполнения сценария, потому что запускается новый процесс php.

PHP в режиме командной строки также перечитывает содержимое php.ini при каждом запуске.

Замена файла php.ini

Можно создать отдельную версию php.ini для различных способов запуска PHP. Если создать файл с именем php-*SAPI*.ini (нужно заменить *SAPI* на соответствующее имя SAPI), он будет использоваться вместо глобального php.ini.

Например, чтобы задать опции только для командной строки PHP, нужно создать файл с именем php-cli.ini. Для модуля Web-сервера Apache файл должен называться php-apache.ini. На платформе Windows сначала используется файл php.ini в каталоге Apache, а потом тот, что находится в C:\WINDOWS. Это позволяет задать различные установки PHP для разных Web-серверов на одном хосте.

Чтобы установить принудительное использование конфигурационного файла, нужно вызвать php с опцией -c. В сценариях оболочки можно изменить первую строчку на ту, что приведена ниже. Тогда данный сценарий всегда запускается со специальным конфигурационным файлом:

```
#!/usr/local/bin/php -c /path/to/php.ini
```

Настройка для отдельного каталога

Специальный файл .htaccess позволяет изменять настройки Web-сервера Apache. С помощью .htaccess также можно менять глобальные настройки php.ini.

С помощью инструкции php_value можно задать новое значение директивы из файла php.ini. Для этого после нее нужно указать инструкцию и новое значение. В следующей строке файла .htaccess директиве max_execution_time устанавливается значение 60 секунд:

```
php_value max_execution_time 60
```



Использование .htaccess

Нужно внимательно следить за синтаксисом при изменении настроек в php.ini и .htaccess. В php.ini между инструкцией и значением должен быть знак равенства. В .htaccess значение следует после названия директивы без знака равенства.

Изменения, внесенные файлом .htaccess, влияют только на текущий каталог и его подкаталоги. Все установки в .htaccess заменяют глобальные установки php.ini, а также все установки в родительском каталоге.

Динамическая настройка

Значения, установленные в файле php.ini, можно менять во время выполнения с помощью функции ini_set. На вход нужно подать два аргумента: имя инструкции и новое значение. При изменении значения директивы функция ini_set возвращает предыдущее значение этой директивы.

В следующем примере изменяется значение директивы memory_limit для текущего сценария. Это сделано для блока кода, который требует больше системных ресурсов:

```
$limit = ini_set("memory_limit", "128M");
// Далее следует код, который требует этих изменений
ini_set("memory_limit", $limit);
```

Предыдущее значение сохраняется в переменной. После завершения директива принимает начальное значение.

Функция ini_get позволяет узнать текущее значение директивы в php.ini.

Директивы настройки

В коротком уроке нельзя охватить все настройки php.ini, их очень много. Поэтому ниже рассматриваются только самые используемые. Полную справку можно получить по адресу: www.php.net/manual/en/ini.php.

Настройка PHP-окружения

В следующем разделе приводится список наиболее используемых инструкций, которые влияют на окружение PHP. Все инструкции приводятся со стандартными значениями из php.ini, который поставляется с PHP 5.

Стиль дескрипторов PHP

Эти установки позволяют выбрать стиль дескрипторов, который используется в сценарии PHP:

- **short_open_tag = On** — инструкция short_open_tag включает или отключает возможность использовать <? для открывающего дескриптора. Если она выключена, в сценарии нужно использовать полный дескриптор <?php.

  Так как <? может иметь другое значение внутри Web-страницы, нужно избегать использовать short_open_tag. Кроме того, в последующих версиях PHP эту опцию могут отключить по умолчанию.

- **asp_tags = Off** — asp_tags устанавливает возможность использовать <% для открывающего и %> для закрывающего дескриптора PHP. Нужно включить эту установку в php.ini, чтобы использовать такой стиль.

Ограничение системных ресурсов

Следующие инструкции позволяют управлять системными ресурсами, доступными в сценарии PHP.

- **max_execution_time = 30** — директива max_execution_time задает максимальное время выполнение сценария в секундах. Если предел превышается, происходит ошибка, и выполнение завершается.

  Если нет сценариев, которые требуют большего времени на выполнение, не нужно изменять эту инструкцию. Случайный бесконечный цикл в сценарии может забрать на себя много системных ресурсов, а max_execution_time позволяет застраховаться от таких проблем.

  Если загрузка Web-страницы занимает больше 30 секунд, многие посетители не будут ждать окончания. Конечно, если не запрашивается специфическая информация, для которой известно, что генерация занимает определенное время.

- **memory_limit = 8M** — каждый сценарий PHP имеет ограниченный доступ к системным ресурсам. Это сделано для защиты системы в целом. Большинство сценариев использует небольшое количество памяти. Количество используемой памяти можно узнать с помощью функции memory_get_usage.

Суффикс M указывает на значение в мегабайтах. Приставки K и G задают, соответственно, значения в килобайтах и гигабайтах. Ограничение на память можно полностью убрать, установив для memory_limit значение -1.

Обработка форм

С помощью этих инструкций, можно изменить способ взаимодействия PHP с формами.

- **magic_quotes** — Директива magic_quotes дает указание PHP автоматически разделять кавычки. Это нужно для безопасной подстановки строк. Ниже приведены стандартные значения:

  magic_quotes_gpc = On

  magic_quotes_runtime = Off

  magic_quotes_sybase = Off

  Директива magic_quotes_gpc используется для значений из форм и cookies (gpc составлено из GET, POST и COOKIE). Инструкция magic_quotes_runtime принуждает PHP *разделять кавычки для данных, которые генерируются сценарием*. Это может быть строка запроса к базе данных или команда для сервера.

  Обычно кавычки разделяются с помощью обратной косой черты. Но в некоторых базах данных, например Sybase, для разделения используется еще один символ кавычки. Если включить magic_quotes_sybase, результат будет в виде '', вместо \'.

- **register_globals = Off** — опция register_globals отключена по умолчанию, начиная с PHP 4.2. Если она включена, все значения суперглобальных массивов $_ENV, $_GET, $_POST, $_COOKIE и $_SERVER дублируются в область глобальных переменных. Имя переменной берется из ключа соответствующего элемента.

- **variables_order = "EGPCS"** — директива variables_order задает порядок в каком переменные заносятся в глобальную область из суперглобальных массивов. Если включена директива register_globals, значение cookie email заносится в глобальную область позже, чем значение из формы с таким же именем. По

этому переменная $email в сценарии содержит значение cookie.

Так как register_globals создает значение, которое не различается по источнику, лучше использовать суперглобальные массивы. Тогда точно известно, что $_POST["email"] содержит значение, полученное из формы, а $email может получить значение из нескольких источников.

- **register_long_arrays = On** — в ранних версиях PHP использовались массивы $HTTP_GET_VARS, $HTTP_POST_VARS, $HTTP_SERVER_VARS и т.п. вместо новых суперглобальных массивов. Инструкция register_long_arrays определяет, нужно ли создавать эти массивы. Эта функциональность поддерживается для обратной совместимости.

Подключение файлов

Можно использовать инструкцию include_path, чтобы задать список мест, где искать файлы для выражений include и require.

Эти значения разделяются с помощью двоеточия в Linux/Unix системах и точкой с запятой — в Windows.

Часто нужно дать возможность подключить файл, который содержится за пределами Web-сервера. В примере ниже задается путь для подключения. Он содержит каталог, параллельный корневому каталогу Web-сайта /home/chris/public_html:

```
php_value include_path .:/home/chris/include
```

Точка (.) используется для задания текущего каталога. В этом примере она имеет больший приоритет, чем заданный путь подключения. Если выражение include найдет нужный файл в обоих каталогах, подключается файл в рабочем каталоге. Этот механизм позволяет использовать одинаковые библиотеки для различных сайтов. Их можно заменять при необходимости.

Установки auto_prepend_file и auto_append_file позволяют задать файлы, которые автоматически добавляются при старте и завершении любого сценария PHP. Размещение файлов нужно задать в include_path или указать полный путь.

Обычно auto_prepend_file используется для того, чтобы подключить часть HTML-разметки до любого вывода в сценарии, чтобы все страницы выглядели одинаково. Директива auto_prepend_file является механизмом PHP. Поэтому подключать можно только файлы, которые обрабатываются PHP. Для статических HTML-страниц такой возможности нет.



HTTP-заголовки

После отправки вывода на браузер нельзя использовать функцию header для HTTP-заголовков, а также другие механизмы PHP, которые работают с заголовками. Например, сеансы или cookies. Поэтому, если нужно посылать специальные HTTP-заголовки, все сценарии, которые подключаются с помощью auto_prepend_file, не должны генерировать вывод.

Ведение учета ошибок

В уроке 22, "Обработка ошибок", было показано, как настроить PHP для ограничения вывода сообщений об ошибках.

Значение директивы error_reporting является побитовой маской, составленной из значений, приведенных в табл. 22.1. Можно использовать логические операторы, чтобы комбинировать эти значения, как показано ниже:

```
error_reporting = E_ALL & ~E_NOTICE & ~E_STRICT
```

Директивы display_errors и log_errors устанавливают вывод ошибок на экран и в файл журнала, соответственно.

В стандартной конфигурации ошибки выводятся на экран и не записываются в файл журнала:

```
display_errors = On
log_errors = Off
```

С помощью инструкции error_log можно задать альтернативный файл журнала:

```
error_log = /tmp/php_log
```


Настройка расширений PHP

Некоторые расширения PHP имеют собственные директивы настройки в php.ini, которые позволяют изменять их поведение.

Для большей ясности в конфигурационном файле используются заголовки разделов. Они разделяют установки по расширениям. Например, все установки для расширения MySQL находятся в разделе php.ini, который начинается с [MySQL].

Каждое имя инструкции содержит префикс, содержащий название расширения (например: mysql.connect_timeout или session.cookie_path).

Документацию по специфическим инструкциям для расширений можно найти в электронном справочнике для каждого расширения.

Настройка безопасности системы

В этом уроке не рассматриваются директивы php.ini. Это относится к safe_mode и другим настройкам безопасности, которые ограничивают функциональность на Web-сервере. Подробнее о них рассказывается в уроке 24, "Безопасность PHP".

Подгружаемые модули

PHP позволяет загружать отдельные расширения во время выполнения. Так можно расширять функциональность PHP без перекомпиляции исходных файлов.

Загрузка расширения по требованию

С помощью функции dl можно динамически подгружать модуль расширения. Для этого нужно собирать расширение как динамически загружаемый объект при компиляции PHP. Это реализуется с помощью опции --with-*EXTENSION*=shared. Например, следующее выражение configure устанавливает компиляцию PHP с поддержкой MySQL. Но сокеты будут поддерживаться для динамической загрузки:

```
./configure --with-mysql --with-sockets=shared
```

Функции dl нужно передать аргумент — имя файла расширения. Для сокетов этот файл называется sockets.so на Linux/Unix системе и php_sockets.dll для Windows.



Подгружаемые расширения

Чтобы функция dl работала, нужно включить директиву enable_dl в php.ini. На некоторых общих серверах эту возможность отключают.

Функция extension_loaded позволяет проверить, загрузилось ли нужное приложение. На вход нужно подать название расширения, а на выходе получим TRUE или FALSE в зависимости от наличия данного расширения. Отметим, что PHP не может определить, как загрузилось расширение: с помощью dl или оно скомпилировано с PHP.

Загрузка модулей при старте

Если есть подгружаемый модуль и нужно чтобы он загружался в PHP без вызова dl в каждом сценарии, можно воспользоваться директивой extension в php.ini. Она позволяет задать список расширений, которые нужно загрузить при старте.

Каждое расширение задается в отдельной строке. Ограничений на количество расширений нет. В следующих строках из php.ini дается указание автоматически загрузить модули sockets и imap для Linux/Unix сервера:

```
extension=imap.so
extension=sockets.so
```

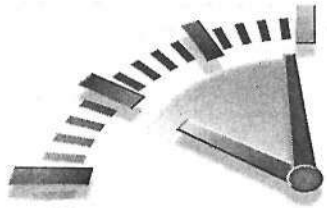
Для Web-сервера под управлением Windows эти инструкции выглядят по-другому:

```
extension=php_imap.dll
extension=php_sockets.dll
```

Резюме

В этом уроке вы узнали о настройке PHP во время работы. В следующем уроке вы узнаете о том как работает безопасный режим и как увеличить безопасность Web-сервера.

Урок 24



Безопасность при использовании PHP

Несомненно, PHP является очень мощным языком написания сценариев для Web-сервера. Но с большими возможностями появляется большая ответственность. В этом уроке вы узнаете, как использовать безопасный режим PHP, чтобы отключить потенциально опасные возможности PHP.

Безопасный режим

Безопасный режим PHP позволяет реализовать базовую безопасность для общего окружения, когда несколько пользователей используют один Web-сервер с поддержкой PHP.

Если PHP работает в безопасном режиме, выполнение некоторых функций ограничивается, а некоторые отключаются вообще.

Ограничения, налагаемые безопасным режимом

Функции для работы с файловой системой имеют ряд ограничений в безопасном режиме. Процессы Web-сервера работают под одним пользовательским ID для всего Web-пространства и должны иметь права на чтение или изменение, чтобы получить доступ к файлу. Это требование операционной системы и не имеет никакого отношения к PHP.

В безопасном режиме при попытке прочитать или записать в локальный файл PHP проверяет, совпадает ли владелец файла и сценария. Если они отличаются, операция запрещается.



Права на запись

Кроме того что безопасный режим в PHP запрещает различным владельцам использовать файлы, нужно обеспечить соответствующие права на уровне операционной системы, чтобы иметь возможность читать или записывать в файлы. Нужно помнить, что пользователь с возможностью работы в командной оболочке на сервере сможет прочитать любой файл, который доступен для Web-сервера, и записать в любой файл, для которого установлены глобальные права на запись.

Следующие функции для работы с файловой системой ограничиваются этим правилом:

```
chdir                   move_uploaded_file
chgrp                   parse_ini_file
chown                   rmdir
copy                    rename
fopen                   require
highlight_file          show_source
include                 symlink
link                    touch
mkdir                   unlink
```

На функции, которые являются частью расширений PHP и работают с файловой системой, также налагаются эти ограничения.



Подгружаемые модули

Функция `dl` не работает в безопасном режиме, независимо от владельца файла расширения. Расширение нужно загружать при старте PHP с помощью соответствующей инструкции в `php.ini`.

Функции, которые выполняют программы сервера, работают только для каталогов, перечисленных в директиве `safe_mode_exec_dir`. Подробнее о ней можно узнать ниже. Даже если установлено право на выполнение, все аргументы командной строки автоматически передаются функции `escapeshellcmd`.

Следующие функции, выполняющие системные программы, ограничиваются этим правилом:

```
exec                    shell_exec
passthru                system
popen
```

Кроме того, отключен оператор апостроф (`).

Функция `putenv` не работает в безопасном режиме. Но если ее выполнить, сообщения об ошибке не выводятся. Другие функции, изменяющие окружение PHP, такие как `set_time_limit` и `set_include_path`, также игнорируются.

Активизация безопасного режима

Безопасный режим включается с помощью директивы `safe_mode` в `php.ini`. Чтобы включить его для всех пользователей на общем Web-сервере, используется следующая инструкция:

```
safe_mode = On
```

Как рассказывалось выше, функции для работы с файловой системой проверяют владельца файла. По умолчанию проверяется соответствие пользовательских ID. Это можно заменить на проверку ID группы (GID) с помощью директивы `safe_mode_gid`.

Если в файловой системе есть общая библиотека, нужно использовать директиву `safe_mode_include_dir`. В ней задается список каталогов, которые не проверяются на соответствие UID/GID при операциях `include` или `require`.



Каталоги для подключения

Если нужно указать несколько каталогов в инструкции `safe_mode_include_dir`, можно разделить их с помощью двоеточия в Linux/Unix и точки с запятой для Windows системы. Это аналогично тому, что делается для директивы `include_path`.

Инструкция ниже разрешает подключать файлы из каталога `/usr/local/include/php` в безопасном режиме:

```
safe_mode_include_dir = /usr/local/include/php
```

Директива `safe_mode_exec_dir` задает места, из которых можно выполнять программы сервера.

Чтобы разрешить выполнение программ из /usr/local/php-bin в безопасном режиме, используется следующая директива:

```
safe_mode_exec_dir = /usr/local/php-bin
```



Возможность выполнения

Не нужно разрешать выполнение всех программ из каталога /usr/bin или другого системного каталога. Лучше создать новый и поместить туда копии или ссылки на необходимые программы.

Директива safe_mode_allowed_env_vars позволяет указать список переменных окружения, которые можно изменять. Заданные значения являются префиксами и по умолчанию это только переменные с PHP_ в начале. Несколько значений разделяются запятой.

В инструкции ниже добавляется разрешение изменять значение часового пояса TZ:

```
safe_mode_allowed_env_vars = PHP_,TZ
```

Другие возможности безопасности

Кроме безопасного режима в PHP есть ряд функций, которые позволяют наложить ограничение на возможности PHP.

Сокрытие PHP

С помощью инструкции expose_php в php.ini можно запретить выводить версию PHP в заголовках Web-сервера:

```
expose_php = On
```

С помощью этой инструкции можно помешать автоматическим сценариям атаковать Web-сервер. Обычно в HTTP-заголовках содержится строка, которая выглядит примерно так:

```
Server: Apache/1.3.33 (Unix) PHP/5.0.3 mod_ssl/2.8.16
OpenSSL/0.9.7c
```


При активации expose_php версия PHP не включается в заголовок.

Конечно, расширение .php явно укажет посетителю, что на Web-сервере используется PHP. Если нужно полностью изменить расширение файлов, нужно найти следующую строку в httpd.conf:

```
AddType application/x-httpd .php
```

Теперь нужно поменять значение .php на любое другое. Можно задать несколько расширений через пробел. Чтобы PHP обрабатывал файлы .html и .htm и, таким образом, скрывал, какой язык используется на Web-сервере, применяется следующая инструкция:

```
AddType application/x-httpd .html .htm
```



Обработка HTML

Возможность обрабатывать HTML-файлы как PHP может быть очень удобной. Но это снижает производительность сервера, в случае статического файла, потому что Web-сервер запускает обработчик PHP. Чтобы этого избежать, для статических страниц нужно использовать другое расширение.

Безопасность файловой системы

Безопасный режим предоставляет доступ к файлу только владельцу. С помощью директивы open_basedir можно задать каталог, в котором должны находиться файлы. Если это сделать, PHP отклоняет все попытки доступа к папке, которая не находится в заданном каталоге или подкаталоге. Инструкция open_basedir работает независимо от безопасного режима.

Чтобы ограничить доступ к файловой системе на Web-сервере только каталогом /tmp, используется следующая инструкция:

```
open_basedir = /tmp
```

Контроль доступа к функциям

С помощью директивы disable_functions можно задать список PHP-функций, разделенных запятой, которые нужно отключить.

Эта установка работает независимо от безопасного режима. Чтобы отключить функцию dl без включения безопасного режима, достаточно использовать инструкцию:

```
disable_functions = dl
```

Таким же образом можно отключить доступ к классам с помощью инструкции disable_classes.

Безопасность базы данных

В уроке 18, "Выполнение программ на Web-сервере", показывалось, как злоумышленник может выполнить нужную команду на сервере; чтобы защититься от этого, следует использовать функцию escapeshellcmd.

Аналогичный механизм работает и для баз данных в PHP. Пусть строки ниже формируют MySQL-запрос на основе значений, полученных из формы:

```
$sql = "UPDATE mytable SET col1 = " . $_POST["value"] . "
       WHERE col2 = 'somevalue'";
$res = mysql_query($sql, $db);
```

Ожидается, что $_POST["value"] будет содержать целое значение для столбца col1. Но злоумышленник может ввести точку с запятой и запрос, который нужно выполнить.

Например, в $_POST["value"] находится такое значение:

```
0; INSERT INTO admin_users (username, password)
VALUES ('me', 'mypassword');
```

Теперь выполняется такой SQL-запрос (для ясности он приводится в отдельных строках):

```
UPDATE mytable SET col1 = 0;
INSERT INTO admin_users (username, password)
VALUES ('me', 'mypassword');
WHERE col2 = 'somevalue';
```

Это очень неприятная ситуация. Первое выражение установит в ноль значение col1 для всех строк в mytable. Это плохо, но второе выражение делает еще хуже — пользователь получает возможность выполнить выражение INSERT, которое создает пользователя с правами администратора.

Третье выражение неправильное, но когда анализатор SQL доберется до него, деструктивные действия уже выполнятся. Такой тип атаки называется *SQL-подстановкой*.

Конечно, для того чтобы нанести ущерб системе, пользователю нужно знать структуру базы данных. В этом примере злоумышленник знает, что в базе есть таблица admin_users с полями username и password, а также то, что пароль не шифруется.

Обычно посетители Web-сайта не знают такой информации о структуре базы, которая строится самостоятельно. Но если на сайте используются компоненты программ с открытым кодом, например бесплатный форум, часть структуры базы может быть известна другим пользователям.

Более того, если сценарий выводит ошибку при неправильном выполнении запроса, это может показать важные детали структуры базы данных. На рабочем Web-сайте нужно установить опцию display_errors в положение off и воспользоваться инструкцией log_errors, чтобы сохранять сообщения об ошибках в файл журнала.

Чтобы избежать SQL-подстановки, нужно убедиться в том, что данные, приходящие от пользователя и подставляющиеся в запрос, не могут оборвать SQL-выражение.

В предыдущем примере показывалось, как обновляется целое значение. Если это строка, заключенная в одинарные кавычки, злоумышленнику нужно отправить закрывающую кавычку перед точкой с запятой, а после нее вставить SQL-выражение.

Но при включении директивы magic_quotes_gpc кавычка, полученная из формы, автоматически разделяется.



Права на базу данных

Очень важно подключаться к базе с правами пользователя, который имеет только необходимые для работы права.

Нельзя соединяться с базой данных через администраторскую учетную запись. Если это сделать, атакующий может получить полный доступ ко всем базам данных на сервере. Кроме того, атакующий может выполнить команды GRANT или CREATE USER и открыть себе полный доступ за пределами сценария.

Для защиты от SQL-подстановок из данных формы нужно также проверять соответствие формата полученных данных. Если в запросе ожидается числовое значение, нужно выполнить проверку с помощью функции is_numeric или использовать функцию settype, чтобы изменить строку на число. Это удаляет все символы, которые могут обмануть SQL.

Если работать с несколькими значениями формы в одном SQL-выражении, лучше использовать функцию sprintf. Это позволяет составить SQL-выражение при помощи символов, которые задают тип данных для каждого значения:

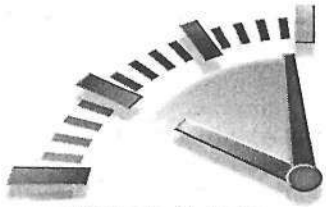
```
$sql = sprintf("UPDATE mytable SET col1 = %d
                WHERE col2 = '%s'",
                $_POST["number"],
                mysql_escape_string($_POST["string"]));
```

Пример выше написан для базы данных MySQL, поэтому строка передается функции mysql_escape_string. Для других баз данных нужно правильно разделить символ кавычки с помощью функции addslashes или другого подходящего способа.

Резюме

В этом уроке вы узнали, как обеспечить безопасность динамических страниц с использованием PHP. В следующем уроке вы узнаете о PEAR — базовом источнике библиотек PHP от сторонних разработчиков.

Урок 25



Использование PEAR

В этом уроке вы узнаете о библиотеке расширений и приложений PHP — PEAR (PHP Extension and Application Repository) .

Введение в PEAR

PEAR — это набор компонент и система распространения повторно используемых PHP-пакетов. В PEAR входят следующие компоненты:

- структурированная библиотека открытых кодов для PHP-разработчиков;
- система распространения и сборки кода в пакеты;
- стандарт кодирования PEAR (PCS);
- фундаментальные классы PHP (PFC);
- поддержка сообщества пользователей PEAR через Web-сайт и списки рассылки.

Библиотека кодов PEAR

PEAR вобрала в себя много различных открытых проектов. Каждый из них составляет отдельный пакет. У всех пакетов PEAR есть составители и разработчики. Они определяют список необходимых изменений и даты выхода финальных версий. Но структура пакета согласовывается для всех проектов PEAR.

С PHP поставляется инсталлятор PEAR. Он позволяет автоматически загрузить и устанавливать пакеты PEAR, указав имя пакета. О том, как работать с инсталлятором PEAR, рассказывается далее в этом уроке.

Пакет может зависеть от других пакетов PEAR. Это отдельно оговаривается в документации, даже если из-за названия кажется, что пакеты связаны.

Пакеты в PEAR имеют древовидную структуру. Каждый уровень иерархии отделяется символом подчеркивания (_). Например, пакет HTTP состоит из различных утилит. В свою очередь, пакет HTTP_Header работает конкретно с HTTP-заголовками.

Распространение и сборка пакетов

Все пакеты PEAR регистрируются в центральной базе данных на сайте http://pear.php.net. На Web-сайте PEAR есть удобный интерфейс для поиска в базе данных. Искать можно по имени пакета, категории и дате выпуска.

Сборщики пакетов PEAR управляют своими проектами с помощью Web-сайта PEAR. CVS-сервер позволяет нескольким разработчикам работать совместно над одним исходным кодом. Последнюю версию пакета всегда можно загрузить с этой центральной точки.

Стандарты кодирования PEAR

Открытые проекты, которые могут использоваться всем сообществом PHP, разрабатываются различными командами программистов. Чтобы облегчить понимание исходного кода, был разработан стандарт кодирования PCS.

Документация по PCS определяет стиль и структуру кода. Его должны придерживаться все разработчики при написании пакета для PEAR-проекта.

Стандарт достаточно детальный и описывает большинство правил стиля. В нем есть соглашение об именовании переменных, а также согласованный стиль для определения функций и классов.

Это может поначалу отпугнуть, но при усложнении сценария сразу заметны преимущества хорошего стиля кодирования. После этого уже не нужно принуждать себя разрабатывать чистый код. Документация по PCS просто задает ряд правил для написания читабельного PHP-кода.

Описание PCS можно найти по адресу: http://pear.php.net/manual/en/standards.php.

Важнейшие классы PHP

PFC является подмножеством пакетов PEAR. Для того чтобы войти в него, класс должен соответствовать набору таких критериев.

- **Качество** — пакет должен быть в стабильном состоянии.
- **Обобщенность** — пакеты не должны привязываться к определенному типу окружения.
- **Возможность взаимодействия** — пакеты должны нормально функционировать на различных системах и иметь стандартизированное API.
- **Совместимость** — пакеты должны проектироваться с обратной совместимостью при добавлении новых возможностей.

В данный момент только инсталлятор PEAR поставляется с PHP. Но позже некоторые классы могут войти в стандартную поставку. Скорее всего это будут классы PFC.

Оперативная поддержка по PEAR

На Web-сайте PEAR по адресу http://pear.php.net есть отличная электронная документация по проекту PEAR и удобный поиск через Web-интерфейс. Разработчики пакетов могут авторизироваться и обновить информацию о проекте.

Есть множество почтовых рассылок пользователей PEAR, сборщиков, основных разработчиков и Web-мастеров. К ним можно присоединиться с помощью формы на странице: http://pear.php.net/support/lists.php.

Использование PEAR

В следующем разделе рассказывается о том, как с помощью PEAR найти и установить пакет в системе. Кроме того описывается, как отправить проект для включения в PEAR.

Поиск пакета PEAR

На каждой странице Web-сайта PEAR есть поисковое окошко, с помощью которого можно найти нужный пакет в базе данных. Нужно просто ввести название или его часть, и все подходящие пакеты покажутся на экране.



Поиск пакетов

Чтобы выполнить расширенный поиск по имени пакета, разработчика или дате выпуска, следует воспользоваться формой на странице: `http://pear.php.net/package-search.php`.

Нужный пакет можно выбрать из результатов поиска. После этого показывается страница с ключевой информацией. На ней описываются все возможности, номер текущей версии и состояние, а также информация о зависимостях. Последняя приводится, если для работы PEAR-пакета требуются другие пакеты.

Закладки в верхней части страницы с описанием пакета содержат ссылку на документацию. Если в резюмирующей информации плохо описаны возможности пакета, можно обратиться к страницам документации.

На странице `http://pear.php.net/packages.php` можно просмотреть все пакеты PEAR, разбитые для удобства на категории.

Использование инсталлятора PEAR

Чтобы загрузить пакет, нужно найти соответствующую закладку в верхней части страницы с описанием пакета. Но лучше воспользоваться инсталлятором PEAR. Он позволяет легко управлять набором пакетов PEAR. Инсталлятор может найти и загрузить последнюю версию пакета, а после этого автоматически его установить.

Инсталлятор PEAR называется `pear`. Для его запуска нужно набрать `pear` в командной строке и добавить нужные аргументы. Опция `list` позволяет увидеть все установленные пакеты:

```
$ pear list
```



Опции команды

Если запустить `pear` без аргументов, на экране появится список доступных опций.

На выходе получим похожий вывод:

```
Installed packages:
===================
Package          Version State
DB               1.6.2   stable
HTTP             1.2.2   stable
Net_DNS          1.00b2  beta
Net_SMTP         1.2.6   stable
Net_Socket       1.0.1   stable
PEAR             1.3.2   stable
SQLite           1.0.2   stable
```

Выводится название пакета, номер версии и статус. Реальный список установленных пакетов может отличаться от показанного выше.

Команда `search` позволяет искать по базе данных PEAR. Чтобы найти все пакеты, содержащие строку `mail`, нужно ввести команду:

```
# pear search mail
```

На выходе получим все соответствующие пакеты, их последние версии и короткое резюме. Поиск не зависит от регистра. Команда `list-all` позволяет увидеть все стабильные пакеты PEAR:

```
# pear list-all
```

На выходе будет длинный список!

Чтобы загрузить и установить пакет, используется команда `install` с именем пакета. Для установки пакета `Mail_Queue` нужно ввести следующую команду:

```
# pear install Mail_Queue
```

Для нормальной работы некоторых пакетов нужны дополнительные пакеты. Поэтому установка может прерваться, если они не будут найдены. Ниже показывается вывод попытки установить `Mail_Queue` без установленного пакета `Mail`:

```
# pear install Mail_Queue
downloading Mail_Queue-1.1.3.tar ...
Starting to download Mail_Queue-1.1.3.tar (-1 bytes)
.....done: 98,816 bytes
requires package `Mail'
Mail_Queue: Dependencies failed
```

Некоторые зависимости опциональные. Если установить пакет `Mail`, чтобы устранить ошибку зависимости в предыдущем сообщении, PEAR предложит улучшить функцио-

нальность пакета Mail с помощью установки пакета Net_SMTP:

```
# pear install Mail
downloading Mail-1.1.4.tar ...
Starting to download Mail-1.1.4.tar (-1 bytes)
.....done: 73,728 bytes
Optional dependencies:
package `Net_SMTP' version >= 1.1.0 is recommended to
utilize some features.
install ok: Mail 1.1.4
```

Команда upgrade загружает и обновляет последнюю версию установленного пакета. Чтобы проверить наличие новой версии пакета Mail, используется следующая команда:

```
# pear upgrade Mail
```

При обнаружении новой версии происходит автоматическое обновление.



Обновление пакетов

Команда upgrade-all обновляет все установленные пакеты PEAR до последней версии.

Если нужно полностью удалить пакет PEAR, используется команда uninstall.

Как сделать собственный вклад в проект PEAR

Тот, у кого есть собственный, полезный другим PHP-проект, может отправить предложение включить его в PEAR.

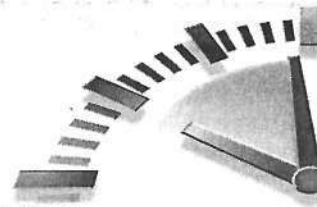
В электронной документации (по адресу http://pear.php.net/manual/en/guide-newmaint.php) есть инструкция для создателей проектов. В ней описываются требования к проекту и стилю кодирования.

Нужно сначала прочитать эту инструкцию, а затем писать пакет, подходящий для других разработчиков. Даже если проект не соответствует требованиям PEAR, в справочнике можно найти идеи по улучшению дизайна и внедрению стандартов кодирования. Это поможет улучшить качество пакета.

Резюме

В этом уроке вы научились работать с PEAR. Окончив чтение этой книги, вы получили все преимущества от использования множества свободно доступных классов PEAR. Это позволяет решать широкий класс задач с помощью PHP. Удачного кодирования!

Приложение А



Установка PHP

Это приложение предназначено для тех, кто хочет самостоятельно установить PHP. Здесь приводится пошаговая инструкция установки для Linux/Unix и Windows платформ.

Установка на Linux/Unix

Эта инструкция рассказывает об установке PHP из исходных кодов для Web-сервера Apache. Чтобы провести установку, нужно получить права пользователя `root`. Для этого запускается команда `su` и вводится пароль суперпользователя.

Сборка Apache из исходных кодов

Если Web-сервер Apache уже установлен с поддержкой динамически разделяемых объектов (DSO), можно пропустить этот раздел. Команда, приведенная ниже, определяет наличие этой возможности на Web-сервере:

```
$ httpd -l
```

Если в выводе есть строка `mod_so.c`, то поддержка DSO активирована. Отметим, что нужно указать полный путь к `httpd` (например, `/usr/local/apache/bin/httpd`).

Начнем с загрузки последней версии исходных кодов Apache с `http://httpd.apache.org`. На время написания последней версией была 2.0.52. Поэтому нужно загрузить файл `httpd-2.0.52.tar.bz2`. Если доступна новая версия, нужно подставить правильный номер, чтобы ее загрузить.

Этот файл нужно сохранить в каталоге /usr/local/src или другом месте для хранения исходных кодов. Архив распаковывается командой bunzip2:

```
# bunzip2 httpd-2.0.52.tar.bz2
```



Распаковка

Если в системе нет программы bunzip2, нужно загрузить файл httpd-2.0.52.tar.gz. Этот архив занимает больше места, но его распаковщик gzip есть в большинстве систем.

После этого расширение файла .bz2 исчезнет. Архив разворачивается командой tar:

```
# tar xvf httpd-2.0.52.tar
```

Файлы извлекаются в каталог httpd-2.0.52. Нужно перейти в него, перед тем как продолжить:

```
# cd httpd-2.0.52
```

Теперь нужно выполнить команду configure с необходимыми параметрами. Например, чтобы изменить каталог для установки, используется опция -prefix и имя каталога после нее. Команда configure -help позволяет увидеть список всех опций.

Нужно добавить опцию --enable-module=so, чтобы установить поддержку DSO. Это позволит загрузить модуль PHP. Нужно ввести следующую команду и необходимые опции, которые нужно включить:

```
# ./configure --enable-module=so
```

Команда configure создает несколько полных экранов вывода при поиске лучших настроек компиляции для системы. После окончания снова появляется командная строка оболочки, и можно продолжить установку.

Команда make запускает процесс компиляции:

```
# make
```

Снова на экране появляется множество различных строк. Время компиляции зависит от скорости Web-сервера. После завершения сборки появляется такая надпись и командная строка:

```
make[1]: Leaving directory `/usr/local/src/httpd-2.0.52'
```


Теперь нужно установить собранную программу. С помощью команды make install файлы автоматически копируются в нужную системную область:

```
# make install
```

С помощью команды apachectl start нужно запустить Web-сервер Apache. После этого нужно ввести в адресной строке браузера IP-адрес сервера, чтобы убедиться в успешной инсталляции. Если стандартное место установки не менялось, команда ниже запускает Web-сервер:

```
# /usr/local/apache/bin/apachectl start
```

Компиляция и установка PHP

Последнюю версию PHP можно загрузить с адреса: www.php.net/downloads.php. На время написания этой книги последней версией была 5.0.3. Поэтому нужно загрузить файл с именем php-5.0.3.tar.bz2. Если есть более новая версия, возьмите файл с ее номером.

Этот файл нужно сохранить в каталоге /usr/local/src или в другом месте для хранения исходных кодов. Архив распаковывается командой bunzip2:

```
# bunzip2 php-5.0.3.tar.bz2
```

После распаковки у файла исчезает расширение .bz2. Командой tar нужно извлечь информацию из распакованного файла:

```
# tar xvf php-5.0.3.tar
```

Файлы извлекаются в каталог php-5.0.3. Перейдем в этот каталог, чтобы продолжить работу:

```
# cd php-5.0.3
```

Теперь нужно выполнить команду configure с требуемыми опциями. Например, для включения поддержки MySQL через расширение MySQLi используется опция --with-mysqli. После нее указывается путь к программе mysql_config. Полный список опций configure выводит строка configure --help.

Также нужно добавить опции --with-apxs или --with-apxs2 (для Apache 2.0) и указать место, где находится программа apxs. Вот пример настроек для стандартной установки Apache:

```
# ./configure --with-apxs2=/usr/local/apache2/bin/apxs
# ./configure --with-apxs2=/usr/local/apache/bin/apxs
```

Команда configure создает несколько полных экранов вывода во время выбора оптимальных настроек компиляции для данной системы. После окончания появляется командная строка, и можно продолжить установку.

Команда make запускает компиляцию:

```
# make
```

Снова получаем много экранов вывода. Продолжительность процесса зависит от быстродействия системы. После окончания сборки на экране появится следующая надпись и командная строка оболочки:

```
Build complete.
(It is safe to ignore warnings about tempnam and tmpnam).
```

Последний шаг — установить только что собранный модуль PHP на Web-сервере. Для этого нужно ввести make install, и файлы автоматически скопируются в нужную системную область:

```
# make install
```

Для завершения инсталляции нужно изменить конфигурационный файл Web-сервера. Нужно настроить, чтобы файлы .php обрабатывались модулем PHP. В файле httpd.conf нужно добавить такую строку:

```
AddType application/x-httpd-php .php
```

При необходимости можно указать альтернативные расширения в дополнение к .php.

Теперь, если перезапустить Web-сервер с помощью apachectl restart, загрузится расширение PHP. Чтобы проверить PHP, можно создать простой сценарий /usr/local/apache2/htdocs/index.php примерно такого вида:

```
<?php
phpinfo();
?>
```

Чтобы зайти на index.php, нужно ввести в Web-браузере IP-адрес сервера. При этом на экране появится подробная информация о конфигурации PHP.

Установка на платформе Windows

Инструкции в этом разделе показывают, как установить PHP для Web-сервера Apache на платформе Windows.

Установка Apache

Если Apache уже установлен, можно пропустить этот раздел.

Последнюю версию Apache можно загрузить с адреса: httpd.apache.org. Это пакет-инсталлятор MSI под названием apache_2.0.52-win32-x86-no_ssl.msi для текущей версии Apache 2.0.52. Сохраните его на рабочем столе и запустите двойным щелчком. Это запустит процесс установки.

Установка проходит при содействии мастера и не вызывает сложностей.

Нужно принять лицензионное соглашение и продолжить установку, после того как появятся сведения о версии. Нажмите Next после прочтения и введите информацию о Web-сервере.

Нужно ввести имя домена Web-сервера, название и адрес электронной почты. При установке на персональный компьютер нужно указать localhost и localdomain в информации о сервере. Следует оставить рекомендуемый порт 80 для работы Apache.

После этого необходимо выбрать стандартный тип установки.

Теперь нужно выбрать место установки файлов Apache. По умолчанию это C:\Program Files\Apache Group. Apache готов к установке, и можно щелкнуть на кнопке Install, чтобы система начала копировать нужные файлы и настраивать их на сервере.

По завершению установки запускается сервер Apache и программа мониторинга. Эти пиктограммы находятся в панели задач. Если дважды щелкнуть на значке, появится программа Apache Service Monitor. С ее помощью можно запускать и останавливать процессы Web-сервера. Зеленый свет свидетельствует о работе сервера.

Установка PHP

Последнюю версию PHP можно загрузить из раздела для Windows с адреса: `www.php.net/downloads.php`. Нужно выбрать файл с расширением .zip, а не установочный пакет. Он называется `php-5.0.3-Win32.zip` для последней версии PHP, которая на время написания книги была 5.0.3. Если появится более новая, нужно загрузить ее, выбрав соответствующий файл.

Zip-файл нужно сохранить на рабочем столе и извлечь содержимое в `C:\php`. Можно выбрать другое место, но тогда нужно внести соответствующие изменения в инструкцию ниже.

Теперь нужно подключить модуль PHP к Apache. С помощью Проводника нужно открыть конфигурационный каталог Apache (для стандартной установки это `C:\Program Files\Apache Group\Apache2\conf`) и отредактировать файл `httpd.conf`, добавив следующую строку в конец файла:

```
LoadModule php5_module c:/php/php5apache2.dll
AddType application/x-httpd-php .php
```

После перезапуска Web-сервера при помощи программы мониторинга Apache загрузится модуль PHP. Чтобы проверить PHP, в каталоге `htdocs` нужно создать простой сценарий `index.php` с таким содержимым:

```
<?php
phpinfo();
?>
```

Теперь нужно зайти на `http://localhost/index.php`, чтобы увидеть всю информацию о конфигурации PHP.

Решение проблем

Если возникли проблемы с установкой, сначала нужно убедиться, что вы точно следовали всем шагам инструкции выше. Если проблемы не исчезли, обратитесь к следующим Web-сайтам, на которых можно найти помощь:

- `http://httpd.apache.org/docs-2.0/faq/support.html`
- `www.php.net/manual/en/faq.build.php`

Предметный указатель

A

addslashes, *232*
alnum, *83*
alpha, *83*
and, *38*
Apache, *241*
array, *72*
array_diff, *77*
array_intersect, *77*
array_key_exists, *78*
array_merge, *77*
array_reverse, *76*
array_search, *77*
array_unique, *77*
arsort, *76*
ASCII, *63*
asort, *76*
asp_tags, *219*
AuthType, *143*
AuthUserFile, *144*
auto_append_file, *221*
auto_prepend_file, *221*

B

basename, *164*
Bash, *200*
boolean, *31*
Bourne Again Shell, *200*
Bourne Shell, *200*

C

Cache-Control, *156*
case, *40*
ceil, *57*
CGI, *199*
chdir, *170*
CHECKBOX, *109*
chop, *173*
class, *101*
CLI, *199*
closedir, *169*
continue, *43*
Cookies, *135*
copy, *164*
count, *77*
crypt, *149*
CSV-формат, *168*

D

date, *21*, *93*
DELETE, *193*
digit, *83*
DIRECTORY_SEPARATOR, *175*
dirname, *164*
disable_classes, *230*
disable_functions, *230*
display_errors, *209*, *222*
dl, *223*
do, *42*
double, *31*

E

E_ALL, *208*
E_COMPILE_ERROR, *208*
E_COMPILE_WARNING, *208*
E_CORE_ERROR, *208*
E_CORE_WARNING, *208*
E_ERROR, *207*
E_NOTICE, *208*
E_PARSE, *207*
E_STRICT, *208*
E_USER_ERROR, *208*
E_USER_NOTICE, *208*
E_USER_WARNING, *208*
E_WARNING, *207*
echo, *21*, *23*
elseif, *39*
Email адрес, *87*
enable_dl, *224*
ereg, *82*
ereg_replace, *89*
error_log, *212*
error_reporting, *207*, *222*
escapeshellcmd, *178*, *230*
exec, *173*

exit, *154*
Expires, *157*
explode, *96*, *168*
expose_php, *228*
extension, *224*
extension_loaded, *224*

F

FALSE, *31*, *36*
fclose, *167*
fgetcsv, *168*
fgets, *166*
file_get_contents, *165*
file_put_contents, *165*
fileatime, *163*
filectime, *163*
filegroup, *163*
fileinode, *163*
filemtime, *163*
fileowner, *163*
fileperms, *163*
filesize, *163*
filetype, *163*
float, *31*
floor, *57*
fopen, *165*
for, *42*
foreach, *74*
fputcsv, *169*
fputs, *168*
fread, *166*
fseek, *167*
ftell, *167*
function, *46*
fwrite, *168*

G

GET, *107*
getcwd, *170*
getdate, *96*
gettype, *31*
global, *51*
gmmktime, *95*
GMT, *95*
GTK+, *206*

H

header, *130*, *153*
headers_list, *156*
headers_sent, *155*
HIDDEN, *114*
htmlentities, *123*
Htpasswd, *144*
HTTP_USER_AGENT, *159*
HTTP_X_FORWARDED_FOR, *159*
HTTP-заголовок, *136*, *153*

I

if, *35*
in_array, *77*
include, *51*
include_once, *52*
include_path, *52*, *221*
ini_set, *218*
INSERT, *193*
integer, *31*
is_array, *130*
is_executable, *163*
is_file, *163*
is_float, *56*
is_int, *56*
is_link, *163*
is_numeric, *56*
is_readable, *163*
is_writeable, *163*
isset, *112*

J

JavaScript, *20*

K

krsort, *76*
ksort, *76*

L

Last-Modified, *157*
localhost, *180*, *245*
Location, *130*, *154*
log_errors, *209*, *222*
lower, *84*

M

magic_quotes, *220*
magic_quotes_gpc, *220*, *231*
magic_quotes_runtime, *220*
magic_quotes_sybase, *220*
mail, *48*, *115*
max, *58*
max_execution_time, *217*, *219*
memory_get_usage, *219*
memory_limit, *218*, *219*
min, *58*
mktime, *94*
mt_rand, *59*
mt_srand, *59*
MySQL, *179*
mysql_affected_rows, *182*
mysql_close, *181*
mysql_connect, *180*
mysql_data_seek, *184*
mysql_errno, *185*
mysql_error, *185*
mysql_escape_string, *232*
mysql_fetch_array, *184*
mysql_num_rows, *183*
mysql_query, *181*
mysql_result, *182*
mysql_select_db, *180*
mysqli, *180*

N

new, *101*
NULL, *57*

O

open_basedir, *229*
opendir, *169*
or, *38*

P

passthru, *171*
PASSWORD, *148*
PATH_SEPARATOR, *175*
PCRE, *81*
PCS, *233*
PEAR, *233*
PEAR DB, *189*
PFC, *233*
PHP, *19*
php.ini, *215*
PHP_OS, *174*
php_sapi_name, *200*
PHP-GTK, *206*
phpinfo, *47*
POSIX-синтаксис, *81*
POST, *107*
PostgreSQL, *192*
print, *84*
print_r, *72*
printf, *64*
punct, *84*
putenv, *175*, *227*

R

RADIO, *109*
rand, *59*
RAND_MAX, *59*
readdir, *169*
realpath, *164*
register_globals, *220*
register_long_arrays, *221*
REMOTE_ADDR, *159*
rename, *164*
REQUEST_URI, *158*
require, *52*
Require group, *144*
Require user, *144*
require valid-user, *144*
require_once, *52*
rewind, *167*
rewinddir, *169*
round, *58*
rsort, *76*
rtrim, *167*

S

safe_mode, *227*
safe_mode_allowed_env_vars, *228*
safe_mode_exec_dir, *226*, *227*
safe_mode_gid, *227*
safe_mode_include_dir, *227*
SELECT, *193*
serialize, *78*
SERVER_ADDR, *160*
SERVER_PORT, *160*

session_cache_limiter, *157*
session_start, *140*
set_error_handler, *210*
set_include_path, *227*
set_time_limit, *227*
setcookie, *138*
Set-Cookie, *154*
settype, *31*
Sh, *200*
short_open_tag, *219*
shuffle, *76*
sort, *75*
space, *84*
sprintf, *64*, *67*
SQL-запрос, *181*
SQL-подстановка, *231*
srand, *59*
SSL-соединение, *138*
stderr, *172*
STDERR, *205*
STDIN, *205*
STDOUT, *205*
string, *31*
strlen, *69*
strpos, *69*
strtolower, *68*
strtotime, *96*
strtoupper, *68*, *205*
SUBMIT, *109*
substr, *68*
switch, *40*

T

tempnam, *164*
TEXT, *110*
time, *92*
trigger_error, *208*, *211*
TRUE, *31*, *36*

U

ucfirst, *68*
ucwords, *68*
unlink, *164*
unserialize, *78*
UPDATE, *193*
upper, *84*

V

variables_order, *220*

W

Web-сервер Apache, *241*
while, *41*

X

xor, *38*
X-Powered-By, *154*

А

Апостроф
  `, *61*, *172*
Аргументы командной строки, *203*
Аргументы функции, *47*
Ассоциативный массив, *74*
Атрибут
  ACTION, *107*
  CHECKED, *109*, *118*
  MAXLENGTH, *108*
  METHOD, *107*
  MULTIPLE, *124*
  NAME, *109*
  SELECTED, *110*, *121*
  TYPE, *108*
  класса, *100*
Аутентификация с помощью сеансов, *145*

Б

Базовая HTTP-аутентификация, *143*
Безопасное дневное время, *95*
Безопасный режим, *225*
Бесконечный цикл, *41*
Библиотека открытых кодов, *233*
Библиотечный файл, *51*
Булевы значения, *36*

В

Возвращаемые значения функции, *47*

Время жизни, *138*
Выполнение запросов, *193*
Выражение, *29*

Г

Глобальные переменные, *50*

Д

Двухмерный массив, *79*
Дескриптор
  ?>, *21*
  <%, *21*
  <?, *21*, *22*
  <?php, *21*, *22*
  <FORM>, *107*
  <INPUT>, *108*, *110*, *148*
  <OPTION>, *121*
  <PRE>, *66*, *73*
  <SELECT>, *110*
  <TEXTAREA>, *110*
  PHP, *22*
  SCRIPT, *21*
  в стиле ASP, *21*
  короткий, *21*
Директива
  asp_tags, *219*
  auto_append_file, *221*
  auto_prepend_file, *221*
  disable_classes, *230*
  disable_functions, *230*
  display_errors, *222*
  enable_dl, *224*
  error_reporting, *222*
  expose_php, *228*
  extension, *224*
  include_path, *52*, *221*
  log_errors, *222*
  magic_quotes, *220*
  magic_quotes_gpc, *220*, *231*
  magic_quotes_runtime, *220*
  magic_quotes_sybase, *220*
  max_execution_time, *217*, *219*
  memory_limit, *218*
  open_basedir, *229*
  php_value, *217*
  register_globals, *220*
  register_long_arrays, *221*
  safe_mode, *227*
  safe_mode_allowed_env_vars, *228*
  safe_mode_exec_dir, *226*, *227*
  safe_mode_gid, *227*
  safe_mode_include_dir, *227*
  short_open_tag, *219*
  variables_order, *220*
  настройки, *218*

З

Зависимость от регистра, *28*
Заголовок
  Cache-Control, *156*
  Expires, *157*
  Last-Modified, *157*
  Location, *154*
  no-cache, *129*
  Set-Cookie, *136*, *154*
  X-Powered-By, *154*
Закрытый метод, *100*
Запуск Apache, *243*

И

Извлечение выбранных данных, *193*
Именование переменных, *28*
Индекс, *71*
Интерфейс командной строки, *199*

К

Кавычки
  двойные, *29*
  одинарные, *29*
Класс, *100*
Класс DB, *189*
Класс символов, *83*
Ключ, *71*
Кнопка submit, *114*
Код возврата, *173*
Команда
  apachectl start, *243*
  bunzip2, *242*
  chmod, *161*
  chown, *162*
  configure, *242*
  finger, *177*

gzip, *242*
hostname, *171*
make, *242*
su, *241*
Командная строка, *199*
Комбинированные операторы, *55*
Комментарий, *25*
#, *25*
*/, *25*
/*, *25*
//, *25*
Конкатенация, *62*
Константа
DIRECTORY_SEPARATOR, *175*
PATH_SEPARATOR, *175*
PHP_OS, *174*
RAND_MAX, *59*
Конструктор, *102*
Круглые скобки, *56*

Л

Логические операторы, *37*
Локальные переменные, *50*
Локальный запуск PHP, *23*

М

Массив, *71*
Метод, *100*
Множественное условное ветвление, *38*
Модульность, *46*

Н

Наследование, *100*
Настройки ведения журнала, *215*
Необязательные аргументы, *50*
Необязательные аргументы функции, *49*

О

Область видимости переменных, *50*
Обрыв выполнения цикла, *43*
Объектно-ориентированное PHP, *99*
Объявление переменной, *27*
Ограничение запроса, *198*
Округление чисел, *57*
Окружение сервера, *174*
Оператор, *29*
!, *38*
!=, *37*
!==, *37*
#!, *201*
%, *54*, *64*
%=, *55*
&&, *38*
*, *53*, *85*
*=, *55*
., *85*
., *164*
.., *164*
.=, *62*
/=, *55*
@, *213*
||, *38*
+, *29*, *53*, *85*
++, *41*, *54*
+=, *55*
<, *35*, *37*
<=, *37*
=, *37*
-=, *55*
==, *37*, *63*
===, *37*, *57*
>, *37*
->, *102*
>=, *37*
break, *40*
case, *40*
continue, *43*
default, *41*
elseif, *39*
exit, *154*
foreach, *74*
global, *51*
if, *35*
include, *51*
include_once, *52*
new, *101*
require, *52*
require_once, *52*
switch, *40*
вычитания, *53*
декремента, *54*

деления, *53*
инкремента, *54*
конкатенации, *30*, *62*
остатка от деления, *54*
подавления ошибки, *213*
сравнения, *37*
суммирования, *29*, *53*
тройное равенство, *57*
умножения, *53*
Определение функции, *46*
Открытый метод, *100*
Отступы, *36*, *43*
Ошибка соединения, *186*, *193*

П

Пакет
HTTP, *234*
HTTP_Header, *234*
Mail, *237*
Mail_Queue, *237*
Net_SMTP, *238*
Переключатель, *109*
Переменная, *27*
Переменная переменной, *33*
Переменные окружения, *174*
Поддомен, *137*
Поиск и замена, *89*
Последовательности, *197*
Поток ошибок, *172*
Потоки ввода-вывода, *204*
Почтовый индекс, *86*
Приведение типа, *32*
Приоритет операторов, *55*
Прототипы, *46*

Р

Работа с кавычками, *197*
Разделитель выражений, *21*
Раскрывающийся список, *110*
Регулярное выражение, *81*
Режимы извлечения, *195*

С

Свойство класса, *100*
Символ
/, *53*
?, *86*
\\n, *25*, *62*
\\t, *62*
^, *83*, *84*
доллар, *27*
перевода строки, *24*
подчеркивание, *29*
Символы
[: :], *83*
Скалярные переменные, *71*
Скобки
(, *36*
), *36*
[, *36*
], *36*
{, *30*
}, *30*
квадратные, *36*
круглые, *36*
фигурные, *36*
Событие
onChange, *124*
Совместимость между базами данных, *195*
Соответствие набору символов, *82*
Сортировка массива, *75*
Спецификатор формата, *64*
Стандартный поток вывода, *172*
Строка, *61*
Суперглобальные массивы, *112*
Сценарий оболочки, *200*, *202*

Т

Табуляция, *62*
Телефонный номер, *87*
Тело функции, *47*
Тип
boolean (булев), *31*
double (двойной точности с плавающей точкой), *31*
float (одинарной точности с плавающей точкой), *31*
integer (целый), *31*
string (строчный), *31*
Типы данных, *31*
Точка с запятой, *21*

У

Уровень уведомления об ошибках, *207*
Условный оператор, *35*

Ф

Файл
  .htaccess, *143*, *209*, *217*
  httpd.conf, *229*, *244*, *246*
  php.ini, *209*, *215*, *227*
  php_sockets.dll, *224*
  php-apache.ini, *217*
  php-cli.ini, *217*
  sockets.so, *224*
Файлы cookies, *135*
Фигурные скобки, *30*, *33*, *36*
Флаг secure, *138*
Флаг чтения, *161*
Формат временной метки, *92*
Форматирование строк, *64*
Функция, *45*
  addslashes, *232*
  array, *72*
  array_diff, *77*
  array_intersect, *77*
  array_key_exists, *78*
  array_merge, *77*
  array_reverse, *76*
  array_search, *77*
  array_unique, *77*
  arsort, *76*
  asort, *76*
  basename, *164*
  ceil, *57*
  chdir, *170*
  chop, *173*
  closedir, *169*
  copy, *164*
  count, *77*
  crypt, *149*
  date, *21*, *93*
  dirname, *164*
  dl, *223*
  echo, *21*, *23*
  ereg, *82*
  ereg_replace, *89*
  error_log, *212*
  error_reporting, *207*
  escapeshellcmd, *178*, *230*
  exec, *173*
  explode, *96*, *168*
  extension_loaded, *224*
  fclose, *167*
  fgetcsv, *168*
  fgets, *166*
  file_exists, *162*
  file_get_contents, *165*
  file_put_contents, *165*
  fileatime, *163*
  filegroup, *163*
  fileinode, *163*
  filemtime, *163*
  fileowner, *163*
  fileperms, *163*
  filesize, *163*
  filetype, *163*
  floor, *57*
  fopen, *165*
  fputcsv, *169*
  fputs, *168*
  fread, *166*
  fseek, *167*
  ftell, *167*
  fwrite, *168*
  getcwd, *170*
  getdate, *96*
  gettype, *31*
  gmmktime, *95*
  header, *130*, *153*
  headers_list, *156*
  headers_sent, *155*
  htmlentities, *123*
  in_array, *77*
  ini_set, *218*
  is_array, *130*
  is_executable, *163*
  is_file, *163*
  is_float, *56*
  is_int, *56*
  is_link, *163*
  is_numeric, *56*
  is_readable, *163*
  is_writeable, *163*
  isset, *112*
  krsort, *76*
  ksort, *76*
  mail, *48*, *115*
  max, *58*

memory_get_usage, *219*
memory_limit, *219*
min, *58*
mktime, *94*
mt_rand, *59*
mt_srand, *59*
mysql_affected_rows, *182*
mysql_close, *181*
mysql_connect, *180*
mysql_data_seek, *184*
mysql_errno, *185*
mysql_error, *185*
mysql_escape_string, *232*
mysql_fetch_array, *184*
mysql_num_rows, *183*
mysql_query, *181*
mysql_result, *182*
mysql_select_db, *180*
opendir, *169*
passthru, *171*
php_sapi_name, *200*
phpinfo, *47*
print_r, *72*
printf, *64*
putenv, *175*, *227*
rand, *59*
readdir, *169*
realpath, *164*
rename, *164*
rewind, *167*
rewinddir, *169*
round, *58*
rsort, *76*
rtrim, *167*
serialize, *78*
session_cache_limiter, *157*
session_start, *140*
set_error_handler, *210*
set_include_path, *227*
set_time_limit, *227*
setcookie, *138*
settype, *31*
shuffle, *76*
sort, *75*
sprintf, *64*, *67*
srand, *59*
strlen, *69*
strpos, *69*
strtolower, *68*
strtotime, *96*
strtoupper, *68*, *205*
substr, *68*
tempnam, *164*
time, *92*
trigger_error, *208*, *211*
ucfirst, *68*
ucwords, *68*
unlink, *164*
unserialize, *78*

Ц

Цикл
  do, *42*
  for, *42*
  while, *41*

Ч

Часовой пояс, *176*

Э

Электронный справочник, *45*
Эпоха Unix, *92*

Научно-популярное издание

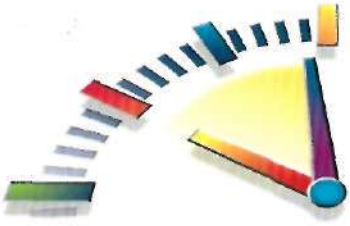
Крис Ньюман
Освой самостоятельно PHP 10 минут на урок

Литературный редактор *Ж.Е. Прусакова*
Верстка *А.Н. Полинчик*
Художественный редактор *В.Г. Павлютин*
Корректор *В.В. Смоляр*

Издательский дом "Вильямс"
101509, г. Москва, ул. Лесная, д. 43, стр. 1

Подписано в печать 14.11.2005. Формат 84x108/32.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 14,3. Уч.-изд. л. 9,4.
Тираж 5000 экз. Заказ N 99.16.

Отпечатано с готовых диапозитивов
в ООО "СЕВЕРО-ЗАПАДНЫЙ ПЕЧАТНЫЙ ДВОР"
188350, Ленинградская обл., г.Гатчина, ул.Солодухина, д. 2



PHP

10 минут на урок

Научись:

- Использовать основные операторы и конструкции PHP
- Выполнять операции с различными типами данных
- Использовать регулярные выражения для обработки строк
- Выполнять обработку данных, полученных из пользовательских форм
- Использовать cookies и сеансы для взаимодействия с пользователем
- Реализовывать аутентификацию пользователя
- Работать с файловой системой Web-сервера
- Использовать команды для работы с базой данных MySQL
- Выполнять сценарии PHP в командной строке
- Разрабатывать безопасные сценарии PHP

Короткие уроки — быстрые результаты

SAMS Освой самостоятельно

Эта книга — простое практическое пособие для тех, кто хочет быстро получить результат. Потратив всего 10 минут на каждый урок, вы узнаете все необходимое для того, чтобы начать применение на практике огромной мощи языка программирования PHP. Приведенные в книге примеры подходят для PHP 4.1.0 и более поздних версий.

Советы показывают короткие пути к решениям

Предостережения помогают избежать распространенных ошибок

Замечания просто и доходчиво объясняют новые термины и определения

Категория: Программирование для Web

Содержание: PHP

Уровень: для пользователей начального и среднего уровня

ISBN 5-8459-0937-6

66,30 руб

06023