От основ к мастерству

Предисловие Дэмиана Конвея

Изучаем глубже Perl

Символ®
O'REILLY®

Рэндал Л. Шварц,
Брайан Д. Фой и Том Феникс

Intermediate Perl

Second Edition

Randal L. Schwartz, brian d foy
& Tom Phoenix

O'REILLY®

Perl
изучаем глубже

Второе издание

Рэндал Л. Шварц, Брайан Д. Фой
и Том Феникс

Санкт-Петербург — Москва
2008

Рэндал Л. Шварц, Брайан Д. Фой и Том Феникс

Perl: изучаем глубже, 2-е издание

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*О. Цилюрик*
Редактор	*В. Овчинников*
Корректура	*О. Макарова*
Верстка	*Д. Орлова*

Шварц Р., Фой Б., Феникс Т.
Perl: изучаем глубже, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 320 с., ил.
ISBN-13: 978-5-93286-093-9
ISBN-10: 5-93286-093-6

Книга «Perl: изучаем глубже» – продолжение мирового бестселлера «Learning Perl» («Изучаем Perl»), известного под названием «Лама». Издание поможет вам перешагнуть грань, отделяющую любителя от профессионала, и научит писать на Perl настоящие программы, а не разрозненные сценарии. Материал изложен компактно и в занимательной форме, главы завершаются упражнениями, призванными помочь закрепить полученные знания. Рассмотрены пакеты и пространства имен, ссылки и области видимости, создание и использование модулей. Вы научитесь с помощью ссылок управлять структурами данных произвольной сложности, узнаете, как обеспечить совместимость программного кода, написанного разными программистами. Уделено внимание и ООП, которое поможет повторно использовать части кода. Обсуждаются создание дистрибутивов, аспекты тестирования и передача собственных модулей в CPAN.

Книга адресована широкому кругу программистов, знакомых с основами Perl и стремящихся повысить свою квалификацию как в написании сценариев, так и в ООП, и призвана помочь им научиться писать эффективные, надежные и изящные программы.

ISBN-13: 978-5-93286-093-9
ISBN-10: 5-93286-093-6
ISBN 0-596-10206-2 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 19.10.2007. Формат 70×100 1/16 . Печать офсетная.
Объем 20 печ. л. Тираж 2000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Вступительное слово

Объектно-ориентированный механизм языка программирования Perl являет собой пример ловкости рук без всякого обмана. Он берет набор возможностей, не связанных с объектно-ориентированным стилем программирования, таких как пакеты, ссылки, хеши, массивы, подпрограммы и модули, и с помощью несложных заклинаний превращает их в полнофункциональные объекты, классы и методы.

Благодаря этому трюку программист, основываясь на своих познаниях языка Perl, может легко и просто перейти к объектно-ориентированному стилю программирования, не преодолевая горы нового синтаксиса и не переплывая океаны новых технологий. Это также означает, что объектно-ориентированные возможности языка Perl можно осваивать постепенно, по мере необходимости отбирая те, что наилучшим образом подходят для решения поставленных задач.

Однако здесь кроется одна проблема. Поскольку все эти пакеты, ссылки, хеши, массивы, подпрограммы и модули составляют основу объектно-ориентированного механизма, для использования объектно-ориентированных возможностей языка Perl необходимо знать принципы работы с пакетами, ссылками, хешами, подпрограммами и модулями.

Трудность именно в этом. Кривая обучения не исчезла, она всего лишь сократилась на полдесятка шагов.

Какие же *не*объектно-ориентированные возможности языка Perl надо изучить, чтобы можно было взяться за объектно-ориентированные?

Ответу на этот вопрос и посвящена данная книга. На ее страницах Рэндал, опираясь на 20-летний опыт работы с языком Perl и 40-летний опыт просмотра фильмов «Остров Джиллигана» и «Мистер Эд», описывает компоненты языка Perl, которые все вместе составляют фундамент его объектно-ориентированных возможностей. И, что еще лучше, на примерах показывает, как из этих компонентов создавать классы и объекты.

Итак, если объекты языка Perl вызывают у вас чувства, подобные тем, которые испытал Джиллиган на необитаемом острове, эта книга – как раз то, что доктор прописал.

Кроме того, вся информация в ней прямо из первых рук.

– Дэмиан Конвей (Damian Conway), май 2003

Предисловие

Более десяти лет тому назад (практически вечность по меркам Интернета) Рэндал Шварц написал первое издание книги «Learning Perl»[1]. За прошедшие годы сам Perl из «крутого» языка сценариев, используемого в первую очередь системными администраторами UNIX, вырос в полноценный объектно-ориентированный язык программирования, способный функционировать на практически любой платформе, известной человечеству.

Объем всех четырех изданий «Learning Perl» оставался практически неизменным (примерно 300 страниц), как в основном неизменным оставался и ее материал, рассчитанный на начинающих программистов. Однако времена изменились, и теперь о Perl можно рассказать значительно больше, чем когда появилось первое издание книги.

Рэндал назвал первое издание книги «Learning Perl Objects, References, and Modules», а сейчас книга получила название «Intermediate Perl» (Perl средней сложности), но на наш взгляд ей больше подошло бы название «Learning More Perl» (Изучаем Perl глубже)[2]. Данная книга продолжает обсуждение тем с того места, где оно было закончено в книге «Learning Perl». Здесь мы покажем вам, как писать большие программы на языке Perl.

Как и в «Learning Perl», мы старались сделать каждую главу настолько маленькой, чтобы ее можно было прочитать за час-другой. Каждая глава заканчивается серией упражнений, которые помогут вам на практических примерах закрепить только что прочитанный материал. Кроме того, в конце книги вы найдете приложение, в котором содержатся решения всех упражнений. Как и в «Learning Perl», материал этой книги подается в той же последовательности, что и в курсах обучения языку Perl, которые проводятся нами в компании Stonehenge Consulting Services.

---

[1] Рэндал Шварц, Том Кристиансен «Изучаем Perl». – Пер. с англ. – BHV-Киев, 1999.

[2] Не спрашивайте, почему книга не была названа именно так. Мы получили более 300 предложений по этой теме. На самом деле невозможно прекратить изучение Perl, поэтому название «Изучаем Perl глубже» фактически ничего не говорит о книге. Наш редактор выбрал название, которое говорит о том, чего следует ожидать от книги.

Чтобы извлечь максимум пользы из этой книги, вам необязательно быть гуру в UNIX и даже необязательно быть пользователем UNIX. Все, о чем говорится в этой книге, одинаково хорошо подходит как для Windows ActivePerl, так и для любой другой современной реализации. Чтобы иметь возможность пользоваться этой книгой, вам необходимо ознакомиться с книгой «Learning Perl» и гореть желанием продолжать двигаться вперед.

Структура книги

Данную книгу следует читать, начиная с первых глав, в том порядке, в каком они следуют, останавливаясь для выполнения упражнений. Материал каждой главы основан на предыдущих главах, и при обсуждении новой темы мы будем исходить из предположения, что предыдущие главы уже были вами прочитаны.

Глава 1 «Введение»

Содержит вводные положения.

Глава 2 «Основы»

Описывает некоторые промежуточные положения, знание которых потребуется при прочтении оставшейся части книги.

Глава 3 «Модули»

Описывает порядок работы с основными модулями Perl и с модулями сторонних производителей. Позже в этой же книге мы покажем, как создавать собственные модули, но в данной главе мы остановимся на использовании существующих модулей.

Глава 4 «Введение в ссылки»

Рассказывает о том, как организовать перенаправление, чтобы один и тот же программный код мог работать с различными наборами данных.

Глава 5 «Ссылки и области видимости»

Рассказывает о том, как Perl работает с указателями на данные, и дает краткое введение в анонимные структуры данных и автовивификацию.

Глава 6 «Управление сложными структурами данных»

Описывает создание структур данных с произвольной глубиной вложенности, включая массивы массивов и хеши хешей, обращение к ним и вывод их содержимого.

Глава 7 «Ссылки на подпрограммы»

Описывает поведение анонимных подпрограмм, которые могут создаваться динамически для последующего использования.

Глава 8 «Ссылки на дескрипторы файлов»

Описывает, как можно хранить дескрипторы файлов в скалярных переменных для передачи между различными частями программы или для сохранения в структурах данных.

Глава 9 «Практические приемы работы со ссылками»

Сложные операции сортировки, преобразование Шварца и работа с рекурсивно определенными данными.

Глава 10 «Разработка больших программ»

Рассматривает вопросы создания больших программ из нескольких файлов с программным кодом, разнесенным по разным пространствам имен.

Глава 11 «Введение в объекты»

Работа с классами, вызов методов, наследование и переопределение.

Глава 12 «Объекты и данные»

Экземпляры данных, конструкторы и методы доступа.

Глава 13 «Уничтожение объектов»

Описывает поведение объектов при уничтожении, включая объекты, существующие постоянно.

Глава 14 «Дополнительные сведения об объектах»

Множественное наследование, автоматические методы и ссылки на дескрипторы файлов.

Глава 15 «Экспортирование»

Как работает директива `use`, как определить, что нужно экспортировать, и как создать собственную процедуру импорта.

Глава 16 «Создание дистрибутивов»

Описывает порядок создания модулей, готовых к распространению, включая платформонезависимые инструкции по установке.

Глава 17 «Основы тестирования»

Описывает порядок тестирования программного кода с целью проверки его функциональности.

Глава 18 «Дополнительные сведения о тестировании»

Описываются более сложные аспекты тестирования программного кода и метаданных, такие как документация и покрытие тестами.

Глава 19 «Передача модулей в CPAN»

Описывает, как можно отправить свои разработки в CPAN.

Приложение А содержит решения всех упражнений.

Типографские соглашения

В книге приняты следующие соглашения по оформлению текста:

`Моноширинным шрифтом`

Выделены имена функций, модулей, файлов, переменных окружения, фрагменты программного кода и пр.

Курсивом

Выделены наиболее важные моменты и вновь вводимые термины.

Примеры программного кода

Данная книга призвана помочь вам в работе. Вы можете вставлять примеры программного кода из этой книги в свои приложения и в документацию, и для этого не надо обращаться в издательство O'Reilly за разрешением. Например, если вы пишете программу и заимствуете несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Не требуется разрешение и для цитирования данной книги или примеров из нее при ответе на вопросы.

Если же вы собираетесь воспроизводить значительные фрагменты программного кода из этой книги (например, в документации), то разрешение необходимо. Разрешение нужно получить и в случае, если вы планируете продавать или распространять компакт-диски с примерами из этой книги.

Мы не требуем добавлять ссылку на первоисточник при цитировании (но совсем не против этого). Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN, например «Intermediate Perl, by Randal L. Schwartz, Brian D. Foy, and Tom Phoenix. Copyright 2006 O'Reilly Media, Inc., 0-596-10206-2».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу *permissions@oreilly.com*.

Отзывы и предложения

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в Соединенных Штатах Америки или в Канаде)
(707) 829-0515 (международный)
(707) 829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на сайте книги:

http://www.oreilly.com/catalog/intermediateperl

Свои пожелания и вопросы технического характера отправляйте по адресу:

bookquestions@oreilly.com

Дополнительную информацию о книгах, обсуждения, центр ресурсов издательства O'Reilly вы найдете на сайте:

http://www.oreilly.com

Safari Enabled

Если на обложке книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности

Рэндал. В предисловии к первому изданию книги «Learning Perl» я выразил свою признательность Бивертону Мак-Менамину (Beaverton McMenamin) – владельцу паба Cedar Hills (Кедровые холмы), что находится рядом с моим домом, за бесплатно предоставленный кабинет, где я имел обыкновение писать черновики книги на моем Powerbook 140. Этот паб стал для меня талисманом, приносящим удачу. Практически все свои книги (включая и эти слова) я писал здесь и очень надеюсь, что удача мне не изменит и на этот раз!

Теперь в этом пабе подают прекрасное пиво, сваренное тут же в маленькой пивоварне, и сладкие сэндвичи, но пропала моя любимая хлебная пицца, которую заменили ежевичным коктейлем (местный рецепт) и острой джамбалайей. (Кроме того, здесь появились два новых кабинета и несколько столов.) Ну а вместо Powerbook 140 у меня теперь более современный Titanium Powerbook, у которого диск в 1000 раз больше, оперативной памяти в 500 раз больше и процессор в 200 раз быстрее. На нем установлена полноценная UNIX-подобная операционная система (OS X) вместо ограниченной версии Mac OS. Все свои черновики (включая и этот) я отправляю через модем сотового телефона на скорости 144 K, могу напрямую общаться со своими рецензентами, и мне не нужно возвращаться домой к моему модему, передающему данные по телефонной линии со скоростью 9600 бод. Как изменились времена!

Еще раз большое спасибо всем, кто работает в «Кедровых холмах», за их неизменное гостеприимство.

Как и в четвертом издании книги «Learning Perl», я должен отметить, что многим обязан своим студентам из Stonehenge Consulting Services за их каверзные (?) вопросы, которые появлялись, когда сложность материала превышала уровень их подготовки. Благодаря им я смог продолжить совершенствование материала, который лег в основу этой книги.

Следует заметить, что все началось с полудневного курса «Что нового в Perl 5?» Марджи Левин (Margie Levine) из Silicon Graphics, а также моего собственного четырехдневного курса «Лама» (Llama) (в то время основанного на Perl версии 4). Со временем у меня возникла идея превратить эти короткие заметки в полноценный курс и подтолкнуть сотрудника компании Stonehenge Джозефа Холла (Joseph Hall) к участию в решении этой задачи. (Он один из тех, кто отбирал примеры программного кода для курса.) Джозеф разработал двухдневный курс для Stonehenge и одновременно написал прекрасную книгу «Effective Perl Programming», которая затем стала использоваться как учебник.

За эти годы в разработке курса «Пакеты, ссылки, объекты и модули» принимали участие многие преподаватели из Stonehenge, включая Чипа Зальценберга (Chip Salzenberg) и Тэда Мак-Клеллана (Ted McClellan). Но большая часть изменений и дополнений была внесена Томом Фениксом (Tom Phoenix), который становился «служащим месяца» в компании Stonehenge настолько часто, что мне, наверное, придется уступить ему мое привилегированное место на парковке. Том отлично управляется с материалами (как Тэд управляется с делами), благодаря чему я могу спокойно сосредоточиться на своих обязанностях президента и дворника компании Stonehenge.

Том Феникс написал большую часть примеров для этой книги и своевременно предоставлял свои рецензии во время моей работы над книгой. Его рукой были написаны целые абзацы, так что мне оставалось только вставлять их на место той бессмыслицы, что была написана мною. У нас получилась отличная команда, которая дружно работает и в аудитории, и над книгой. Именно за приложенные усилия мы признали Тома соавтором, но я готов взять всю вину на себя, если какая-либо часть книги вам не понравится, потому что, скорее всего, в этом нет вины Тома.

И последний, но не в последнюю очередь, кому я хотел бы выразить свою благодарность, – это Брайан Д. Фой (brian d foy), который примкнул к работе над книгой, начиная со второго ее издания, и предложил массу изменений и дополнений к этому изданию.

Разумеется, книга не состоялась бы, не будь темы для обсуждения и каналов распространения, поэтому я хочу выразить свою признательность Ларри Уоллу (Larry Wall) и Тиму О'Рейли (Tim O'Reilly).

Спасибо вам, ребята, за то что вы создали компанию, которая оплачивала мне мои затраты в течение 15 последних лет.

И, как обычно, отдельное спасибо Лейле и Джеку, которые научили меня всему, что я знаю о писательской деятельности, и убедили меня в том, что я должен быть чуть большим (?), чем программист, который умеет писать. Благодаря им я стал писателем, который умеет программировать. Спасибо вам.

Спасибо и вам, уважаемый читатель. Именно для вас я трудился долгие часы, потягивая холодное пиво и поедая пудинг, стараясь не залить клавиатуру моего ноутбука. Спасибо вам за то, что вы читаете мою книгу. Я искренне надеюсь, что внес свой вклад (пусть и незначительный) в ваше образование. Если вы встретите меня когда-нибудь на улице, просто скажите: «Привет!».[1] Мне это будет приятно. Спасибо вам.

Брайан. В первую очередь я хотел бы сказать спасибо Рэндалу, так как впервые я познакомился с Perl благодаря первому изданию его книги «Learning Perl» и многое узнал, преподавая курсы «Лама» и «Альпака» в компании Stonehenge Consulting. Учить других – часто лучший способ научиться самому.

Мне удалось убедить Рэндала в необходимости обновить книгу «Learning Perl», а когда эта работа была закончена, я заметил ему, что пора обновить и эту книгу. Наш редактор Элисон Рэндал (Allison Randal) согласилась с этим и приложила максимум усилий, чтобы не нарушить наш график.

Отдельное спасибо Стейси (Stacey), Бастеру (Buster), Мими (Mimi), Роско (Rosco), Амелии (Amelia), Лиле (Lila) и всем тем, кто пытался отвлечь меня от работы, когда я был занят.

От нас обоих. Спасибо нашим рецензентам: Дэвиду Адлеру (David H. Adler), Стефену Дженкинсу (Stephen Jenkins), Кевину Мельтцеру (Kevin Meltzer), Мэттью Масгрову (Matthew Musgrove), Эндрю Сэвиджу (Andrew Savige) и Риккардо Сигнесу (Ricardo Signes) за их комментарии к рукописи этой книги.

Спасибо нашим студентам, которые помогли нам понять, какие части курса необходимо пересмотреть и дополнить. Именно благодаря вам мы испытываем чувство гордости за свою работу.

Спасибо членам группы Perl Mongers, кто принимал нас в своих городах как родных. Давайте встретимся еще когда-нибудь.

И наконец, огромное спасибо Ларри Уоллу за его большие и мощные игрушки, которые позволили нам сделать свою работу намного быстрее, проще и с увлечением.

---

[1] К тому же вы можете спросить меня что-нибудь о Perl. Я не возражаю.

1

Введение

Добро пожаловать в следующий этап изучения языка программирования Perl. Вероятно, вы здесь или для того, чтобы научиться писать программы длиннее 100 строк, или по принуждению своего босса.

Наша предыдущая книга «Learning Perl» была такой большой, потому что она представляет собой введение в язык программирования Perl и его использование для написания маленьких и средних программ (которые, по нашим наблюдениям, составляют большую часть кода, написанного на языке Perl). Чтобы избежать увеличения объема книги под кодовым названием «Лама», мы намеренно и очень аккуратно оставили за бортом довольно много информации.

На следующих страницах вы найдете «продолжение истории», изложенной в том же дружественном стиле. Она содержит сведения, знать которые совершенно необходимо, если вы собираетесь писать программы длиной от 100 до 10 000 строк.

Например, вы узнаете, как организовать коллективную работу над проектом. Это просто здорово, потому что если вы не собираетесь работать по 35 часов каждый день, вам наверняка потребуется помощь при решении крупных задач. Вам также придется обеспечивать совместимость программного кода, написанного различными программистами, чтобы свести результаты коллективного труда в единое приложение.

В этой книге также показано, как работать с большими и сложными структурами данных, которые мы небрежно называем «хешами хешей», или «массивами массивов хешей массивов». Познакомившись со ссылками поближе, вы сможете управлять структурами данных произвольной сложности.

Затем мы перейдем к заслуживающим внимания понятиям объектно-ориентированного программирования (ООП), которое поможет вам повторно использовать части своего (а может быть, и чужого) программно-

го кода с минимальными переделками. Вы без труда разберетесь в этой теме, даже если никогда раньше не сталкивались с объектами.

Немаловажным аспектом коллективной разработки является цикличность выпуска новых версий и разработка тестов для проведения модульного и интеграционного тестирования. Здесь вы получите основные сведения о создании дистрибутивных пакетов и разработке модульных тестов, которые могут применяться как в процессе работы над приложением, так и для окончательного тестирования готового продукта.

И наконец, точно так же, как и в предыдущих изданиях «Learning Perl», мы будем развлекать вас интересными примерами и плохими каламбурами. (Мы отправили-таки Фреда (Fred), Барни (Barney), Бетти (Betty) и Уилму (Wilma) по домам. А на новые роли пригласили звезд.)

Что вы должны знать?

Предполагается, что вы уже прочитали книгу «Learning Perl» или по крайней мере делаете вид, что уже достаточно наигрались с Perl и обладаете базовыми знаниями. В этой книге, например, не рассказывается, как обращаться к элементам массива или как вернуть некоторое значение из подпрограммы.

Как минимум вы должны знать:

- Как запускать в своей системе программы, написанные на Perl
- Три основных типа переменных в Perl: скаляры, массивы и хеши
- Конструкции управления ходом исполнения, такие как `while`, `for` и `foreach`
- Что такое подпрограммы
- Операторы языка Perl, такие как `grep`, `map`, `sort` и `print`
- Функции работы с файлами, такие как `open`, функции чтения из файлов и `-X` (тестирование файлов)

В этой книге вы сможете получить более глубокие знания по данным темам, но мы полагаем, что основы вы уже знаете.

Как быть со сносками?

Как и в книге «Learning Perl», некоторые дополнительные сведения, знание которых не обязательно при первом прочтении, оформлены в виде сносок.[1] При первом прочтении их можно пропустить, но при повторном было бы желательно прочитать и их. В сносках нет ничего такого, что было бы необходимо для понимания остального материала.

[1] Таких, как эта.

Как быть с упражнениями?

Тренировки помогают усвоить материал. А лучший способ потренироваться – выполнить несколько упражнений после каждого получасового изучения очередной темы. Разумеется, если вы читаете достаточно быстро, то до конца главы доберетесь немного быстрее, чем за полчаса. Приостановитесь, сделайте передышку и выполните упражнения!

Каждое упражнение едва ли займет больше пары минут. Это время соответствует середине колоколообразной кривой, однако будет совсем неплохо, если у вас это займет чуть больше или чуть меньше времени. Иногда время решения упражнений зависит лишь от того, насколько часто вам приходилось сталкиваться с решением подобных задач. Так что это время – всего лишь ориентир.

Ответы к упражнениям приведены в приложении. Однако попробуйте не заглядывать туда, иначе вы снизите обучающую ценность упражнения.

Что делать, если я преподаю Perl?

Если вы преподаете язык программирования Perl и решили использовать эту книгу в качестве учебного пособия, то должны понимать, что каждый набор упражнений слишком короток для большинства студентов, чтобы занять их на полные 45 минут. Одни упражнения отнимут больше времени, другие меньше. Мы обнаружили этот факт лишь после того, как расставили эти маленькие числа в квадратных скобках.

Итак, приступим. Обучение начнется, как только вы перевернете страницу...

2

Основы

Прежде чем приступить к изучению основного материала, рассмотрим некоторые понятия языка Perl, которыми мы будем оперировать в этой книге и которые обычно не попадают в фокус внимания начинающих программистов. Попутно мы представим персонажей, которые встретятся нам в этой книге.

Операторы списков

Вы уже наверняка знаете несколько операторов Perl, предназначенных для работы со списками, но скорее всего вы и предположить не могли, что эти операторы работают именно со списками. Из них чаще остальных, пожалуй, применяется оператор `print`. Он может принимать один или несколько аргументов и связывает их в единое целое.[1]

```
print 'Кораблекрушение потерпели двое ', 'Джиллиган', ' и ', 'Шкипер', "\n";
```

Для работы со списками предназначен еще целый ряд операторов, о которых вы уже узнали из книги «Learning Perl». Оператор `sort` упорядочивает входной список. В известной песне[2] потерпевшие кораблекрушение упоминаются не в алфавитном порядке, однако мы можем исправить этот недостаток с помощью оператора `sort`.

```
my @castaways = sort qw(Джиллиган Шкипер Джинджер Профессор Мери-Энн);
```

---

[1] В этой книге текстовые константы в программном коде переведены на русский язык. Работоспособность кода была проверена в трех операционных системах – Windows 98, Linux Mandriva 2006 и QNX 6.3.0 – в тех реализациях Perl 5.xx, которые были последними для каждой из ОС на момент перевода книги. – *Примеч. науч. ред.*

[2] В балладе об острове Джиллигана («The Ballad of Gilligan's Isle»), написанной Джорджем Уайлом (George Wyle) и Шервудом Шварцем (Sherwood Schwartz).

Оператор reverse возвращает список в обратном порядке.

```
my @castaways = reverse qw(Джиллиган Шкипер Джинджер Профессор Мери-Энн);
```

В языке Perl имеется масса других операторов, которые работают со списками, и, как только вы научитесь использовать их, вы обнаружите, что объем ввода с клавиатуры уменьшился, а ваши намерения стали выражаться более ясно.

Фильтрация списков с помощью grep

Оператор grep принимает список значений и «условное выражение». Он извлекает из списка одно значение за другим и помещает их в переменную $_. После этого производится вычисление условного выражения в скалярном контексте. Если в результате получается «истина», grep передает значение $_ в выходной список.

```
my @lunch_choices = grep &is_edible($_), @gilligans_posessions.
```

В списочном контексте оператор grep возвращает список всех прошедших проверку элементов, а в скалярном – количество отобранных элементов.

```
my @results = grep EXPR, @input_list;
my $count = grep EXPR, @input_list;
```

В данном случае EXPR означает любое скалярное выражение, которое должно выполнить проверку значения переменной $_ (явно или неявно). Например, чтобы отыскать все числа больше 10, в условном выражении можно сравнить переменную $_ со значением 10.

```
my @input_numbers = (1, 2, 4, 8, 16, 32, 64);
my @bigger_than_10 = grep $_ > 10, @input_numbers;
```

В результате будет получена последовательность чисел 16, 32 и 64. Здесь имеет место явное обращение к переменной $_. В следующем примере показано косвенное обращение к этой переменной из оператора поиска по шаблону:

```
my @end_in_4 = grep /4$/, @input_numbers;
```

Теперь мы получим числа 4 и 64.

Оператор grep запоминает оригинальное значение переменной $_ и восстанавливает его по окончании работы. Переменная $_ – это не просто копия элемента данных, на самом деле ее можно рассматривать как псевдоним фактического элемента, чем-то похожего на переменную цикла foreach.

Если условное выражение достаточно сложное, его можно оформить в виде подпрограммы:

```
my @odd_digit_sum = grep digit_sum_is_odd($_), @input_numbers;

sub digit_sum_is_odd {
    my $input = shift;
```

```
    my @digits = split //, $input; # Предполагается, что элемент списка
                                   # содержит только цифровые символы
    my $sum;
    $sum += $_ for @digits;
    return $sum % 2;
}
```

Теперь мы получим список, содержащий числа 1, 16 и 32. Суммы цифр этих чисел при делении на 2, выполняемом в последней строке подпрограммы, дают остаток 1, что расценивается оператором `grep` как «истина».

Синтаксис оператора `grep` имеет две формы. Только что мы продемонстрировали форму выражения, а сейчас покажем блочную. Применяя такую форму записи вместо явного определения подпрограммы, которая служит для проверки в единственном месте, мы можем поместить тело подпрограммы прямо в оператор `grep`:[1]

```
my @results = grep {
    блок;
    программного;
    кода;
} @input_list;

my $count = grep {
    блок;
    программного;
    кода;
} @input_list;
```

Точно так же, как и при записи в форме выражения, оператор `grep` на время помещает очередной элемент входного списка в переменную `$_`. Затем исполняется блок программного кода. Результат последнего выражения в блоке определяет конечный результат выполнения всего блока. (Как и любое другое условное выражение, оно вычисляется в скалярном контексте.) Поскольку теперь у нас имеется целый блок, мы можем внутри него работать с переменными, область видимости которых будет ограничена данным блоком. Перепишем предыдущий пример, придерживаясь блочного синтаксиса:

```
my @odd_digit_sum = grep {
    my $input = $_;
    my @digits = split //, $input; # Предполагается, что элемент списка
                                   # содержит только цифровые символы
    my $sum;
    $sum += $_ for @digits;
    $sum % 2;
  } @input_numbers;
```

[1] В блочной форме записи символ запятой между блоком и входным списком не ставится. Когда оператор `grep` записывается в форме выражения, запятая между условным выражением и списком ставится обязательно.

Обратите внимание на два отличия: входные значения теперь извлекаются из переменной $_, а не из списка переданных аргументов, а кроме того, мы убрали оператор return. Оставить его было бы ошибкой, потому что теперь мы находимся уже не внутри подпрограммы, а в блоке кода.[1] Разумеется, мы можем оптимизировать этот блок, отказавшись от промежуточных переменных:

```
my @odd_digit_sum = grep {
    my $sum;
    $sum += $_ for split //;
    $sum % 2;
  } @input_numbers;
```

Не бойтесь описывать ход исполнения более явно, если это поможет вам и вашим коллегам точнее понимать и сопровождать программный код. Это главный приоритет.

Преобразование списка с помощью оператора map

Оператор map обладает похожим синтаксисом, и во многом его действия напоминают оператор grep. Например, он временно помещает элементы списка друг за другом в переменную $_ и допускает две формы записи: в форме выражения и в форме блока.

Однако это *выражение отображения* (*mapping expression*), а не условное выражение. Выражение оператора map вычисляется в списочном контексте (а не в скалярном, как в grep). Каждое обращение к выражению дает часть полного результата. Полный результат представляет собой список из частных результатов. В скалярном контексте map возвращает количество элементов, возвращаемых в списочном контексте. Однако оператор map очень редко встречается в скалярном контексте (если вообще встречается).

Начнем с простого примера:

```
my @input_numbers = (1, 2, 4, 8, 16, 32, 64);
my @result = map $_ + 100, @input_numbers;
```

Оператор map поместит каждый из семи элементов списка в переменную $_, в результате мы получим список, где каждому входному числу будет соответствовать единственный результат – число, которое больше исходного на 100. Таким образом, в списке @result будут находиться числа 101, 102, 104, 108, 116, 132 и 164.

Однако каждому элементу входного списка может соответствовать и несколько элементов в выходном списке. Посмотрим, что произойдет, если в результате обработки каждого входного элемента будут получаться два выходных элемента:

---

[1] Вызов оператора return привел бы к выходу из подпрограммы, которая содержит данный блок кода. Некоторые из нас сталкивались с этой ошибкой в своих первых работах.

```
my @result = map { $_, 3 * $_ } @input_numbers;
```

Теперь каждому входному элементу у нас соответствуют два выходных: 1, 3, 2, 6, 4, 12, 8, 24, 16, 48, 32, 96, 64 и 192. Мы можем сохранить эти пары чисел в виде хеша, если потребуется определить значения чисел в три раза больших степеней двойки:

```
my %hash = @result;
```

Или, если обойтись без промежуточных переменных:

```
my %hash = map { $_, 3 * $_ } @input_numbers;
```

Как видите, оператор `map` отличается высокой степенью гибкости – он позволяет создать для каждого входного элемента произвольное число выходных элементов. При этом для каждого входного элемента число выходных элементов может быть и разным. Посмотрим, что произойдет, если мы попытаемся разбить числа на отдельные цифры:

```
my @result = map { split //, $_ } @input_numbers;
```

Внутри блока числа разделяются на цифры. Для чисел 1, 2, 4 и 8 мы получим по одному результату. Для чисел 16, 32 и 64 – по два. Когда оператор `map` объединит результаты, получится список 1, 2, 4, 8, 1, 6, 3, 2, 6 и 4.

Если в результате вычисления выражения для некоторого элемента входного списка получается пустой список, оператор `map` просто не вставляет его в конечный результат. Эту особенность можно использовать для выборки определенных элементов. Допустим, что нам надо отобрать только отделенные (см. выше) цифры, если последняя из них 4:

```
my @result = map {
    my @digits = split //, $_;
    if ($digits[-1] == 4) {
        @digits;
    } else {
        ( );
    }
} @input_numbers;
```

Если она равна 4, то в результате оценки выражения `@digits` (в списочном контексте) возвращаются отделенные цифры. Если же не равна, возвращается пустой список, и результат для заданного входного элемента удаляется. Таким образом, оператор `map` может применяться вместо оператора `grep`, но не наоборот.

Разумеется, все, что можно сделать с помощью операторов `grep` и `map`, можно сделать и с помощью циклов `foreach`. Но точно так же можно программировать и на ассемблере или даже в машинных кодах.[1] Дело не в том, как можно реализовать тот или иной алгоритм, а в том, что

[1] Если вы достаточно давно работаете с вычислительной техникой, чтобы знать о существовании машинных кодов.

операторы grep и map позволяют уменьшить сложность программы и сконцентрироваться на решении задач высокого уровня, не отвлекаясь на детали.

Организация ловушек ошибок с помощью eval

Нередко исполнение обычных строк программного кода приводит к аварийному завершению программы, если что-то идет не так, как ожидалось.

```
my $average = $total / $count;  # деление на ноль?
print "okay\n" unless /$match/; # ошибочный шаблон?

open MINNOW, '>ship.txt'
or die "Невозможно создать файл 'ship.txt': $!";  # недостаточно прав?

&implement($_) foreach @rescue_scheme; # ошибка внутри подпрограммы?
```

Однако незапланированный ход событий вовсе не означает, что надо смириться с аварийным завершением программы. Для вылавливания разного рода ошибок Perl предоставляет оператор eval.

```
eval { $average = $total / $count } ;
```

Если во время исполнения блока eval произойдет ошибка, это уже не приведет к завершению всей программы, просто управление будет передано строке, следующей сразу же за блоком eval. Обычно после исполнения оператора eval проверяется значение переменной $@, которая будет содержать пустое значение (в случае отсутствия ошибки) или строку с сообщением об ошибке, например «divide by zero».

```
eval { $average = $total / $count } ;
print "Продолжение после ошибки: $@" if $@;

eval { &rescue_scheme_42 } ;
print "Продолжение после ошибки: $@" if $@;
```

Сразу за блоком eval необходимо ставить символ точка с запятой, поскольку eval – это функция (а не управляющая конструкция, как, например, if или while). Но сам блок является самым настоящим блоком, и в нем можно определять локальные переменные (переменные my) и любые другие операторы. Как и любая другая функция, eval имеет возвращаемое значение (значение последнего выражения или значение, возвращаемое оператором return). Разумеется, если в процессе исполнения блока кода произойдет ошибка, никакого значения возвращено не будет, то есть будет получено значение undef в скалярном контексте или пустой список в списочном. Это позволяет оформлять безопасное вычисление выражений так:

```
my $average = eval { $total / $count } ;
```

Теперь переменная $average будет содержать либо частное от деления, либо значение undef в зависимости от того, насколько успешно завершилась операция деления.

Perl поддерживает даже вложенные блоки eval. Это позволяет отлавливать ошибки во вложенных подпрограммах. Но eval не может перехватывать фатальные ошибки, которые приводят к краху самого Perl. Сюда можно отнести получение сигнала,[1] который нельзя перехватить, ошибка нехватки памяти и прочие катастрофические ситуации. Оператор eval не может применяться и для вылавливания синтаксических ошибок, поскольку компиляция блока eval производится одновременно с остальной частью программы, а не во время исполнения. Он не может перехватывать предупреждения (хотя Perl дает такую возможность с помощью $SIG{__WARN__}).

Исполнение программного кода, созданного динамически

Существует еще одна форма обращения к оператору eval, когда в качестве параметра выступает не блок кода, а строка. В этом случае компиляция и исполнение строки производятся уже во время исполнения программы. Такая возможность выглядит очень удобной и полезной, но она чрезвычайно опасна, если в строку могут попасть данные, полученные из ненадежного источника. Мы не рекомендуем вычислять строковые выражения с помощью оператора eval, разве что в исключительных случаях. Мы рассмотрим эту возможность чуть позже, и, кроме того, вы наверняка встретите такую форму записи оператора eval в чужих программах, поэтому мы покажем, как она применяется на практике:

```
eval '$sum = 2 + 2';
print "Сумма = $sum\n";
```

Perl выполнит это выражение в контексте окружающего программного кода, т. е. практически точно так же, как если бы это выражение было оформлено обычным образом. Результатом оператора eval является результат вычисления последнего выражения, поэтому совершенно необязательно вставлять весь блок операторов в строку eval.

```
#!/usr/bin/perl

foreach my $operator ( qw(+ - * /) ) {
    my $result = eval "2 $operator 2";
    print "2 $operator 2 = $result\n";
}
```

Здесь в цикле выполняется перебор операторов + - * /, и каждый из них поочередно работает внутри блока кода оператора eval. В строке, которая передается оператору eval, значение переменной $operator интерполируется в строку. После этого eval исполняет программный код, заключенный в строковую переменную, и возвращает результат последнего выражения, который записывается в переменную $result.

[1] Имеются в виду сигналы UNIX, например SIGKILL. – *Примеч. науч. ред.*

Если eval не сможет скомпилировать и исполнить программный код строки, которая ему была передана, мы получим точно такое же значение переменной $@, как и в случае использования eval в блочной форме. В следующем примере мы хотели перехватить ошибку деления на ноль, но в результате не смогли разделить число на «ничто» (еще одна разновидность ошибки).

```
print 'Частное: ', eval '5 /', "\n";
warn $@ if $@;
```

Оператор eval перехватит синтаксическую ошибку и запишет сообщение в переменную $@, что мы проверяем сразу же после вызова eval.

```
Частное:
syntax error at (eval 1) line 2, at EOF
```

Далее, в главах 10, 17 и 18, мы рассмотрим применение eval для загрузки необязательных модулей. Если Perl оказывается не в состоянии загрузить какой-либо модуль, он обычно останавливает исполнение программы. Мы будем перехватывать эту ошибку и продолжать работу программы.

На всякий случай, если вы пропустили наше предыдущее предупреждение, напомним еще раз: используйте эту форму оператора eval только в исключительных случаях. Если есть возможность выполнить те же самые действия другим способом, попробуйте сначала реализовать его. Мы обратимся к данной возможности в главе 10 для загрузки программного кода из внешнего файла, но при этом мы продемонстрируем другой, более интересный способ сделать то же самое.

Упражнения

Ответы на эти вопросы вы найдете в приложении А в разделе «Ответы к главе 2».

Упражнение 1 [15 мин]

Напишите программу, которая принимала бы из командной строки список файлов и отбирала бы с помощью оператора grep те из них, размер которых не превышает 1000 байт. Воспользуйтесь оператором map, чтобы перед каждым полученным в результате именем файла вставить четыре пробела и символ перевода строки после имени. Выведите получившийся список.

Упражнение 2 [25 мин]

Напишите программу, которая просила бы пользователя ввести шаблон (регулярное выражение). Строка должна вводиться с клавиатуры, а не приниматься в виде аргумента командной строки. Из заранее определенного каталога (например, */etc* или *c:* *windows*) выведите спи-

сок файлов, чьи имена совпадают с введенным шаблоном. Программа должна продолжать работу до тех пор, пока пользователь не введет в качестве шаблона пустую строку. Пользователь не должен вводить символы слэша, которые в Perl традиционно служат для разграничения шаблонов. Вводимые шаблоны должны разделяться символом перевода строки. Постарайтесь обеспечить устойчивость программы к ошибкам в шаблонах, таким как отсутствие парных скобок.