Непревзойденная мощь в обработке текстов и создании сценариев

4-е издание

охватывает версию 5.14

Программирование на

Perl

СИМВОЛ®

O'REILLY®

Том Кристиансен, брайан д фой, Ларри Уолл и Джон Орвант

Programming Perl

Fourth Edition

Tom Christiansen, brian d foy,
Larry Wall and Jon Orwant

O'REILLY®

Программирование на Perl

Четвертое издание

Том Кристиансен, брайан д фой,
Ларри Уолл и Джон Орвант

Санкт-Петербург — Москва
2014

Том Кристиансен, брайан д фой,
Ларри Уолл и Джон Орвант

Программирование на Perl, 4-е издание

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научные редакторы	*М. Зислис, А. Киселев*
Редактор	*М. Зислис*
Верстка	*Д. Орлова*
Корректоры	*Т. Иванкова, В. Логунова*

Кристиансен Т., д фой б., Уолл Л., Орвант Дж.

Программирование на Perl, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2014. – 1048 с., ил.

ISBN 978-5-93286-214-8

Четвертое издание «Программирование на Perl» ждали в России и программисты, и системные администраторы. И вот обновление описания языка Perl, продолжавшего активно развиваться в течение последних пяти лет, перед вами. На этот раз в «Кэмэл» обсуждается текущая версия Perl 5.14 и дается обзор некоторых особенностей готовящейся к выходу версии Perl 5.16.

Все большую значимость в обработке текстов приобретает Юникод, а Perl предлагает лучшую и самую безболезненную поддержку этого стандарта, тесно интегрируя Юникод во все сферы, в том числе в такой популярный механизм языка Perl, как регулярные выражения. И Юникоду, и регулярным выражениям уделено много внимания в книге.

Четвертое издание охватывает такие важные особенности языка Perl, как новые ключевые слова и синтаксические конструкции, уровни ввода/вывода и кодировки, новые escape-последовательности, поддержка стандарта Unicode 6.0, групповые графемы и свойства символов Юникода, именованные сохраняющие группы в регулярных выражениях, рекурсивные и грамматические шаблоны, расширенный обзор архива CPAN и современные передовые приемы программирования. Материал иллюстрируется множеством интересных примеров. И если вы ищете справочник по языку Perl, то он перед вами.

ISBN 978–5–93286–214–8
ISBN 978–0–596–00492–7 (англ)

© Издательство Символ-Плюс, 2014
Authorized Russian translation of the English edition of Programming Perl, Fourth Edition ISBN 9780596004927 © 2013 O'Reilly Media, Inc. All rights reserved. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 380-5007, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 12.11.2013. Формат 70×100 1/16.
Печать офсетная. Объем 67,5 печ. л.

Оглавление

Предисловие . . . 13

I. Общий обзор . . . 31

1. Обзор Perl . . . 33

Введение . . . 33
Естественные и искусственные языки . . . 34
Пример вычисления среднего . . . 48
Дескрипторы файлов . . . 51
Операторы . . . 53
Управляющие конструкции . . . 60
Регулярные выражения . . . 68
Чего вы не знаете, то вам (сильно) не навредит . . . 76

II. Анатомия Perl . . . 77

2. Всякая всячина . . . 79

Атомы . . . 79
Молекулы . . . 80
Встроенные типы данных . . . 82
Переменные . . . 84
Имена . . . 85
Скалярные значения . . . 90
Контекст . . . 101
Списочные значения и массивы . . . 104
Хеши . . . 108
Таблицы имен и дескрипторы файлов . . . 110
Операторы ввода . . . 111

3. Унарные и бинарные операторы . . . 117

Термы и списочные операторы (влево) . . . 119
Оператор «стрелка» . . . 121
Автоинкрементирование и автодекрементирование . . . 122
Возведение в степень . . . 123
Идеографические унарные операторы . . . 123
Операторы связывания . . . 124

Мультипликативные операторы . . . 125
Аддитивные операторы . . . 126
Операторы сдвига . . . 127
Именованные унарные операторы и операторы проверки файлов . . . 127
Операторы сравнения . . . 132
Операторы равенства . . . 132
Оператор интеллектуального сопоставления . . . 133
Операторы поразрядного действия . . . 138
Логические операторы (короткого пути) в стиле C . . . 139
Оператор диапазона . . . 140
Условный оператор . . . 142
Операторы присваивания . . . 144
Оператор запятой . . . 146
Списочные операторы (вправо) . . . 146
Логические and, or, not и xor . . . 147
Операторы C, отсутствующие в Perl . . . 148

4. Операторы и объявления . . . 149

Простые операторы . . . 150
Составные операторы . . . 151
Операторы if и unless . . . 152
Оператор given . . . 153
Операторы циклов . . . 159
Оператор goto . . . 168
Окаменевшие switch/case . . . 169
Оператор многоточия . . . 171
Глобальные объявления . . . 172
Объявления с областью видимости . . . 174
Прагмы . . . 182

5. Поиск по шаблону . . . 185

Бестиарий регулярных выражений . . . 186
Операторы поиска по шаблону . . . 189
Метасимволы и метазнаки . . . 209
Классы символов . . . 219
Квантификаторы . . . 229
Позиции . . . 232
Захват и группировка . . . 236
Группировка без сохранения . . . 243
Управление процессом . . . 246
Замысловатые шаблоны . . . 259
Определение собственных утверждений . . . 281

6. Юникод . . . 285

Не рассказывай, а показывай . . . 289
Доступ к данным в Юникоде . . . 291
Ошибочные представления о регистре . . . 296

Графемы и нормализация . . . 299
Сравнение и сортировка строк Юникода . . . 306
Дополнительные возможности . . . 314
Ссылки . . . 320

7. Подпрограммы . . . 321

Синтаксис . . . 321
Семантика . . . 323
Передача ссылок . . . 329
Прототипы . . . 331
Атрибуты подпрограмм . . . 339

8. Ссылки . . . 341

Что такое ссылка? . . . 341
Создание ссылок . . . 343
Использование жестких ссылок . . . 349
Символические ссылки . . . 359
Фигурные скобки, квадратные скобки и кавычки . . . 360

9. Структуры данных . . . 365

Массивы массивов . . . 365
Хеши массивов . . . 373
Массивы хешей . . . 375
Хеши хешей . . . 377
Хеши функций . . . 380
Более сложные записи . . . 380
Сохранение структур данных . . . 383

10. Пакеты . . . 385

Таблицы имен . . . 387
Квалифицированные имена . . . 391
Пакет по умолчанию . . . 392
Изменение пакета . . . 393
Автозагрузка . . . 395

11. Модули . . . 398

Загрузка модулей . . . 399
Выгрузка модулей . . . 401
Создание модулей . . . 401
Замещение встроенных функций . . . 407

12. Объекты . . . 409

Краткая памятка по объектно-ориентированному жаргону . . . 409
Система объектов Perl . . . 411
Вызов методов . . . 411
Создание объектов . . . 417
Наследование классов . . . 421

Деструкторы экземпляров . . . 431
Управление данными экземпляров . . . 433
Управление данными класса . . . 440
Лось в посудной лавке (Moose) . . . 443
Резюме . . . 445

13. Перегрузка . . . 446

Прагма overload . . . 447
Обработчики перегрузки . . . 447
Перегружаемые операторы . . . 449
Конструктор копирования (=) . . . 456
Когда обработчик перегрузки отсутствует (nomethod и fallback) . . . 457
Перегрузка констант . . . 458
Открытые функции перегрузки . . . 459
Наследование и перегрузка . . . 460
Перегрузка на этапе выполнения . . . 460
Диагностика перегрузки . . . 460

14. Связанные переменные . . . 461

Связывание скаляров . . . 463
Связывание массивов . . . 471
Связывание хешей . . . 477
Связывание дескрипторов файлов . . . 482
Неочевидная ловушка при отвязывании . . . 493
Модули для связывания в CPAN . . . 495

III. Perl как технология . . . 497

15. Межпроцессные взаимодействия . . . 499

Сигналы . . . 500
Файлы . . . 505
Каналы . . . 512
System V IPC . . . 520
Сокеты . . . 523

16. Компиляция . . . 532

Жизненный цикл программ на Perl . . . 533
Компиляция кода . . . 534
Выполнение кода . . . 540
Серверы компиляторов . . . 542
Генераторы кода . . . 543
Средства разработки кода . . . 545
Компилятор и интерпретатор: авангардизм и ретро . . . 547

17. Интерфейс командной строки . . . 552

Обработка команд . . . 552
Переменные среды . . . 569

18. Отладчик Perl . . . 577

Использование отладчика . . . 578
Команды отладчика . . . 580
Настройка отладчика . . . 588
Автоматическое выполнение . . . 591
Поддержка отладчика . . . 593
Профилировщик Perl . . . 595

19. CPAN . . . 600

История . . . 600
Обзор репозитория . . . 601
Экосистема CPAN . . . 604
Установка модулей из CPAN . . . 607
Создание дистрибутивов для CPAN . . . 610

IV. Perl как культура . . . 615

20. Защита данных . . . 617

Обработка ненадежных данных . . . 618
Обработка ошибок синхронизации . . . 630
Работа с ненадежным кодом . . . 637

21. Распространенные приемы программирования . . . 646

Обычные промахи новичков . . . 646
Эффективность . . . 657
Стиль программирования . . . 667
Беглый разговор на Perl . . . 671
Генераторы программ . . . 680

22. Переносимость программ Perl . . . 684

Перевод строки . . . 686
Старшинство байтов и ширина чисел . . . 687
Файлы и файловые системы . . . 688
Взаимодействие с системой . . . 689
Межпроцессные взаимодействия (IPC) . . . 690
Внешние подпрограммы (XS) . . . 690
Стандартные модули . . . 691
Дата и время . . . 691
Интернационализация . . . 692
Стиль . . . 692

23. Документация в формате POD . . . 694

Вкратце о pod . . . 694
Трансляторы и модули pod . . . 702
Создание собственных инструментов для работы с pod . . . 703
Ловушки pod . . . 709
Документирование программ Perl . . . 710

24. Культура Perl . . . 712
История практичности . . . 712
Поэзия Perl . . . 716
Достоинства программиста на Perl . . . 717
События . . . 718
Где и как получить помощь . . . 719

V. Справочный материал . . . 721

25. Специальные имена . . . 723
Специальные имена, сгруппированные по типам . . . 723
Специальные переменные в алфавитном порядке . . . 727

26. Форматы . . . 749
Форматы строк . . . 749
Двоичные форматы . . . 755
Форматы шаблонов . . . 765

27. Функции . . . 773
Функции Perl по категориям . . . 776
Функции Perl в алфавитном порядке . . . 778

28. Стандартная библиотека Perl . . . 926
Библиотечное дело . . . 926
Обзор библиотеки Perl . . . 928

29. Модули прагм . . . 934
attributes . . . 935
autodie . . . 936
autouse . . . 936
base . . . 937
bigint . . . 938
bignum . . . 939
bigrat . . . 939
blib . . . 939
bytes . . . 939
charnames . . . 940
constant . . . 944
deprecate . . . 946
diagnostics . . . 946
encoding . . . 948
feature . . . 948
fields . . . 949
filetest . . . 949
if . . . 950
inc::latest . . . 950
integer . . . 950

less . . . 951
lib . . . 952
locale . . . 953
mro . . . 954
open . . . 954
ops . . . 955
overload . . . 956
overloading . . . 956
parent . . . 957
re . . . 957
sigtrap . . . 959
sort . . . 962
strict . . . 962
subs . . . 965
threads . . . 965
utf8 . . . 967
vars . . . 967
version . . . 967
vmsish . . . 968
warnings . . . 969
Пользовательские прагмы . . . 972

Глоссарий . . . 975

Алфавитный указатель . . . 1017

Предисловие

В погоне за счастьем

Perl – язык, с помощью которого вы сделаете свою работу.

Конечно, если эта работа – программирование, то теоретически ее можно сделать с помощью любого «полного» компьютерного языка. Но опыт показывает, что компьютерные языки различаются не столько *возможностью* что-либо сделать, сколько *легкостью*, с которой это достигается. На одном полюсе находятся так называемые языки четвертого поколения, с помощью которых можно легко делать одни вещи и почти невозможно другие. На другом полюсе – так называемые языки с промышленными возможностями (industrial-strength languages), посредством которых одинаково трудно делать почти все.

Perl не таков. Если сказать кратко, по задумке его создателей *на этом языке легко решать простые задачи, сохраняя возможность решать и сложные*.

Что это за «простые задачи», которые должны решаться легко? Разумеется, те, которые мы решаем изо дня в день. Нам нужен язык, с помощью которого легко работать с числами и текстом, файлами и каталогами, компьютерами и сетями, а в особенности – с программами. Он должен позволять легко запускать внешние программы и просматривать результаты их работы в поисках интересных данных. Он должен позволять легко отправлять эти интересные данные другим программам, способным обрабатывать их особым образом. Он должен также позволять нам легко разрабатывать собственные программы, изменять их и производить отладку. И конечно, наши собственные программы должны легко компилироваться и запускаться, а также быть переносимыми на любую современную операционную систему.

Все это, а также многое другое делает Perl.

Первоначально разработанный как интегрирующий язык для UNIX, Perl давно распространился на большинство других операционных систем. Поскольку Perl выполняется почти везде, он является одной из наиболее переносимых сред программирования, существующих сегодня. Чтобы писать переносимые программы на C или C++, необходимо расставить все эти странные пометки `#ifdef` для каждой операционной системы. Для обеспечения переносимости программ на Java нужно разбираться в индивидуальных особенностях всех реализаций этой платформы. Для создания переносимых сценариев командной оболочки нужно помнить синтаксис всех команд для каждой версии операционной системы и пытаться найти общий знаменатель, благодаря которому они, как можно надеяться,

будут работать всюду. А чтобы создавать переносимые программы на Visual Basic, потребуется дать более гибкое определение понятия «переносимость». :-)

Perl позволяет нам счастливо избежать таких проблем, сохраняя при этом многие преимущества других языков и добавляя собственные чудеса. У этих чудес много источников: практичность набора функций Perl, изобретательность сообщества Perl и неистребимый энтузиазм движения open source в целом. Однако в значительной мере чудеса обусловлены гибридной природой Perl. У Perl смешанное происхождение, и многообразие средств в этом языке всегда считалось плюсом, а не слабостью. Perl – это язык, говорящий: «Дайте мне ваших усталых, ваших бедных»[1]. Если вы чувствуете себя словно в теснящейся толпе и стремитесь «дышать свободно», то Perl – для вас.

Perl охватывает различные культуры. Его взрывное распространение в значительной мере питалось стремлением *бывших* системных UNIX-программистов взять с собой как можно больше из «старого мира». Для них Perl является переносимой квинтэссенцией культуры UNIX, оазисом в пустыне «невозможности перейти из одного места в другое». Существует, однако, и движение в обратном направлении: веб-дизайнеры, работающие в Windows, с удовольствием обнаруживают возможность запускать свои Perl-программы на UNIX-сервере своей компании без доработки.

Хотя Perl особенно популярен среди системных программистов и веб-разработчиков, это связано лишь с тем, что они первыми его открыли; аудитория Perl значительно шире. Получив при создании скромный статус языка обработки текста, Perl развился в сложный язык программирования общего назначения с богатой средой разработки программ, укомплектованной отладчиками, профилировщиками, компоновщиками, компиляторами, библиотеками, редакторами с подсветкой синтаксиса и другими атрибутами «настоящего» языка программирования – если они вам требуются. Но все они относятся к поддержке возможностей решения сложных задач, с чем справляются многие другие языки. Уникальность Perl в том, что он никогда не отступал от идеи легких решений для простых задач.

Поскольку Perl является одновременно мощным и доступным средством, он постоянно используется во всех мыслимых сферах – от аэрокосмической техники до молекулярной биологии, от математики до лингвистики, от графики до обработки документов, от управления базами данных до сетевого администрирования. Perl используется теми, кому позарез нужно быстро проанализировать или преобразовать большие объемы данных, будь то последовательность генов ДНК, набор веб-страниц или контракты на поставку свинины.

Существует много слагаемых успеха этого языка. Perl как открытый проект стал успешным еще до того, как движение open source получило свое название. Perl свободно распространяется, и так будет всегда. Каждый может работать с Perl так, как сочтет удобным, и на основе очень либеральной политики лицензирования. Если вы занимаетесь коммерческой деятельностью и хотите воспользоваться Perl, можете приступать. Язык Perl разрешается встраивать в коммерческие приложения бесплатно и без ограничений. А для тех, у кого возникнет проблема,

---

[1] «Give me your tired, your poor, Your huddled masses yearning to breathe free...» – эти слова являются частью текста стихотворения, выбитого на табличке, которая находится в помещении внутри пьедестала Статуи Свободы. См. фото таблички по адресу *http://en.wikipedia.org/wiki/The_New_Colossus*. – *Прим. ред.*

которую сообщество Perl не сможет решить, существует безотказная страховка – сам исходный код. Сообщество Perl не занимается продажей своих профессиональных тайн под видом «обновлений». Сообщество Perl никогда не «выйдет из дела» и не оставит вас с брошенным на произвол судьбы продуктом.

Безусловно, популярности Perl способствует его бесплатное распространение. Но этого недостаточно для объяснения феномена Perl, поскольку большой успех приходит далеко не ко всем бесплатно распространяемым пакетам. Дело не в том, что он бесплатен; он доставляет удовольствие. Люди чувствуют желание творить на Perl, поскольку он дает свободу самовыражения: можно выбирать между целями оптимизации – скоростью работы компьютера или скоростью программирования, между многословием и выразительностью, между «читабельностью» и простотой поддержки или повторного использования, или переносимости, или простотой изучения, или поучительностью. Можно оптимизировать даже непонятность, если принять участие в конкурсе на самую непонятную программу – Obfuscated Perl Contest.

Perl способен предоставить все эти степени свободы, поскольку является языком с раздвоением личности. Это одновременно и очень простой, и очень богатый язык. Perl заимствует лучшие идеи практически отовсюду и объединяет их в простую логическую систему. Для тех, кому он просто нравится, Perl – это *Practical Extraction and Report Language* (практический язык извлечения данных и создания отчетов). Для тех, кто любит его, Perl – это *Pathologically Eclectic Rubbish Lister* (паталогически эклектичный язык для распечатки чепухи). А минималистам Perl кажется проявлением бесцельной избыточности. Но это хорошо. Редукционисты должны существовать (в основном среди физиков). Редукционисты стремятся разъять целое на части. Мы, все остальные, просто пытаемся собрать целое из частей.

Во многих отношениях Perl – простой язык. Не требуется знать множество особых заклинаний, чтобы скомпилировать программу на Perl – ее можно просто выполнить как пакетный файл или сценарий оболочки. Типы и структуры Perl просты в использовании и понимании. Perl не налагает свои произвольные ограничения на данные – строки и массивы могут быть сколь угодно велики, лишь бы хватило оперативной памяти, и их организация позволяет им легко увеличиваться по мере надобности. Perl не требует изучения новых синтаксиса и семантики, в значительной мере заимствуя их из других языков, с которыми вы можете быть знакомы (например, C, *awk*, BASIC, Python, английский и греческий). На практике почти любой программист сможет прочесть хорошо написанный код на Perl и составить себе представление о том, что он делает.

Очень важно, что нет необходимости изучать Perl полностью, чтобы начать писать полезные программы. Изучение Perl можно начать с «тонкого конца». Вы можете программировать на Perl версии «Детский Лепет», и мы обещаем не смеяться над этим. Точнее, мы обещаем смеяться не более чем над первыми попытками ребенка творчески подходить к миру. Многие идеи Perl заимствованы из естественного языка, и одна из лучших его черт состоит в том, что он позволяет использовать лишь подмножество языка, если его достаточно, чтобы передать мысль. В культуре Perl приемлема любая степень владения языком. Полицию по охране языка мы к вам не пришлем. Сценарий Perl будет «правильным», если выполнит задачу прежде, чем начальник вас уволит.

Будучи во многих отношениях простым, Perl является и богатым языком, в котором можно долго совершенствоваться. Это расплата за возможность решать сложные задачи. Хотя понадобится некоторое время на освоение всех средств Perl, вы будете рады иметь в своем распоряжении расширенные возможности, когда они вдруг понадобятся.

Благодаря своему происхождению Perl был богатым языком уже тогда, когда считался «просто» языком преобразования данных, предназначенным для ориентирования в файлах, просмотра больших объемов текста, создания и получения динамических данных и вывода легко форматируемых отчетов, основанных на этих данных. Но в какой-то момент начался расцвет Perl. Он стал также и языком для работы с файловой системой, управления процессами, администрирования баз данных, программирования в архитектуре клиент-сервер, создания безопасных программ, управления данными в Сети и даже для объектно-ориентированного и функционального программирования. Эти возможности не были просто механически присоединены к Perl – каждая новая синергически работает с остальными, поскольку с самого начала Perl проектировался как интегрирующий язык.

Но Perl умеет объединять в единое целое не только собственные функции. Он создавался как модульный, расширяемый язык. Perl позволяет быстро проектировать, программировать, отлаживать и разворачивать приложения, а также без труда расширять функциональные возможности этих приложений при необходимости. Perl можно встраивать в другие языки, а другие языки можно встраивать в Perl. С помощью механизма импорта модулей можно использовать эти внешние определения, как если бы они были встроенными функциями Perl. Объектно-ориентированные внешние библиотеки сохраняют свою объектную ориентированность в Perl.

Perl помогает разработчику и в других отношениях. В отличие от строго интерпретируемых языков, таких как командные файлы и сценарии оболочки, которые компилируют и выполняют лишь одну команду за раз, Perl сначала быстро компилирует всю программу в промежуточный формат. Подобно любому другому компилятору, он осуществляет различного вида оптимизации и мгновенно реагирует на любые ошибки – от синтаксических и семантических до неудачи при связывании с библиотеками. Когда компилирующий интерфейс Perl удовлетворен вашей программой, он передает промежуточный код на выполнение интерпретатору (либо какому-либо из нескольких модулей генераторов, способных создавать текст на C или байт-код). Все это выглядит сложным, однако компилятор и интерпретатор работают весьма эффективно, и обычный цикл компиляции-прогона-исправления занимает считанные секунды. В совокупности с мощной поддержкой амортизации отказов столь короткий цикл делает Perl языком, на котором действительно возможно быстрое прототипирование. Позже, по ходу совершенствования программы, вы сможете повысить требования к себе и программировать больше за счет дисциплины, чем за счет интуиции. Perl и в этом окажет содействие, если его вежливо об этом попросить.

Perl также способствует созданию более защищенных программ. Помимо всех обычных интерфейсов защиты, предоставляемых другими языками, Perl защищает от случайных ошибок в системе безопасности посредством уникального механизма трассировки данных, автоматически определяющего данные, которые поступили из ненадежного источника, и предотвращающего выполнение опасных операций. Наконец, Perl позволяет создавать специальные защищенные отсеки,

в которых можно безопасно выполнять код сомнительного происхождения с ограничением опасных операций.

Парадоксально, но самая большая помощь, которую Perl может оказать программисту, связана не столько с языком Perl, сколько с людьми, которые с ним работают. Скажем откровенно, сообщество Perl составляют люди, которые более чем кто-либо другой готовы прийти на помощь. Если считать, что в движении Perl есть что-то благочестивое, то именно это и есть его основная ценность. Ларри хотел, чтобы сообщество было чем-то вроде рая, и в целом его желание пока осуществляется. Внесите и свой вклад, чтобы оно таким и оставалось.

Изучаете ли вы Perl ради спасения мира, или из любопытства, или по приказу вашего начальника – в любом случае этот учебник позволит освоить как основы, так и сложные вопросы. И хотя мы не намереваемся учить вас программированию, проницательный читатель что-то приобретет как от искусства, так и от науки программирования. Мы рекомендуем вам развивать в себе три великие добродетели программиста: *лень*, *нетерпеливость* и *высокомерие* (*laziness*, *impatience*, *hubris*). Мы надеемся, что, читая эту книгу, вы найдете ее местами довольно занимательной (а местами – крайне занимательной). Если этого окажется недостаточно, чтобы вы не заснули, постоянно напоминайте себе, что изучение Perl повысит ценность вашего резюме. Так что читайте дальше.

Что нового в этом издании

Проще сказать, что старого! Прошло достаточно много времени с момента выхода предыдущего издания. В свое оправдание мы можем лишь сказать, что у нас была на то пара причин, но теперь все в порядке.

Третье издание вышло в середине 2000 года, как раз когда вышла версия Perl 5.6. Когда мы пишем эти строки 12 лет спустя, к выходу готовится версия Perl 5.16. Много воды утекло за эти годы: вышло несколько новых версий Perl 5 и случилось маленькое событие, которое мы называем Perl 6. Однако эта шестерка обманчива. В действительности, Perl 6 – это «младший брат» Perl 5, а не важное обновление Perl 5, как можно было бы заключить из номера версии. Но в этой книге не рассказывается об этом другом языке. Она все еще посвящена Perl 5 – версии, которой вполне успешно пользуется большинство людей во всем мире (и даже парни из проекта Perl 6!).[1]

Чтобы рассказать, что нового в этой книге, придется рассказать, что нового в Perl. Это издание – не просто «косметический ремонт», призванный повысить продажи книги. Это долгожданное обновление описания языка, продолжавшего активно развиваться в течение последних пяти лет. Мы не будем перечислять все изменения (при необходимости обратитесь к страницам *perldelta*), но есть кое-что, о чем нам хотелось бы рассказать отдельно.

В Perl 5 мы начали добавлять новые функциональные возможности, одновременно создавая средства защиты старых программ от новых инструкций. Например,

---

[1] Поскольку мы ленивы, а вы уже знаете, что эта книга посвящена Perl 5, следует упомянуть, что мы не всегда будем точно указывать номер версии «Perl v5.n» – в оставшейся части книги, если вы увидите лишь номер версии, начинающийся с «v5», просто знайте, что речь идет об этой версии Perl.

мы наконец смягчились в отношении частых просьб реализовать инструкцию, подобную инструкции switch. Однако, как это принято в мире Perl, мы сделали ее лучше и удобнее, предоставив вам более полный контроль над тем, что вы делаете. Мы назвали ее given-when, но эта инструкция будет доступна, только если вы явно попросите об этом. Любая из следующих директив включает доступ к этой новой возможности:

```
use v5.10;
use feature qw(switch);
use feature qw(:5.10);
```

а включив ее, вы получаете «заряженный» оператор switch:

```
given ($item) {
    when (/a/)   { say "Matched an a" }
    when (/bee/) { say "Matched a bee" }
}
```

В главе 4 вы познакомитесь поближе с этой и другими новыми особенностями, так как там их обсуждение более уместно.

Даже при том что поддержка стандарта Юникод (Unicode) существует в Perl начиная с версии v5.6, она была значительно улучшена в последних версиях. В частности, это касается более полной, чем в других языках программирования на данный момент, поддержки Юникода в регулярных выражениях. Благодаря постоянному улучшению поддержки этого стандарта Perl иной раз используется даже для испытаний будущих наработок «Консорциума Юникода». В предыдущем издании этой книги весь материал, посвященный Юникоду, уместился в единственной главе, а в этом издании обсуждение этой темы встречается везде, где это уместно.

Стали еще лучше регулярные выражения – одна из особенностей, устойчиво ассоциирующихся у программистов именно с языком Perl. Другие языки заимствовали язык шаблонов из Perl и дали ему название Perl Compatible Regular Expressions (регулярные выражения, совместимые с Perl), но при этом добавили некоторые свои особенности. Мы, в свою очередь, заимствовали некоторые из этих особенностей, продолжая традицию вбирать в Perl все самое лучшее отовсюду. Вы также познакомитесь с мощными новыми функциями для работы с Юникодом в шаблонах регулярных выражений.

Потоки выполнения (threads) также претерпели значительные изменения. Perl поддерживает две модели многопоточного выполнения: одну мы назвали 5005threads (по номеру версии, в которой она была добавлена), а другая – потоки интерпретатора. Начиная с версии v5.10 поддерживаются только потоки интерпретатора. Однако по различным причинам мы решили не включать обсуждение этой темы в книгу, а больше внимания уделить другим особенностям. Если у вас появится желание изучить потоки выполнения, обратитесь к странице *perlthrtut* справочного руководства[1], которая содержит практически все, что мы могли бы поместить в главу, описывающую потоки выполнения. Возможно, в будущем мы добавим эту главу в качестве бесплатного приложения.

---

[1] Доступ к соответствующим страницам можно получить при помощи команды *perldoc perlthrtut* или по адресу *http://perldoc.perl.org/perlthrtut.html*. – *Прим. ред.*

С течением времени одни особенности появлялись, другие исчезали. Некоторые экспериментальные особенности оказывались неудачными, и мы заменяли их другими экспериментальными особенностями. Так были убраны и забыты псевдохеши[1]. Если вы не знаете, что это такое, не переживайте об этом и не ищите их в этом издании.

И еще: с момента последнего обновления этой книги произошла крупная революция (или две) в практике программирования на Perl, как и в культуре тестирования. Архив CPAN (Comprehensive Perl Archive Network – обширный сетевой архив ресурсов для Perl) продолжает свой экспоненциальный рост, что делает его «убойной особенностью» Perl. Хотя эта книга и не об архиве CPAN, мы все же будем рассказывать о некоторых модулях из него, когда это потребуется. Не пытайтесь реализовать все на голом Perl, без использования дополнительных модулей.

Мы исключили из этого издания две главы: список модулей стандартной библиотеки и список диагностических сообщений (главы 32 и 33 в предыдущем издании). Обе они устареют еще до того, как эта книга попадет на вашу книжную полку. Мы расскажем вам, как самостоятельно получить этот список. Что касается диагностических сообщений, их можно найти на странице *perldiag* справочного руководства[2] или включить вывод подробных предупреждений прагмой `diagnostics`.

Часть I «Общий обзор»

Начать всегда труднее всего. В этой части базовые идеи Perl излагаются в неформальном виде – устройтесь поудобнее в вашем любимом кресле. Не претендуя на роль полного учебного руководства, эта часть предлагает скоростное введение в Perl, что устроит не всякого читателя. В разделе «Печатная документация» (ниже) поищите книги, которые лучше сочетаются с вашим стилем учебы.

Часть II «Анатомия Perl»

В этой части проводится глубокое и ничем не ограниченное обсуждение внутреннего устройства языка на всех уровнях абстракции – от типов данных, переменных и регулярных выражений до подпрограмм, модулей и объектов. Читатель получит хорошее представление о том, как работает язык, а также несколько советов по правильному проектированию программ. (А тех, кто никогда не использовал язык с поиском по шаблону, ждет особое удовольствие.)

Часть III «Perl как технология»

Многое можно делать с помощью одного только Perl, но в этой части вы изучите волшебство более высокого уровня. Узнаете о том, как заставить Perl пройти через все препятствия, которые поставит перед ним ваш компьютер, – от обработки Юникода, взаимодействия процессов и многопоточности до компилирования, вызова, отладки и профилирования, а также создания собственных внешних расширений на C или C++ или интерфейсов к имеющимся API. Perl будет счастлив побеседовать с любым интерфейсом на вашем компьютере, да, пожалуй, и любом другом компьютере в Интернете, если позволят погодные условия.

---

[1] Псевдохеши – виртуальные ассоциативные массивы. – *Прим. ред.*

[2] Воспользуйтесь командой *perldoc perldiag* или адресом *http://perldoc.perl.org/perldiag.html*. – *Прим. ред.*

Часть IV «Perl как культура»

Каждому ясно, что у культуры должен быть свой язык, но сообществу Perl всегда было ясно, что у языка должна быть культура. В этой части мы рассматриваем программирование на Perl как человеческую деятельность, являющуюся частью реального мира людей. Мы также даем много советов относительно того, как заниматься самосовершенствованием и как сделать, чтобы ваши программы приносили больше пользы людям.

Часть V «Справочный материал»

Здесь собраны главы, в которых читатель сможет найти что-либо в алфавитном порядке – от специальных переменных и функций до стандартных модулей и прагм. Глоссарий будет особенно полезен тем, кто не знаком с жаргоном вычислительной техники. Например, те, кто не знает, что такое «прагма», могут прямо сейчас посмотреть значение этого слова. (А тем, кто не знает значение слова «такое», мы не можем помочь ничем.)

Стандартный дистрибутив

Официальная политика Perl, как отмечается в странице *perlpolicy* справочного руководства[1], заключается в поддержке двух последних официальных версий. Поскольку на момент написания этих строк текущей была версия v5.14, это означает, что официально поддерживаются обе версии, v5.12 и v5.14. Когда будет выпущена версия v5.16, официальная поддержка версии v5.12 прекратится.

В настоящее время большинство производителей операционных систем включают Perl в качестве стандартной составляющей своей системы, хотя их цикл выпуска новых версий может не совпадать с циклом выпуска новых версий Perl. На момент написания данной книги Perl входит в стандартные дистрибутивы AIX, BeOS, BSDI, Debian, DG/UX, DYNIX/ptx, FreeBSD, IRIX, LynxOS, Mac OS X, OpenBSD, OS390, RedHat, SINIX, Slackware, Solaris, SuSE и Tru64. Некоторые компании поставляют Perl на отдельных CD с бесплатным программным обеспечением или через группы обслуживания клиентов. Сторонние производители, такие как ActiveState, предоставляют откомпилированные дистрибутивы для ряда операционных систем, в том числе производимых Microsoft.

Даже если производитель включил Perl в стандартный дистрибутив, в конечном итоге, возможно, понадобится откомпилировать и установить Perl самостоятельно. В результате вы будете знать, что ваша версия является самой свежей, и сможете сами выбрать, куда установить библиотеки и документацию. Также можно будет решить, следует ли скомпилировать Perl с поддержкой дополнительных расширений, таких как поддержка многопоточной модели выполнения, большие файлы или множество низкоуровневых опций отладки, доступ к которым осуществляется через ключ командной строки *-D*. (Отладчик уровня пользователя поддерживается всегда.)

Проще всего загрузить комплект исходного кода Perl, указав браузеру домашнюю страницу на *www.perl.org*, где на видном месте располагается информация о загружаемых файлах, а также доступны ссылки на компилированные двоичные модули для платформ, компиляторы C для которых затерялись.

---

[1] Команда *perldoc perlpolicy* или *http://perldoc.perl.org/perlpolicy.html*. – *Прим. ред.*

Можно также направиться прямо в архив CPAN, описанный в главе 19, по адресу *http://www.cpan.org*. Если работа с ним окажется слишком медленной (а это может случиться, поскольку он очень популярен), следует найти зеркальный сервер CPAN поблизости от себя. На странице *http://www.cpan.org/SITES.html* приводится список всех сайтов архива CPAN, откуда вы можете выбрать удобное для вас зеркало. Некоторые зеркала доступны по FTP, другие по HTTP (что может иметь значение для тех, кто выходит в Интернет из корпоративной сети, защищенной брандмауэром). Мультиплексор *http://www.cpan.org* попытается принять решение автоматически, однако при желании вы легко сможете изменить этот выбор.

Получив исходный код и распаковав его в каталог, следует прочесть файлы *README* и *INSTALL*, чтобы узнать, как выполнить сборку Perl. В каталоге может также иметься файл *INSTALL.platform*, где *platform* представляет платформу вашей операционной системы.

Если данная платформа является разновидностью UNIX, то команды, необходимые для получения, конфигурирования, сборки и установки Perl, могут быть примерно следующие. Во-первых, необходимо выбрать команду, с помощью которой будет получен исходный код. Загрузить пакет можно с помощью браузера или инструмента командной строки:

```
% wget http://www.cpan.org/src/5.0/maint.tar.gz
```

Теперь нужно распаковать, сконфигурировать, собрать и установить:

```
% tar zxf latest.tar.gz      # или сначала gunzip, а затем tar xf.
% cd perl-5.14.2             # или 5.* для других версий.
% sh Configure -des          # принимает ответы по умолчанию.
% make test && make install  # обычно требует привилегий суперпользователя
```

Для вашей платформы могут иметься уже готовые пакеты, не требующие выполнения всех этих операций (а также включающие исправления и расширения для вашей платформы). Кроме того, многие платформы уже включают предустановленный Perl, так что описанные действия могут оказаться ненужными.

Если Perl уже установлен, но вам хочется установить другую версию, можно избежать лишней работы, воспользовавшись инструментом *perlbrew*. Он автоматизирует все описанные действия и выполняет установку в каталог, куда вы имеете право устанавливать файлы, если не обладаете привилегиями администратора. Этот инструмент доступен в CPAN под названием `App::perlbrew`, но вы можете установить его, выполнив следующие действия, как описывается в документации:

```
% curl -L http://xrl.us/perlbrewinstall | bash
```

После установки просто позвольте этому инструменту выполнить всю работу за вас:

```
% ~/perl5/perlbrew/bin/perlbrew install perl-5.14.2
```

Однако этим возможности инструмента *perlbrew* не ограничиваются, поэтому за дополнительной информацией обращайтесь к документации.

Существуют также расширенные версии стандартного дистрибутива Perl. Компания ActiveState предлагает ActivePerl (*http://www.activestate.com/activeperl/downloads*) – бесплатные версии для Windows, Mac OS X и Linux и платные – для Solaris, HP-UX и AIX.

Strawberry Perl (*http://strawberryperl.org/*) – версия для Windows, включающая различные инструменты, необходимые для компиляции и установки сторонних модулей Perl из CPAN.

Citrus Perl (*http://www.citrusperl.com/*) – дистрибутив для Windows, Mac OS X и Linux, включающий инструментарий wxPerl для создания графических интерфейсов. Он предназначен для тех, кто желает создавать на языке Perl программы с графическим интерфейсом. А в распространении этих приложений вам поможет другой инструмент, Cava Packager (*http://www.cava.co.uk/*), также входящий в состав этого дистрибутива.

Электронная документация

Обширная электронная документация по Perl входит в состав его стандартного дистрибутива. (О печатной документации говорится в следующем разделе.) Дополнительная документация появляется, как только устанавливается новый модуль из CPAN.

Упоминая в этой книге «страницы руководства Perl», мы имеем в виду комплект электронных страниц руководства по Perl, который находится на вашем компьютере. Под *страницей электронного руководства* (*manpage*) будем понимать просто файл с документацией, для чтения которого необязательно иметь UNIX-программу *man*. Страницы руководства Perl могут быть установлены даже как страницы HTML, особенно в системах, отличных от UNIX.

Электронные страницы руководства по Perl разделены на несколько секций, поэтому можно легко найти нужное, не продираясь через сотни страниц текста. Поскольку страница верхнего уровня называется просто *perl*, то в UNIX команда `man perl` должна привести именно на нее.[1] Эта страница, в свою очередь, обозначает страницы, посвященные конкретным темам. Например, `man perlre` выведет страницу руководства по регулярным выражениям Perl. Команда `perldoc` часто работает в тех системах, в которых не работает команда `man`. В вашем дистрибутиве могут также содержаться страницы руководства по Perl в формате HTML или родном для системы формате подсказки. Уточните этот вопрос у своего системного администратора – если, конечно, сами не являетесь им.

Навигация по стандартным страницам руководства

В незапамятные времена (если говорить о Perl, то имеется в виду 1987 год) страница руководства *perl* была кратким документом объемом около 24 печатных страниц. Например, раздел по регулярным выражениям занимал всего два абзаца. (Этого было достаточно при условии знакомства с *egrep*.) В некоторых отношениях с тех пор изменилось почти все. Если считать стандартную документацию, различные утилиты, сведения о переносах на различные платформы и множество стандартных модулей, то наберется несколько тысяч печатных страниц документации, разбросанных по многим отдельным страницам электронного руководства.

[1] Если при этом открывается нечто необозримое, то, вероятно, вы обращаетесь к древнему руководству версии 4. Проверьте, не указывает ли переменная среды `MANPATH` на места, где можно производить археологические раскопки. (Введите `perldoc perl`, чтобы узнать, как настроить `MANPATH` соответственно выдаче команды `perl -V:man.dir`.)

(И это без учета модулей из CPAN, которые могут быть у вас установлены, и в немалом количестве.)

Но в других отношениях не изменилось ничего: по-прежнему жива страница руководства *perl* и по-прежнему она является наилучшей отправной точкой, когда вы не знаете, откуда начать. Разница в том, что, попав туда, нельзя остановиться. Документация – это больше не кустарное производство, это торговый пассаж с сотнями магазинов. Войдя в дверь, необходимо найти плакат, который поможет определить, где находится лавка или универсальный магазин, продающий то, зачем вы пришли. Конечно, освоившись в торговом центре, как правило, заранее знаешь, куда направиться.

Вот некоторые вывески:

Таблица 1. Некоторые страницы руководства по Perl

Страница руководства	Что освещает
perl	Доступные страницы руководства по Perl
perldata	Типы данных
perlsyn	Синтаксис
perlop	Операторы и их приоритеты
perlre	Регулярные выражения
perlvar	Предопределенные переменные
perlsub	Подпрограммы
perlfunc	Встроенные функции
perlmod	Как использовать модули Perl
perlref	Ссылки
perlobj	Объекты
perlipc	Взаимодействие между процессами
perlrun	Как выполнять команды Perl, а также ключи
perldebug	Отладка
perldiag	Диагностические сообщения

Это лишь небольшой отрывок, но в нем содержится самое необходимое. Как видно, если нужны сведения о том или ином операторе, вам потребуется *perlop*. А если нужно что-то выяснить о предопределенных переменных, следует искать в *perlvar*. Если получено непонятное диагностическое сообщение, обращайтесь к *perldiag*. И так далее.

В стандартное руководство по Perl входит список часто задаваемых вопросов (FAQ). Он разбит на девять разных страниц, перечисленных в табл. 2:

Таблица 2. Страницы из списка часто задаваемых вопросов

Страница руководства	Что освещает
perlfaq1	Общие вопросы о Perl
perlfaq2	Получение Perl и сведения о нем
perlfaq3	Инструменты программирования

Таблица 2 (продолжение)

Страница руководства	Что освещает
perlfaq4	Обработка данных
perlfaq5	Файлы и форматы
perlfaq6	Регулярные выражения
perlfaq7	Общие особенности языка Perl
perlfaq8	Взаимодействие с системой
perlfaq9	Сетевое взаимодействие

Некоторые страницы руководства (табл. 3) содержат замечания по специфике платформ:

Таблица 3. Страницы с замечаниями по специфике платформ

Страница руководства	Что освещает
perlamiga	Версия для Amiga
perlcygwin	Версия для Cygwin
perldos	Версия для MS-DOS
perlhpux	Версия для HP-UX
perlmachten	Версия для Power MachTen
perlos2	Версия для OS/2
perlos390	Версия для OS/390
perlvms	Версия для DEC VMS
perlwin32	Версия для MS-Windows

Относительно версий для разных платформ см. также главу 22 и каталог ports в архиве CPAN (*http://www.cpan.org/ports/index.html*), описанном выше.

Страницы руководства, не принадлежащие Perl

Когда мы ссылаемся на документацию, не принадлежащую Perl, как в случае *getitimer*(2), ссылка указывает на страницу *getitimer* из раздела 2 руководства Unix Programmer's Manual.[1] Страницы руководства для системных вызовов, вроде *getitimer,* могут отсутствовать в системах, отличных от UNIX, но ведь системные вызовы UNIX в таких операционных системах все равно использовать не получится. Для тех, кому действительно требуется документация по команде, системному вызову или библиотечной функции UNIX, скажем, что многие организации поместили свои страницы руководства в Интернете. Выполнив поиск по строке `crypt(3)` в Google, можно найти множество копий данной конкретной страницы.

[1] В разделе 2 должны содержаться только сведения о прямых вызовах операционной системы. (Они часто называются *системными обращениями* («system calls»), но мы неуклонно называем их в этой книге *системными вызовами* (syscalls), чтобы не спутать с функцией `system`, не имеющей отношения к системным вызовам. Однако между системами есть некоторые различия в том, какие вызовы реализованы как системные, а какие – как обращения к библиотекам C, поэтому есть вероятность, что `getitimer(2)` обнаружится в разделе 3.

Хотя страницы руководства по Perl верхнего уровня обычно устанавливаются в разделе 1 стандартных каталогов *man*, мы не будем добавлять (1) к названиям таких страниц руководства в нашей книге. Их легко узнать, поскольку они имеют вид «`perl`*бубубу*».

Печатная документация

Тем, кто хочет больше узнать о Perl, рекомендуем некоторые издания:

- «Perl 5 Pocket Reference», by Johan Vromans; O'Reilly Media (5th Edition, July 2011). Эта маленькая брошюра служит удобным справочником по Perl.
- «Perl Cookbook», by Tom Christiansen and Nathan Torkington, O'Reilly Media (2nd Edition, August 2003).[1] Эта книга дополняет ту, которая сейчас у вас в руках. Она содержит готовые рецепты программирования на Perl.
- «Learning Perl», by Randal Schwartz, brian d foy, and Tom Phoenix; O'Reilly Media (6th Edition, June 2011).[2] Эта книга учит программистов тем 30% основ Perl, которые пригождаются в 70% случаев. Она ориентирована на тех, кто пишет автономные программы, содержащие порядка пары сотен строк.
- «Intermediate Perl», by Randal Schwartz, brian d foy, and Tom Phoenix; O'Reilly Media (August 2012).[3] Эта книга является продолжением книги «Learning Perl» («Изучаем Perl») и знакомит читателей со ссылками, структурами данных, пакетами, объектами и модулями.
- «Mastering Perl», by brian d foy; O'Reilly Media (July 2007). Это последняя книга трилогии, включающей также книги «Learning Perl» («Изучаем Perl») и «Intermediate Perl» («Изучаем Perl глубже»). Вместо основ языка она рассматривает вопросы применения Perl для решения повседневных задач.
- «Modern Perl», by chromatic; Oynx Neon (October 2010). Эта книга содержит обзор современных тем и приемов программирования на Perl для тех, кто уже занимается программированием, но не уделяет внимания последним событиям в мире Perl.
- «Mastering Regular Expressions», by Jeffrey Friedl; O'Reilly Media (3rd Edition, August 2006).[4] Хотя в книге не освещены последние нововведения в регулярных выражениях Perl, она является ценным справочником для всех, кто хочет знать, как работают регулярные выражения.
- «Object Oriented Perl», by Damian Conway, Manning, 1999. Для начинающих и опытных разработчиков объектно-ориентированных программ. Эта книга излагает обычные и тайные приемы создания мощных объектных систем на Perl.

---

[1] Т. Кристиансен, Н. Торкингтон «Perl: библиотека программиста». – Пер. с англ. – СПб.: Питер, 2001.

[2] Р. Шварц, брайан д фой и Т. Феникс «Изучаем Perl», 5-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2009.

[3] Р. Шварц, брайан д фой и Т. Феникс «Perl: изучаем глубже», 3-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2014.

[4] Д. Фридл «Регулярные выражения», 3-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2008.

- «Mastering Algorithms with Perl», by Jon Orwant, Jarkko Hietaniemi, and John Macdonald, O'Reilly, 1999. Все полезные приемы из курса вычислительных алгоритмов, но без тягостных доказательств. Книга освещает фундаментальные и полезные алгоритмы, относящиеся к графам, текстам, множествам и ряду других областей.

Существует много других книг и публикаций по Perl, и по старческой дряхлости мы наверняка позабыли упомянуть некоторые хорошие. (Из милосердия мы не стали отмечать некоторые плохие издания.)

Помимо перечисленных публикаций, относящихся к Perl, мы рекомендуем следующие книги. Они не посвящены непосредственно Perl, но их бывает удобно иметь под рукой в качестве справочника, источника советов или вдохновения.

- «The Art of Computer Programming», by Donald Knuth, Volumes 1–4A: «Fundamental Algorithms», «Seminumerical Algorithms», «Sorting and Searching», and «Combinatorial Algorithms»; Addison-Wesley (2011).[1]
- «Introduction to Algorithms», by Thomas Cormen, Charles Leiserson, and Ronald Rivest; MIT Press and McGraw-Hill (1990).
- «Algorithms in C», by Robert Sedgewick, Addison-Wesley (1990).
- «The Elements of Programming Style», by Brian Kernighan and P.J. Plauger; Prentice-Hall (1988).
- «The Unix Programming Environment», by Brian Kernighan and Rob Pike; Prentice-Hall (1984)[2] (3rd Edition, August 2006).
- «POSIX Programmer's Guide», by Donald Lewine, O'Reilly Media (1991).
- «Advanced Programming in the UNIX Environment», by W. Richard Stevens and Stephen A. Rago, Addison-Wesley (3rd Edition, May 2013).[3]
- «TCP/IP Illustrated», vols. 1–3, by W. Richard Stevens, Addison-Wesley (1992–1996).
- «The Lord of the Rings», by J. R. R. Tolkien, Houghton Mifflin (U.S.) and Harper Collins (U.K.) (последнее издание: 2005).[4]

Дополнительные источники

Интернет – чудесное изобретение, и все мы продолжаем открывать возможности его наиболее полного использования. (Конечно, некоторые предпочитают «открывать» Интернет так, как Толкин открывал Средиземье.)

---

[1] Д. Кнут «Искусство программирования», тома 1–4A: «Основные алгоритмы», «Получисленные алгоритмы», «Сортировка и поиск», «Комбинаторные алгоритмы, часть 1». – Пер. с англ. – Вильямс, 2013.

[2] Брайан Керниган и Роб Пайк «UNIX. Программное окружение». – Пер. с англ. – СПб.: Символ-Плюс, 2003.

[3] У. Ричард Стивенс и Стивен Раго «UNIX. Профессиональное программирование», 3-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2014.

[4] Дж. Р.Р. Толкин «Властелин Колец», АСТ, 2009.

Perl в Сети

Посетите домашнюю страницу Perl *http://www.perl.org/*. Она предлагает новости мира Perl, исходные коды и скомпилированные версии для разных платформ, тематические статьи, документацию, расписания конференций и многое другое.

Посетите также веб-страницу Perl Mongers *http://www.pm.org*, чтобы взглянуть на Perl с точки зрения широких масс, так сказать, на «корни», которые бурно разрастаются во всех частях света, за исключением Южного полюса, где их приходится держать в закрытом помещении. Местные группы PM проводят регулярные небольшие встречи, на которых можно обменяться профессиональным опытом с другими хакерами Perl, живущими в той же части света.

Сообщения об ошибках

В том маловероятном случае, если ошибка обнаружена в самом Perl, а не в вашей программе, нужно постараться воспроизвести ее в минимальном по объему контрольном примере и сообщить о ней с помощью программы *perlbug*, поставляемой с Perl. Дополнительные сведения можно найти на *http://bugs.perl.org*.

В действительности, команда *perlbug* является лишь интерфейсом к инструменту учета дефектов под названием RT.[1] Кроме того, отчет об ошибке можно отправить на адрес *perlbug@perl.org* без привлечения дополнительных инструментов, однако *perlbug* вместе с отчетом отправляет дополнительную информацию о вашей версии Perl, такую как номер версии и флаги компиляции, которые могут помочь разработчикам определить источник вашей проблемы.

Можно также заглянуть в список существующих дефектов. Возможно, кто-то уже столкнулся с вашей проблемой. Начните со страницы *https://rt.perl.org/* и перейдите по ссылке *perl5*.

Если речь идет о стороннем модуле из архива CPAN, следует воспользоваться другой версией инструмента RT по адресу: *https://rt.cpan.org/*. Однако следует учитывать, что не все модули в архиве CPAN пользуются бесплатной услугой RT, поэтому всегда проверяйте документацию, где могут приводиться дополнительные инструкции, касающиеся отправки отчетов об ошибках.

Соглашения, принятые в этой книге

Некоторые из принятых нами соглашений более подробно обсуждаются в соответствующих разделах. Соглашения по коду обсуждаются в разделе «Стиль программирования» главы 21. Можно сказать, что наши лексические соглашения представлены в глоссарии (наш словарь).

В книге приняты следующие типографские соглашения:

ЗАГЛАВНЫЕ СИМВОЛЫ

Используются для обозначения формальных названий символов Юникода и логических операторов.

---

[1] Проект Best Practical, в рамках которого осуществляется разработка инструмента Request Tracker, или RT, бесплатно предоставляет свою платформу основным проектам Perl, включая сам *perl* и все пакеты в архиве CPAN.

Курсив

Применяется для URL-адресов, страниц руководства, имен файлов и программ. Новые термины тоже выделяются курсивом при первом появлении в тексте. Для многих из этих терминов можно найти альтернативные определения в глоссарии, если недостаточно тех, которые имеются в тексте. Курсив также используется для выделения команд и ключей командной строки. Это, к примеру, позволяет отличить ключ *-w*, включающий вывод предупреждений, от параметра `-w` (проверки наличия файла).

`Моноширинный`

Используется в примерах и в обычном тексте для выделения программного кода. Данные обозначаются `моноширинным шрифтом` и заключаются в кавычки (`"`), не являющиеся частью значения.

`Моноширинный курсив`

Служит для обозначения общих элементов кода, вместо которых необходимо подставить конкретные значения. Он также используется в некоторых примерах, чтобы выделить вывод, производимый программой.

`Моноширинный полужирный`

Этот шрифт используется в примерах для обозначения текста, который вводится в командной строке.

`Моноширинный полужирный курсив`

Используется для выделения вывода программы, когда требуется отличать его от текста, вводимого в командной строке.

Мы приводим множество примеров, и в большинстве случаев это фрагменты более крупных программ. Некоторые примеры являются завершенными программами, что можно увидеть по начальной строке `#!`. Почти все наши более длинные программы начинаются со строки:

```
#!/usr/bin/perl
```

В других примерах приводится текст, который следует ввести в командной строке. Мы используем `%` для обозначения приглашения оболочки:

```
% perl -e 'print "Hello, world.\n"'
Hello, world.
```

Этот стиль – типичный образец стандартной командной строки UNIX, в которой одиночные кавычки представляют «наиболее закавыченный» формат. В других системах соглашения по использованию кавычек и шаблонных символов могут быть иными. Например, многие интерпретаторы команд в MS-DOS и VMS требуют двойные, а не одиночные кавычки при определении аргументов, содержащих пробелы и шаблонные символы.

Благодарности

Здесь мы публично приносим благодарность нашим советникам, консультантам и рецензентам, что должно компенсировать все грубости, сказанные им в частном порядке. Это: Эбигайл (Abigail), Мэтью Барнетт (Matthew Barnett), Пирс Коули (Piers Cawley), chromatic (хроматик), Дамиан Конвей (Damian Conway), Дейв Кросс

(Dave Cross), Хоакин Ферреро (Joaquin Ferrero), Джеремиа Фостер (Jeremiah Foster), Джефф Хеймер (Jeff Haemer), Юрий Маленький (Yuriy Malenkiy), Нуно Мендес (Nuno Mendes), Стеффен Мюллер (Steffen Müller), Энрике Нелл (Enrique Nell), Дэвид Никол (David Nicol), Флориан Рагвитц (Florian Ragwitz), Эллисон Рэндал (Allison Randal), Крис Роедер (Chris Roeder), Кит Томпсон (Keith Thompson), Леон Тиммерманс (Leon Timmermans), Натан Торкингтон (Nathan Torkington), Йохан Вроманс (Johan Vromans) и Карл Уильямсон (Karl Williamson).

Особую благодарность хотим выразить всем сотрудникам издательства O'Reilly Media за их героические усилия в преодолении бесчисленных, неожиданных и необычных проблем, связанных с опубликованием этой книги в современном постмодернистском издательском мире. В первую очередь мы хотели бы сказать спасибо нашему выпускающему редактору Холли Бауэр (Holly Bauer) за ее бесконечное терпение в отношении тысяч маленьких уточнений и изменений, сделанных уже после сдачи рукописи. Мы благодарны начальнику производственного отдела Дэну Фоксмиту (Dan Fauxsmith) за его упорство в поиске редких шрифтов, необходимых для многочисленных примеров использования Юникода, и за поддержание плавности хода производственного конвейера. Мы признательны заведующему производством Адаму Уитверу (Adam Witwer) за то, что он, засучив рукава, самоотверженно боролся с программным комплексом Antenna House, применявшимся для создания готовой к тиражированию копии книги. Наконец, благодарим издателя Лори Петрицки (Laurie Petrycki): она не просто поддержала всех причастных к созданию книги, о которой мечтали авторы, но и воодушевила самих авторов на создание книг, прочтение которых приносит людям массу удовольствия.

Safari® Books Online

Safari Books Online – это виртуальная библиотека, которая позволяет легко и быстро находить ответы на вопросы среди более чем 7500 технических и справочных изданий и видеороликов.

Подписавшись на услугу, вы сможете загружать любые страницы из книг и просматривать любые видеоролики из нашей библиотеки. Читать книги на своих мобильных устройствах и сотовых телефонах. Получать доступ к новинкам еще до того, как они выйдут из печати. Читать рукописи, находящиеся в работе, и посылать свои отзывы авторам. Копировать и вставлять отрывки программного кода, работать с закладками, загружать отдельные главы, помечать ключевые разделы, оставлять примечания, печатать страницы и пользоваться массой других преимуществ, позволяющих экономить ваше время.

Благодаря усилиям O'Reilly Media данная книга также доступна через службу Safari Books Online. Чтобы получить полный доступ к электронной версии этой книги, а также к другим книгам схожей тематики, изданным O'Reilly и другими издательствами, подпишитесь бесплатно по адресу *http://my.safaribooksonline.com*.

Хотим услышать ваши отзывы

Мы протестировали и выверили все сведения, представленные в этой книге так тщательно, как могли, но читатели могут обнаружить, что некоторые функции

претерпели изменения (или даже что мы допустили ошибки!). Просим сообщать обо всех найденных ошибках, а также пожеланиях для последующих изданий по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
1-800-998-9938 (в США и Канаде)
1-707-829-0515 (международный/местный)
1-707-829-0104 (факс)

Книга имеет свой веб-сайт, где можно найти список обнаруженных ошибок и другую информацию:

http://shop.oreilly.com/product/9780596004927.do

Здесь же вы найдете все примеры программного кода, доступные для загрузки, что избавит вас от необходимости набирать их вручную, как это делали мы.

Свои комментарии и вопросы технического характера, касающиеся этой книги, направляйте по адресу:

bookquestions@oreilly.com.

Дополнительную информацию о книгах, конференциях, центрах ресурсов и сети издательства O'Reilly вы найдете на сайте:

http://www.oreilly.com

I

Общий обзор

1

Обзор Perl

Введение

Мы считаем Perl языком, который легко освоить и применять, и надеемся убедить в этом и вас. Одна из особенностей Perl, делающих его легким языком, заключена в возможности без обиняков сказать то, что необходимо сказать. Во многих языках программирования приходится объявлять типы, переменные и подпрограммы, которые предполагается использовать, прежде чем написать хотя бы одну строчку выполняемого кода. Подобные объявления – разумный подход, если речь идет о сложных задачах, требующих сложных структур данных. Но для многих более простых, повседневных задач хотелось бы иметь язык программирования, на котором можно просто сказать:

```
print "Howdy, world!\n";
```

и программа это сделает.

Таким языком и является Perl. На практике этот пример представляет собой законченную программу[1], и если передать ее интерпретатору Perl, тот выведет на экран `"Howdy, world!"`. (`\n` в этом примере обеспечивает перевод строки в конце вывода.)

Вот и все. *После* того как вы сказали, что хотели, тоже не требуется произносить много слов. В отличие от многих языков программирования, Perl считает, что «вывалиться» из программы, когда закончился исходный текст, – нормальный способ ее завершения. Конечно, при желании вы можете явно вызвать функцию `exit`, как *можете* и объявить некоторые переменные, либо даже сделать *обязательным* объявление всех переменных. Но право выбора принадлежит вам. В Perl можно поступать «так, как должно» согласно вашему личному пониманию этих слов.

Есть много других причин, по которым Perl легко использовать, но перечислять сейчас их все бессмысленно, поскольку этому и посвящена наша книга. Говорят,

---

[1] Или «сценарий» (script), «приложение» (application), «исполняемый объект» (executable), «штуковина» (doohickey). Кому что больше нравится.

что дьявол проявляет себя в частностях, но Perl старается облегчить вашу жизнь и в трудных ситуациях. На любом уровне Perl заботится о том, чтобы доставить вас туда, куда нужно, с минимумом суеты и максимумом комфорта. Вот почему многие программисты на Perl ходят с глупыми ухмылками на физиономиях.

Эта глава представляет собой обзор, поэтому мы не пытаемся представить Perl с рациональной стороны. Равно как не претендуем на полноту изложения или логичность. Для этого предназначены следующие главы. Вулканцам, андроидам и другим лицам аналогичного склада ума следует пропустить этот обзор и перейти сразу к главе 2, где плотность информации выше. С другой стороны, тем, кому нужен учебник, где все постепенно раскладывается по полочкам, возможно, следует обратиться к книге «Learning Perl»[1]. Но и данную книгу пока не выбрасывайте.

Эта глава представляет Perl *другому* полушарию вашего мозга, тому, которое называют ассоциативным, артистическим, управляющим эмоциями или просто впитывающим. Для этого Perl рассматривается с разных точек зрения, чтобы дать читателю о нем такое же ясное представление, какое слепцы могли получить о слоне (ощупывая его). Возможно, мы продвинемся дальше мудрецов: мы все-таки имеем дело с верблюдом (см. обложку). Надеемся, что хотя бы одно из этих представлений о Perl поможет вам меньше горбатиться.

Естественные и искусственные языки

Языки были изобретены людьми и для блага людей. В анналах компьютерных наук этот факт случайно оказался позабытым.[2] Поскольку (допуская вольность речи) можно сказать, что создателем Perl является лингвист, этот язык, по задумке, должен работать так же гладко, как естественные языки. Само собой, это охватывает множество аспектов, поскольку естественные языки хорошо работают одновременно на нескольких уровнях. Можно было бы перечислить многие из этих лингвистических принципов, но самым важным принципом проектирования языка является тот, что простые вещи должны делаться просто, а сложные вещи должны быть реализуемы. (На самом деле, здесь два принципа.) Они могут показаться вам очевидными, однако многие языки программирования не удовлетворяют то одному, то другому условию.

Естественные языки удовлетворяют обоим условиям, поскольку люди непрерывно пытаются выражать как простые, так и сложные мысли, и поэтому языки развиваются так, чтобы справляться с обеими задачами. Perl проектировался, прежде всего, как язык развивающийся, и он действительно эволюционировал со времени своего создания. Вклад в эволюцию Perl вносили многие люди на протяжении ряда лет. Мы часто шутим, что верблюд – это лошадь, спроектированная комитетом, но если вдуматься, верблюд достаточно хорошо приспособлен для жизни в пустыне. Эволюция привела верблюда к относительной самодостаточности. (С другой стороны, в результате эволюции верблюд не стал лучше пахнуть. Как и Perl.) Вот одна из многих странных причин, почему мы сделали верблюда талисманом Perl, но к лингвистике это имеет отдаленное отношение.

---

[1] Рэндал Шварц, Том Феникс, брайан д фой «Изучаем Perl», 5-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2009.

[2] Точнее, об этом факте вспоминают от случая к случаю.

Далее. Когда кто-то произносит слово «лингвистика», многие начинают думать о словах либо о предложениях. Но слова и предложения – это лишь пара удобных способов членения речи. Те и другие можно разбить на более мелкие смысловые единицы или объединить в более крупные. А смысл каждой единицы существенно зависит от синтаксического, семантического и практического контекста, в котором находится эта единица. В естественном языке есть слова разных видов: существительные, глаголы и прочее. Если произнести слово «dog» (собака) изолированно, в отрыве от контекста, представляется существительное, хотя можно использовать это слово и в других смыслах. То есть существительное может выполнять функции глагола, прилагательного или наречия в зависимости от контекста. «If you dog a dog during the dog days of summer, you'll be a dog tired dogcatcher.»[1] Perl также по-разному интерпретирует слова в зависимости от контекста. Далее мы увидим, как он это делает. Помните просто, что Perl пытается понять, о чем идет речь, как это делает любой хороший слушатель. Perl со своей стороны усиленно старается соблюсти условия сделки. Скажите, чего вы хотите, а Perl, как правило, «сообразит». (Если только вы не начнете нести бессмыслицу – анализатор Perl понимает Perl значительно лучше, чем английский или суахили.)

Но вернемся к существительным. Существительное может быть названием конкретного объекта или родового класса объектов – и само по себе оно не содержит информации о том, который из вариантов используется. Это различие проводится в большинстве языков программирования, когда конкретная величина называется значением, а общая – переменной. Значение где-то просто существует, а вот переменная в течение срока своей жизни связывается с одним или более значений. Поэтому механизм интерпретации обязан отслеживать связь переменной с конкретными значениями. Интерпретатор может находиться в мозгу человека или в компьютере.

Синтаксис переменных

Переменная – это просто удобный способ хранить что-либо, именованный ящик, в котором программист может найти нечто требующееся ему, когда оно позднее понадобится. Как и в реальной жизни, есть ячейки хранения разного типа: одни являются, по существу, тайными, а другие открыты для всех. Некоторые хранилища временные, а другие – более постоянные. Специалисты в области информатики любят обсуждать «области видимости» переменных, подразумевая под этим выражением именно то, что в нем сказано. В Perl имеются достаточно удобные средства решения проблем области видимости, с которыми читатель в будет иметь удовольствие познакомиться в должное время. Оно еще не наступило. (Лю-

---

[1] Пример многозначности слова «dog» и зависимости от контекста. Здесь оно используется в роли глагола; в роли существительного; в составе идиомы, которая вообще переводится без применения слова «собака» (dog days, дождливые дни или дни повышенной влажности); в составном слове dogcatcher, которое может означать сотрудника ведомства по надзору за дикими животными или выражать презрительное отношение к бестолковому политику; а также для описания сильной усталости (устал как собака). – *Прим. перев.*

Возможно, читатель устал как собака от всей этой лингвистической трескотни. Но мы хотим, чтобы было понятно, чем Perl, чтоб его собаки подрали, отличается от обычных компьютерных языков!

бопытные могут поискать прилагательные `local`, `my`, `our` и `state` в главе 27 либо заглянуть в раздел «Объявления, ограниченные областью видимости» главы 4.)

В данный момент полезнее провести классификацию переменных по типам данных, которые они способны хранить. Как и в английском языке, в Perl основное различие проводится между данными в единственном и во множественном числе. Строки и числа являются данными в единственном числе, а списки строк или чисел – во множественном. (А когда мы доберемся до объектно-ориентированного программирования, вы обнаружите, что типичный объект выглядит единичным снаружи, но множественным изнутри – как группа студентов.) Переменную в единственном числе мы называем *скаляром*, а во множественном – *массивом*. Поскольку строка может храниться в скалярной переменной, можно написать такую, несколько более длинную (и снабженную комментариями) версию нашего первого примера:

```
my $phrase = "Howdy, world!\n"; # Создать новую переменную.
print $phrase;                  # Вывести переменную.
```

Прилагательное `my` сообщает интерпретатору Perl, что `$phrase` – совершенно новая переменная, поэтому ему не нужно пытаться искать переменную с этим именем. Обратите внимание, что нам не пришлось предварительно определять, какой тип имеет переменная `$phrase`. Символ `$` указывает Perl на то, что `phrase` представляет собой скалярную переменную, т.е. содержит единственное значение. Напротив, имя переменной-массива должно начинаться символом `@`. (Возможно, вам будет проще запомнить, что `$` – это стилизованная буква «s», от «scalar», тогда как @ – стилизованная «a», от «array».)[1]

В Perl имеются некоторые другие типы переменных с такими непривлекательными именами, как «хеш» (hash), «дескриптор» (handle) и «запись таблицы имен» (typeglob). Как и в случае скаляров и массивов, переменным этих типов предшествуют забавные символы, которые более широко известны как разыменовывающие префиксы. В табл. 1.1 приводится полный список разыменовывающих префиксов, которые могут вам встретиться:

Таблица 1.1. Типы переменных и области их использования

Тип	Символ	Пример	Что обозначает
Скаляр	`$`	`$cents`	Отдельное значение (число или строку)
Массив	`@`	`@large`	Список значений с числовым ключом
Хеш	`%`	`%interest`	Группа значений со строковым ключом
Подпрограмма	`&`	`&how`	Фрагмент кода Perl, который может быть вызван
Typeglob	`*`	`*struck`	Все, что имеет имя `struck`

Некоторые пуристы от языков программирования считают эти разыменовывающие префиксы причиной отвращения к Perl. Это поверхностный взгляд. У этих символов много достоинств, не последним из которых является возможность подстановки, или интерполяции (interpolation), этих символов в строки без всякого дополнительного синтаксиса. Кроме того, сценарии Perl легко читать (тем, кто

[1] Это упрощенное изложение настоящей истории разыменовывающих префиксов, которая будет рассказана в главе 2.

потрудился изучить Perl!), поскольку существительные отличаются от глаголов. В язык могут добавляться новые глаголы без ущерба для старых сценариев. (Мы уже говорили, что в Perl на этапе проектирования была заложена возможность развития.) И аналогия с существительными приведена не зря: есть масса примеров в английском и других языках, когда требуются грамматические маркеры существительных. Вот так мы думаем! (Как нам кажется.)

Единственное число

В приведенном выше примере мы видели, что скаляру можно присваивать новое значение с помощью оператора = подобно тому, как это делается во многих других языках программирования. Скалярным переменным можно присваивать скалярные значения любого вида: целые числа, числа с плавающей запятой, строки и даже такие эзотерические вещи, как ссылки на другие переменные или объекты. Есть много способов создания этих значений, которые должны присваиваться.

Как и в командной оболочке UNIX[1], можно использовать различные механизмы расстановки кавычек для создания значений разного вида. Двойные кавычки (double quotes) выполняют *интерполяцию переменных*[2] и *интерполяцию обратного слэша* (например, превращение \n в символ перевода строки), тогда как одиночные кавычки подавляют интерполяцию. А обратные кавычки (наклоненные влево) вызывают выполнение внешней программы и возврат ее выдачи, которая перехватывается как одна строка, содержащая все строки выдачи.

```
my $answer = 42;                # целое число
my $pi = 3.14159265;            # вещественное число
my $avocados = 6.02e23;         # экспоненциальная запись
my $pet = "Camel";              # строка
my $sign = "I love my $pet";    # строка с интерполяцией
my $cost = 'It costs $100';     # строка без интерполяции
my $thence = $whence;           # значение другой переменной
my $salsa = $moles * $avocados; # гастрохимическое выражение
my $exit = system("vi $file");  # числовой код результата выполнения команды
my $cwd = `pwd`;                # строка выдачи команды
```

И хотя мы еще не рассказали о необычных значениях, следует указать, что скаляры могут содержать ссылки на другие структуры данных, в том числе подпрограммы и объекты.

```
my $ary = \@myarray;            # ссылка на именованный массив
my $hsh = \%myhash;             # ссылка на именованный хеш
my $sub = \&mysub;              # ссылка на именованную подпрограмму

my $ary = [1,2,3,4,5];          # ссылка на анонимный массив
```

---

[1] Здесь и всюду, говоря «UNIX», мы подразумеваем любую операционную систему, похожую на UNIX, в том числе BSD, Mac OS X, Linux, Solaris, AIX и, конечно, UNIX.

[2] Иногда создатели сценариев командной оболочки называют это «подстановкой», но мы предпочитаем сохранить это слово для использования в Perl в других целях. Поэтому, пожалуйста, называйте это интерполяцией. Мы используем этот термин в текстологическом смысле («этот отрывок является гностической интерполяцией»), а не в математическом («данная точка на графике получена интерполяцией между двумя другими точками»).

```
my $hsh = {Na => 19, Cl => 35}; # ссылка на анонимный хеш
my $sub = sub { print $state }; # ссылка на анонимную подпрограмму
my $fido = Camel->new("Amelia");# ссылка на объект
```

Когда создается новая скалярная переменная, но до того как ей присвоено значение, она автоматически инициализируется значением `undef`, которое, как и следовало ожидать, означает «не определено» (undefined). В зависимости от контекста, неопределенное значение может интерпретироваться как пустое значение: `""` или `0`. В более общем случае, в зависимости от контекста использования, переменные автоматически интерпретируются как строки, числа или значения «true» (истина) или «false» (ложь), обычно называемые булевыми, или логическими значениями. Вспомните, как важен контекст в естественном языке. В Perl разные операторы предполагают получение в качестве параметров различного вида одиночных значений, и мы будем говорить, что данные операторы предоставляют для этих параметров скалярный контекст. Иногда мы будем более конкретно говорить, что оператор предоставляет числовой, строковый или логический контекст для этих параметров. (Ниже мы расскажем о списочном контексте, который противопоставляется скалярному.) Perl автоматически преобразует данные к виду, требуемому текущим контекстом, руководствуясь здравым смыслом. Предположим, например, что вы сказали следующее:

```
my $camels = '123';
print $camels + 1, "\n";
```

Первоначальным значением переменной `$camels` является строка, но она преобразуется в число, чтобы можно было прибавить к ней единицу, а затем обратно в строку, чтобы можно было вывести ее как `124`. Символ перевода строки, `"\n"`, тоже находится в строковом контексте, но поскольку он изначально является строкой, преобразование не требуется. Обратите, однако, внимание на двойные кавычки: использование одиночных кавычек, т.е. `'\n'`, привело бы к появлению строки из двух символов, а именно обратного слэша и «`n`», которая никак не может представлять перевод строки.

Итак, в некотором смысле, двойные и одиночные кавычки предоставляют еще один способ уточнения контекста. Интерпретация содержимого строки в кавычках зависит от того, какого вида кавычки используются. (Далее мы рассмотрим некоторые другие операторы, синтаксически работающие как кавычки, но использующие строку некоторым особым способом, например, для поиска по шаблону или подстановки. Все они тоже действуют как строки в двойных кавычках. Контекст *двойных кавычек* является в Perl «интерполирующим» и предоставляется многими операторами, которые совершенно не похожи на двойные кавычки.)

Аналогично ссылка ведет себя как ссылка в «разыменовывающем» контексте и как обычная скалярная величина в других случаях. Например, можно сказать так:

```
my $fido = Camel->new("Amelia");
if (not $fido) { die "dead camel"; }
$fido->saddle();
```

Здесь создается ссылка на объект `Camel`, которая помещается в переменную `$fido`. В следующей строке мы проверяем значение `$fido` как логической величины на равенство «true» и возбуждаем исключительную ситуацию (т.е. жалуемся, что величина не истинна); в данном случае это означает, что конструктор `Camel->new` не смог создать требуемый объект `Camel`. Однако в последней строке мы обращаемся

с $fido как со ссылкой, вызывая метод saddle() объекта, содержащегося в $fido и представляющего собой объект Camel, поэтому Perl ищет метод saddle() для объектов Camel. Подробнее об этом сказано ниже. Сейчас просто запомните, что в Perl важен контекст, поскольку благодаря ему Perl узнает о ваших потребностях, не ожидая явных указаний, необходимых во многих других языках программирования.

Множественное число

В некоторых типах переменных содержатся множественные значения, связанные вместе логически. В Perl есть два типа многозначных переменных: массивы и хеши. Во многих случаях они ведут себя как скаляры: новые можно создать, например, с помощью объявления my, и они автоматически инициализируются пустым значением. Но в отличие от скаляров, при присвоении значений многозначным переменным в правой части оператора присваивания контекст является *списочным*, а не скалярным.

Массивы и хеши также различаются между собой. Массив следует использовать, когда нужно найти значение по его порядковому номеру. Хеш применяется, чтобы найти что-либо по названию. Эти два понятия дополняют друг друга. Часто можно наблюдать использование массива для перевода номера месяца в название и соответствующего хеша для перевода названия месяца обратно в номер. (Однако хеши могут содержать не только числа. К примеру, можно создать хеш, который будет переводить названия месяцев в названия камней, соответствующих месяцам.)

Массивы. *Массив* – это упорядоченный список скаляров, к которым обращаются по их местоположению в списке[1]. Список может содержать числа, строки или смесь того и другого. (Он может также содержать ссылки на вложенные массивы (субмассивы) и вложенные хеши (субхеши).) Чтобы присвоить массиву значение в виде списка, нужно просто объединить значения с помощью круглых скобок:

```
my @home = ("кушетка", "стул", "стол", "печка");
```

И наоборот, используя @home в списочном контексте, например, в правой части присваивания списку, мы получим на выходе тот самый список, который ввели. Например, присвоить значения, взятые из массива, четырем скалярным переменным можно так:

```
my ($potato, $lift, $tennis, $pipe) = @home;
```

Это называется списочным присваиванием. Логически оно происходит параллельно, поэтому можно обменять значениями две переменные с помощью такой фразы:

```
($alpha,$omega) = ($omega,$alpha);
```

Как и в языке C, нумерация элементов массивов начинается с нуля, поэтому, работая с элементами массива с первого по четвертый, следует обращаться к ним по индексам от 0 до 3.[2] Индексы массивов заключаются в квадратные скобки,

---

[1] Или по ключу (порядковому номеру, нижнему индексу), определяющему местоположение. Выберите вариант по душе.

[2] Если это кажется вам непривычным, считайте индекс смещением, т.е. числом предшествующих элементов массива. Очевидно, перед первым элементом нет других элементов, поэтому его смещение равно 0. Так уж думают компьютеры. (Как нам кажется.)

[вот такие], поэтому при выборе отдельного элемента массива нужно ссылаться на него так – $home[*n*], где *n* является значением индекса (на единицу меньшим номера элемента), который нам нужен. Посмотрите на следующий пример. Поскольку мы имеем дело со скалярным элементом, ему обязательно должен предшествовать символ $.

Если вы хотите присваивать значения элементам массива по одному, приведенное выше списочное присваивание можно записать так:

```
my @home;
$home[0] = "кушетка";
$home[1] = "стул";
$home[2] = "стол";
$home[3] = "печка";
```

Как видно из примера, переменные можно создавать с помощью директивы my, не присваивая начальное значение. (Для отдельных элементов массива директива my не используется, потому что массив уже существует и он сам знает, как создавать элементы по мере необходимости.)

Поскольку массивы упорядочены, над ними можно выполнять различные полезные действия, такие как стековые операции push и pop. Стек, в сущности, является просто упорядоченным списком с началом и концом. Что особенно важно, с концом. Perl рассматривает конец массива как вершину стека. (Хотя большинство программистов на Perl представляют себе массивы горизонтальными последовательностями, у которых вершина стека справа.)

Хеши. *Хеш* представляет собой неупорядоченный набор скаляров, к которым обращаются по некоторому строковому значению[1], связанному с каждым скаляром. По этой причине хеши часто называют *ассоциативными массивами*. Но это слишком длинное название для ввода с клавиатуры, а упоминаются ассоциативные массивы столь часто, что мы решили дать им какое-нибудь краткое и броское название. Другой причиной, по которой мы выбрали название «хеш», было желание подчеркнуть тот факт, что подобные массивы не упорядочены. (Так совпало, что в их внутренней реализации используется поиск по хеш-таблице, в результате чего работа с ними происходит столь быстро вне зависимости от числа хранимых значений.) Однако применить к хешу операцию push или pop нельзя: это не имеет смысла. У хеша нет начала и конца. Тем не менее хеши – это невероятно мощный и полезный инструмент. Пока вы не начали думать на языке хешей, вы, в сущности, не начали думать на Perl. На рис. 1.1 показаны упорядоченные элементы массива и неупорядоченные (но поименованные) элементы хеша.

Поскольку ключи для хеша не задаются автоматически порядком значений, при заполнении хеша необходимо задавать как значение, так и ключ. Можно присваивать хешу список, так же как и массиву, но при этом каждая *пара* элементов списка будет интерпретироваться как ключ и значение. Поскольку мы работаем с парами элементов, имена хешей помечаются разыменовывающим префиксом %. (Если внимательно посмотреть на символ %, можно заметить ключ и значение, разделенные косой чертой. Если не получается, попробуйте прищуриться.)

---

[1] Или по ключу (порядковому номеру, нижнему индексу), определяющему местоположение. Выберите вариант по душе.