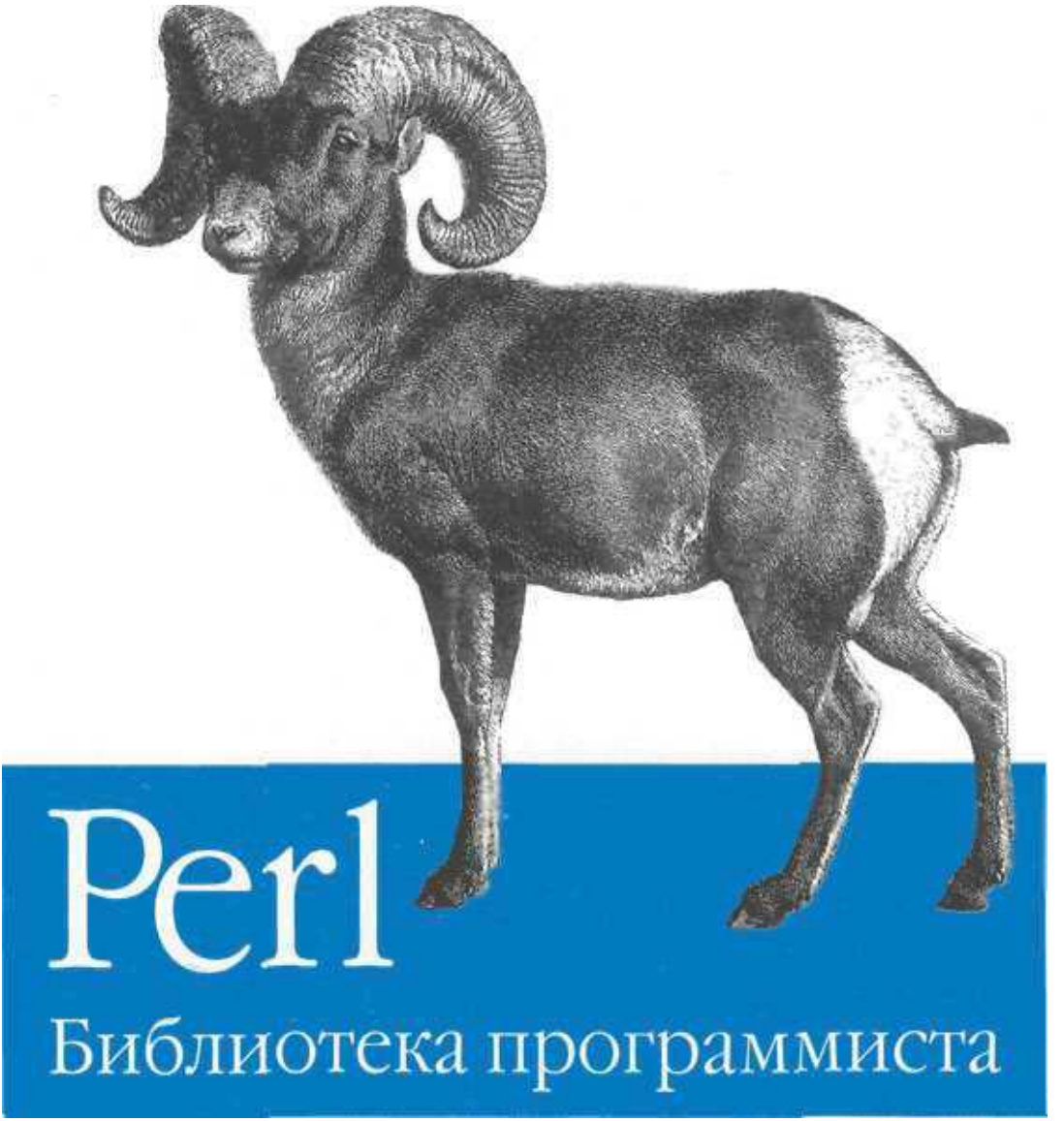
Решения и примеры для программистов



Т. Кристиансен
Н. Торкингтон

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Т. Кристиансен
Н. Торкингтон

БИБЛИОТЕКА ПРОГРАММИСТА

Perl

ПИТЕР®

2001

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Том Кристиансен, Натан Торкингтон

Perl: библиотека программиста

Перевел с английского Е. Матвеев

Главный редактор *В. Усманов*
Заведующий редакцией *Е. Строганова*
Ведущий редактор *А. Пасечник*
Научный редактор *С. Реентенко*
Корректор *В. Листова*
Верстка *Р. Гришанов*

ЭЛЕКТРОННАЯ БИБЛИОТЕКА
Прочитал? Удали!
LIBRUS.RU

ББК 32.973.2-018
УДК 681.3.06

Кристиансен Т., Торкингтон Н.
К82 Perl: библиотека программиста — СПб: Питер, 2001. — 736 с.: ил.
ISBN 5-8046-0094-X

Книга содержит обширную коллекцию путей решения большинства проблем, возникающих при работе с языком Perl. Рассматривается широкий круг вопросов: от основ техники программирования до профессиональных тонкостей, от манипуляций со строками, числами и массивами до создания баз данных SQL, от сценариев CGI и Интернет-приложений до разработки серьезных систем клиент-сервер. Наиболее удачные и красивые решения, представляющие квинтэссенцию опыта десятков профессионалов, окажутся в вашем распоряжении.

Original English language Edition Copyright© 1998 O'Reilly & Associates, Inc.
© Перевод на русский язык, Е. Матвеев, 2000
© Серия, оформление, Издательский дом "Питер", 2001

Права на издание получены по соглашению с O'Reilly & Associates, Inc.
Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.
Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 5-8046-0094-X
ISBN 1-56592-243-3 (англ.)

ЗАО "Питер Бук". 196105, Санкт-Петербург, ул. Благодатная, 67.
Лицензия ИД № 01940 от 05.06.00.
Подписано в печать 20.09.00. Формат 70X100 1/16. Усл. п. л. 46. Доп. тираж 5000 экз. Заказ № 1800.
Отпечатано с фотоформ в ГПП "Печатный двор" Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Краткое содержание

Глава 1 • Строки 26
Глава 2 • Числа 67
Глава 3 • Дата и время 94
Глава 4 • Массивы 114
Глава 5 • Хэши 151
Глава б • Поиск по шаблону 179
Глава 7 • Доступ к файлам 239
Глава 8 • Содержимое файлов 289
Глава 9 • Каталоги 323
Глава 10 • Подпрограммы 348
Глава 11 • Ссылки и записи 376
Глава 12 • Пакеты, библиотеки и модули 405
Глава 13 • Классы, объекты и связи 449
Глава 14 • Базы данных 496
Глава 15 • Пользовательские интерфейсы 521
Глава 16 • Управление процессами и межпроцессные взаимодействия 556
Глава 17 • Сокеты 604
Глава 18 • Протоколы Интернета 643
Глава 19 • Программирование CGI ббб
Глава 20 • Автоматизация в Web 703
Алфавитный указатель 729

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Содержание

Предисловие 15
Введение 17
Благодарности 23

Глава 1 Строки 26

1.1. Работа с подстроками ... 28
1.2. Выбор значения по умолчанию 31
1.3. Перестановка значений без использования временных переменных ...33
1.4. Преобразование между символами и ASCII-кодами 34
1.5. Посимвольная обработка строк ...36
1.6. Обратная перестановка слов или символов 38
1.7. Расширение и сжатие символов табуляции 40
1.8. Расширение переменных во входных данных 41
1.9. Преобразование регистра 43
1.10. Интерполяция функции и выражений в строках 46
1.11. Отступы во встроенных документах 47
1.12. Переформатирование абзацев .51
1.13. Служебные преобразования символов 53
1.14. Удаление пропусков в обоих концах строки 54
1.15. Анализ данных, разделенных запятыми 55
1.16. Сравнение слов с похожим звучанием 57
1.17. Программа: fixstyle 58
1.18.

Глава 2 Числа 67

2.1. Проверка строк на соответствие числам 68
2.2. Сравнение чисел с плавающей запятой 70
2.3. Округление чисел с плавающей запятой 71
2.4. Преобразования между двоичной и десятичной системами счисления.. 72
2.5. Действия с последовательностями целых чисел 74
2.6. Работа с числами в римской записи 75
2.7. Генератор случайных чисел 76
2.8. Раскрутка генератора случайных чисел 77
2.9. Повышение фактора случайности 78
2.10. Генерация случайных чисел с неравномерным распределением 78
2.11. Выполнение тригонометрических вычислений в градусах..... 81
2.12. Тригонометрические функции 82
2.13. Вычисление логарифмов . 83
2.14. Умножение матриц . 84
2.15. Операции с комплексными числами 86
2.16. Преобразования восьмеричных и шестнадцатеричных чисел 87

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

2.17. Вывод запятых в числах 88
2.18. Правильный вывод во множественном числе 89
2.19. Программа: разложение на простые множители 91

Глава 3 Дата и время 94

3.1. Определение текущей даты 96
3.2. Преобразование полного времени в секунды с начала эпохи 98
3.3. Преобразование секунд с начала эпохи в полное время 99
3.4. Операции сложения и вычитания для дат 100
3.5. Вычисление разности между датами 101
3.6. Определение номера недели, или дня недели/месяца/года 102
3.7. Анализ даты и времени в строках 104
3.8. Вывод даты 105
3.9. Таймеры высокого разрешения. 107
3.10. Короткие задержки 109
3.11. Программа: hopdelta 110

Глава 4 Массивы 114

4.1. Определение списка в программе 115
4.2. Вывод списков с запятыми 117
4.3. Изменение размера массива 119
4.4. Выполнение операции с.каждым элементом списка 121
4.5. Перебор массива по ссылке 124
4.6. Выборка уникальных элементов из списка 125
4.7. Поиск элементов одного массива, отсутствующих в другом массиве... 127
4.8. Вычисление объединения, пересечения и разности уникальных списков 129
4.9. Присоединение массива 131
4.10. Обращение массива 132
4.11. Обработка нескольких элементов массива 133
4.12. Поиск первого элемента списка, удовлетворяющего некоторому критерию . 1 3 4
4.13. Поиск всех элементов массива, удовлетворяющих определенному критерию 136
4.14. Числовая сортировка массива 137
4.15. Сортировка списка по вычисляемому полю 139
4.16. Реализация циклических списков 143
4.17. Случайная перестановка элементов массива 144
4.18. Программа: words 145
4.19. Программа: permute 147

Глава 5 Хэши 151

5.1. Занесение элемента в хэш 153
5.2. Проверка наличия ключа в хэше 154

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

5.3. Удаление из хэша 156
5.4. Перебор хэша 157
5.5. Вывод содержимого хэша 160
5.6. Перебор элементовхэша в порядке вставки 161
5.7. Хэши с несколькими ассоциированными значениями 162
5.8. Инвертирование хэша . 164
5.9. Сортировка хэша ... 165
5.10. Объединение хэшей 166
5.11. Поиск общих или различающихся ключей в двух кэшах 168
5.12. Хэширование ссылок 169
5.13. Предварительное выделение памяти для хэша 170
5.14. Поиск самых распространенных значений 171
5.15. Представление отношений между данными 172
5.16. Программа:

Глава 6 Поиск по шаблону 179

6.1. Копирование с подстановкой 185
6.2. Идентификация алфавитных символов 186
6.3. Поиск слов 187
6.4. Комментирование регулярных выражений 188
6.5. Поиск N-го совпадения 190
6.6. Межстрочный поиск 193
6.7. Чтение записей с разделением по шаблону 195
6.8. Извлечение строк из определенного интервала 197
6.9. Работа с универсальными символами командных интерпретаторов 200
6.10. Ускорение интерполированного поиска 201
6.11. Проверка правильности шаблона 205
6.12. Локальный контекст в регулярных выражениях 207
6.13. Неформальный поиск 209
6.14. Поиск от последнего совпадения 210
6.15. Максимальный и минимальный поиск 211
6.16. Поиск повторяющихся слов 213
6.17. Логические AND, OR и NOT в одном шаблоне 216
6.18. Поиск многобайтовых символов 220
6.19. Проверка адресов электронной почты 224
6.20. Поиск сокращений 226
6.21. Программа: urlify 228
6.22. Программа: ...229
6.23. Копилка регулярных выражений 236

Глава 7 Доступ к файлам 239

7.1. Открытие файла 243
7.2. Открытие файлов с нестандартными именами 247
7.3. Тильды в именах файлов 248
7.4. Имена файлов в сообщениях об ошибках ... 249

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.5. Создание временных файлов 250
7.6. Хранение данных в тексте программы 252
7.7. Создание фильтра 253
7.8. Непосредственная модификация файла с применением временной копии 258
7.9. Непосредственная модификация файла с помощью параметра -i.... 259
7.10. Непосредственная модификация файла без применения временного файла 261
7.11. Блокировка файла 262
7.12. Очистка буфера 265
7.13. Асинхронное чтениеиз нескольких манипуляторов 267
7.14. Асинхронный ввод/вывод :. 269
7.15. Определение количества читаемых байтов 270
7.16. Хранение файловых манипуляторов в переменных 272
7.17. Кэширование открытых файловых манипуляторов 275
7.18. Одновременный вывод через несколько файловых манипуляторов 276
7.19. Открытие и закрытие числовых файловых дескрипторов 277
7.20. Копирование файловых манипуляторов 278
7.21. Программа: netlock 280
7.22. Программа:

Глава 8 Содержимое файлов 289

8.1. Чтение строк с символами продолжения 293
8.2. Подсчет строк (абзацев, записей) в файле 294
8.3. Обработка каждого слова в файле 295
8.4. Чтение файла по строкам или абзацам в обратном направлении 297
8.5. Чтение из дополняемого файла 298
8.6. Выбор случайной строки из файла 300
8.7. Случайная перестановка строк 301
8.8. Чтение строки с конкретным номербм 301
8.9. Обработка текстовых полей переменной длины 305
8.10. Удаление последней строки файла. 306
8.11. Обработка двоичных файлов 307
8.12. Ввод/вывод с произвольным доступом 309
8.13. Обновление файла с произвольным доступом 309
8.14. Чтение строки из двоичного файла 311
8.15. Чтение записей фиксированной длины 312
8.16. Чтение конфигурационных файлов 314
8.17. Проверка достоверности файла 316
8.18. Программа: tailwtmp 319
8.19. Программа: tctee 319
8.20. Программа: laston 321

Глава 9 Каталоги 323

9.1. Получение и установка атрибутов времени 328
9.2. Удаление файла 329
9.3. Копирование или перемещение файла 330

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9.4. Распознавание двух'имен одного файла 332
9.5. Обработка всех файлов каталога 333
9.6. Получение списка файлов по шаблону 335
9.7. Рекурсивная обработка всех файлов каталога 336
9.8. Удаление каталога вместе с содержимым 339
9.9. Переименование файлов 340
9.10. Деление имени файла на компоненты 342
9.11. Программа: symirror.. 343
9.12. Программа: 1st 344

Глава 10 Подпрограммы 348

10.1. Доступ к аргументам подпрограммы 349
10.2. Создание закрытых переменных в функциях 351
10.3. Создание устойчивых закрытых переменных 353
10.4. Определение имени текущей функции 354
10.5. Передача массивов и хэшей по ссылке 356
10.6. Определение контекста вызова 357
10.7. Передача именованных параметров 358
10.8. Пропуск некоторых возвращаемых значений 359
10.9. Возврат нескольких массивов или хэшей 360
10.10. Возвращение признака неудачного вызова 361
10.11. Прототипы функций 362
10.12. Обработка исключений 364
10.13. Сохранение глобальных значений 366
10.14. Переопределение функции 369
10.15. Перехват вызовов неопределенных функций с помощью AUTOLOAD 371
10.16. Вложенные подпрограммы... 372
10.17. Сортировка почты 373

Глава 11 Ссылки и записи 376

11.1. Ссылки на массивы 381
11.2. Создание хэшей массивов 383
11.3. Получение ссылок на хэши 384
11.4. Получение ссылок на функции 385
11.5. Получение ссылок на скаляры 388
11.6. Создание массивов ссылок на скаляры 389
11.7. Применение замыканий вместо объектов 390
11.8. Создание ссылок на методы. 392
11.9. Конструирование записей 393
11.10. Чтение и сохранение записей в текстовых файлах 395
11.11. Вывод структур данных 396
11.12. Копирование структуры данных 398
11.13. Сохранение структур данных на диске 399
11.14. Устойчивые структуры данных 401
11.15. Программа: бинарные деревья 402

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Глава 12 Пакеты, библиотеки и модули 405

12.1. Определение интерфейса модуля 410
12.2. Обработка ошибок
12.3. Отложенное использование модуля 414
12.4. Ограничение доступа к переменным модуля 416
12.5. Определение пакета вызывающей стороны 419
12.6. Автоматизированное выполнение завершающего кода 421
12.7. Ведение собственного каталога модулей 422
12.8. Подготовка модуля к распространению 425
12.9. Ускорение загрузки модуля с помощью SelfLoader 427
12.10. Ускорение загрузки модуля с помощью AutoLoader 428
12.11. Переопределение встроенных функций. 429
12.12. Вывод сообщений об ошибках и предупреждений по аналогии со встроенными функциями 430
12.13. Косвенные ссылки на пакеты 432
12.14. Применение h2ph для преобразования заголовочных файлов C 433
12.15. Применение h2xs для создания модулей с кодом C 436
12.16. Документирование модуля в формате pod 439
12.17. Построение и установка модуля CPAN 441
12.18. Пример: шаблон модуля 443
12.19. Программа: поиск версий и описаний установленных модулей 445

Глава 13 Классы, объекты и связи 449

13.1. Конструирование объекта 456
13.2. Уничтожение объекта 457
13.3. Работа с данными экземпляра 459
13.4. Управление данными класса 462
13.5. Использование класса как структуры 464
13.6. Клонирование объектов 467
13.7. Косвенный вызов методов. 469
13.8. Определение принадлежности субкласса 470
13.9. Создание класса с поддержкой наследования 472
13.10. Вызов переопределенных методов 474
13.11. Генерация методов доступа с помощью AUTOLOAD 475
13.12. Решение проблемы наследования данных 477
13.13. Использование циклических структур данных 479
13.14. Перегрузка операторов 482
13.15. Создание "магических" переменных функцией tie 488

Глава 14 Базы данных 496

14.1. Создание и использование DBM-файла 498
14.2. Очистка DBM-файла 501
14.3. Преобразование DBM-файлов 502
14.4. Объединение DBM-файлов 503
14.5. Блокировка DBM-файлов 504

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14.6. Сортировка больших DBM-файлов 506
14.7. Интерпретация текстового файла в виде строковой базы данных ... 507
14.8. Хранение сложных структур данных в DBM-файлах 511
14.9. Устойчивые данные 513
14.10. Выполнение команд SQL с помощью DBI и DBD 515
14.11. Программа: ggh — поиск в глоба льном журнале Netscape 517

Глава 15 Пользовательские интерфейсы 521

15.1. Лексический анализ аргументов 523
15.2. Проверка интерактивного режима 525
15.3. Очистка экрана 526
15.4. Определение размера терминала или окна 527
15.5. Изменение цвета текста 528
15.6. Чтение с клавиатуры 530
15.7. Предупреждающие сигналы 531
15.8. Использование termios 532
15.9. Проверка наличия входных данных 534
15.10. Ввод пароля :... 535
15.11. Ред актирование входных данных 536
15.12. Управление экраном 538
15.13. Управление другой программой с помощью Expect 540
15.14. Создание меню с помощью Tk 542
15.15. Создание диалоговых окон с помощью Tk 545
15.16. Обработка событий масштабирования в Tk 548
15.17. Удаление окнасеанса DOS в Perl/Tk для Windows 550
15.18. Программа: tcapdemo 551
15.19. Программа: tkshufflepod 552

Глава 16 • Управление процессами и межпроцессные взаимодействия 556

16.1. Получение вывода отп р о г р а м м ы . 559
16.2. Запуск другой программы 560
16.3. Замена текущей программы 562
16.4. Чтение или запись в другой программе 563
16.5. Фильтрация выходных данных 565
16.6. Предварительная обработка ввода 567
16.7. Чтение содержимого STDERR 568
16.8. Управление потоками ввода и вывода другой программы 571
16.9. Управление потоками ввода, вывода и ошибок другой программы.... 573
16.10. Взаимодействие между родственными процессами 575
16.11. Имитация файла на базе именованного канала 580
16.12. Совместное использование переменных в разных процессах 584
16.13. Получение списка сигналов 586
16.14. Посылка сигнала 587
16.15. Установка обработчика сигнала 588
16.16. Временное переопределение обработчика сигнала 589

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

16.17. Написание обработчика сигнала 590
16.18. Перехват Ctrl+C 593
16.19. Уничтожение процессов-зомби 594
16.20. Блокировка сигналов 596
16.21. Тайм-аут 598
16.22. Программа: sigrand 598

Глава 17 Сокеты 604

17.1. Написание клиента TCP 606
17.2. Написание сервера TCP 608
17.3. Передача данных через TCP 611
17.4. Создание клиента UDP 614
17.5. Создание сервера UDP 616
17.6. Использование сокетов UNIX 618
17.7. Идентификация другого конца сокета 620
17.8. Определение вашего имени и адреса 621
17.9. Закрытие сокета после разветвления 622
17.10. Написание двусторонних клиентов 623
17.11. Разветвляющие серверы 625
17.12. Серверы с предварительным ветвлением 627
17.13. Серверы без ветвления 629
17.14. Написание распределенного сервера 633
17.15. Создание сервера-демона 635
17.16. Перезапуск сервера по требованию 636
17.17. Программа: backsniff 637
17.18. Программа: fwdport 638

Глава 18 Протоколы Интернета 643

18.1. Простой поиск в DNS 644
18.2. Клиентские операции FTP 647
18.3. Отправка почты 650
18.4. Чтение и отправка новостей Usenet 653
18.5. Чтение почты на серверах POP3 656
18.6. Программная имитация сеанса telnet 658
18.7. Проверка удаленного компьютера 660
18.8. Применение whois для получения данных от InterNIC 662
18.9. Программа: expn и vrfy 663

Глава 19 Программирование CGI 666

19.1. Написание сценария CGI 670
19.2. Перенаправление сообщений об ошибках 672
19.3. Исправление ошибки 500 Server Error 673
19.4. Написание безопасных программ CGI 677
19.5. Повышение эффективностисценариев CGI 680
19.6. Выполнение команд безобращений к командному интерпретатору 681

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

19.7. Форматирование списков и таблиц средствами HTML.... 683
19.8. Перенаправление клиентского броузера 686
19.9. Отладка на уровне HTTP 688
19.10. Работа с cookies 690
19.11. Создание устойчивых элементов 692
19.12. Создание многостраничного сценария CGI 693
19.13. Сохранение формы в файле или канале 696
19.14. Программа: chemiserie 698

Глава 20 • Автоматизация в Web 703

20.1. Выборка URL из сценария Perl 704
20.2. Автоматизация подачи формы 706
20.3. Извлечение URL 708
20.4. Преобразование ASCII в HTML 710
20.5. Преобразование HTML в ASCII 711
20.6. Удаление тегов HTML 712
20.7. Поиск устаревших ссылок 714
20.8. Поиск свежих ссылок 715
20.9. Создание шаблонов HTML 717
20.10. Зеркальное копирование Web-страниц 720
20.11. Создание робота 721
20.12. Анализ файла журнала Web-сервера 722
20.13. Обработка серверных журналов 724
20.14. Программа: htmlsub 727
20.15. Программа:

Алфавитный указатель 729

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Предисловие

Говорят, метафорами легко увлечься. Но некоторые метафоры настолько хороши, что в таком увлечении нет ничего плохого. Вероятно, к их числу относится и метафора поваренной книги — по крайней мере, в данном случае. Меня смущает лишь одно: представленная работа настолько монументальна, что все сказанное мной будет либо повторением, либо пустыми словами.

Впрочем, прежде меня это никогда не останавливало.

Вероятно, кулинария является самым скромным из всех искусств; но я всегда считал скромность достоинством, а не недостатком. Великий художник, как и великий повар, всегда работает с конкретными выразительными средствами. И чем скромнее средства, тем скромнее должен быть художник, чтобы вывести их за рамки обыденного. И еда, и язык программирования относятся к скромным средствам; они состоят из внешне разрозненных ингредиентов. И все же в руках мастера, наделенного творческим мышлением и дисциплиной, из самых обыденных вещей — картошки, макарон и Perl — возникают произведения искусства, которые не просто справляются со своей задачей, но и делают это так, что ваше странствие по жизни становится чуть более приятным.

Кроме того, кулинария принадлежит к числу самых древних искусств. Некоторые современные художники полагают, что так называемое эфемерное искусство изобрели совсем недавно, однако кули нария всегда былаэфемерным искусстврм. Мы пытаемся сохранить произведения искусства, продлить их существование, но даже пища, захороненная вместе с фараонами, со временем приходит в негодность. Итак, плоды нашего программирования на Perl тоже вчем-то эфемерны. Этот аспект "кухни Perl" часто порицают. Если хотите — называйте его "программированием на скорую руку", но миллиардные обороты в кафе быстрого обслуживания позволяют надеяться, что быстрая еда может быть качественной (нам хотелось бы в это верить).

Простые вещи должны быть простыми, а сложные... возможными. На один рецепт быстрых блюд приходится бесчисленное множество обычных рецептов. Одна из прелестей жизни в Калифорнии — в том, что мне доступна практически любая национальная кухня. Но даже в границах одной культуры У Каждой Задачи Всегда Найдется Несколько Решений. Как говорят в России, "Сколько поваров, столько и рецептов борща", и я этому верю. Рецепт моей мамы даже обходится без свеклы! И это вполне нормально. Борщ становится неким разделителем культур, а культурное разнообразие интересно, познавательно, полезно и увлекательно.

Итак, Том и Нат в этой книге не всегда делают все так, как это бы сделал я. Иногда они даже не могут прийти к единому решению — и это тоже сила, а не слабость. Признаюсь, из этой книги я узнал кое-что новое. Более того, наверняка и сейчас я знаю далеко не все (и надеюсь, не узнаю в ближайшее время). Мы часто говорим о культуре Perl так, словно она является чем-то единым, непоколебимым, хотя в действительности существует множество здоровых субкультур Perl, не говоря уже о всевозможных сочетаниях суб-субкультур, суперкультур и околокультур, наследующих друг от друга атрибуты и методы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Итак, поваренная книга не готовит пищу за вас (она этого не умеет) и даже не учит вас готовить (хотя и помогает в этом). Она лишь передает различные культурные фрагменты, которые оказалисьполезными, и, возможно, отфильтровывает другие "культуры", которые выросли в холодильнике по беспечности хозяев. В свою очередь, вы поделитесь этими идеями с другими людьми, пропустите их через собственный опыт и личные вкусы, ваше творческое мышление и дисциплину. У вас появятся собственные рецепты, которые вы передадите собственным детям. Не удивляйтесь, когда они придумают что-то свое и спросят, что вы об этом думаете. Постарайтесь не корчить недовольную гримасу.

Рекомендую вам эти рецепты. Когда я читал их, у меня не было особых поводов для недовольных гримас.

Ларри Уолл
июнь 1998 г.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты (издательство "Питер", компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на Web-сайте издательства

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Введение

Бизнесмены осторожно разглядывали докладчика, скепсис на лицах переходил в заинтересованность, и наоборот.

"Ваш рискованный план обещает выгоду, — заключил председатель. — Но он очень дорог и основан на одних предположениях. Наши математики не подтверждают ваших цифр. Почему мы должны доверить вам свои деньги? Что вы знаете такого, чего не знаем мы?"

"Прежде всего, — ответил он, — я знаю, как вертикально поставить яйцо без внешней опоры. А вы?" С этими словами докладчик залез в сумку и осторожно вынул свежее куриное яйцо. Он передал яйцо финансовым магнатам, которые передавали его из рук в руки, пытаясь справиться с несложной задачей. Все попытки оказались тщетными. Раздавались отчаянные возгласы: "Это невозможно! Никому не удастся поставить яйцо вертикально!"

Докладчик взял яйцо у рассерженных бизнесменов и поставил яйцо на дубовый стол, прочно удерживая его в руках. После легкого, но уверенного нажатия скорлупа слегка потрескалась. Когда докладчик убрал руку, яйцо осталось на месте — слегка продавленное, но определенно устойчивое. "Что здесь невозможного?" — спросил он.

"Но это же обычный фокус, — закричали бизнесмены. — Такое может сделать любой!"

"Ваша правда, — последовал ответ. — Но это относится ко всему. Пока вы не знаете решения, задача кажется невозможной. А решение выглядит так просто, что вы не понимаете, почему это раньше не приходило вам в голову. Так позвольте мне показать простое решение, чтобы другие могли легко пойти тем же путем. Вы мне доверяете?"

Скептически настроенные капиталисты убедились, что предприниматель действительно на что-то способен, и выделили деньги на его проект. Из маленького андалузского порта отправились в море "Нинья", "Пинта" и "Санта Мария". Их вел предприниматель с надбитым яйцом и своими идеями — Христофор Колумб.

За ним последовали многие.

Задачи программирования часто похожи на колумбово яйцо. Пока никто не покажет решения, вы сидите и смотрите, как яйцо (то бишь программа) падает снова и снова, ни на шаг не приближаясь к решению. Это особенно справедливо для таких идиоматических языков, как Perl.

Эта книга не задумывалась как полный справочник по Perl, хотя мы опишем некоторые недокументированные аспекты Perl. Любая поставка Perl содержит свыше 1000 страниц электронной документации. Если их не окажется под рукой, обратитесь к системному администратору.

Итак, эта книга — для тех, кто хочет *лучше* узнать Perl. Перед вами не справочник и не учебник, хотя книга окажется полезным дополнением к ним. Она предназначена для людей, которые изучили основы языка и теперь пытаются связать ингредиенты в готовую программу. На протяжении 20 глав и свыше 300 отдельных тем, именуемых "Рецептами", вы найдете тысячи решений для повседневных задач, с которыми сталкиваются как новички, так и опытные программисты.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Мы постарались сделать так, чтобы книга подходила и для последовательного, и для произвольного доступа. Каждый рецепт вполне самостоятелен, но если вам понадобится дополнительная информация, вы найдете в конце рецепта список ссылок. Глава обычно начинается с простых, повседневных рецептов, а книга начинается с простых глав. Рецепты, посвященные типам данным и операторам Perl, особенно полезны для новичков. Постепенно мы перейдем к темам и решениям, рассчитанным на более опытных программистов. Но там и сям встречается материал, способный вдохновить даже настоящего знатока Perl.

Главы начинаются с краткого обзора. За введением следует основная суть главы, ее рецепты. В духе лозунга Perl — "Всегда существует несколько решений" — во многих рецептах продемонстрированы разные способы решения той же самой или аналогичной задач. Рецепты простираются от конкретных решений в стиле "коротко, но мило" до углубленных мини-учебников. Там, где приведено несколько вариантов, мы часто объясняем преимущества и недостатки каждого подхода.

Предполагается, что к этой книге (как и к обычным поваренным книгам) читатель обращается более или менее произвольно. Если вы хотите научиться что-то делать, загляните в нужный рецепт. Даже если конкретное решение не подойдет к вашей задаче, оно по крайней мере дастпредставление о возможных направлениях поисков.

Каждая глава завершается одной или несколькими законченными программами. Хотя некоторые рецепты уже содержат маленькие программы, эти приложения выделяют основную тему главы; кроме того, в них, как и в любой реальной программе, используются приемы из других глав. Все этипрограммы полезны, некоторые из них используются ежедневно. Некоторые программы даже помогли нам в работе над книгой.

О чем рассказано в этой книге

Первая четверть книги, занимающая более пяти глав, посвящена базовым типам данных Perl. В главе 1 "Строки" рассматриваются такие вопросы, как работа с подстроками, расширение вызовов функций в строках и анализ данных, разделенных запятыми. Глава 2 "Числа" описывает некоторые странности представления с плавающей запятой, разделение разрядов запятыми и процесс генерации псевдослучайных чисел. Глава 3 "Дата и время" демонстрирует преобразования между числовыми и строковыми форматами даты и применение таймеров. В главе 4 "Массивы" рассматривается все, что относится к операциям со списками и массивами, в том числе поиск уникальных элементов, эффективная сортировка и случайные перестановки элементов. Глава 5 "Хэши" завершает основы языка и представляет самый полезный тип данных — ассоциативные массивы. В ней показано, как обращаться с элементами хэша в порядке вставки, как отсортировать хэш по значению и как хранить несколько ассоциированных значений для одного ключа.

Глава 6 "Поиск по шаблону" занимает больше всего места. Рецепты описывают преобразование универсальных символов командного интерпретатора в шаблон, поиск букв и слов, многострочные совпадения, отказ от максимализма при поиске и поиск строк, которые близки к искомым, но не совпадают с ними. Хотя

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

глава и так получилась самой длинной, она могла бы стать еще длиннее — в каждой главе вы найдете примеры использования регулярных выражений. Это часть того, что придает Perl его неповторимость.

Три следующие главы относятся к файловой системе. В главе 7 "Доступ к файлам" показано, как открыть файл, заблокировать его для параллельной работы, модифицировать его на месте и сохранить файловый манипулятор в переменной. В главе 8 "Содержимое файлов" обсуждается проблема поиска конца увеличивающегося файла, чтение конкретной строки файла и двоичный ввод/вывод с произвольным доступом. Наконец, в главе 9 "Каталоги" описаны приемы копирования, перемещения и удаления файлов, изменения атрибутов времени файла и рекурсивной обработки всех файлов каталога.

Основное внимание в главах 10-13 уделено тому, как сделать программы более гибкими и функциональными. Глава 10 "Подпрограммы" содержит рецепты для создания устойчивых локальных переменных, передачи параметров по ссылке, косвенного вызова функций и обработки исключений. Глава 11 "Ссылки и записи" посвящена структурам данных; продемонстрированы основные операции со ссылками на данные и функции. Также в ней показано, как создавать аналоги конструкции struct языка С, как сохранятьи загружать их из устойчивого хранилища. В главе 12 "Пакеты библиотеки и модули", рассматривается деление программы на отдельные файлы; создание переменных и функций, действующих только в границах данного модуля; замена встроенных функций, перехват обращений к отсутствующим модулям и использование утилит *h2ph* и *h2xs* для использования кода, написанного на С и С++. Наконец, в главе 13 "Классы, объекты и связи" рассматриваются основные принципы построения объектно-ориентированных модулей для создания пользовательских типов, обладающих конструкторами, деструкторами и возможностями наследования. В других рецептах показаны примеры использования циклических структур данных, перегрузки операторов и связанных типов данных.

Две следующие главы посвящены интерфейсам: первая — интерфейсам к базам данных, вторая — к визуальным устройствам. В главе 14 "Базы данных" описана методика работы с индексированными текстовыми файлами, блокировка файлов DBM и хранение в них информации, а также продемонстрирован интерфейс Perl к базам данных SQL. В главе 15 "Пользовательские интерфейсы" рассматриваются такие темы, как очистка экрана, обработка параметров командной строки, посимвольный ввод, перемещение курсора средствами *termcap* и *curses* и независимое от платформы графическое программирование с применением *Tk*.

Последняя четверть книги посвящена взаимодействию с другими программами и устройствами. В главе 16 "Управление процессами и межпроцессные взаимодействия" говорится о запуске других программ и получении их вывода, об уничтожении процессов-зомби, именованных каналах, обработке сигналов и совместному использованию переменных работающими процессами. Глава 17 "Сокеты" показывает, как установить потоковое соединение или использовать датаграммы при разработке низкоуровневых сетевых приложений "клиент/сервер". В главе 18 "Протоколы Интернета" рассматриваются протоколы высокого уровня — mail, FTP, Usenet и Telnet. Глава 19 "Программирование CGI" содержит рецепты для обработки Web-форм, перехвата ошибок, повышения безопасности за счет отказа

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

от обращений к командному интерпретатору, использования cookies, обслуживания электронных магазинов и сохранения форм в файлах или каналах. В последней главе книги "Автоматизация в Web" описана неинтерактивная работа в Web. В числе рецептов — выборка URL, автоматизация подачи форм в сценариях, извлечение URL из Web-страниц, удаление тегов HTML, поиск свежих или устаревших ссылок и обработка серверных файлов журналов.

Платформы

Книга создавалась на основе Perl 5.004_04, что означает старшую версию 5, младшую 004 и исправления уровня 4. Большинство программ и примеров было протестировано в BSD, Linux и SunOS, но это не значит, что они будут работать только в этих системах. Perl *проектировался* как язык, независимый от платформы. Если вы ограничиваетесь базовыми операциями с переменными, шаблонами, подпрограммами и высокоуровневым вводом/выводом, ваша программа должна одинаково работать везде, где работает Perl, то есть практически везде. Первые две трети книги посвящены именно такому общему программированию.

Изначально Perl задумывался как высокоуровневый кросс-платформенный язык системного программирования. Хотя с того времени Perl вышел далеко за пределы исходного предназначения, он продолжает широко использоваться в системном программировании в родных системах семейства UNIX и на других длатформах. Для обеспечения максимальной переносимости основное внимание уделялось открытым системам, соответствующим стандарту POSIX (Portable Operating System Interface), — к их числу принадлежат практически все разновидности UNIX и множество других операционных систем. Большинство рецептов будет работать в любой POSIX-системе без каких-либо изменений (или с минимальными изменениями).

Perl может использоваться для системного программирования даже в системах, не соответствующих стандарту POSIX. Для этого вам понадобятся специализированные модули для этих систем, однако в этой книге они не рассматриваются. Это объясняется тем, что такие программы не переносимы, — и, честно говоря, еще и тем, что в распоряжении авторов таких систем нет. Информация о специализированных модулях приведена в документации, прилагаемой к вашей версии Perl.

Не беспокойтесь — большинство рецептов, связанных с системным программированием, работает и в системах, не соответствующих стандарту POSIX (особенно рецепты, относящиеся к базам данных, сетевым средствам и работе в Web). Используемые в этих областях модули маскируют различия между платформами. Исключение составляют в первую очередь немногочисленные рецепты и программы, основанные на многозадачных конструкциях, и в первую очередь — на мощной функции fork, стандартной в семействе POSIX и редко реализуемой в других системах.

Во многих операциях со структурированными файлами используется удобная база данных */etc/passwd.* При чтении текстовых файлов используется */etc/motd,* а там, где была нужна внешняя программа с выходными данными, — *who(1).* Эти файлы были выбраны лишь для демонстрации общих принципов, действующих независимо от того, присутствуют эти файлы в вашей системе или нет.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Условные обозначения, использованные в книге

Условные обозначения в программах

Мы твердо верим, что каждая нетривиальная программа на Perl должна содержать флаг командной строки -w и директиву use strict. Практически все наши программы начинаются так:

```
#!/usr/bin/perl -w
usestrict;
```

В книге приведено множество примеров. Большинство из них представляет собой фрагменты кода. Некоторые примеры являются полноценными программами, их нетрудно узнать по начальной строке #!.

Однако некоторые примеры должны вводиться в приглашении командной строки. Приглашение обозначается символом %:

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

Подобный стиль характерен для стандартных командных строк UNIX. Правила определения строк и универсальные символы в других системах могут быть другими. Например, большинство стандартных командных интерпретаторов в DOS и VMS требует, чтобы для группировки аргументов с пробелами или универсальными символами использовались кавычки вместо апострофов. Внесите соответствующие исправления.

Шрифтовое выделение

В книге использованы следующие условные обозначения:

Курсив

Выделение имен файлов, названий команд и URL, а также новых терминов (там, где они впервые встречаются в тексте).

Жирный шрифт

Параметры командной строки.

`Моноширинный шрифт`

Имена функций и методов, а также их аргументы. В примерах обозначает непосредственно вводимый текст, а в тексте книги — элементы программного кода.

`Моноширинный полужирный шрифт`

Выходные данные в примерах.

Документация

Самая свежая и полная документация по Perl распространяется вместе с Perl. В печатном виде эта объемистая антология займет свыше 1000 страниц и внесет заметный вклад в глобальную вырубку лесов. К счастью, распечатывать ее не нужно, поскольку вся документация находится в удобном электронном виде с возможностями поиска.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Говоря о "стран ицах руководства" в этой книге, мы имеем в виду набор электронных документов. Название чисто условное; для их чтения необязательно использовать программу *man,* традиционную для UNIX. Также подойдет команда *perldoc,* распространяемая с Perl. Страницы руководства даже могут устанавливаться в виде HTML-страниц, особенно в системах, не принадлежащих к семейству UNIX. Если вам известно местонахождение электронной документации, вы сможете искать в ней информацию с помощью утилиты [1]. HTML-версия электронной документации также имеется в Web по адресу *http://www.perl.com/CPAN/doc/manual.html/*.

Когда мы ссылаемся на документацию, не относящуюся к Perl (например, "См. страницу руководства kill(2) вашей системы"), речь идет о странице *kill из* раздела 2 руководства "UNIX Programmer's Manual" (системные функции). Для систем, не входящих в семейство UNIX, эта документация недоступна, но в этом нет ничего страшного, поскольку вам все равно не удастся ей воспользоваться. Если вам действительно понадобится документация по системной или библиотечной функции, многие организации размещают свои man-страницы в Web, и простейший поиск вида +crypt(3)+manual в AltaVista даст желаемые результаты.

[1] Если утилита ьзуйтесь программой приведенной в конце главы 6.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Благодарности

Эта книга появилась на свет лишь благодаря множеству людей, компетентных и некомпетентных, стоявших за спинами авторов. Во главе этого легиона стоит наш редактор, Линда Май (Linda Mui), с кнутом в одной руке и пряником в другой. Она была бесподобна.

Ларри Уолл как автор Perl был нашим судьей в высшей инстанции. Он следил за тем, чтобы мы не документировали то, что онсобирался изменить, и помогал в выборе формулировок и стиля. Если временами в этой книге вам послышится голос Ларри, вероятно, вы не ошиблись.

Глория, жена Ларри — литературный критик. Как ни поразительно, она прочитала каждое слово в этой книге... и одобрила большинство из них. Вместе с Шерон Хопкинс (Sharon Hopkins), поэтессой Perl по призванию, она помогла справиться с нашей патологической склонностью к предложениям, которые можно было бы умеренно описать как нечто среднее между невообразимо сложным и безнадежно запутанными. В результате наши невразумительные высказывания стали понятны даже тем, чьим родным языком не был ассемблер PDP-11 или средневековый испанский.

Трое самых усердных рецензентов, Марк-Джейсон Доминус (Mark-Jason Dominus), Джон Оруэнт (Jon Orwant) и Эбигейл (Abigail), трудились вместе с нами практически все время работы над книгой. Их суровые стандарты, ужасающий интеллект и практический опыт программирования на Perl принесли бесценную помощь. Дуг Эдварде (Doug Edwards) педантично протестировал каждый фрагмент кода в семи начальных главах книги и нашел неочевидные частные случаи, о которых никто даже не подумал. В числе других ведущих рецензентов были Энди Догерти (Andy Dougherty), Энди Орам (Andy Oram),
Брайан Баас (Bryan Buus), Джайсл Аас (Gisle Aas), Грэхем Барр (Graham Barr), Джефф Хемер (Jeff Haemer), Джеффри Фридл Friedl), Линкольн Стейн (Lincoln Stein), Марк Мильке (Mark Mielke), Мартин Бреч (Martin Маттиас Неерахер (Matthias Neeracher), Майк Сток (Mike Stok), Нат Патвардхан (Nate Patwardhan), Пол Грасси (Paul Grassie), Питер Приммер (Peter Prymmer), Рафаэль Манфреди (Raphael M Уитби (Rod Whitby).

И это далеко не все. Многие бескорыстные личности поделились с нами своими техническими познаниями. Некоторые из них прочитали целые главы и составили формальные рецензии; другие давали содержательные ответы на короткие технические вопросы там, где мы выходили за рамки своей компетенции. Кое-кто даже присылал нам программы. Приведем лишь частичный список тех, кто был нам полезен: Аарон Харш (Aaron Harsh), Али Райл (AH Rayl), Аллигатор Декарт (Alligator Descartes), Эндрю Hume), Эндрю Стребков (А Энди Уордли (Andy Wardley), Эштон МакЭндрюс (Ashton
Бен Герцфилд (Ben Gertzfield), Бенджамин Хольцман (Benjamin Holzman), Брэд Хьюджес (Brad Hughes), Чейм Френкель Чарльз Бейли (Charles Bailey), Крис Нандор (Chris Nandor), Клинтон Вонг (Clinton Wong), Дэн Клейн (Dan Klein), Дэн Сугальски (Dan Sugalski), Дэниел Грисинджер (Daniel Grisinger), Деннис Тейлор (Dennis Taylor), Дуг МакИчерн (Doug MacEachern), Дуглас Дэвен-

[1] А также вставлял сноски. - *Примеч. авт.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

порт (Douglas Davenport), Дрю Экхардт Дилан Нортрап (Dylan Northrup), Эрик Эйзенхарт (Eric Eisenhart), Грег Бэкон (Greg Bacon), Гурусами Сарати (Gurusamy Sarathy), Генри Спенсер (Henry Spencer), Джейсон Стюарт (Jason Stewart), Джоэл Нобл (Joel Noble), Джонатан Коэн (Jonathan Cohen), Джонатан Скотт Дафф (Jonatathan Scott Duff), Джош Пуринтон Josh Purinton), Джулиан Андерсон (Julian Anderson), Кейт Уинстейн (Keith Winstein), Кен Лунд (Ken Lunde), Кирби Хьюджес (Kirby Hughes), Ларри Рослер (Larry Rosier), Лес Петере (Les Peters), Марк Хесс (Mark Hess), Марк Джеймс (Mark James), Мартин Бреч (Martin Мэри Кутски (Mary Koutski), Майкл Паркер (Michael Parker), Ник Инг-Симмонс (Nick Ing-Simmons), Пол Маркесе (Paul Marquess), Питер Коллинсон (Peter Collinson), Питер Озел (Peter Osel), Фил Бошамп (Phil Beauchamp), Пирс Коули (Piers Cawley), Рэндал Шварц (Randal Schwartz), Рич Рауэнзан (Rich Rauenzahn), Ричард Аллан (Richard Allan), Рокко Капуто (Rocco Caputo), Родерик Шертлер (Roderick Schertler), Роланд Уокер (Roland Walker), Ронан Уэйд (Ronan Waide), Стивен Лиди (Stephen Lidie), Стивен Owens (Stephen Owens), Салливан Бек (Sullivan Beck), Тим Бунс (Tim Bunce), Тодд Миллер (Todd Miller), Трои Денкингер (Troy Denkinger) и Вилли Гримм (Willy Grimm).

Нельзя не упомянуть и сам Perl, без которого эта книга никогда не была бы написана. Мы написали на Perl множество мелких утилит, помогавших нам в работе над книгой. Они преобразовывали наш текст из формата pod в формат troff для отображения и проверки и в формат FrameMaker на стадии подготовки к печати. Другая программа Perl проверяла синтаксис в каждом фрагменте кода, встречающемся в книге. С помощью Tk-расширения Perl была написана графическая утилита для перемещения между рецептами посредством перетаскивания мышью. Кроме того, мы также написали бесчисленное множество мелких утилит для других целей. Назовем лишь некоторые из них — поиск блокировок RCS, поиск повторяющихся слов и некоторых разновидностей грамматических ошибок, управление почтовыми папками с сообщениями от рецензентов, построение предметного указателя и содержания, поиск текста с пересечением границы строки или ограниченного определенным разделом и т. д. Некоторые из этих утилит описаны в книге.

Том

Прежде всего благодарю Ларри и Глорию за то, что они пожертвовали частью своего отпуска в Европе для работы над книгой, а также моих друзей и семью — Брайана, Шерон, Брента, Тодда и Дрю — за то, что они терпели меня в течение двух последних лет и выдержали бесчисленные проверки ошибок.

Хочу поблагодарить Натана за то, что он выдержал свои еженедельные поездки, мою пикантную вегетарианскую кухню, а также за то, что он упрямо исследовал темы, которых я старался избегать.

Благодарю наших безвестных титанов, Денниса, Линуса, Кирка, Эрика и Рича, которые тратили время на мои глупые вопросы об операционной системе и *troff,* Они проделали громадную работу, без которой эта книга никогда бы не была написана.

Также благодарю моих учителей, которые доблестно отправлялись в опасные места типа Нью-Джерси и преподавали Perl. Благодарю Тима О'Рейли (Tim O'Reilly) и Фрэнка Уиллисона (Frank Willison): во-первых, за то, что они подда-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

лись на уговоры и согласились опубликовать эту книгу, и во-вторых — за то, что они ставили на первое место качество, а не скорость работы. Также благодарю Линду, нашего ошеломляюще честного редактора, за то, что ей удалось совершить невероятное — соблюсти сроки издания.

Благодарю свою мать, Мэри, за то, что она оторвалась от работы по восстановлению прерий и преподавания информатики и биологии и помогла нормально организовать мою деловую и домашнюю жизнь в течение времени, достаточного для написания этой книги.

Наконец, хочу поблагодарить Иоганна Себастьяна Баха, который был для меня бесконечным источником поэзии и вдохновения, лекарством для ума и тела. Отныне при виде этой книги я всегда буду вспоминать звуки музыки, навечно запечатленные в моей памяти.

Нат

Без любви и терпения своей семьи я ничего не достиг бы в этой жизни. Спасибо вам! От своих друзей — Жюля, Эми, Раджа, Майка, Кефа, Сая, Роберта, Звана, Понди, Марка и Энди — я узнал много нового. Я глубоко благодарен своим коллегам в Сети, от которых я получал ценные технические советы и где познакомился со своей женой (впрочем, относительно нее никаких советов мне не давали). Также благодарю свою фирму, Front Range Internet, за интересную работу, с которой мне не хотелось бы уходить.

Том был великолепным соавтором. Без него эта книга была бы отвратительной, тупой и короткой. Напоследок хочу поблагодарить Дженни. Мы были женаты около года, когда я принял предложение насчет книги, и с тех пор практически не виделись. Никто так не порадуется окончанию этой работы, как она.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

1 Строки

..И открыл легкомысленно уста свои,
и безрассудно расточает слова,

Книга Иова, 35:16

Введение

Многие языки программирования заставляют нас мыслить на неудобном низком уровне. Вам понадобилась строка, а язык хочет, чтобы вы работали с указателем или байтовым массивом. Впрочем, не отчаивайтесь — Perl не относится к языкам низкого уровня, и в нем удобно работать со строками.

Perl *проектировался* для обработки текста. В сущности, в Perl существует такое количество текстовых операций, что их невозможно описать в одной главе. Рецепты обработки текста встречаются и в других главах. В частности, обратитесь к главе 6 "Поиск по шаблону" и главе 8 "Содержимое файлов" — в них описаны интересные приемы, не рассмотренные в этой главе.

Фундаментальной единицей для работы с данными в Perl является скаляр (scalar), то есть отдельное значение, хранящееся в отдельной (скалярной) переменной. В скалярных переменных хранятся строки, числа и ссылки. Массивы и хэши представляют собой соответственно списки или ассоциативные массивы скаляров. Ссылки используются для косвенных обращений к другим величинам; они отчасти похожи на указатели в языках низкого уровня. Числа обычно хранятся в формате вещественных чисел с двойной точностью. Строки в Perl могут иметь произвольную длину (ограниченную только объемом виртуальной памяти вашего компьютера) и содержат произвольные данные — даже двоичные последовательности с нулевыми байтами.

Строка не является массивом байт; к отдельному символу нельзя обратиться по индексу, как к элементу массива — для этого следует воспользоваться функцией substr. Строки, как и все типы данных Perl, увеличиваются и уменьшаются в размерах по мере необходимости. Неиспользуемые строки уничтожаются системой сборки мусора Perl (обычно при выходе переменной, содержащей строку, за пределы области действия или после вычисления выражения, в которое входит стро-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ка). Иначе говоря, об управлении памятью можно небеспокоиться — об этом уже позаботились до вас.

Скалярная величина может быть определенной или неопределенной. Определенная величина может содержать строку, число или ссылку. Единственным неопределенным значением является undef, все остальные значения считаются определенными — даже 0 и пустая строка. Однако определенность не следует путать с логической истиной; чтобы проверить, определена ли некоторая величина, следует воспользоваться функций defined. Логическая истина имеет особое значение, которое проверяется логическими операторами && и ||, а также в условии блока while.

Две определенные строки считаются ложными: пустая строка ("") и строка единичной длины, содержащая цифру "ноль" ("О"). Возможно, второе вас несколько удивит, но это связано с тем, что Perl выполняет преобразования между числами и строками по мере необходимости. Числа 0.0.00 и 0.00000000 без кавычек считаются ложными значениями, но в строках они становятся истинными (так, строка "0.00" считается истинной, а не ложной). Все остальные определенные значения (например, "false", 15 и\$x) истинны.

В строковом контексте значение undef интерпретируется как пустая строка ("). В числовом контексте undef интерпретируется как 0, а вссылочном — как нуль-ссылка. При этом во всех случаях оно считается ложным. Использование неопределенной величины там, где Perl ожидает получить определенную, приводит к записи в STDERR предупреждения времени выполнения (если был использован флаг -w). Для простого вопроса о том, является ли нечто истинным или ложным, предупреждение не выдается. Некоторые операции не выдают предупреждений при использовании переменных, содержащих неопределенные значения. К их числу относятся операции автоматического увеличения и уменьшения, ++ и --, а также сложение и конкатенация с присваиванием, += и =.

В программах строки записываются в апострофах или кавычках, в форме q// или qq// или "встроенных документов" Апострофы используются в простейшей форме определения строк с минимальным количеством специальных символов: ' — завершает строку, \ — вставляет в нее апостроф, а \\ — обратную косую черту:

```
$string = '\n';              #Два          символа, \ и n
$string = 'Jon \'Maddog\' Orwant';  # Внутренние апострофы
```

В строках, заключенных в кавычки, возможна интерполяция имен переменных (но не вызовов функций — о том, как это делается, см. рецепт1.10). В них используется множество служебных символов: "\n" — символ перевода строки, "\033" — символ с восьмеричным кодом 33, "\cJ" — Ctrl+J и т. д. Полный список приведен в странице руководства*perlop(1)*.

```
$string = "\n";              #          Символ перевода строки
$string = "Jon \"Maddog\" Orwant";  #  Внутренние кавычки
```

Операторы q// и qq// позволяют чередовать разделители строк с апострофами и кавычками. Например, строку с внутренними апострофами проще записать в следующем виде, вместо того чтобы использовать служебные символы \':

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$string = q/Jon 'Maddog' Orwant/; # Внутренние апострофы
```

В качестве разделителей могут использоваться одинаковые символы, как в этом примере, или парные (для различных типов скобок):

```
$string = q[Jon 'Maddog' Orwant]; # Внутренние апострофы
$string = q{Jon 'Maddog' Orwant}; # Внутренние апострофы
$string = q(Jon 'Maddog' Orwant); # Внутренние апострофы
$string = q<Jon 'Maddog' Orwant>; # Внутренние апострофы
```

Концепция "встроенных документов" позаимствована из командных интерпретаторов (shell) и позволяет определять строки, содержащие большое количество текста. Текст может интерпретироваться по правилам для строк, заключенных в апострофы или кавычки, и даже как перечень исполняемых команд — в зависимости от того, как задается завершающий идентификатор. Например, следующий встроенный документ будет интерпретироваться по правилам для строк, заключенных в кавычки:

```
$a = <<"EOF";
This is a document
terminated by EOF on a line by itself
EOF
```

Обратите внимание: после завершающего EOF точка с запятой не ставится. Встроенные документы более подробно рассматриваются в рецепте 1.11.

Предупреждение для программистов из других стран: в настоящее время Perl не обладает прямой поддержкой многобайтовых кодировок (в версии 5.006 ожидается поддержка Unicode), поэтому в тексте книги понятия *байт* и *символ* считаются идентичными.

1.1. Работа с подстроками

Проблема

Требуется получить или модифицировать не целую строку, а лишь ее часть. Например, вы прочитали запись с фиксированной структурой и теперь хотите извлечь из нее отдельные поля.

Решение

Функция `substr` предназначена для чтения и записи отдельных байтов строки:

```
$value = substr($string, $offset, $count);
$value = substr($string, $offset);

substr($string, $offset, $count) = $newstring;
substr($string, $offset) = $newtail;
```

Функция `unpack` ограничивается доступом только для чтения, но при извлечении нескольких подстрок работает быстрее:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
ft Получить 5-байтовую строку, пропустить 3,
№ затем две 8-байтовые строки, затем все остальное
($leading, $s1, $s2, $trailing) =
    unpack("A5 x3 A8 A8 A*", $data);

# Деление на группы из пяти байт
@fivers = unpack("A5" x (length($string)/5), $string);

# Деление строки на отдельные символы
@chars = unpack("A1" x length($string), $string);
```

Комментарий

В отличие от многих языков, в которых строки представлены в виде массива байтов (или символов), в Perl они относятся к базовым типам данных. Это означает, что для работы с отдельными символами или подстроками применяется функция unpack или substr.

Второй аргумент функции subst г (смещение) определяет начало интересующей вас подстроки; положительные значения отсчитываются от начала строки, а отрицательные — с конца. Если смещение равно 0, подстрока начинается с начала. Третий аргумент определяет длину подстроки.

```
$string = "This is what you have";
# +01234567890123456789 0 Прямое индексирование (слева направо)
# 109876543210987654321- Обратное индексирование (слева направо)
О соответствует 10, 20 и т. д.

$first = suPstr($string, 0, 1); # "T"
$start =.substr($string, 5, 2); # "is"
                                13); # "you have"
$last = substr($string, -1); # "e"
$end = substr($string, -4); # "have"
$piece = substr($string, -8, 3); ft "you"
```

Однако функция subst г позволяет не только просматривать части строки, но и изменять их. Дело в том, что subst г относится к экзотической категории *левосторонних функций,* то есть таких, которым при вызове можно присвоить значение. К тому же семейству относятся функции vec, pos и keys (начиная с версии 5.004). При некоторой фантазии функции local и ту также можно рассматривать как левосторонние.

```
$string = "This is what you have";
print $string;
This is what you have
substr($string, 5, 2) = "wasn't"; # заменить "is" на "wasn't"
This wasn't what you have
substr($string, -12) = "ondrous"; # "This wasn't wondrous"
This wasn't wondrous
substr($string, 0, 1) = "";  # Удалить первый символ
his wasn't wondrous
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
substr($string, -10) =      "";      # Удалить последние 10 символов
his  wasn'
```

Применяя оператор =~ в сочетании с операторами s///, m// или tr///, можно заставить их работать только с определенной частью строки:

```
# =~ применяется для поиска по шаблону
if (substr($string, -10) =" /pattern/) {
    print "Pattern matches in last 10 characters\n";


# подставить "at" вместо "is", ограничиваясь первыми пятью символами
substr($string, 0, 5) =" s/is/at/g;
```

Более того, подстроки даже можно поменять местами, используя с каждой стороны присваивания несколько вызовов substr:

```
# Поменять местами первый и последний символ строки
$a = "make a hat";
(substr($a,0,1), substr($a,-1)) = (substr($a, -1), substr($a,0, 1));
print $a;
take a ham
```

Хотя функция unpack не является левосторонней, она работает значительно быстрее substr, особенно при одновременном извлечении нескольких величин. В отличие от substr она неподдерживает непосредственные смещения. Вместо этого символ "x" нижнего регистра с числом пропускает заданное количество байт в прямом направлении, а символ "X" верхнего регистра — в обратном направлении.

```
# Извлечение подстроки функцией unpack
$a = "To be or not to be";
$b = unpack("x6 A6", $a); # Пропустить 6 символов, прочитать 6 символов
print $b;
or  not

($b, $c) = unpack("x6 A2 X5 A2", $a); # Вперед 6, прочитать 2;
                                      # назад 5, прочитать 2
print "$b\n$c\n";
or
be
```

Иногда строка "режется" на части в определенных позициях. Предположим, вам захотелось установить позиции разреза перед символами 8, 14, 20, 26 и 30 — в каждом из перечисленных столбцов начинается новое поле. В принципе можно вычислить форматную строку unpack - "A7 A6 A6 A4 A*", но программист на Perl по природе ленив и не желает попусту напрягаться. Пусть за него работает Perl. Воспользуйтесь приведенной ниже функцией *cut2fmt:*

```
sub cut2fmt {
    my(@positions) = @_;
    my $template = '';
    my $lastpos =1;
            $place(@positions) {
        $template .= "A" . ($place - $lastpos) . " ";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        $lastpos   = $place;
    }
    $template .= "A*";
            $template;
>
$fmt = cut2fmt(8, 14, 20, 26, 30);
print "$fmt\n";
A7 A6 A6 A6 A4 A*
```

Возможности функции unpack выходят далеко за пределы обычной обработки текста. Она также обеспечивает преобразование между текстовыми и двоичными данными.

Смотри также

Описание функций unpack и substr в perlfunc(l); подпрограмма cut2fmt из рецепта 1.18. Применение unpack для двоичных данных демонстрируется в рецепте 8.18.

1.2. Выбор значения по умолчанию

Проблема

Требуется закрепить за скалярной переменной значение по умолчанию, но лишь в том случае, если оно не было задано ранее. Довольно часто требуется, чтобы стандартное значение переменной жестко кодировалось в программе, но его можно было переопределить из командной строки или переменной окружения.

Решение

Воспользуйтесь оператором || или ||=,работающим как со строками, так и с числами:

```
# Использовать $b, /если значение $Ь истинно, и $c в противном случае
$a = $b || $c;

# Присвоить $x значение $y, но лишь в томслучае,
# если $x не является истинной
$x ||= $y;
```

Если ваша переменная может принимать значения 0 или "O", воспользуйтесь функцией defined:

```
# Использовать $b, если значение $b определено, и $c в противном случае
$a = defined($b) ? $Ь : $c;
```

Комментарий

Главное отличие между этими двумя приемами (defined и ||) состоит, прежде всего, в том, что именно проверяется — определенность или истинность. В мире Perl три определенных значения являются ложными: 0, "0" и "". Если ваша переменная содержит одну из этих величин, но вы не хотите изменять ее, || не подои-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

дет — приходится выполнять неуклюжие проверки с defi ned. Часто бывает удобно организовать программу так, чтобы принималась в расчет истинность или ложность переменных, а не их определенность.

В отличие от других языков, где возвращаемые значения ограничиваются О и 1, оператор ‖ Perl обладает более интересным свойством: он возвращает первый (левый) операнд, если тот имеет истинное значение; в противном случае возвращается второй операнд. Оператор && ведет себя аналогично (для второго выражения), но этот факт используется реже. Для операторов несущественно, что представляют собой их операнды — строки, числа или ссылки; подойдет любое скалярное значение. Они просто возвращают первый операнд, из-за которого все выражение становится истинным или ложным. Возможно, это расходится с возвращаемым значением в смысле булевой алгебры, но такими операторами удобнее пользоваться.

Это позволяет установить значение по умолчанию для переменной, функции или более длинного выражения в том случае, если первый операнд не подходит. Ниже приведен пример использования ‖, в котором $foo присваивается либо $bar, либо, если значение $bar ложно, — строка "DEFAULT VALUE":

```
$foo = $bar || "DEFAULT VALUE"
```

В другом примере переменной $dir присваивается либо первый аргумент командной строки программы, либо "/tmp", если аргумент не указан:

```
$dir = shift(@ARGV) || "/tmp"
```

То же самое можно сделать и без изменения @ARGV:

```
$dir = $ARGV[0] || "/tmp"
```

Если 0 является допустимым значением $ARGV[0], использовать ‖ нельзя, потому что вполне нормальное значение будет интерпретировано как ложное. Приходится обращаться к тернарному оператору выбора:

```
$dir = defined($ARGV[0]) ? shift(@ARGV) : "/tmp";
```

То же можно записать и иначе, со слегка измененной семантикой:

```
$dir = @ARGV ? $ARGV[0] : "/tmp"
```

Мы проверяем количество элементов в @ARGV. В условии оператора выбора (?:) @ARGV интерпретируется в скалярном контексте. Значение будет ложным лишь при нулевом количестве элементов, в этом случае будет использоваться "/tmp". Во всех остальных ситуациях переменной (когда пользователь вводит аргумент) будет присвоен первый аргумент командной строки.

Следующая строка увеличивает значение %count, при этом в качестве ключа используется значение $shell, а если оно ложно — строка "/bin/sh".

```
$count{ $shell || "/bin/sh" }++;
```

В одном условии можно объединить несколько альтернативных вариантов, как показывает следующий пример. Результат совпадает с первым операндом, имеющим истинное значение.

```
# Определить имя пользователя в системе UNIX
$user = $ENV{USER}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
|| $ENV{LOGNAME}
|| getlogin()
|| (getwuid($<))[0]
|| "Unknown uid number $<";
```

Оператор && работает аналогично; он возвращает первый операнд, если этот операнд ложен. В противном случае возвращается второй операнд. Поскольку ложные значения представляют интерес существенно реже, чем истинные, это свойство используется не так часто. Одно из возможных применений продемонстрировано в рецепте 8.13.

Оператор присваивания ||= выглядит странно, но работает точно так же, как и остальные операторы присваивания. Практически для всех бинарных операторов Perl `$VAR OP= VALUE` означает `$VAR = $VAR OP VALUE`; например, `$a += $b` — то же, что и `$a = $a + $b`. Следовательно, оператор ||= может использоваться для присваивания альтернативного значения переменной. Поскольку || выполняет простую логическую проверку (истина или ложь), у него не бывает проблем с неопределенными значениями, даже при использовании ключа *-w*.

В следующем примере ||= присваивает переменной `$starting_point` значение заданоранее. Предполагается, что `$starting_point` не принимает значений 0 или "0", а если принимает — то такие значения должны быть заменены:

```
$starting_point ||=
```

В операторах присваивания || нельзя заменять оператором or, поскольку or имеет слишком низкий приоритет. Выражение `$a = $b or $c` эквивалентно `($a = $b) or $c`. В этом случае переменной `$b` всегда присваивается `$a`, а это совсем не то, чего вы добивались.

Не пытайтесь распространить это любопытноеприменение || и ||= соскалярных величин на массивы и хэши. У вас ничего не выйдет, потому что левый операнд интерпретируется в скалярном контексте. Приходится делать что-нибудь подобное:

```
@a = @b unless @a;      # Копировать, если массив пуст
@a = @b ? @>b : @c;     # Присвоить @b, если он не пуст, иначе @c
```

> Смотри также

Описание оператора || в *perlop(1)*; описание функций defined и exists в *perlfunc(i)*.

1.3. Перестановка значений без использования временных переменных

Проблема

Требуется поменять значения двух скалярных переменных, но вы не хотите использовать временную переменную.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь присваиванием по списку:

```
($VAR1, $VAR2) = ($VAR2, $VAR1);
```

Комментарий

В большинстве языков программирования перестановка значений двух переменных требует промежуточного присваивания:

```
$temp = $a;
$a    = $b;
$b    = $temp;
```

В Perl дело обстоит иначе. Язык следит за обеими сторонами присваивания и за тем, чтобы ни одно значение не было случайно стерто. Это позволяет избавиться от временных переменных:

```
$a    = "alpha";
$b    = "omega";
($a, $b) = ($b, $a);    # Первый становится последним - и наоборот
```

Подобным способом можно поменять местами сразу несколько переменных:

```
($alpha, $beta, $production) = qw(January March August);
# beta перемещается в alpha,
# production - в beta,
# alpha - в production
($alpha, $beta, $production) = ($beta, $production, $alpha);
```

После завершения этого фрагмента значения переменных `$alpha`, `$beta` и `$production` будут равны соответственно "March", "August" и "January".

Смотри также

Раздел "List value constructors" perlop(l).

1.4. Преобразование между символами и ASCII-кодами

Проблема

Требуется вывести код, соответствующий некоторому символу в кодировке ASCII, или наоборот — символ по ASCII-коду.

Решение

Воспользуйтесь функцией ord для преобразования символа в число или функцией chr — для преобразования числа в символ:

```
$num  = ord($char);
$char = chr($num);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Формат %c в функциях printf и sprintf также преобразует число в символ:

```
$char = sprintf("%c", $num);      # Медленнее, чем chr($num)
printf("Number %d is character %c\n", $num, $num);
Number 101 is character e
```

Шаблон C*, используемый в функциях pack и unpack, позволяет быстро преобразовать несколько символов:

```
@ASCII = unpack("C*", $string);
@STRING = pack("C*", $ascii);
```

Комментарий

В отличие от низкоуровневых, нетипизованных языков вроде ассемблера, Perl не считает эквивалентными символы и числа; эквивалентными считаются *строки* и числа. Это означает, что вы не можете произвольно присвоить вместо символа его числовое представление, или наоборот. Для преобразования между символами и их числовыми значениями в Perl существуют функции chr и ord, взятые из Pascal:

```
$ascii_value = ord("e");    # Теперь 101
$character   = chr(101);    # Теперь "e"
```

Символ в действительности представляется строкой единичной длины, поэтому его можно просто вывести функцией print или с помощью формата %s функций printf и sprintf. Формат %c заставляет printf или sprintf преобразовать число в символ, однако он не позволяет вывести символ, который уже хранится в символьном формате (то есть в виде строки).

```
printf("Number %d is character %c\n", 101, 101);
```

Функции pack, unpack, chr и ord работают быстрее, чем sprintf. Приведем пример практического применения pack и unpack:

```
@ascii_character_numbers = unpack("C*", "sample");
print "@ascii_character_numbers\n";
115 97 109 112 108 101

$word = pack("C*", @ascii_character_numbers);
$word = pack("C*", 115, 97, 109, 112, 108, 101);  # То же самое
print "$word\n"
sample
```

А вот как превратить HAL в IBM:

```
$hal = "HAL";
@ascii = unpack("C*", $hal);
                (@ascii) {
    $val++;       #      Увеличивает каждый ASCII-код на 1
}
$ibm = pack("C*", @ascii);
print "$ibm\n";      #     Выводит "IBM"
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Функция ord возвращает числа от 0 до 255. Этот диапазон соответствует типу данных unsigned char языка C.

Смотри также

Описание функций chr, ord, printf, sprintf, pack и unpack в *perlfunc(1)*.

1.5. Посимвольная обработка строк

Проблема

Требуется последовательно обрабатывать строку по одному символу.

Решение

Воспользуйтесь функцией split с пустым шаблоном, чтобы разбить строку на отдельные символы, или функцией unpack, если вам нужны лишь их ASCII-коды:

```
@array = split(//, $string);

@array = unpack("C*", $string);
```

Или последовательно выделяйте очередной символ в цикле:

```
while (/(.)/g) { # .здесь не интерпретируется как новая строка
    # Сделать что-то полезное с $1
}
```

Комментарий

Как говорилось выше, фундаментальной единицей текста *в* Perl является строка, а не символ. Необходимость посимвольной обработки строк возникает достаточно редко. Обычно такие задачи легче решаются с помощью высокоуровневых операций Perl (например, поиска по шаблону). Пример приведен в рецепте 7.7, где для поиска аргументов командной строки используются подстановки.

Если вызвать split с шаблоном, который совпадает с пустой строкой, функция возвращает список отдельных символов строки. При намеренном использовании эта особенность оказывается удобной, однако с ней можно столкнуться и случайно. Например, /X*/ совпадает с пустой строкой. Не исключено, что вам встретятся и другие ненамеренные совпадения.

Ниже приведен пример, который выводит символы строки "an apple a day", отсортированные в восходящем порядке ASCII-кодов:

```
%seen = ();
$string = "an apple a day";
        $byte (split //, $string) {
    $seen{$1}++;
}
print "unique chars        ", sort(keys %seen), "\n";
unique chars are:  adelnpy
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решения с функциями split и unpack предоставляют массив символов, с которым можно работать. Если массив не нужен, воспользуйтесь поиском по шаблону в цикле while с флагом /д, который будет извлекать по одному символу:

```
%seen = ();
$string = "an apple a day";
while ($string =~ /(.)/g) {
    $seen{$1}++;
}
print "unique chars ", sort(keys %seen), "\n";
unique chars adelnpy
```

Как правило, посимвольная обработка строк не является оптимальным решением. Вместо использования index/substr или split/unpack проще воспользоваться шаблоном. В следующем примере 32-разрядная контрольная сумма вычисляется вручную, но лучше поручить работу функции unpack — она сделает то же самое намного эффективнее.

Следующий пример вычисляет контрольную сумму символов $string в цикле Приведенный алгоритм не оптимален; просто мы используем традиционную и относительно легко вычисляемую сумму. За более достойной реализацией контрольной суммы обращайтесь к модулю MD5 на CPAN.

```
$sum = 0;
(unpack("C*", $string)) {
    $sum += $ascval;
}
print "sum is $sum\n";
# Для строки "an apple a day" выводится сумма 1248
```

Следующий вариант делает то же самое, но намного быстрее:

```
$sum = unpack("%32C", $string);
```

Это позволяет эмулировать программу вычисления контрольной суммы SysV:

```
#!/usr/bin/perl
# sum - Вычисление 16-разрядной контрольной суммы всех входных файлов
$checksum = 0;
while (o) { $checksum += unpack("%16C*", $_) }
$checksum %= (2 ** 16) - 1;
print "$checksum\n";
```

На практике пример использования выглядит так:

```
% perl sum /etc/termcap
1510
```

Если у вас установлена GNU-версия sum, для получения идентичного ответа для того же файла ее следует вызвать с параметром *-sysv:*

```
% sum -sysv /etc/termcap
1510 851 /etc/termcap
```

В примере 1.1 приведена еще одна крошечная программа, в которой также реализована посимвольная обработка входных данных. Идея заключается в том, что-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

бы вывод каждого символа сопровождался небольшой паузой — текст будет появляться перед аудиторией в замедленном темпе, и его будет удобнее читать.

Пример 1.1. slowcat

```
#!/usr/bin/perl
8 slowcat - з а м е д л е н н ы й вывод
' и использование: slowcat [-DELAY] [files...],
Я где DELAY - задержка
$DELAY = ($ARGV[0] =' /'-([.\d]+)/) ? (shift, $1) : 1;
$I = 1;
while (o) {
    for (split(//)) {
        print;
        select(undef,undef,undef, 0.005 * $DELAY);
```

Смотри также

Описание функций split и unpack в *perlfunc(1);* применение select для организации задержки объясняется в рецепте 3.10.

1.6. Обратная перестановка слов или символов

Проблема

Требуется изменить порядок символов или слов в строке на противоположный.

Решение

Для перестановки байтов воспользуйтесь функцией скалярном контексте:

Для перестано вки слов воспользуйтесь списковом контексте с функциями split и join:

= join(" ", split(" ", $string));

Комментарий

У функции существуют два варианта применения. В скалярном контексте функция объединяет аргументы и возвращает полученную строку в обратном порядке. В списковом контексте функция возвращает аргументы в обратном порядке. При использовании перестановки символов в неочевидной ситуации используйте функцию scalar для форсированного применения скалярного контекста.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

$gnirts = Перестановка символов $string

@sdrow = Перестановка элементов @sdrow

$confused = Перестановка букв в join("", @words)

Приведем пример обратной перестановки слов в строке. Пробел (" ") в качестве шаблона split является особым случаем. Он заставляет split использовать в качестве разделителя смежные пропуски (whitespace) и отбрасывать начальные пустые поля (по аналогии с awk). Обычно split отбрасывает только конечные пустые поля.

```
# Обратная перестановка слов
$string = 'Yoda said, "can you see this?"';
@allwords = split(" ", $string);
          = join(" ",          @allwords);
print $revwords, "\n";
this?" see you "can said, Yoda
```

Временный массив @allwords можно убрать и сделать все в одной строке:

```
          = join(" ",          split(" ", $string));
```

Смежные пропуски в $string превращаются в один пробел в $revw ords. Чтобы сохранить существующие пропуски, поступите так:

```
          = join("",           split (/(\s+)/, $string));
```

С помощью функции можно проверить, является ли слово палиндромом (то есть читается ли одинаково в обоих направлениях):

```
$word =
$is_palindrome = ($word eq
```

Программа для поиска длинных палиндромов в файле */usr/dict/words* записывается в одну строку:

```
% perl -nle                    length >5' /usr/dict/words
deedeed
degged
deified
denned
hallah
kakkak
murdrum

rotator
sooloos
tebbet

tut-tut
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Описание функций split, scalar в *perlfunc(1);* раздел "Switches" *perlrun(1).*

1.7. Расширение и сжатие символов табуляции

Проблема

Требуется преобразовать символы табуляции в строке в соответствующее количество пробелов, или наоборот. Преобразование пробелов в табуляцию сокращает объем файлов, имеющих много смежных пробелов. Преобразование символов табуляции в пробелы может понадобиться при выводе на устройства, которые не понимают символов табуляции или считают, что они находятся в других позициях.

Решение

Примените подстановку весьма странного вида:

```
while ($string =" s/\t+/' ' x (length($&) * 8 - length($') % 8)/e) {
    # Выполнять пустой цикл до тех пор,
    # пока выполняется условие подстановки
}
```

Также можно воспользоваться стандартным модулем Text::Tabs:

```
use Text::Tabs;
@expanded_lines = expand(@lines_with_tabs);
@tabulated_lines = unexpand(@lines_without_tabs);
```

Комментарий

Если позиции табуляции следуют через каждые N символов (где N обычно равно 8), их несложно преобразовать в пробелы. В стандартном, "книжном" методе не используется модуль Text::Tabs, однако разобраться в нем непросто. Кроме того, в нем используется переменная $', одно упоминание которой замедляет поиск по шаблону в программе. Причина объясняется в разделе "Специальные переменные" введения к главе 6.

```
while (<>) {
    1 while s/\t+/' ' x (length($&) * 8 - length($`) %8)/e;
    print;
}
```

Вы смотрите на второй цикл while и не можете понять, почему его нельзя было включить в конструкцию s///g? Потому что вам приходится каждый раз заново пересчитывать длину от начала строки (хранящуюся в $`), а не от последнего совпадения.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Загадочная конструкция 1 while CONDITION эквивалентна while (CONDITION) {}, но более компактна. Она появилась в те дни, когда первая конструкция работала в Perl несравнимо быстрее второй. Хотя сейчас второй вариант почти не уступает по скорости, первый стал удобным и привычным.

Стандартный модуль Text::Tabs содержит функции преобразований в обоих направлениях, экспортирует переменную $tabstop, которая определяет число пробелов на символ табуляции. Кроме того, это не приводит к снижению быстродействия, потому что вместо $& и $' используются $1 и $2:

```
use Text::Tabs;
$tabstop = 4;
while (<>) { print expand($_) }
```

Модуль Text::Tabs также может применяться для "сжатия" табуляции. В следующем примере используется стандартное значение $tabstop, равное 8:

```
use Text::Tabs;
while (<>) { print unexpand($_) }
```

Смотри также

Страница руководства модуля Text::Tabs; описание оператора s/// в *perlre*(1) *и perlop*(1).

1.8. Расширение переменных во входных данных

Проблема

Имеется строка, внутри которой присутствует ссылка на переменную:

```
You owe $debt to me.
```

Требуется заменить имя переменной $debt в строке ее текущим значением.

Решение

Если все переменные являются глобальными, воспользуйтесь подстановкой с символическими ссылками:

```
$text =~ s/\$(\w+)/${$1}/g;
```

Но если среди переменных могут встречаться лексические (ту) переменные, следует использовать /ее:

```
$text =~ s/(\$\w+)/$1/gee;
```

Комментарий

Первый способ фактически сводится к следующему: мы ищем нечто похожее на имя переменной, а затем интерполируем ее значение посредством символическо-

somevar, ${$1}

дет равно содержимому $somevar. Такой вариант не будет работать при действую-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

щей директиве use st rict ' refs',потому что она запрещает символическое разыменование.

Приведем пример:

```
use vars qw($rows $cols);
no strict 'refs';      # для приведен ного ниже ${$1}
my $text;

($rows, $cols) = (24, 80);
$text = q(I am $ rows high and $cols long); # апострофы!
$text =~ s/\$(\w+)/${$1}/g;
print $text;
I am 24 high and 80 long
```

Возможно, вам уже приходилось видеть, как модификатор подстановки /e используется для вычисления заменяющего выражения, а не строки. Допустим, вам потребовалось удвоить, каждое целое число в строке:

```
$text = "I am 17 years old";
$text =~ s/(\d+)/2 * $1/eg;
```

Перед запуском программы, встречая /e при подстановке, Perl компилирует код заменяющего выражения вместе с остальной программой, задолго до фактической подстановки. При выполнении подстановки $1 заменяется найденной строкой. В нашем примере будет вычислено следующее выражение:

```
2 * 17
```

Но если попытаться выполнить следующий фрагмент:

```
$text = 'I am $AGE years old'; # Обратите внимание на апострофы!
$text =~ s/(\$\w+)/$1/eg;    #    НЕВЕРНО
```

при условии, что $text содержит имя переменной $AGE, Perl послушно заменит $1 на $AGE и вычислит следующее выражение:

```
'$AGE'
```

В результате мы возвращаемся к исходной строке. Чтобы получить значение переменной, необходимо *снова* вычислить результат. Для этого в строку добавляется еще один модификатор /e:

```
$text =~ s/(\$\w+)/$1/eeg;    #    Находит переменные my()
```

Да, количество модификаторов /e может быть любым. Только первый модификатор компилируется вместе с программой и проверяется на правильность синтаксиса. В результате он работает аналогично конструкции eval {BLOCK}, хотя и не перехватывает исключений. Возможно, лучше провести аналогию с do {BLOCK}.

Остальные модификаторы /e ведут себя иначе и больше напоминают конструкцию eval "STRING". Они не компилируются до выполнения программы. Маленькое преимущество этой схемы заключается в том, что вам не придется вставлять в блок директиву no strict 'refs'. Есть идругое огромное преимущество: этот

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

механизм позволяет находить лексические переменные, созданные с помощью my, — символическое разыменование на это не способно.

В следующем примере модификатор /x разрешает пропуски и комментарии в шаблоне подстановки, а модификатор /e вычисляет правостороннее выражение на программном уровне. Модификатор /e позволяет лучше управлять обработкой ошибок или других экстренных ситуаций:

```
# Расширить переменные в $text. Если переменная не определена,
# вставить сообщение об ошибке.
$text =~ s{
     \$              #              Найти знак доллара
    (\w+)            #              Найти "слово" и сохранить его в $1
}{
    no strict 'refs';
    if (defined $$1) {
        $$1;                    И Расширять только глобальные переменные
    } else {
        "[NO VARIABLE: \$$1]";  " Сообщение об ошибке
    }
>egx;
```

Обратите внимание на изменение синтаксиса $$1 в Perl 5.004; когда-то это выражение означало ${$}1, а теперь оно означает ${$1}. Для обеспечения обратной совместимости в строках оно сохраняет старый смысл (но выдает предупреждение с -w). Запись ${$1} используется в строках для того, чтобы предотвратить разыменование PID. Если значение $$ равно 23448, то $$1 в строке превращается в 234481, а не в значение переменной, имя которой хранится в $1.

Смотри также

Описание оператора s/// в *perlre*(1) и *perlop*(1); описание функции eval в *perlfunc(1)*. Аналогичное использование подстановок встречается в рецепте 20.9.

1.9. Преобразование регистра

Проблема

Строку с символами верхнего регистра необходимо преобразовать в нижний регистр, или наоборот.

Решение

Воспользуйтесь функциями lc и uc со служебными командами \L и \U:

```
use locale;                 # Необходимо в 5.004 и выше

$big = uc($little);      #      "bo peep" -> "BO PEEP"
$little = lc($big);         # "JOHN"    -> "john"
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$big = "\U$little";     #     "bo peep" -> "BO PEEP"
$little = "\L$big";     #     "JOHN"    -> "john"
```

Для замены отдельного символа используйте функции lcfirstи ucfirst со служебными командами \l и \u:

```
$big = "\u$little";     #     "bo"      -> "Bo"
$little = "\l$big";     #     "BoPeep   -> "boPeep"
```

Комментарий

Функции и служебные команды выглядят по-разному, но делают одно и то же. Допускается указание регистра как первого символа, так и целой строки. Вы даже можете форсировать применение верхнего регистра для первого символа и нижнего — для всех остальных.

Встречая директиву use locale, функции преобразования регистра Perl и механизм поиска по шаблону начинают "уважать" правила вашего национального языка. Благодаря ей становится возможным поиск символов с диакритическими элементами и т. д. Одна из распространенных ошибок — преобразование регистра с помощью tr///.Да, мы хорошо помним, что в одном из старых изданий этой книги рекомендовали использовать tr/A-Z/a-z/. В свое оправдание можем лишь сказать, что в то время другого способа не существовало. Такое решение работает не всегда, поскольку из него выпадают все символы с умляутами, седилями и прочими диакритическими элементами, встречающимися во многих языках. Команды преобразования регистра uc и \U понимают эти символы и обеспечивают их правильное преобразование (по крайней мфе, если в программе присутствует директива use locale). Исключение составляет немецкий язык; символ Я в верхнем регистре выглядит как SS, но в Perl такое преобразование не поддерживается.

```
use locale;

$beast    = "dromedary";
Я Изменить регистр разных символов $beast
$capit    = ucfirst($beast);      # Dromedary
$capit    = "\u\L$beast";     #     (то же)
$capall   = uc($beast);       #     DROMEDARY
$capall   = "\U$beast";       #     (то же)
          = lcfirst(uc($beast)); # dROMEDARY
            "\l\U$beast";
```

Как правило, служебные команды обеспечивают согласованное применение регистра в строке:

```
# Преобразовать первый символ каждого слова в верхний регистр,
# а остальные символы - в нижний
$text = "tHIS is a loNG liNE";
$text =~ s/(w+)/\u\L$1/g;
print $text;
This Is A Long Line
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Ими также можно пользоваться для выполнения сравнений без учета регистра:

```
if (uc($a) eq uc($b)) {
    print "a and b are the same\n";
}
```

Программа *randcap* из примера 1.2 случайным образом преобразует в верхний регистр примерно 20 процентов вводимых символов. Пользуясь ей, можно свободно общаться с 14-летними WaREz d00Dz.

Пример 1.2. randcap

```
#!/usr/bin/perl -p
# randcap: фильтр, который случайным образом
# преобразует к верхнему регистру 20% символов
# В версии 5.004 вызов srand() необязателен.
BEGIN {srand(time() " ($$ + ($$ << 15))) }
sub randcase { rand(100) < 20 ? "\u$_[0]" : "\l$_[0]" }
s/(\w)/randcase($1)/ge;

% randcap < genesis | head -9
boOk 01 genesis

001:001 in the BEginning goD

001:002 and the earth wAS without ForM, aND void; AnD darkneSS was
        upon The Face of thedEEp. an the spIrit of GOd movEd upOn
        tHe face of the Waters.

001:003 and god Said, let ligHt: therE wAs LigHt.
```

Более изящное решение — воспользоваться предусмотренной в Perl возможностью применения поразрядных операторов для строк:

```
sub randcase {
    rand(100) < 20 ? ("\040" ^ $1) : $1
}
```

Этот фрагмент изменяет регистр примерно у 20 процентов символов. Однако для 8-разрядных кодировок онработает неверно. Аналогичная проблема существовала и в исходной программе randcase, однако она легко решалась применением директивы use locale.

Следующий пример поразрядных строковых операций быстро отсекает у всех символов строки старшие биты:

```
$string &= "\177" x length($string);
```

Впрочем, о человеке, ограничивающем строки 7-разрядными символами, будут говорить все окружающие — и не в самых лестных выражениях.

Смотри также

Описание функций uc, lc, ucfirst и lcfirst в *perlfunc(1);* описание метасимволов \L, \U, \l и \u в разделе "Quote and Quote-like Operators" *perlop*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

1.10. Интерполяция функций и выражений в строках

Проблема

Требуется интерполировать вызов функции или выражение, содержащиеся в строке. По сравнению с интерполяцией простых скалярных переменных это позволит конструировать более сложные шаблоны.

Решение

Выражение можно разбить на отдельные фрагменты и произвести конкатенацию:

```
$answer = $var1 . func(). $var2;  # Только для скалярных величин
```

Также можно воспользоваться неочевидными расширениями `@{[LIST EXPR]}` или `${\(SCALAR EXPR)}`:

```
$answer = "STRING @{[LIST EXPR]} MORE STRING"
$answer = "STRING ${\(SCALAR EXPR)} MORE STRING";
```

Комментарий

В следующем фрагменте продемонстрированы оба варианта. В первой строке выполняется конкатенация, а во второй — фокус с расширением:

```
$phrase = "I have " . ($n + 1) . "guanacos.";
$phrase = "I have ${\($n + 1)} guanacos.";
```

В первом варианте строка-результат образуется посредством конкатенации более мелких строк; таким образом, мы добиваемся нужного результата без интерполяции. Функция `print` фактически выполняет конкатенацию для всего списка аргументов, и, если вы собираетесь вызвать `print $phrase`, можно было бы просто написать:

```
print "I have ", $n + 1 . "guanacos.\n";
```

Если интерполяция абсолютно неизбежна, придется воспользоваться вторым вариантом, изобилующим знаками препинания. Только символы `@`, `$` и `\` имеют особое значение в кавычках и обратных апострофах. Как и в случаях с `m//` и `s///`, синоним `qx()` не подчиняется правилам расширения для кавычек, если в качестве ограничителя использованы апострофы! `$home = qx'echo home is HOME';` возьмет переменную `$HOME` из командного интерпретатора, а не из Perl! Итак, единственный способ добиться расширения произвольных выражений — **расширить** `${}` или `@{}`, в чьих блоках присутствуют ссылки.

Однако вы можете сделать нечто большее, чем просто присвоить переменной значение, полученное в результате интерполяции. Так, в следующем примере мы конструируем строку с интерполированным выражением и передаем результат функции:

```
some_func("What you want is @{[                    items");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Интерполяция может выполняться и во встроенных документах:

```
die "Couldn't send mail" unless send_mail(<<"EOTEXT", $target);
To: $naughty
From: Your bank
Cc: @{ [get_manager_list($naughty)] }
Date: @{[ do { my $now = `date`; chomp $now; $now} ]} (today)

Dear $naughty,

Today, you bounced check number @{[ 500 + int rand(100) ]} to us.
Your account is now closed.


the management
EOTEXT
```

Расширение строк в обратных апострофах (`` ` ``) оказывается особенно творческой задачей, поскольку оно часто сопровождается появлением ложных символов перевода строки. Создавая блок в скобках за `@` в разыменовании анонимного массива `@{[]}`, как это было сделано в последнем примере, вы можете создавать закрытые (private) переменные.

Все эти приемы работают, однако простое разделение задачи на несколько этапов или хранение всех данных во временных переменных почти всегда оказывается более понятным для читателя.

В версии 5.004 Perl в выражении `${\EXPR }` значение `EXPR` ошибочно вычислялось в списковом, а не скалярном контексте. Ошибка была исправлена в версии 5.005.

Смотри также
perlref(1).

1.11. Отступы во встроенных документах

Проблема

При использовании механизма создания длинных строк *(встроенных документов)* текст должен выравниваться вдоль левого поля; в программе это неудобно. Требуется снабдить отступами текст документа в программе, но исключить отступы из окончательного содержимого документа.

Решение

Воспользуйтесь оператором s// / для отсечения начальных пропусков:

```
# Все сразу
($var = <<HERE_TARGET) =~ s/^\s+//gm;
    далее следует
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    ваш текст
HERE_TARGET

# Или за два этапа
$var = <<HERE_TARGET;
    далее следует
    ваш текст
HERE_TARGET
$var =~ s/^\s+//gm;
```

Комментарий

Подстановка получается весьма прямолинейной. Она удаляет начальные пропуски из текста встроенного документа. Модификатор /m позволяет символу ^ совпадать с началом каждой строки документа, а модификатор /g заставляет механизм поиска повторять подстановку с максимальной частотой (то есть для каждой строки встроенного документа).

```
($definition = 'FINIS')   =~s/^\s+//gm;
    The five variations of camelids
                    camel, his frieds
    the llama and the alpaca, and the
    rather less well-known guanaco
    and vicuca.
FINIS
```

Учтите: во всех шаблонах этого рецепта используется модификатор \s, разрешающий совпадение с символами перевода строки. В результате из встроенного документа будут удалены все пустые строки. Если вы не хотите этого, замените в шаблонах \s на [^\S\n].

В подстановке используется то обстоятельство, что результат присваивания может использоваться в левой стороне =~. Появляется возможность сделать все в одной строке, но она работает лишь при присвоении переменной. При непосредственном использовании встроенный документ интерпретируется как неизменяемый объект, и вы не сможете модифицировать его. Более того, содержимое встроенного документа нельзя изменить *без* предварительного сохранения его в переменной.

Впрочем, для беспокойства нет причин. Существует простой обходной путь, особенно полезный при частом выполнении этой операции. Достаточно написать подпрограмму:

```
sub fix {
    my $string = shift;
    $string =" s/"\s+//gm;



print fix(<<"END");
    Наш документ
END
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Если функция была объявлена заранее, скобки можно опустить:
print fix "END";
    Наш документ
END
```

Как и во всех встроенных **документах,** маркер конца документа (END в нашем примере) должен быть выровнен по левому полю. Если вы хотите снабдить отступом и его, в документ придется добавить соответствующее количество пропусков:

```
($quote = '     FINIS') =~s/^\s+//gm;
        ...we will have peace, when you and all you works have
        perished--and the works of your dark master to whom you would
        deliver          liar, Saruman, and a corrupter of men's
        hearts.  --Theoden in /usr/src/perl/taint.c

    FINIS
$quote =~ s/\s+--/\n--; # Перенести на отдельную строку
```

Если эта операция выполняется с документами, содержащими программный код для eval или просто выводимый текст, массовое удаление всех начальных пропусков нежелательно, поскольку оно уничтожит отступы в тексте. **Конечно,** это безразлично для eval, но не для читателей.

Мы подходим к следующему усовершенствованию — префиксам для строк, которые должны снабжаться отступами. Например, в следующем примере каждая строка начинается с @@@ и нужного отступа:

```
if ($REMEMBER_THE_MAIN) {
    $perl_main_C = dequote<<'  MAIN_INTERPRETER_LOOP';
        @@@ int
        @@@ runops() {
        @@@     SAVEI32(runlevel);
        @@@     runlevel++;
        @@@    while ( op = (*op->op_ppaddr)() ) ;
        @@@     TAINT_NOT;
        @@@
        @@@ }
    MAIN_INTERPRETER_LOOP
    # При желании добавьте дополнительный код
}
```

При уничтожении отступов также возникают проблемы со стихами.

```
sub dequote;
$poem=dequote<<EVER_ON_AND_ON;
       Now far ahead the Road has gone,
          And I must follow, if I can,
       Pursuing it with eager feet,
          Until it joins some larger way
                 paths and errands meet.
          And whither then? I cannot say.
                --Bilbo in /usr/src/perl/pp_ctl.c
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
EVER_ON_AND_ON
print          your poem:\n\n$poem\n";
```

Результат будет выглядеть так:

```
Here's your poem:
Now far ahead the Road has gone,
    And I must follow, if I can,
Pursuing it with eager feet,
    Until it joins some larger way
            paths and errands meet.
    And whither then? I cannot say.
          Bilbo   in /usr/src/perl/pp_ctl.c
```

Приведенная ниже функция dequote справляется со всеми описанными проблемами. При вызове ей в качестве аргумента передается встроенный документ. Функция проверяет, начинается ли каждая строка с общей подстроки (префикса), и если это так — удаляет эту подстроку. В противном случае она берет начальный пропуск из первой строки и удаляет его из всех последующих строк.

```
sub dequote {
    local $_ = shift;
    my ($white, $leader); tt пропуск и префикс, общие для всех строк
    if (/^\s*(?:([^\w\s]+)(\s*).*\n)(?:\s*\1\2?.*\n)+$/) {
        ($white, Header) = ($2, quotemeta($1));
    } else {
        ($white, $leader) = (/^(\s+)/, '');
    }
    s/^\s*?$leader(?:$white)?//gm;

}
```

Если при виде этого шаблона у вас стекленеют глаза, его всегда можно разбить на несколько строк и добавить комментарии с помощью модификатора /x:

```
if (m{
                                # начало строки
    '    \s *                  # 0 и более символов-пропусков
         (?:                    # начало первой несохраненной группировки
             (          #          начать сохранение $1
               [^\w\s]          #     один байт - не пробел и не буквенный символ
                +               #      1 или более
             )          #          закончить сохранение $1
             ( \s* )            #    занести 0 и более пропусков в  буфер $2
             .*       \n        #   искать до конца первой строки
         )          #           конец первой группировки
         (?:         #          начало второй несохраненной группировки
            \s *         #         0 и более символов-пропусков
            \1          #          строка, предназначенная для $1
            \2 ?         #         то, что будет в $2, но дополнительно
            .*      \n      #      искать до конца строки
         ) +                    # повторить идею с группами 1 и более раз
         $           #           до конца строки
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
      }x
    )
{
   ($white, $leader) = ($2, quotemeta($1));
} else {
  ($white, $leader) = (/^(\s+)/, '');
>
s{
                                8 начало каждой строки (из-за /m)
     \s *            #            любое количество начальных пропусков
        ?              #            с минимальным совпадением
      $leader                    # сохраненный префикс
     (?:             #            начать несохраненную группировку
        $white           #          то же количество
     ) ?             #             если после префикса следует конец строки
}{}xgm;
```

Разве не стало понятнее? Пожалуй, нет. Нет смысла уснащать программу банальными комментариями, которые просто дублируютКод. Возможно, перед вами один из таких случаев.

Смотри также

Раздел "Scalar Value Constructors" *perldata*(1); описание оператора s/// в *perlre*(1) и *perlop*(1).

1.12. Переформатирование абзацев

Проблема

Длина текста не позволяет разместить его в одной строке. Требуется разделить его на несколько строк без переноса слов. Например, сценарий проверки стиля читает текстовый файл по одному абзацу и заменяет неудачные обороты хорошими. Замена оборота "применяет функциональные возможности" словом "использует" приводит к изменению количества символов, поэтому перед выводом абзаца его придется переформатировать.

Решение

Воспользуйтесь стандартным модулем Text::Wrap для расстановки разрывов строк в нужных местах:

```
use Text::Wrap;
@OUTPUT = wrap($LEADTAB, $NEXTTAB, @PARA);
```

Комментарий

В модуле Text::Wrap присутствует функция wrap (см. пример 1.3), которая получает список строк и переформатирует их в абзац с длиной строки не более `$Text::Wrap::columns` символов. Мы присваиваем переменной `$columns` значение 20; это гарантирует, что ни одна строка не будет длиннее 20 символов.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Перед списком строк функции wrap передаются два аргумента: один определяет отступ первой строки абзаца, а второй — отступы всех последующих строк.

Пример 1.3. wrapdemo

```
/usr/bin/perl  -w
Я wrapdemo - демонстрация работы Text::Wrap

@input = ("Folding and splicing is the work of an editor,",
          "not a mere collection of silicon",
          "and",
          "mobile electrons!");

use Text::Wrap qw($columns &wrap);

$columns = 20;
print "0123456789" x 2, "\n";
print wrap("    ", "  ", @input), "\n";
```

Результат выглядит так:

```
01234567890123456789
    Folding and
  splicing is the
  work of an
  editor, not a
  mere collection
  of silicon and
  mobile electrons!
```

В результате мы получаем один абзац, в которой каждая строка, кроме последней, завершается символом перевода строки:

```
# Объединение нескольких строк с переносом текста
use Text::Wrap;
undef $/;
print wrap('', '', split(/\s*\n\s*/, <>));
```

Если на вашем компьютере установлен модуль Term::ReadKey с CPAN, вы можете воспользоваться им для определения размеров окна, чтобы длина строк соответствовала текущему размеру экрана. Если этого модуля нет, размер экрана иногда можно взять из $ENV{COLUMNS} или определить по выходным данным команды stty.

Следующая программа переформатирует и слишком короткие, и слишком длинные строки абзаца по аналогии с программой fmt. Для этого разделителем входных записей $/ назначается пустая строка (благодаря чему о читает целые абзацы), а разделителем выходных записей $\ — два перевода строки. Затем абзац преобразуется в одну длинную строку посредством замены всех символов перевода строки (вместе с окружающими пропусками) одиночными пробелами. Наконец, мы вызываем функцию wrap с пустыми отступами первой и всех последующих строк.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Text::Wrap        qw(&wrap $columns);
use Term::ReadKey     qw(GetTerminalSize);
($columns) = GetTerminalSize();
($/, $\) = ('', "\n\n");     # Читать по абзацам, выводить два перевода строки
while (o) {                  # Читать весь абзац
    s/\s*\n\s*/ /g;          " Заменить промежуточные переводы строк пробелами
    print wrap('', '', $_);  # и отформатировать
```

> Смотри также

Описание функций split и join в *perlfunc*(1), страница руководства стандартного модуля Text::Wrap. Применение модуля Term::ReadKey с CPAN продемонстрировано в рецепте 15.6.

1.13. Служебные преобразования символов

Проблема

Некоторые символы выводимой строки (апострофы, запятые и т. д.) требуется преобразовать к специальному виду. Предположим, вы конструируете форматную строку для sprintf и хотите преобразовать символы % в %%.

Решение

Воспользуйтесь подстановкой, которая снабжает префиксом \ или удваивает каждый преобразуемый символ:

```
# Обратная косая черта
$var ="  s/([CHARLIST])/\\$1/g;

# Удвоение
$var =- s/([CHARLIST])/$1$1/g;
```

Комментарий

В приведенных выше решениях $var — модифицируемая переменная, а CHARLIST — список преобразуемых символов, который может включать служебные комбинации типа \t или \п. Если преобразуется всего один символ, можно обойтись без скобок:

```
$string =~  s/%/%%/g;
```

Преобразования, выполняемые в следующем примере, позволяют подготовить строку для передачи командному интерпретатору. На практике преобразование символов ' и " еще не сделает произвольную строку полностью безопасной для командного интерпретатора. Правильно собрать весь список символов так сложно, а риск так велик, что для запуска программ лучше воспользоваться списковыми формами system и exec (см. рецепт 16.11) — в этом случае вы вообще избегаете взаимодействия с интерпретатором.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$string = q(Mom said, "Don't do that.");
$string =- s/(['"])/\\$1/g;
```

Две обратные косые черты в секции заменителя были использованы потому, что эта секция интерпретируется по правилам для строк в кавычках. Следовательно, чтобы получить одну обратную косую черту, приходится писать две. Приведем аналогичный пример для VMS DCL, где дублируются все апострофы и кавычки:

```
$string = q(Mom said, "Don't do that.");
$string =- s/(['"])/$1$1/g;
```

С командными интерпретаторами Microsoft дело обстоит еще сложнее. В DOS и Windows *COMMAND.COM* работает с кавычками, но не с апострофами; не имеет представления о том, как поступать с обратными апострофами, а для превращения кавычек в литерал используетс я обратная косая черта. Почти все бесплатные или коммерческие Unix-подобные интерпретаторы для Windows пытаются исправить эту удручающую ситуацию.

Кроме того, можно определить интервал с помощью символа -, а затем инвертировать его с помощью символа ". Следующая команда преобразует все символы, не входящие в интервал от A до Z:

```
$string =- s/([^A-Z])/\\$1/g;
```

Для преобразования всех неалфавитных символов следует воспользоваться метасимволами \Q и \E или функцией `quotemeta`. Например, следующие команды эквивалентны:

```
$string = "this \Qis a test!\E";
$string = "this is\\ a\\ test!";
$string = "this " . quotemeta("is a test!");
```

> Смотри также

Описание оператора s/// в *perlre*(1) и *perlop*(1); описание функции quotemeta рассматривается в *perlfunc*(1). В рецепте 19.1 рассматривается преобразование служебных символов в HTML, а в рецепте 19.6 — о том, как обойтись без передачи интерпретатору строк со служебными символами.

1.14. Удаление пропусков в обоих концах строки

Проблема

В полученную строку могут входить начальные или конечные пропуски. Требуется удалить их.

Решение

Воспользуйтесь парой подстановок:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$string =~ s/^\s+//;
$string =~ s/\s+$//;
```

Также можно написать специальную функцию, которая возвращает нужное значение:

```
$string = trim($string);
@many   = trim(@many);

sub trim {
        my @out = @_;
        for (@out) {
            s/^\s+//;
            s/\s+$//;
    >
            wantarray  @out   $out[0];
)
```

Комментарий

У этой проблемы имеются различные решения, однако в большинстве случаев приведенный вариант является наиболее эффективным.

Для удаления последнего символа из строки воспользуйтесь функцией chop. В версии 5 была добавлена функция chomp, которая удаляет последний символ в том и только в том случае, если он содержится в переменной $/ (по умолчанию — "\n"). Чаще всего она применяется для удаления завершающего символа перевода строки из введенного текста:

```
# Вывести полученный текст заключенным в ><
while(<STDIN>) {
    chomp;
    print  ">$_<\n";
```

> Смотри также

Описание оператора s/// в *perlre*(1) и *perlop*(1); описание функций chop и chomp в *perlfunc*(1). Начальные пропуски удаляются в функции getnum из рецепта 2.1 и при разделении элементов списка в рецепте 4.1.

1.15. Анализ данных, разделенных запятыми

Проблема

Имеется файл данных, поля которого разделены запятыми. Однако в полях могут присутствовать свои запятые (находящиеся внутри строк или снабженные служебными префиксами). Многие электронные таблицы и программы для работы с

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

базами данных поддерживают списки полей, разделенных запятыми, в качестве стандартного формата для обмена данными.

Решение

Воспользуйтесь следующей процедурой:

```
sub parse_csv {
    my $text = shift;      # Запись со значениями, разделенными запятыми
    my @new = ();
    push(@new, $+) while $text =~ m{
        Я Первая часть группирует фразу в кавычках
        "([^\"\\]*(?:\\.[^\"\\]*)*)",?
           I  ([^,]+),?
           |
       }дх;
       push(@new, undef) if substr($text,-1,1) eq ',';
            @new;       #        значений, которые разделялись запятыми
}
```

Также можно воспользоваться стандартным модулем Text:ParseWords:

```
use Text::ParseWords;

sub parse_csv {
          quoteword(",",0, $_[0]);
}
```

Комментарий

Ввод данных, разделе нных запятыми, — коварная и непростая задача. Все выглядит просто, но в действительности приходится использовать довольно сложную систему служебных символов, поскольку сами поля могут содержать внутренние запятые. В результате подстановка получается весьма сложной, а простая функция split /,/ вообще исключается.

К счастью, модуль Text::ParseWords скрывает от вас все сложности. Передайте функции qoutewords два аргумента и строку разделенных данных. Первый аргумент определяет символ-разделитель (в данном случае — запятая), а второй — логический флаг, который показывает, должна ли возвращаемая строка содержать внутренние кавычки.

Если кавычки должны присутствовать внутри поля, также ограниченного кавычками, воспользуйтесь префиксом \: "like \"this\". Кавычки, апострофы и обратная косая черта — единственные символы, для которых этот префикс имеет специальное значение. Все остальные экземпляры \ остаются в итоговой строке.

Ниже показан пример использования процедуры parse_csv. q<> — всего лишь хитроумный заменитель кавычек, благодаря которому нам не придется расставлять повсюду символы \.

```
$line = q<XYZZY, "", "O'Reilly, Inc", "Wall, Larry", "a \"glug\" bit,", 5,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    "Error,        Dumped">;
@fields = parse_csv($line);
for ($i = 0;$i < @fields; $i++)
    print "$i : $fields[$i]\n";
```

```
XYZZY

O'Reilly, Inc
Wall, Larry
a \"glug\" bit,
5
Error, Core Dumped
```

▷ Смотри также

Описание синтаксиса регулярных выражений в *perlre*(1); документация по стандартному модулю Text::ParseWords.

1.16. Сравнение слов с похожим звучанием

Проблема

Имеются две английские фамилии. Требуется узнать, звучат ли они похожим образом (независимо от написания). Это позволит выполнять неформальный поиск в телефонной книге, в результатах которого наряду со Smith будут присутствовать и другие похожие имена — например, Smythe, Smite и Smote.

Решение

Воспользуйтесь стандартным модулем Text::Soundex:

```
use Text::Soundex;

$CODE = soundex($STRING);
@CODE = soundex(@LIST);
```

Комментарий

Алгоритм soundex хэширует слова (особенно английские фамилии) в небольшом пространстве с использованием простой модели, имитирующей произношение по правилам английского языка. Грубо говоря, каждое слово сокращается до четырехсимвольной строки. Первый символ является буквой верхнего регистра, а прочие — цифры. Сравнивая значения для двух строк, можно определить, звучат ли они похожим образом.

Следующая программа предлагает ввести имя и ищет в файле паролей имена с похожим звучанием. Аналогичный подход может использоваться для баз данных имен, поэтому при желании можно индексировать базу данных по ключам soundex. Конечно, такой индекс не будет уникальным.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Text::Soundex;
use User::pwent;

print "Lookup user: ";
chomp($user = <STDIN>);
exit unless defined $user;
$name_code = soundex($user);

while($uent = getpwent()) {
    ($firstname, $lastname) = $uent->gecos =" /(\w+)[^,]*\b(\w+)/;

    if ($name_code eq soundex($uent->name) ||
        $name_code eq soundex($$lastname)  ||
        $name_code eq soundex($firstname)  )
    {
        printf "%s: %s %s\n", $uent->name, $firstname, $lastname;
```

Смотри также

Документация по стандартным модулям Text::Soundex и User::pwent; man-страница *passwd*(5) вашей системы; "Искусство программирования", том 3, глава 6.

1.17. Программа: fixstyle

Представьте себе таблицу *с* парами устаревших и новых слов.

Старые слова	Новые слова
bonnet	hood
rubber	eraser
lorrie	truck
trousers	pants

Программа из примера 1.4 представляет собой фильтр, который заменяет все встречающиеся в тексте слова из первого столбца соответствующими элементами второго столбца.

При вызове без файловых аргументов программа выполняет функции простого фильтра. Если в командной строке передаются имена файлов, то в них помещаются результаты, а прежниеверсии сохраняются в файлах с расширениями **.orig* (см. рецепт 7.9). При наличии параметра командной строки -v сообщения обо всех изменениях записываются в STDERR.

Таблица пар "исходное слово/заменитель" хранится в основной программе, начиная с END (см. рецепт 7.6). Каждая пара преобразуется в подстановку и накапливается в переменной $code так же, как это делается в программе popgrep2 из рецепта 6.10.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Параметр -t выводит сообщение об ожидании ввода с клавиатуры при отсутствии других аргументов. Если пользователь забыл ввести имя файла, он сразу поймет, чего ожидает программа.

Пример 1.4. fixstyle

```
#!/usr/bin/perl -w
# fixstyle - замена строк секции <DATA> парными строками
# использование: $0 [-v] [файлы...]
use strict;
my $verbose = (@ARGV && $ARGV[0] eq '-v' && shift);

if (@ARGV) {
    $^I = ".orig";          " Сохранить старые файлы
> else {
    warn "$0: Reading from stdin\n" if -t STDIN;


my $code = "while (<>) {\n";
" Читать данные и строить код для eval
while (<DATA>) {
    chomp;
    my ($in, $out) = split /\s*=>\s*/;
    next unless $in && $out;
    $code .= "s{\\Q$in\\E}{$out}g";
    $code .= "&& printf STDERR qq($in => $out at \$ARGV line \$.\\n)"
                                                    if $verbose;
    $code .= ";\n";
}
$code .= "printf;\n}\n";

eval "{ $code > 1" || die;

__END__
analysed        => analyzed
built-in        => builtin
chastized       => chastised
commandline     => command-line
de-allocate     => deallocate
dropin          => drop-in
hardcode        => hard-code
meta-data       => metadata
multicharacter  => multi-character
multiway        => multi-way
non-empty       => nonempty
non-profit      => nonprofit
non-trappable   => nontrappable




turnkey         => turn-key
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Небольшое предупреждение: программа работает быстро, но не в тех случаях, когда количество замен измеряется сотнями. Чем больше секция DATA, тем больше времени потребуется. Несколько десятков замен не вызовут существенного замедления. Более того, для малого количества замен эта версия работает быстрее следующей. Но если запустить программу с несколькими сотнями замен, она начнет заметно отставать.

В примере 1.5 приведена следующая версия программы. При малом количестве замен она работает медленнее, а при большом — быстрее.

Пример 1.5. fixstyle2

```
#!/usr/bin/perl -w
# fixstyle2 = аналог fixstyle для большого количества замен
use strict;
my $verbose = (@ARGV && $AR6V[0] eq '-v' && shift);
my %change = ();
while (<DATA>) {
    chomp;
    my ($in, $out) = split /\s*=>\s*/;
    next unless $in && $out;
    $change{$in} = $out;
}
if (@ARGV) {,
    $^I = ".orig";
} else {
    warn "$0: Reading from stdin\n" if -t STDIN;


while (<>) {
    my $i =0;
    s/^(\s+)// && print $1;     # Выдать начальный пропуск
    for (split /(\s+)/, $_, -1) {
        print( ($i++ & 1) ? $_ : ($change{$_} || $_));



__END__
analysed        => analyzed
built-in        => builtin
chastized       => chastised
commandline     => command-line
de-allocate     => deallocate
dropin          => drop-in
hardcode        => hard-code
meta-data       => metadata
multicharacter  => multi-character
multiway        => multi-way
non-empty       => nonempty
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
non-profit      => nonprofit
non-trappable   => nontrappable

turnkey         => turn-key
```

В новой версии программы каждая строка разбивается на пропуски и слова (относительно медленная операция). Затем слова используются для поиска замены в хэше, что выполняется существенно быстрее подстановки. Следовательно, первая часть работает медленнее, а вторая — быстрее. Выигрыш в скорости зависит от количества совпадений.

Если бы мы не старались сохранить количество пропусков, разделяющих слова, нетрудно сделать так, чтобы вторая версия не уступала первой по скорости даже при небольшом количестве замен. Если вам хорошо известны особенности входных данных, пропуски можно заменить одиночными пробелами. Для этого применяется следующий цикл:

```
# Работает очень быстро, но со сжатием пропусков
while (o) {
    for (split) {
        print $change{$_} || $_, " ";
    }
    print "\n";
}
```

В конце каждой строки появляется лишний пробел. Если это нежелательно, воспользуйтесь методикой рецепта 16.14 и создайте входной фильтр. Вставьте следующий фрагмент перед циклом while, сжимающим пропуски:

```
my $pid = open(STDOUT, "|=");
die "cannot fork: $!" unless defined $pid;
unless ($pid) {
        while (<STDIN>) {
        s/ $//;
        print;
    }
    exit;
```

1.18. Программа:

Многие программы (в том числе *ps*, *netstat*, *ls –l*, *find –ls* и *tcpdump*) часто выдают большие объемы данных. Файлы журналов тоже быстро увеличиваются в размерах, что затрудняет их просмотр. Такие данные можно обработать программой-фильтром типа grep и отобрать из них лишь часть строк, однако регулярные выражения плохо согласуются со сложной логикой — достаточно взглянуть на ухищрения, на которые приходится пускаться в рецепте 6.17.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В частности, нам хотелось бы иметь возможность обращаться с полноценными запросами к выводу программы или файлу журнала. Допустим, вы спрашиваете у ps: "Покажи мне все непривилегированные процессы размером больше 10Кб" или "Какие команды работают на псевдоконсолях?"

Программа psg rep умеет делать все это и бесконечно большее, потому что в ней критерии отбора не являются регулярными выражениями; они состоят из полноценного кода Perl. Каждый критерий последовательно применяется к каждой строке вывода. В результате выводятся лишь те данные, которые удовлетворяют всем аргументам. Ниже приведены примеры критериев поиска и соответствующие им командные строки.

- Строки со словами, заканчивающимися на sh:

```
'/sh\b/'
```

- Процессы с именами команд, заканчивающимися на sh:

```
'command /sh$/'
```

- Процессы с идентификатором пользователя, меньшим 10:

```
'uid 10'
```

- Интерпретаторы с активными консолями:

```
'command 'tty ne "?"'
```

- Процессы, запущенные на псевдоконсолях:

```
'tty =~ /^[p-t]/'
```

- Отсоединенные непривилегированные процессы:

```
'uid "?"'
```

- Большие непривилегированные процессы:

```
'size 2**10' 'uid 0'
```

Ниже показаны данные, полученные при последнем вызове на нашем компьютере. Как и следовало ожидать, в них попал только netscape и его вспомогательный процесс:

```
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN      STA TTY TIME COMMAND
     0   101  9751     1   0   0  14932  9652 do_select S   p1   0:25 netscape
100000   101  9752  9751   0   0  10636   812 do_select S   p1   0:00 (dns
                                                                      helper)
```

В примере 1.6 приведен исходный текст программы psgrep.

Пример 1.6. ps

```
#!/usr/bin/perl -w
#        - фильтрация выходных данных ps
ft с компиляцией пользовательских запросов в программный код
#
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

1.18. Программа:

```
use strict;

tt Все поля из заголовка PS
my @fieldnames = qw(FLAGS UID PID PPID PRI NICE SIZE
                    RSS WCHAN STATTTY TIME COMMAND);

# Определение формата распаковки (в примере
ft жестко закодирован формат ps для Linux)
my $fmt = cut2fmt(8, 14, 20, 26, 30, 34, 41, 47, 59, 63, 67, 72);

my %fields;           # Для хранения данных

die "Thanatos unless @ARGV;
usage: $0 criterion ...
    Each criterion is a                involving:
    @fieldnames
    All criteria must be met for a line to be printed.
Thanatos

tt Создать синонимы для uid, size, UID, SIZE и т.д.
tt Пустые скобки необходимы для создания прототипа без аргументов
for my $name (@fieldnames) {
    no strict 'refs';
    *name = *{lc $name} = sub () { $fields{$name} };


my $code = "sub is_desirable { " . join(" and ", @ARGV) . " .} ";
unless (eval $code. 1) <
    die "Error in code: $@\n\t$code\n";


open (PS, "ps wwaxl |")             || die "cannot fork: $!";
print scalar <PS>;          tt Строка-заголовок
while (<PS> {
    @fields{@fieldnames} = trim(unpack($fmt, $_));
    print if is_desirable();        # Строки, удовлетворяющие критериям
}
close(PS)                       |   |  die "ps failed!";

# Преобразовать позиции разреза в формат распаковки
sub cut2fmt {
    my(@positions) = @_;
    my $template  = '';
    my $lastpos   = 1;
            $place(@positions)
        $template .= "A" . ($place - $lastpos) . " ";
        $lastpos   = $place;
    }
    $template .= "A*";
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 1.6 (продолжение)

```
        $template;


sub trim {
    my @out = @_;
    for (@out) {
        s/^\s+//;
        s/\s+$//;
    }
        wantarray   @out   $out[0];


# Следующий шаблон использовался для определения позиций разреза.
# Далее следует пример входных данных
#123456789012345678901234567890123456789012345678901234567890123456789012345
#         1         2         3         4         5         6         7
# Позиции:
#       8     14    20    26  30  34     41    47          59  63  67   72
#       I     I     I     I   I   I      I     I           I   I   I    I
__END__
 FLAGS   UID   PID  PPID PRI  NI   SIZE   RSS WCHAN        STA TTY TIME COMMAND
   100     0     1     0   0   0    760   432 do_select    S   ?   0:02 init
   140     0   187     1   0   0    784   452 do_select    S   ?   0:02 syslogd
100100   101   428     1   0   0   1436   944 do_exit      S   1   0:00 /bin/
                                                                        login
100140    99 30217   402   0   0   1552  1008 posix_lock   S   ?   0:00 httpd
     0   101   593   428   0   0   1780  1260 copy_thread  S   1   0:00 -tosh
100000   101 30639  9562  17   0    924   496              R   P1  0:00 ps axl
     0   101 25145  9563   0   0   2964  2360              S   P2  0:06 trn
100100     0 10116  9564   0   0   1412   928 setup_frame  T   P3  0:00 ssh -C
                                                                        www
100100     0 26560 26554   0   0   1076   572 setup_frame  T   P2  0:00 less
100000   101 19058  9562   0   0   1396   900 setup_frame  T   P1  0:02 nvi /
                                                                        tmp/a
```

В программе *psgrep* объединены многие приемы, представленные в книге. Об удалении начальных и конечных пропусков рассказано в рецепте 1.14. Преобразование позиций разреза в формат unpack для извлечения полей с фиксированным положением рассматривается в рецепте 1.1. Поискурегулярных выражений в строках посвящена вся глава 6.

Многострочный текст, передаваемый die, представляет собой встроенный документ (см. рецепты 1.10 и 1.11). Присваивание @fields{@fieldnames} заносит сразу несколько величин в хэш %fields. Хэши рассматриваются в рецептах 4.7 и 5.10.

Входные данные программы-примера, расположенные под END, описаны в рецепте 7.6. На стадии разработки для тестирования использовались "консервированные" данные, полученные через файловый манипулятор DATA. Когда программа заработала, мы перевели ее на получение данных из присоединенной команды ps, однако исходные данные были оставлены для будущего переноса на другие платформы и сопровождения. Конвейерный запуск других программ рассматривает-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ся в главе 16 "Управление процессами и межпроцессные взаимодействия", особенно в рецептах 16.10 и 16.13.

Настоящая сила и , что в Perl строковые аргументы могут представлять собой не просто строки, а программный код Perl. Похожий прием использован в рецепте 9.9, за исключением того, что в аргументы пользователя "упакованы" в процедуру `is_desirable`. При этом компиляция строк в код Perl выполняется всего один раз — еще перед запуском той программы, чей вывод мы обрабатываем. Например, при запросе UID ниже 10 будет сгенерирована следующая строка:

```
eval "sub is_desirable { uid < 10 } " . 1;
```

Загадочное . 1 в конце присутствует для того, чтобы при компиляции пользовательского кода команда eval возвращала истинное значение. В этом случае нам даже не придется проверять $@ на предмет ошибок компиляции, как это делается в рецепте 10.12.

Использование произвольного кода Perl в фильтрах для отбора записей — невероятно мощная возможность, но она не является абсолютно оригинальной. Perl многим обязан языку программирования awk, который часто применялся для подобной фильтрации. Один из недостатков awk заключался в том, что он не мог легко интерпретировать входные данные в виде полей фиксированной длины (вместо полей, разделенных особыми символами). Другой недостаток — отсутствие мнемонических имен полей; в awk использовались имена $1, $2 и т. д. К тому же Perl может делать многое такое, на что не способен awk.

Пользовательские критерии даже не обязаны быть простыми выражениями. Например, следующий вызов инициализирует переменную $id номером пользователя nobody и затем использует ее в выражении:

```
'no strict "vars";
BEGIN { $id = getpwnam("nobody") }.
uid == $id '
```

Но как использовать эти слова, uid, command и size, даже не снабжая их символом $ для представления соответствующих полей входных записей? Мы напрямую манипулируем с таблицей символов, присваивая замыкания неявным тип-глобам (typeglobs), которые создают функции с соответствующими именами. Замыкания описаны в рецепте 11.4, а их присвоение тип-глобам для создания синонимов функций — в рецепте 10.14.

Однако в встречается нюанс, отсутствующий в этих рецептах, — речь идет о пустых скобках в замыкании. Благодаря скобкам функция может использоваться в выражениях везде, где допускается отдельная величина (например, строка или числовая константа). В результате создается пустой прототип, а функция обращения к полю (например, uid) вызывается без аргументов, по аналогии со встроенной функцией time. Если не создать для функций пустые прототипы, выражения "uid < 10" или "size / 2 > rss" приведут в замешательство синтаксический анализатор — он увидит в них незаконченный глоб (wildcard glob) или шаблон поиска. Прототипы рассматриваются в рецепте 10.11.

Показанная версия получает входные данные от команды ps в формате Red Hat Linux. Чтобы перенести ее в другую систему, посмотрите, в каких столб-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

цах начинаются заголовки. Такой подход не ограничивается спецификой ps или системы UNIX. Это общая методика фильтрации входных записей с использованием выражений Perl, которая легко адаптируется для другой структуры записи. Поля могут быть выстроены в столбцы, разделены запятыми или заключены в скобки.

После небольшого изменения в функциях отбора программа даже подойдет для работы с пользовательской базой данных. Если у вас имеется массив записей (см. рецепт 11.9), пользователь может указать произвольный критерий отбора:

```
sub id()        { $_->{ID}    }
sub title()     { $_->{TITLE} }
sub executive   { title =~/(?:vice-)?president/i }

Я Критерии отбора указываются при вызове
@slowburners                    !executive } @employees;
```

По причинам, связанным с безопасностью и быстродействием, такой подход редко встречается в реальных механизмах, описанных в главе 14 "Базы данных". В частности, он не поддерживается в SQL, но имея в своем распоряжении Perl и некоторую долю изобретательности, нетрудно создать свой собственный вариант. Подобная методика использована в поисковой системе *http://moxperl.com/cgi-bin/MxScreen*, но вместо получения данных от ps записи представляют собой хэши Perl, загружаемые из базы данных.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

2 Числа

Каждый, кто занимается математическими методами получения случайных чисел, несомненно, впадает в грех.

Джон фон Нейман (1951)

Введение

Числа составляют основные типы данных практически в любом языке программирования, однако даже с ними могут возникнуть неожиданные сложности. Случайные числа, числа с дробной частью, числовые последовательности и преобразования строк в числа — все это вызывает немалые затруднения.

Perl старается по возможности облегчить вам жизнь, и его средства для работы с числами не являются исключением из этого правила. Если скалярное значение интерпретируется в программе как число, то Perl преобразует его в числовую форму. Читаете ли вы числовые данные из файла, извлекаете отдельные цифры из строки или иным образом получаете числа из бесчисленных текстовых источников Внешнего Мира, — вам не приходится преодолевать препятствия в виде неудобных ограничений других языков на пути преобразования ASCII-строк в числа.

Если строка используется в числовом контексте (например, в математическом выражении), Perl старается интерпретировать ее как число, однако у него нет возможности сообщить о том, что строка в действительности не соответствует числу. Встречая не-числовой символ, Perl прекращает интерпретацию строки, при этом не-числовые строки считаются равными нулю, поэтому "A7" преобразуется в 0, а "7 A" — в 7 (хотя флаг **-w** предупредит вас о некорректных преобразованиях). Иногда (например, при проверке вводимых данных) требуется узнать, соответствует ли строка числу. Мы покажем как это делается в рецепте 2.1.

В рецепте 2.16 объясняется, как получить число из строк с шестнадцатеричными или восьмеричными представлениями чисел — например, `"0xff"`. Perl автоматически преобразует литералы в программном коде (поэтому `$a = 3 + 0xff` присвоит `$a` значение 258), но это не относится к данным, прочитанным программой. Вы не можете прочитать `"0xff"` в `$b` и затем написать `$a = 3 + $b`, чтобы присвоить `$a` 258.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

А если трудностей с целыми числами окажется недостаточно, числа с плавающей запятой преподнесут целый букет новых проблем. Во внутреннем представлении дробные числа хранятся в формате с плавающей запятой. Они представляют вещественные числа лишь приближенно, с ограниченной точностью. Для представления бесконечного множества вещественных чисел используется конечное пространство, обычно состоящее из 64 бит или около того. Потеря точности неизбежна.

Числа, прочитанные из файла или встретившиеся в программе в виде литералов, преобразуются из десятичного представления (например, 0.1) во внутреннее. Невозможно точно представить 0.1 в видедвоичного числа с плавающей запятой — подобно тому, как 1/3 невозможно точно представить в видеконечного десятичного числа. Следовательно, двоичное представление 0.1 в действительности отличается от 0.1. Для 20 десятичных разрядов оно равно 0.10000000000000000555.

При выполнении арифметических операций с двоичными представлениями чисел с плавающей запятой накапливаются ошибки. Значение выражения `3 * 0.1` не совпадает с двоичной кодировкой числа 0.3. Это означает, что числа с плавающей запятой в Perl нельзя просто сравнивать с помощью ==. Работе с ними посвящены рецепты 2.2 и 2.3.

В рецепте 2.4 показано, как преобразовать ASCII-строку с двоичным представлением числа (например, "1001") в целое (9 для приведенного примера) и обратно. Рецепт 2.5 описывает три способа выполнения некоторой операции с каждым элементом последовательного множества целых чисел. Преобразование чисел в римскую запись и обратно продемонстрировано в рецепте 2.6. -

Случайным числам посвящено сразу несколько рецептов. Функция Perl `rand` возвращает число с плавающей запятой от 0 до 1 или от 0 до своего аргумента. Мы покажем, как получить случайное число в конкретном интервале, как сделать их "еще более случайными" и как заставить rand генерировать новый набор случайных чисел при каждом запуске программы.

Глава завершается рецептами, относящимися к тригонометрии, логарифмам, умножению матриц, комплексным числам. Заодно вы найдете ответ на часто встречающийся вопрос: "Как включить в выводимое число запятую?"

2.1. Проверка строк на соответствие числам

Проблема

Требуется проверить, соответствует ли строка допустимому числу. Эта проблема часто возникает при проверке входных данных (например, в сценариях CGI).

Решение

Сравните строку с регулярным выражением, которое совпадает со всеми интересующими вас разновидностями чисел:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
if ($string =~ /PATTERN/) {
    # является числом
} else {
    # не является числом
}
```

Комментарий

Все зависит от того, что именно понимать под числом. Даже простые на первый взгляд понятия — например, *целое* — заставят вас поломать голову над тем, какие строки следует отнести к этой категории. Например, что делать с начальным + для положительных чисел? Разрешить, сделать обязательным или запретить? А числа с плавающей запятой представляются таким огромным количеством способов, что у вас в голове перегреется процессор.

Сначала решите, какие символы допустимы, а какие — нет. Затем сконструируйте для отобранных сийволов регулярное выражение. Ниже приведены некоторые стандартные конструкции для самых распространенных ситуаций (что-то вроде полуфабрикатов для нашей поваренной книги).

```
# Содержит нецифровые символы
warn "has nondigits"            if   /\D/;
# Не является натуральным числом
warn "not a natural number"     unless /^\d+$/;        # Отвергает -3
# Не является целым числом
warn "not an integer"           unless /^-?\d+$/;      # Отвергает +3
warn "not an integer"           unless /^[+-]?\d+$/;
# Не является десятичным числом
warn "not a decimal number"     unless /^-?\d+\.?\d*$/; # Отвергает .2
warn "not a decimal number"     unless /^-?(?:d+(?:\.\d*)?|\.\d+)$/;
# Не является вещественным числом C
warn "not a C float"
    unless /^([+-]?)(?=\d|\.\d)\d*(\.\d*)?([Ee]([+-]?\d+))?$/;
```

В этих шаблонах не обрабатываются особые случаи Infinity и NaN в записи IEEE. Если вы не боитесь, что члены комитета IEEE придут к вашему компьютеру и начнут бить вас по голове копиями соответствующих стандартов, вероятно, об этих странных "числах" можно забыть.

Для строк с начальными или конечными пробелами эти шаблоны не подходят. Либо вставьте в них соответствующую логику, либо вызовите функцию `trim` из рецепта 1.14.

В POSIX-системах Perl поддерживает функцию POSIX: : `strtod`. Ёе семантика чрезвычайно громоздка, поэтому мы приведем функцию `getnum` для упрощения доступа. Эта функция получает строку и возвращает либо преобразованное число, либо `undef` для строк, не соответствующих вещественным числам C. Интерфейсная функция `is_numeric` упрощает вызов `getnum` в ситуациях, когда вы просто хотите спросить: "Это вещественное число"?

```
sub getnum {
    use POSIX qw(strtod);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        my $str = shift;
        $str =^ s/^\$+//;
        $str =^ s/\s+$//;
        $! = 0;
    my($num, $unparsed) = strtod($str);
        if (($str eq '') || ($unparsed != 0) || $!) {

        } else {
                $num;
        }
}

sub is_numeric { defined scalar &getnum }
```

> Смотри также

Описание синтаксиса регулярных выражений в *perlre*(1); страница руководства *strtod*(3); документация по стандартному модулю POSIX.

2.2. Сравнение чисел с плавающей запятой

Проблема

Арифметика с плавающей запятой не является абсолютно точной. Сравнивая два числа, вы хотите узнать, совпадают ли они до определенного десятичного разряда. Как правило, именно так *следует* сравнивать числа с плавающей запятой.

Решение

Воспользуйтесь функцией sprintf и отформатируйте числа до определенного десятичного разряда, после чего сравните полученные строки:

```
# equal(NUM1, NUM2, ACCURACY); возвращает true, если NUM1 и NUM2
№ совпадают на ACCURACY десятичных разрядов.

sub equal {
    my ($A, $B, $dp) = @_;
        sprintf("%.${dp}g",        sprintf("%.${dp}g", $B);
}
```

Альтернативное решение — преобразовать числа в целые, умножая их на соответствующий коэффициент.

Комментарий

Процедура equal понадобилась из-за того, что в компьютерах многие числа с плавающей запятой представляются с ограниченной точностью. Дополнительная информация приведена в разделе"Введение".

При фиксированном количестве цифр в дробной части (например, в денежных суммах) проблему можно решить преобразованием в целое число. Если

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

сумма 3.50 будет храниться в виде 350, а не 3.5, необходимость в числах с плавающей запятой отпадает. Десятичная точка снова появляется в выводимых данных:

```
$wage = 536;                # $5.36/час
$week = 40 * $wage;         # $214.40
printf("One week's wage is: \$%.2f\n", $week/100);
```

```
One week's wage is: $214.40
```

Редко требуется сравнивать числа более чем до 15 разряда.

> Смотри также

Описание функции sprintf в *perlfunc*(1); описание переменной $# в странице руководства *perlvar*(1); документация постандартному модулю Math::BigFloat. Функция sprintf используется в рецепте 2.3. Также обращайтесь к разделу 4.2.2 тома 2 "Искусство программирования".

2.3. Округление чисел с плавающей запятой

Проблема

Число с плавающей запятой требуется округлить до определенного разряда. Проблема связана с теми же погрешностями представления, которые затрудняют сравнение чисел (см. рецепт 2.2), а также возникает в ситуациях, когда точность ответа намеренно снижается для получения более наглядного результата.

Решение

Для получения непосредственного вывода воспользуйтесь функциями Perl sprintf или printf:

```
$rounded = sprintf("%FORMATf", $unrounded);
```

Комментарий

Округление серьезно отражается на работе некоторых алгоритмов, потому используемый метод должен быть точно указан. В особо важных приложениях (например, в финансовых вычислениях или системах наведения ракет) грамотный программист реализует свою собственную функцию округления, не полагаясь на встроенную логику языка (или ее отсутствие).

Однако во многих ситуациях можно просто воспользоваться функцией sprintf. Формат f позволяет указать количество разрядов, до которого округляется аргумент. Perl округляет последний разряд вверх, еслиследующаяцифра равна 5 и более, и вниз в противном случае.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$a = 0.255
$b = sprintf("%.2f", $a);
print "Unrounded: $a\nRounded: %.2f\n", $a;

Unrounded: 0.255
Rounded:   0.26
Unrounded: 0.255
Rounded:   0.26
```

Существуют три функции, предназначенные для округления чисел с плавающей запятой до целых: int, ceil и floor. Встроенная функция Perlint возвращает целую часть числа с плавающей запятой (при вызове без аргумента она использует $_). Функции модуля **POSIX** floor и ceil округляют аргументы вверх и вниз, соответственно, до ближайшего целого.

```
use POSIX;

print "number\tint\floor\tceil\n";

@a = { 3.3 , 3.5 , 3.7 , -3.3};
        (@a)
    printf( "% .1f\t% .1f\t% .1f\t% .1f\n",
        $_, int($_), floor($_), ceil($_) );
}

number  int     floor   ceil
 3.3     3.0     3.0     4.0
 3.5     3.0     3.0    "4.0
 3.7     3.0     3.0     4.0
-3.3    -3.0   -4.0    -3.0
```

▷ Смотри также

Описание функций sprintf и int в *perlfunc*(1), описание функций floor и ceil в документации по стандартному модулю POSIX. Методика использования sprintf для округления представлена в рецепте 2.2.

2.4. Преобразования между двоичной и десятичной системами счисления

Проблема

Имеется десятичное число, которое необходимо вывести в двоичном представлении, или наоборот, двоичная последовательность, которую требуется преобразовать в десятичное число. Такие задачи часто возникают при отображении не-текстовых данных — например, полученных в процессе взаимодействия с некоторыми системными функциями и программами.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Чтобы преобразовать целое число Perl в строку, состоящую из единиц и нулей, сначала упакуйте его в сетевой формат "N" (с начальным старшим байтом), а затем снова распакуйте по одному биту (формат "B32").

```
sub dec2bin {
    my $str = unpack("B32", pack("N", shift));
    $str =^ s/^0+(?=\d)//;   # В противном случае появятся начальные нули
    return $str;
}
```

Чтобы преобразовать строку из единиц и нулей в целое число Perl, дополните ее необходимым количеством нулей, а затем выполните описанную выше процедуру в обратном порядке:

```
sub bin2dec {
            unpack("N", pack("B32", substr("0" x 32 . shift, -32)));
}
```

Комментарий

Речь идет о преобразовании чисел между строками вида "00100011" и десятичной системой счисления (35). Строка содержит двоичное представление числа. На этот раз функция sprintf не поможет: в ней не предусмотрен формат для вывода чисел в двоичной системе счисления. Следовательно, нам придется прибегнуть к функциям Perl pack и unpackдля непосредственных манипуляций со строковыми данными.

Функции pack и unpack предназначены для работы со строками. Строки можно интерпретировать как последовательности битов, байты, целые, длинные целые, числа с плавающей запятой в представлении IEEE, контрольные суммы — не говоря уже о многом другом. Обе функции, pack и unpack, по аналогии со sprintf получают форматную строку, которая определяет выполняемые с аргументом операции.

Мы используем pack и unpack для интерпретации строк как последовательностей битов и двоичного представления целого числа. Чтобы понять, каким образом строка интерпретируется как последовательность битов, необходимо хорошо разобраться в поведении функции pack. Строка интерпретируется как последовательность байтов, состоящих из восьми бит. Байты всегда нумеруются слева направо (первые восемь бит образуют первый байт, следующие восемь бит — второй и т. д.), однаковнутри каждого байта биты могут нумероваться как слева направо, так и справа налево.

Функция pack с шаблоном "B" работает с битами каждого байта, пронумерованными слева направо. Именно в этом порядке они должны находиться для применения формата "N", которым мы воспользуемся для интерпретации последовательности битов как 32-разрядного целого.

```
$num = bin2dec('0110110')   # $num = 54                    *
$binstr = dec2bin(54);      # $binstr = 110110
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

2.5. Действия с последовательностями целых чисел

Проблема

Требуется выполнить некоторую операцию со всеми целыми между X и Y. Подобная задача возникает при работе с непрерывной частью массива или в любой ситуации, когда необходимо обработать все числа[1] из заданного интервала.

Решение

Воспользуйтесь циклом for или .. в сочетании с циклом foreach:

```
foreach ($X .. $Y) {
    # $_ принимает все целые значения от X до Y включительно
}

                    {
    # $i принимает все целые значения от X до Y включительно
    }

                          $i++) {
    # $i принимает все целые значения от X до Y включительно
}

          =               $i+=7) {
    # $i принимает целые значения от X до Y включительно с шагом 7
}
```

Комментарий

В первых двух методах используется конструкция $X.. $Y, которая создает список всех целых чисел между $X и $Y. Если $X и $Y расположены далеко друг от друга, это приводит к большим расходам памяти (исправлено в версии 5.005). При организации перебора последовательных целых чисел цикл for из третьего способа расходует память более эффективно.

В следующем фрагменте продемонстрированы все три способа. В данном случае мы ограничиваемся выводом сгенерированных чисел:

```
print "Infancy is:";
        (0 ..
    print "$_ ";
}
print "\n";

print "Toddling is: ";
            ..
```

[1] Точнее, все целые числа. Найти все вещественные числа будет нелегко. Не верите — посмотрите у Кантора.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    print "$i ";
}
print "\n";

print "Childhood is: ";
for ($i = 5; $i <= 12; $i++) {
    print "$i ";
}
print "\n";

Infancy is: 0 1 2
Toddling is: 3 4
Childhood is: 5 6 7 89 10 11 12
```

> Смотри также

Описание операторов for и в *perlsyn*(1).

2.6. Работа с числами в римской записи

Проблема

Требуется осуществить преобразование между обычными числами и числами в римской записи. Такая необходимость часто возникает при оформлении сносок и нумерации страниц в предисловиях.

Решение

Воспользуйтесь модулем Roman с CPAN:

```
use Roman;
$roman = roman($arabic);                    # Преобразование
                                            8 в римскую запись
$arabic = arabic($roman) if isroman($roman); # Преобразование
                                            # из римской записи
```

Комментарий

Для преобразования арабских ("обычных") чисел в римские эквиваленты в модуле Roman предусмотрены две функции, Roman и roman. Первая выводит символы в верхнем регистре, а вторая — в нижнем.

Модуль работает только с римскими числами от **1** до **3999** включительно. В римской записи нет отрицательных чисел или нуля, а для числа 5000 (с помощью которого представляется 4000) используется символ, не входящий в кодировку ASCII.

```
use Roman;
$roman_fifteen = roman(15);                    # "xv"
print "Roman for fifteen is $roman_fifteen\n";
$arabic_fifteen = arabic($roman_fifteen);
print "Converted back, $roman_fifteen is $arabic_fifteen\n";
Roman for fifteen is xv
Converted back, xv is 15
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

> Смотри также

Документация по модулю Roman; рецепт 6.23.

2.7. Генератор случайных чисел

Проблема

Требуется генерировать случайные числа в заданном интервале — например, чтобы выбрать произвольный элемент массива, имитировать бросок кубика в игре или сгенерировать случайный пароль.

Решение

Воспользуйтесь функцией Perl `rand`.

```
$random = int( rand( $Y-$X+1 ) ) + $X;
```

Комментарий

Следующий фрагмент генерирует и выводит случайное число в интервале от 25 до 75 включительно:

```
$random = int( rand(51)) + 25;
print "$random\n";
```

Функция rand возвращает дробное число от 0 (включительно) до заданного аргумента (не включается). Мы вызываем ее с аргументом 51, чтобы случайное число было больше либо равно 0, но никогда не было бы равно 51 и выше. Затем от сгенерированного числа берется целая часть, что дает число от 0 до 50 включительно (функция `int` превращает 50,9999... в 50). К полученному числу прибавляется 25, что дает в результате число от 25 до 75 включительно.

Одно из распространенных применений этой методики — выбор случайного элемента массива:

```
$elt = $array[ rand @array ];
```

Также она часто используется для генерации случайного пароля из заданной последовательности символов:

```
@chars = ( "A" .. "Z", "a" .. "z", 0 .. 9, qw(% ! @ $ % ^ & *) );
$password = join("", @chars[ map { rand @chars } ( 1 ..8 ) ]);
```

Мы генерируем восемь случайных индексов `@chars` с помощью функции тар, извлекаем соответствующие символы в виде среза и объединяем их в случайный пароль. Впрочем, в действительности пароль получается *не совсем* случайным — безопасность вашей системы зависит от стартового значения (seed) генератора случайных чисел на момент запуска программы. В рецепте 2.8 показано, как "раскрутить" генератор случайных чисел и сделать генерируемые числа более случайными.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

> Смотри также

Описание функций `int`, `rand` и `join` в *perlfunc*(1). Случайные числа исследуются в рецептах 2.8–2.10, а используются — в рецепте 1.9.

2.8. Раскрутка генератора случайных чисел

Проблема

При каждом запуске программы вы получаете один и тот же набор "случайных" чисел. Требуется "раскрутить" генератор, чтобы Perl каждый раз генерировал разные числа. Это важно практически для любых применений случайных чисел, особенно для игр.

Решение

Воспользуйтесь функцией Perl `srand`:

```
srand EXPR;
```

Комментарий

Генерация случайных чисел — непростое дело. Лучшее, на что способен компьютер без специального оборудования, — генерация псевдослучайных чисел, равномерно распределенных в области своих значений. Псевдослучайные числа генерируются по математическим формулам, а это означает, что при одинаковом стартовом значении генератора две программы сгенерируют одни и те же псевдослучайные числа.

Функция `srand` задает новое стартовое значение для генератора псевдослучайных чисел. Если она вызывается с аргументом, то указанное число будет использовано в качестве стартового. При отсутствии аргумента `srand` использует величину, значение которой трудно предсказать заранее (относится к Perl 5.004 и более поздним версиям; до этого использовалась функция `time`, значения которой совсем не были случайными). Не вызывайте `srand` в программе более одного раза.

Если вы не вызвали `srand` сами, Perl версий 5.004 и выше вызывает `srand` с "хорошим" стартовым значением при первом запуске `rand`. Предыдущие версии этого не делали, поэтому программы всегда генерировали одну и ту же последовательность чисел. Если вы предпочитаете именно такое поведение, вызывайте `srand` с конкретным аргументом:

```
srand( <STDIN> );
```

То, что Perl старается выбрать хорошее стартовое значение, еще не гарантирует криптографической безопасности сгенерированных чисел от усердных попыток взлома. Информацию о построении надежных генераторов случайных чисел можно найти в учебниках по криптографии.

> Смотри также

Описание функции `srand` в *perlfunc*(1). Примеры ее применения приведены в рецептах 2.7 и 2.9.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

2.9. Повышение фактора случайности

Проблема

Требуется генерировать случайные числа, которые были бы "более случайными», чем выдаваемые генератором Perl. Иногда возникают проблемы, связанные с ограниченным выбором стартовых значений в библиотеках C. В некоторых приложениях последовательность псевдослучайных чисел начинает повторяться слишком рано.

Решение

Воспользуйтесь другими генераторами случайных чисел — например, теми, которые присутствуют в модулях Math::Random и Math::TrulyRandom с CPAN:

```
use Math::TrulyRandom;
$random = truly_random_value();

use Math::Random;
$random = random_uniform();
```

Комментарий

Для генерации случайных чисел в Perl используется стандартная библиотечная функция C rand(3) (впрочем, на стадии компоновки это можно изменить). Некоторые реализации функции `rand` возвращают только 16-разрядные случайные числа или используют слабые алгоритмы, не обеспечивающие достаточной степени случайности.

Модуль Math::TrulyRandom генерирует случайные числа, используя погрешности системного таймера. Процесс занимает некоторое время, поэтому им не стоит пользоваться для генерации большого количества случайных чисел.

Модуль Math::Random генерирует случайные числа с помощью библиотеки `randlib`. Кроме того, он содержит многочисленные вспомогательные функции.

> Смотри также

Описание функций srand и rand в *perlfunc*(1); рецепты 2.7—2.8; документация по модулям Math::Random и Math::TrulyRandom с CPAN.

2.10. Генерация случайных чисел с неравномерным распределением

Проблема

Требуется генерировать случайные числа в ситуации, когда одни значения появляются с большей вероятностью, чем другие (неравномерное распределение). Допустим, вы отображаете на своей Web-странице случайный баннер и у вас имеется набор весовых коэффициентов, определяющих частоту появления того или

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

иного баннера. А может быть, вы имитируете нормальное распределение (закон распределения Гаусса).

Решение

Если вам потребовались случайные величины, распределенные по конкретному закону (допустим, по закону Гаусса), загляните в учебник по статистике и найдите в нем нужную функцию или алгоритм. Следующая функция генерируетслучайные числа с нормальнымраспределением, со стандартным отклонением 1 и нулевым математическим ожиданием.

```
sub gaussian_rand {
    my ($u1, $u2);  # Случайные числа с однородным распределением
    my $w;          # Отклонение, затем весовой коэффициент
    my ($g1, $g2);  в Числа с гауссовским распределением

    do {
        $u1 = 2 " rand() - 1;
        $u2 = 2 * rand() - 1;
        $w = $u1*$u1 + $u2*u2
    }'while ($w >= 1);

    $w = sqrt((-2 *log($w)) /$w);
    $g2 = $u1 * $w;
    $g1 = $u2 * $w;
    " Возвратить оба числа или только одно
          wantarray ? ($g1, $g2) : $g1; .
}
```

Если у вас есть список весовых коэффициентов и значений и вы хотите выбирать элементы списка случайным образом, выполните два последовательных шага. Сначала превратите весовые коэффициенты в вероятностное распределение с помощью приведенной ниже функции weight_to_dist, а затем воспользуйтесь функцией weighted_rand длясслучайного выбора чисел.

```
tt weight_to_dist: получает хэш весовых коэффициентов
# и возвращает хэш вероятностей
sub weight_to_dist {
    my %weights = @_;
    my %dist    = (); •
    my $total   = 0;
    my ($key, $weight);
    local $_;

            (values %weights) {
        $total += $_;
    }

    while ( ($key, $weight) = each %weights ) {
        $dist{$key}  =  $weight/$total;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    >                                                  '' '    '' ¦
            %dist;
}

# weighted_ran: получает хэш вероятностей
# и возвращает случайный элемент хэша
sub weighted_rand {
    my %dist = @_;
    my ($key, $weight);

    while (1) {            # Чтобы избежать погрешностей вычислений
                           # с плавающей запятой (см. ниже).
        my $rand = rand;
        while ( ($key, $weight) = each %dist ) {
                $key if ($rand -= $weight) <0;
        }
    }
}
```

Комментарий

Функция gaussian_rand реализует ***полярный метод Бокса—Мюллера*** для преобразования двух независимых случайных чисел с однородным распределением, лежащих **в** интервале от **0** до **1 в** два числа с математическим ожиданием **0** и стандартным отклонением **1** (то есть распределенных по закону Гаусса). Чтобы сгенерировать числа с другим математическим ожиданием и стандартным отклонением, умножьте выходные данные gaussian_rand на нужное стандартное отклонение и прибавьте математическое ожидание:

```
# gaussian_rand - см. выше
$mean = 25;
$sdev = 2;
$salary - gaussian_rand() * $sdev + $mean;
printf("You have been          \$%.2f\n",   salary);
```

Функция weighted_rand получает случайное число из интервала от Одо 1.' Затем она использует вероятности, сгенерированные weight_to_dist, и определяет, какому элементу соответствует это случайное число. Из-за погрешностей представЛения с плавающей запятой накопленные ошибки могут привести к тому, что возвращаемый элемент не будет найден. Поэтому кодразмещается в цикле while, который в случае неудачи выбирает новое случайное число и делает очередную попытку.

Кроме того, модуль Math::Random с CPAN содержит функции, генерирующие случайные числа для многих распределений.

> Смотри также

Описание функции rand в *perlfunc*(1);рецепт 2.7; документация по модулю Math::Random с CPAN.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

2.11. Выполнение тригонометрических вычислений в градусах

Проблема

Требуется, чтобы в тригонометрических функциях использовались градусы вместо стандартных для Perl радианов.

Решение

Создайте функции для преобразований между градусами и радианами (2л радиан соответствуют 360 градусам).

```
BEGIN {
    use constant PI => 3.14159265358979;

    sub deg2rad {
                    shift;

    }

    sub rad2deg {
        my $radians = shift;
            ($radians          180;
    }
}
```

Также можно воспользоваться модулем Math::Trig:

```
use Math::Trig;

$radians =
          = rad2deg($radians);
```

Комментарий

Если вам приходится выполнять большое количество тригонометрических вычислений, подумайте об использовании стандартных модулей Math::Trig или POSIX. В них присутствуют дополнительные тригонометрические функции, которых нет в стандартном Perl. Другой выход заключается в определении приведенных выше функций rad2deg и deg2rad. В Perl нет встроенной константы л, однако при необходимости ее можно вычислить настолько точно, насколько позволит ваше оборудование для вычислений с плавающей запятой. В приведенном выше решении я является константой, определяемой командой use constant.

Синус угла, заданного в градусах, вычисляется следующим образом:

```
Я Функции deg2rad и rad2def приведены выше или взяты из Math::Trig

                    shift;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    my $radians =

}
```

Смотри также

Описание функций sin, cos и atan2 в *perlfunc(1);* стандартная документация по модулям POSIX и Math::Trig.

2.12, Тригонометрические функции

Проблема

Требуется вычислить значения различных тригонометрических функций — таких как синус, тангенс или арккосинус.

Решение

В Perl существуют лишь стандартные тригонометрические функции sin, cos и atan2. С их помощью можно вычислить тангенс (tan) и другие тригонометрические функции:

```
sub tan {
    my $theta = shift;

            sin($theta)/cos($theta);
}
```

В модуле POSIX представлен расширенный набор тригонометрических функций:

```
use POSIX;

$y = acos(3.7);
```

Модуль Math::Trig содержит полный набор тригонометрических функций, а также позволяет выполнять операции с комплексными аргументами (или дающие комплексный результат):

```
use Math::Trig;

$y = acos(3.7);
```

Комментарий

Если значение $theta равно п/2, 3п/2 и т. д., в функции tan возникает исключительная ситуация деления на ноль, поскольку для этих углов косинус равен нулю. Аналогичные ошибки возникают и во многих функциях модуля Math::Trig. Чтобы перехватить их, воспользуйтесь конструкцией eval:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
eval {
    $y = tan($pi/2);
            undef;
```

Смотри также

Описание функций sin, cos и atan2 в perlfunc(1). Тригонометрия в контексте комплексных чисел рассматривается в рецепте 2.15, а использование eval для перехвата исключений — **в** рецепте **10.12.**

2,13. Вычисление логарифмов

Проблема

Требуется вычислить логарифм по различным основаниям.

Решение

Для натуральных логарифмов (по основанию ***e)*** существует встроенная функция log:

```
$log_e = log(VALUE);
```

Чтобы вычислить логарифм по основанию **10,** воспользуйтесь функцией log10 модуля **POSIX:**

```
use POSIX qw(log10);
$log_10 = log10(VALUE);
```

Для других оснований следует использовать соотношение:

$$\log_n(x) = \log_2(x)/\log_2(n)$$

где x — число, логарифм которого вычисляется, n — нужное основание, а e — основание натуральных логарифмов.

```
sub log_base {
    my ($base, $value) = @_;
            log($value)/log($base);
}
```

Комментарий

Функция log_base позволяет вычислять логарифмы по любому основанию. Если основание заранее известно, намного эффективнее вычислить его натуральный логарифм заранее и сохранить для последующего использования, вместо того чтобы каждый раз пересчитывать его заново.

```
# Определение log_base см. выше
$answer = log_base(10, 10000);
print "log10(10,100) = $answer\n";
log10(10,000) * 4
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В модуле Math::Complex для вычисления логарифмов по произвольному основанию существует функция logn(), поэтому вы можете написать:

```
use Math::Complex;
printf "log2(1024) = %lf\n", logn(1024, 2); # Обратите внимание
                                            # на порядок аргументов!
log2(1024) = 10.000000
```

хотя комплексные числа в вычислениях не используются. Функция не очень эффективна, однако в будущем планируется переписать Math::Complex на C для повышения быстроты.

Смотри также

Описание функции **log** в *perlfunc*(1); документация по стандартному модулю **POSIX.**

2.14. Умножение матриц

Проблема

Требуется перемножить два двумерных массива. Умножение матриц часто используется в математических и инженерных вычислениях.

Решение

Воспользуйтесь модулями **PDL** с **CPAN.** Модули **PDL** (Perl Data Language, то есть "язык данных Рёг1") содержат быстрые и компактные матричные и математические функции:

```
use PDL;
ff $a и $b - обьекты pdl
$c = $a * $b;
```

Альтернативный вариант — самостоятельно реализовать алгоритм умножения матриц для двумерных массивов:

```
sub mmult {
    my ($m1,$m2) = @_;
    my ($m1rows,$m1cols) = matdim($m1);
    my ($m2rows,$m2cols) = matdim($m2);

    unless ($m1cols == $m2rows) { # Инициировать исключение
        die "IndexError: matrices don't match: $m1cols != $m2rows";
    }


    my ($i, $j, $k);

    for $i (range($m1rows)) {
        for $j (range($m2cols)) {
            for $k (range($m1cols)) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                $result->[$i][$j] += $m1->[$i][$k] * $m2->[$k][$j];
            }
        }
    }

}

sub range {0 .. ($_[0] - 1 }

sub veclen {
                = $_[0];
    my type = ref
    if ($type ne "ARRAY") {die "$type is bad array ref for $ary_ref" }
        scalar(@$ary_ref);
}

sub matdim {
    my $matrix = $_[0];
    my $rows = veclen($matrix);
    my $cols = veclen($matrix->[0]);
                , $cols);
}
```

Комментарий

Если у вас установлена библиотека PDL, вы можете воспользоваться ее молниеносными числовыми операциями. Они требуют значительно меньше памяти и ресурсов процессора, чем стандартные операции с массивами Perl. При использовании объектов PDL многие числовые операторы (например, + и *) перегружаются и работают с конкретными типами операндов (например, оператор * выполняет так называемое скалярное умножение). Для умножения матриц используется перегруженный оператор x.

```
use PDL;

$a = pdl [
    [ 3, 2, 3 ],
    [ 5, 9, 8 ],
]:

$b = pdl [
    [ 4, 7 ],
    [ 9, 3 ],
    [ 8, 1 ],
];

$c = $a x $b; # Перегруженный оператор x
```

Если библиотека PDL недоступна или вы не хотите привлекать ее для столь тривиальной задачи, матрицы всегда можно перемножить вручную:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# mmult() и другие процедуры определены выше

$x = [
      [ 3, 2, 3 ],
      [ 5, 9, 8 ],
]:

$y = [
      [ 4, 7 ],
      [ 9, 3 ],
   •  [ 8, 1 ],
];

$z = mult($x, $y);
```

Смотри также

Документация по модулю PDL с CPAN.

2.15. Операции с комплексными числами

Проблема

Ваша программа должна работать с комплексными числами, часто используемыми в инженерных, научных и математических расчетах.

Решение

Либо самостоятельно организуйте хранение вещественной и мнимой составляющих комплексного числа, либо воспользуйтесь классом Math::Complex (из стандартной поставки Perl).

Ручное умножение комплексных чисел

```
# $c = $a * $b - моделирование операции
                                  ($a_imaginary   $b_imaginary
$c_imaginary              $b_imaginary              $a_imaginary
```

Math::Complex

```
# Умножение комплексных чисел с помощью Math::Complex
use Math::Complex;
$c = $a * $b;
```

Комментарий

Ручное умножение комплексных числа 3+5i и 2-2i выполняется следующим образом:

```
            $a_imaginary                          5i;
            $b_imaginary = -2;              # 2 - 2i;
$c_real                           $a_imaginary   $b_imaginary
$c_imaginary              $b_imaginary                $a_imaginary
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
print "c = ${c_real}+${c_imaginary}i\n";
```

```
'c = 16+4i
```

То же с применением модуля Math::Complex:

```
use Math::Complex;
$a = Math::Complex->new(3,5);
$b = Math::Complex->new(2,-2);
$c = $a * $b;
print "c = $c\n";
```

```
o - 16+4i
```

Версия 5.004 позволяет создавать комплексные числа с помощью конструктора или экспортированной константы i:

```
use Math::Complex;
$c = cplx(3,5) * cplx(2,-2);          # Лучше воспринимается
$d = 3 + 4*i;                          Я 3 + 4i
printf "sqrt($d) = %s\n", sqrt($d);
```

```
sqrt(3+4i) = 2+i
```

В исходном варианте модуля Math::Complex, распространяемом с версией 5.003, не перегружаются многие функции и операторы версии 5.004. Кроме того, Math::Complex используется модулем Math::Trig (появившимся в версии 5.004), поскольку некоторые функции могут выходить за пределы вещественной оси в комплексную плоскость — например, арксинус 2.

Смотри также

Документация по стандартному модулю Math::Complex.

2.16. Преобразования восьмеричных и шестнадцатеричных чисел

Проблема

Требуется преобразовать строку с восьмеричным или шестнадцатеричным представлением (например, "0x55" или "0755") в правильное число.

Perl воспринимает лишь те восьмеричные и шестнадцатеричные числа, которые встречаются в программе в виде литералов. Если числа были получены при чтении из файла или переданы в качестве аргументов командной строки, автоматическое преобразование не выполняется.

Решение

Воспользуйтесь функциями Perl oct и hex:

```
$number = hex($hexadecimal);          " Шестнадцатеричное число
$number = oct($octal);                # Восьмеричное число
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Функция oct преобразует восьмеричные числа как с начальными нулями, так и без них ("0350" и "350"). Более того, она даже преобразует шестнадцатеричные числа, если у них имеется префикс "Ox". Функция hex преобразует только шестнадцатеричные числа с префиксом "Ox" или безнего — например, "0x255", "3A", "ff" или "deadbeef" (допускаются символы верхнего и нижнего регистров).

Следующий пример получает число в десятичной, восьмеричной или шестнадцатеричной системе счисления и выводит его во всехтрех системах счисления. Для преобразования из восьмеричной системы используется функция oct. Если введенное число начинается с 0, применяется функция hex. Затем функция printf при выводе преобразует число в десятичную, восьмеричную и шестнадцатеричную систему:

```
print "Gimme a number in decimal, octal, or hex: ";
$num = <STDIN>;
chomp $num;
exit unless defined $num;
$num += oct($num) is $num =^ /^0/;   # Обрабатывает как восьмеричные,
                                     # так и шестнадцатеричные числа
printf "%d %x %o\n", $num, $num, $num;
```

Следующий фрагмент преобразует режимы доступа к файлам UNIX. Режим всегда задается в восьмеричном виде, поэтому вместо hex используется функция oct:

```
print "Enter file permission in octal: ";
$permissions = <STDIN>;
die "Exiting ...\n" unless defined $permissions;
chomp $permissions;
$permissions = oct($permissions);   # Режим доступа всегда задается
                                    # в восьмеричной системе
print "The decimal value is $permissions\n";
```

Смотри также

Раздел "Scalar Value Constructors" в *perldata*(1); описание функций oct и hex в *perlfunc*(1).

2.17. Вывод запятых в числах

Проблема

При выводе числа требуется вывести запятые после соответствующих разрядов. Длинные числа так воспринимаются намного лучше, особенно в отчетах.

Решение

Обратите строку, чтобы перебирать символы в обратном порядке, — это позволит избежать подстановок в дробной части числа. Затем, воспользуйтесь регулярным

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

выражением, найдите позиции для запятых и вставьте их с помощью подстановки. Наконец, верните строку к исходному порядку символов.

```
sub commify {
                    $_[0];
    $text =~ s/(\d\d\d)(?=\d)(?!\d*\.)/$1,/g;
          scalar        $text;
}
```

Комментарий

Регулярные выражения намного удобнее использовать в прямом, а не в обратном направлении. Учитывая этот факт, мы меняем порядок символов в строке на противоположный и вносим небольшие изменения в алгоритм, который многократно вставляет запятые через каждые три символа от конца. Когда все вставки будут выполнены, порядок символов снова меняется, а строка возвращается из функции. Поскольку функция reverse учитывает косвенный контекст возврата, мы принудительно переводим ее в скалярный контекст.

Функцию нетрудно модифицировать так, чтобы вместо запятых разряды разделялись точками, как принято в некоторых странах.

Пример использования функции commify выглядит так:

```
# Достоверный счетчик обращений :-)
use Math::TrulyRandom;
$hits = truly_random_value();     # Отрицательное значение!
$output = "Your web page          $hits accesses last month.\n";
print commify($output);
Your web page          -1,740,525,205 accesses last month.
```

Смотри также

perllocale(1); описание функции reverse в *perlfunc*(1).

2.1S. Правильный вывод во множественном числе

Проблема

Требуется вывести фразу типа: "It took $time hours" («Это заняло $time часов»). Однако фраза "It took 1 hours" ("Это заняло 1 часов") не соответствует правилам грамматики. Необходимо исправить ситуацию[1].

Решение

Воспользуйтесь printf и тернарным оператором X?Y:Z, чтобы изменить глагол или существительное.

[1] К сожалению, для русского языка этот рецепт не подойдет, поскольку множественное число в нем образуется по более сложным правилам с большим количеством исключений. — *Примеч. перев.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
printf "It took %d hour%s\n", $time, $time == 1 ? "" : "s";

printf "%d hour%s %s enough.\n", $time,
        $time == 1 ? ""   :  "s";
        $time == 1 ? "is" :
```

Кроме того, можно воспользоваться модулем Lingua::EN::Inflectc CPAN, упоминаемым в комментарии.

Комментарий

Невразумительные сообщения вроде "1 file(s) updated" встречаются только из-за того, что автору программы лень проверить, равен ли счетчик 1.

Если образование множественного числа не сводится к простому добавлению суффикса s, измените функцию printf соответствующим образом:

```
printf "It took %d centur%s", $time, $time == 1 ? "y" : "ies";
```

В простых ситуациях такой вариант подходит, однако вам быстро надоест писать его снова и снова. Возникает желание написать для него специальную функцию:

```
sub noun_plural {
    local $_ = shift;
    ff Порядок проверок крайне важен!
    s/ss$/sses/                          ||
    s/([psc]h)$/${1}es/                  ||
    s/z$/zes/                          | |
    s/ff$/ffs/            '          | |
    s/f$/ves/                          | |
    s/ey$/eys/                           ||
    s/y$/ies/                          | |
    s/ix$/ices/         .                ||
    s/([sx])$/$1es/                      ||
    s/$/s/                    ,          ||
              die "can't get here";

}
*verb_singular = \&noun_plural; tf Синоним функции
```

Однако со временем будут находиться новые исключения и функция будет становиться все сложнее и сложнее. Если у вас возникнет потребность в подобных морфологических изменениях, воспользуйтесь универсальным решением, которое предлагает модуль Lingua:EN::Inflect от CPAN.

```
use  Lingua::EN::Inflect qw(PL classical);
 classical(1);              # Почему не сделать по умолчанию?
while  (<DATA>)  {          # Каждая строка данных
    for  (split)  {        # Каждое слово в строке
        print "One $ , two ", PL($_), ".\n";
    }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
Я И еще один вариант
$_ = 'secretary general';
print "One $_, two ", PL($_), ".\n";

__END__
fish fly ox
species genus jockey
index matrix mythos
phenomenon formula
```

Результат выглядит так:

```
One fish, two fish.
One fly, two flies.
One ox, two oxen.
One species, two species.
One genus, two genera.
One phylum, two phyla.
One cherub, two cherubim.
One radius, two radii.
One jockey, two jockeys.
One index, two indices,
One matrix,- two matrices.
One mythos, two mythoi.
One phenomenon, two phenomena.
One formula, two formulae.
One secretary general, two                    general.
```

Мы рассмотрели лишь одну из многих возможностей модуля. Кроме того, он обрабатывает склонения и спряжения для других частей речи, содержит функции сравнения без учета регистра, выбирает между использованием а и an и делает многое другое.

Смотри также

Описание тернарного оператора выбора в *perlop*(1); документация по модулю Lingua::EN::Inflect с CPAN.

2.19. Программа: разложение на простые множители

Следующая программа получает один или несколько целых аргументов и раскладывает их на простые множители. В ней используется традиционное числовое представление Perl, кроме тех ситуаций, когда представление сплавающей запятой может привести к потере точности. В противном случае (или при запуске с параметром *-b*) используется стандартная библиотека Math::Blight, что позволяет работать с большими числами. Однако библиотека загружается лишь при необходимости, поэтому вместо use используются ключевые слова

это позволяет выполнить динамическую загрузку библиотеки во время выполнения вместо статической загрузки на стадии компиляции.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Наша программа недостаточно эффективна для подбора больших простых чисел, используемых в криптографии.

Запустите программу со списком чисел, и она выведет простые множители для каждого числа:

```
$ factors 8 9 96 2178
8          2**3
9          3**2
96         2**5 3
2178       2 3**2 11**2
```

Программа нормально работает и с очень большими числами:

```
% factors 2393220000000000000000000
+2393220000000000000000000  2**19  3  5**18  +39887
% factors 2393220000000000000000000000
+2500000000000000000000000000  2**24  5**26
```

Исходный текст программы приведен в примере 2.1.

Пример 2.1. bigfact

```
#!/usr/bin/perl
ff bigfact - разложение на простые множители
use strict;
use integer;

use vars qw{ $opt_b $opt_d };
use Getopt::Std;

@ARGV && getopts('bd')        or die "usage; $0 [-b] number ...";

load_biglib() if $opt_b;

ARG:            $orig
    my ($n, $root, %factors, $factor);
    $n = $opt_b ? Math::BigInt->new($orig) : $orig;
    if ($n + 0 ne $n) { # Не используйте здесь -w
        printf STDERR "bignum: %s would become %s\n", $n, $n+0 if $opt_d;
        load_biglib();
        $n = Math::BigInt->new($orig);
    }
    printf "%-10s ", $n;

    # $sqi равно квадрату $i. Используется тот факт,
    ff что ($i + 1) ** 2 == $i ** 2 + 2 * $i + 1.
    for (my ($i, $sqi) = (2, 4); $sqi <= $n; $sqi += 2 * $i ++ + 1) {
        while ($n % $i == 0) {
            $n /= $i;
            print STDERR "<$i>" if $opt_d;
            $factors {$i} ++;
        }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    }

    if ($n != 1 && $n != $orig) { $factors{$n}++ }
    if (! %factors) {
        print "PRIME\n";
        next ARG;
    }
    for $factor ( sort { $a <=> $b } keys %factors ) {
            print "$factor";
        if ($factors{$factor} > 1) {
        print "**$factors{$factor}";
        }
        print " ";
    }
    print "\n",
}

# Имитирует use, но во время выполнения
sub load_biglib {
            Math::BigInt;
    Math::BigInt->import();
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

3 Дата и время

Не следует требовать, чтобы время в секундах с начала эпохи точно соответствовало количеству секунд между указанным временем и началом эпохи.

Стандарт IEEE 1003.1b-1993 (POSIX) раздел B.2.2.2

Введение

Время и дата — очень важные величины, и с ними необходимо уметь работать. "Сколько пользователей регистрировалось за последний месяц?», "Сколько секунд я должен проспать, чтобы проснуться к полудню?" и "Не истек лисрок действия пароля данного пользователя?" — вопросы кажутся тривиальными, однако ответ на них потребует на удивление нетривиальных операций.

В Perl моменты времени представлены в виде интервалов, измеряемых в секундах с некоторого момента, называемого *началом эпохи*. В UNIX и многих других системах начало эпохи соответствует 00 часов 00 минут 1 января 1970 года по Гринвичу (GMT[1]). На Macintosh дата и время измеряется в местном часовом поясе. Функция `gmtime` возвращает правильное время по Гринвичу, основанное на смещении местного часового пояса. Помните об этом, рассматривая рецепты этой главы. На Macintosh количество секунд с начала эпохи позволяет отсчитывать время в интервале от 00:00 1 января 1904 года до 06:28:15 6 февраля 2040 года.

Говоря о времени и датах, мы часто путаем две разные концепции: момент времени (дата, время) и интервал между двумя моментами (недели, дни, месяцы и т. д.). При отсчете секунд с начала эпохи интервалы и моменты представляются в одинаковых единицах, поэтому с ними можно выполнять простейшие математические операции.

Однако люди не привыкли измерять время в секундах с начала эпохи. Мы предпочитаем работать с конкретным годом, месяцем, днем, часом, минутой и секундой. Более того, название месяца может быть как полным, так и сокращенным. Число может указываться как перед месяцем, так и после него. Использование разных форматов затрудняет вычисления, поэтому введенная пользователем или

[1] В наши дни время по Гринвичу также часто обозначается сокращением UTC (Universal Coordinated Time).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

прочитанная из списка строка даты/времени обычно преобразуется в количество секунд с начала эпохи, с ней производятся необходимые операции, после чего секунды снова преобразуются для вывода.

Для удобства вычислений количество секунд с начала эпохи всегда измеряется по Гринвичу. В любых преобразованиях всегда необходимо учитывать, представлено ли время по Гринвичу или в местном часовом поясе. Различные функции преобразования позволяют перейти от гринвичского времени в местное, и наоборот.

Функция Perl `time` возвращает количество секунд, прошедших с начала эпохи... более или менее[1] точно. Для преобразования секунд с начала эпохи в конкретные дни, месяцы, годы, часы, минуты и секунды используются функции localtime и `gmtime`. В списковом контексте эти функции возвращают список, состоящий из девяти элементов.

Переменная	Значение	Интервал
$sec	Секунды	0-60
`$min`	Минуты	0-59
$hours	Часы	0-23
`$mday`	День месяца	1-31
`$month`	Месяц	0–11, 0 == январь
$year	Год, начиная с 1900	1-138 (и более)
$wday	День недели	0–6, 0 -- воскресенье
$yday	День года	1-366
`$isdst`	0 или 1	true, если действует летнее время

Секунды изменяются в интервале 0-60 с учетом возможных корректировок; под влиянием стандартов в любой момент может возникнуть лишняя секунда.

В дальнейшем совокупность «день/месяц/год/час/минута/секунда» будет обозначаться выражением "полное время" — хотя бы потому, что писать каждый раз "отдельные значения дня, месяца, года, часа, минут и секунд" довольно утомительно. Сокращение не связано с конкретным порядком возвращаемых значений.

Perl *не возвращает* данные о годе в виде числа из двух цифр. Он возвращает разность между текущим годом и 1900, которая до 1999 года представляет собой число из двух цифр. У Perl нет своей "проблемы 2000 года", если только вы не изобретете ее сами (впрочем, у вашего компьютера и Perl может возникнуть проблема 2038 года, если к тому времени еще будет использоваться 32-разрядная адресация). Для получения полного значения года прибавьте к его представлению 1900. Не пользуйтесь конструкцией `"19$year"`, или вскоре ваши программы начнут выдавать "год 19102". Мы не можем точно зафиксировать интервал года, потому что все зависит от размера целого числа, используемого вашей системой для представления секунд с начала эпохи. Малые числа дают небольшой интервал; большие (64-разрядные) числа означают огромные интервалы.

Скорее, менее. В момент написания книги функция возвращала на21 секунду меньше. В соответствии со стандартом POSIX функция time не должна возвращать секунды, которые накапливаются из-за замедления вращения Земли, обусловленного воздействием приливов. За дополнительной информацией обращайтесь к разделу 3 sci.astro FAQ по адресу http://astrosun.tn.cornell.edu/students/lazio.sci.astro.3.FAQ.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В скалярном контексте localtime и gmtime возвращают дату и время, отформатированные в виде ASCII-строки:

```
Fri Apr 11 09:27:08 1997
```

Объекты стандартного модуля Time::tm позволяют обращаться к компонентам даты/времени по именам. Стандартные модули Time::localtime и Time::gmtime переопределяют функции localtime и gmtime, возвращающие списки, и заменяют их версиями, возвращающими объекты Time::tm. Сравните два следующих фрагмента:

```
# Массив
print "Today is day ", (localtime()[7], " of the year.\n";
Today is day 117 of the year.

# Объекты Time::tm
use Time::localtime;
$tm = localtime;
print "Today is day ", $tm->yday, " of the year.\n";
Today is day 117 of the year.\
```

Чтобы преобразовать список в количество секунд с начала эпохи, воспользуйтесь стандартным модулем Time::Local. В нем имеются функции timelocal и timegm, которые получают список из девяти элементов и возвращают целое число. Элементы списка и интервалы допустимых значений совпадают с теми, которые возвращаются функциями localtime и gmtime.

Количество секунд с начала эпохи ограничивается размером целого числа. Беззнаковое 32-разрядное целое позволяет представить время по Гринвичу от 20:45:52 13 декабря 1901 года до 03:14:07 19 января 2038 года включительно. Предполагается, что к 2038 году в компьютерах должны использоваться целые числа большей разрядности. Во всяком случае, будем надеяться на это. Чтобы работать с временем за пределами этого интервала, вам придется воспользоваться другим представлением или выполнять операции со отдельными значениями года, месяца и числа.

Модули Date::Calc и Date::Manip с CPAN работают с этими отдельными значениями, но учтите — они не всегда вычитают из года 1900, как это делает localtime, а нумерация месяцев и недель в них не всегда начинается с 0. Как всегда, в страницах руководства можно найти достоверные сведения о том, какая информация передается модулю, а какая — возвращается им. Только представьте, как будет неприятно, если рассчитанные вами финансовые показатели уйдут на 1900 лет в прошлое!

3.1. Определение текущей даты

Проблема

Требуется определить год, месяц и число для текущей даты.

Решение

Воспользуйтесь функцией localtime. Без аргументов она возвращает текущую дату и время. Вы можете вызвать localtime и извлечь необходимую информацию из полученного списка:

```
($DAY, $MONTH, $YEAR) = (localtime)[3,4,5];
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Модуль Time::localtime переопределяет localtime так, чтобы функция возвращала объект Time::tm:

```
use Time::localtime;
$tm = localtime;
($DAY, $MONTH, $YEAR) = ($tm->mday, $tm->mon, $tm->year);
```

Комментарий

Вывод текущей даты в формате ГГГГ-ММ-ДД с использованием стандартной функции localtime выполняется следующим образом:

```
($day, $month, $year) = (localtime)[3,4,5];
printf("The        date is %04d %02d %02d\n", $year+1900, $month+1, $day);
            date
```

Нужные поля из списка, возвращаемого localtime, извлекаются *с* помощью среза. Запись могла выглядеть иначе:

```
($day, $month, $year) = (localtime)[3..5];
```

А вот как текущая дата выводится в формате ГГГГ-ММ-ДД (рекомендованном стандартом ISO 8601) с использованием Time::localtime:

```
use Time::localtime;
$tm = localtime;
printf("The              is %04d-%02d-%02d\n", $tm->year+1900,
    ($tm->mon)+1, $tm->mday);
                    1999-04-28
```

В короткой программе объектный интерфейс выглядит неуместно. Однако при большом объеме вычислений с отдельными компонентами даты обращения по имени заметно упрощают чтение программы.

То же самое можно сделать и хитроумным способом, не требующим создания временных переменных:

```
printf("The                  %04d-%02d-%02\n",
        sub {($_[5]+1900, $_[4]+1. $_[3])}->(localtime));
```

Кроме того, в модуле POSIX имеется функция strftime, упоминаемая в рецепте 3.8:

```
use POSIX qw(strftime);
print strftime "%Y-%m-%d\n", localtime;
```

Функция gmtime работает аналогично localtime, но возвращает время по Гринвичу, а не для местного часового пояса.

Смотри также

Описание функций localtime и gmtime в *perlfunc*(1); документация по стандартному модулю Time::localtime.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

3.2. Преобразование полного времени в секунды с начала эпохи

Проблема

Требуется преобразовать дату/время, выраженные отдельными значениями дня, месяца, года и т. д. в количество секунд с начала эпохи.

Решение

Воспользуйтесь функцией `timelocal` или `timegm` стандартного модуля Time::Local. Выбор зависит от того, относится ли дата/время к текущему часовому поясу или Гринвичскому меридиану:

```
use Time::Local;
$TIME = timelocal($sec, $min, $hours, $mday, $mon, $year);
$TIME = timegm($sec, $min, $hours, $mday, $mon, $year);
```

Комментарий

Встроенная функция `localtime` преобразует количество секунд с начала эпохи в компоненты полного времени; процедура `timelocal` из стандартного модуля Time::Local преобразует компоненты полного времени в секунды. Следующий пример показывает, как определяется количество секунд с начала эпохи для текущей даты. Значения дня, месяца и года получаются от `localtime`:

```
# $hours, $minutes и $seconds задают время для текущей даты
# и текущего часового пояса
use Time::Local;
$time = timelocal($seconds, $minutes, $hours, (localtime)[3,4,5]);
```

Если функции `timelocal` передаются месяц и год, они должны принадлежать тем же интервалам, что и значения, возвращаемые `localtime`. А именно, нумерация месяцев начинается с 0, а из года вычитается 1900.

Функция `timelocal` предполагает, что компоненты полного времени соответствуют текущему часовому поясу. Модуль Time::Local также экспортирует процедуру `timegm`, для которой компоненты полного времени задаются для Гринвичского меридиана. К сожалению, удобных средств для работы с другими часовыми поясами, кроме текущего или Гринвичского, не существует. Лучшее, что можно сделать, — преобразовать время к Гринвичскому и вычесть или прибавить смещение часового пояса в секундах.

В следующем фрагменте демонстрируется как применение `timegm`, так и настройка интервалов года и месяца:

```
# $day - день месяца (1-31)
# $month - месяц (1-12)
9 $year - год, состоящий из четырех цифр (например, 1999)
# $hours, $minutes и $seconds - компоненты времени по Гринвичу
use Time::Local;
$time = timegm($seconds, $minutes, $hours, $day, $month-1, $year-1900);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Как было показано во введении, количество секунд с начала эпохи не может выходить за пределы интервала От 20:45:52 13 декабря 1901 года до 03:14:07 19 ' января 2038 года включительно. Не преобразуйте такие даты — либо воспользуйтесь модулем Date:: с CPAN, либо выполняйте вычисления вручную.

Смотри также

Документация по стандартному модулю Time::Local. Обратное преобразование рассматривается в рецепте 3.3.

3.3. Преобразованиесекунд с начала эпохи в полное время

Проблема

Требуется преобразовать количество секунд с начала эпохи в отдельные значения дня, месяца, года и т. д.

Решение

Воспользуйтесь функцией localtime или gmtime в зависимости от того, хотите ли вы получить дату/время для текущего часового пояса или для Гринвичского меридиана.

```
($seconds, $minutes, $hours, $day_of_month, $year,
    $wday, $yday, $isdst) = localtime($TIME);
```

Стандартные модули Time::timelocal и Time::gmtime переопределяют функции localtime и gmtime так, чтобы к компонентам можно было обращаться по именам:

```
use Time::localtime;        # или Time::gmtime
$tm = localtime($TIME);     # или gmtime($TIME)
$seconds = $tm->sec;
# ...
```

Комментарий

Функции localtime и gettime возвращают несколько странную информацию о годе и месяце; из года вычитается 1900, а нумерация месяцев начинается с 0 (январь). Не забудьте исправить полученные величины, как это делается в следующем примере:

```
($seconds, $minutes, $hours, $day_of_month, $month, $year,
    $wday, $yday, $isdst) = localtime($time);
printf("Dateline: %02d:%02d:%02d-%04d/%02d/%02d\n",
    $hours, $minutes, $seconds, $year+1900, $month+1,
    $day_of_month);
```

Модуль Time::localtime позволяет избавиться от временных переменных:

```
use Time::localtime;
$tm = localtime($time);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
printf("Dateline: %02d:%02d:%02d-%04d/%02d/%02d\n"
    $tm->hour, $tm->min, $tm->sec, $tm->year+1900,
    $tm->mon+1, $tm->mday);
```

Смотри также

Описание функции `localtime` в *perlfunc*(1); документация по стандартным модулям Time::localtime и Time::gmtime. Обратное преобразование рассматривается в рецепте 3.2.

3.4. Операции сложения и вычитания для дат

Проблема

Имеется значение даты/времени. Требуется определить дату/время, отделенную от них некоторым промежутком в прошлом или будущем.

Решение

Проблема решается простым сложением или вычитанием секунд с начала эпохи:

```
$when = $now + $difference;
$then = $now - $difference;
```

Если у вас имеются отдельные компоненты полного времени, воспользуйтесь модулем Date::Calc с CPAN. Если вычисления выполняются только с целыми днями, примените функцию `Add_Delta_Days` (смещение $offset может представлять собой как положительное, так и отрицательное целое количество дней):

```
use Date: Calc qw(Add_Delta_Days),
($y2, $m2, $d2) = Add_Delta_Days($y, $m, $d, $offset),
```

Если в вычислениях используются часы, минуты и секунды (то есть не только дата, но и время), воспользуйтесь функцией `Add_Delta_DHMS`:

```
use Date :DateCalc qw(Add_Delta_DHMS),
($year2, $month2, $day2, $h2, $m2, $s2) =
    Add_Delta_DHMS( $year, $month, $day, $hour, $minute, $seconds,
        $days_offset, $hour_offset, $minute_offset, $seconds_offset );
```

Комментарий

Вычисления с секундами от начала эпохи выполняются проще всего (если не считать усилий на преобразования даты/времени в секунды и обратно). В следующем фрагменте показано, как прибавить смещение (в данном примере — 55 дней, 2 часа, 17 минут и 5 секунд) к заданной базовой дате и времени:

```
$birthtime = 96176750,              # 18 января 1973 года, 03:45:50
$interval  = 5 +                     # 5 секунд
             17 * 60 +               # 17 минут
             2 * 60 * 60 +           Я 2 часа
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                55 * 60 * 60 " 24; •    # и 55 дней
$then = $birthtime + $interval;
print "Then is ", scalar(localtime($then)), "\n";
Then is Wed Mar 14 06:02:55 1973
```

Мы также могли воспользоваться функцией Add_Delta_DHMS и обойтись без преобразований к секундам с начала эпохи и обратно:

```
use Date::Calc qw(Add_Delta_DHMS);
($year, $month, $day, $hh, $mm, $ss) = Add_Delta_OHMS(
     1973, 1, 18, 3, 45, 50, # 18 января 1973 года, O3:45:5O
                55, 2, 17, 5); # 55 дней, 2 часа, 17 минут, 5 секунд
print "To be prcise: $hh:$mm:$ss, $month/$day/$year\n";
To be precise: 6:2:55, 3/14/1973
```

Как обычно, необходимо проследить, чтобы аргументы функции находились в правильных интервалах. Add_Delta_DHMS получает полное значение года (без вычитания 1900). Нумерация месяцев начинается с 1, а не с 0. Аналогичные параметры передаются и функции Add_Delta_Days модуля DateDateCalc:

```
use Date::DateCalc qw(Add_Delta_Days);
($year, $month, $day) = Add_Oelta_Days( 1973, 1, 18, 55);
print "Nat was 55 days old on: $month/$day/$year\n";
Nat was 55 days old on: 3/14/1973
```

Смотри также

Документация по модулю Date:;Calc от CPAN.

3.5. Вычисление разности между датами

Проблема

Требуется определить количество дней между двумя датами или моментами времени.

Решение

Если даты представлены в виде секунд с начала эпохи и принадлежат интервалу от 20:45:52 13 декабря 1901 года до 03:14:07 19 января 2038 года включительно, достаточно вычесть одну дату из другой и преобразовать полученные секунды в дни:

```
$seconds =          = $earlier;
```

Если вы работаете с отдельными компонентами полного времени или беспокоитесь об ограничениях интервалов для секунд с начала эпохи, воспользуйтесь модулем Date::Calc с CPAN. Он позволяет вычислять разность дат:

```
use Date::Calc qw(Delta_DHMS);
($days, $hours, $minutes, $seconds) =
  Delta_DHMS( $year1, $month1, $day1, $hour1, $minute1, $seconds1, # Ранний
                                                                    S момент
              $year2, $month2, $day2, $hour2, $minute2, $seconds2, # Поздний
                                                                    # момент
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Одна из проблем, связанных с секундами с начала эпохи, — преобразование больших целых чисел в форму, понятную для человека. Следующий пример демонстрирует один из способов преобразования секунд с начала эпохи в привычные недели, дни, часы, минуты и секунды:

```
          361535725;          # 04:35:25 16 июня 1981 года
$nat   =   96201950;          # 03:45:50 18 января 1973 года

                        $nat;
print                                   between Nat and Bree\n";
            266802575 seconds

$seconds     =

$minutes
                              $minutes)
$hours
                              $hours)    / 24;
            =
            =                 $days)     / 7;

print "($weeks weeks, $days days, $hours:$minutes:$seconds)\n";
(441 weeks, 0 days, 23: 49: 35)
```

Функции модуля Date::Calc упрощают подобные вычисления. Delta_Days возвращает количество дней между двумя датами. Даты передаются ей в виде списка "год/месяц/день" в хронологическом порядке, то есть начиная с более ранней.

Смотри также

Документация по стандартному модулю Date::Calc с CPAN.

3.6. Определение номера недели или дня недели/месяца/года

Проблема

Имеется дата в секундах с начала эпохи или в виде отдельных компонентов — года, месяца и т. д. Требуется узнать, на какой номер недели или день недели/месяца/года она приходится.

Решение

Если дата выражена в секундах с начала эпохи, день года, день месяца или недели возвращается функцией `localtime`. Номер недели легко рассчитывается по дню года.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
($MONTHDAY, $WEEKDAY, $YEARDAY) = (localtime $DATE) [3,6,7];
$WEEKNUM = int($YEARDAY / 7) + 1;
```

Отдельные компоненты полного времени можно преобразовать в число секунд с начала эпохи (см. рецепт 3.3) и воспользоваться приведенным выше решением. Возможен и другой вариант — применение функций Day_of_Week, Week_Number и Day_of_Year модуля Date::Calc CPAN:

```
use Date::Calc qw(Day_of_Week Week_Number Day_of_Year);
# Исходные величины - $year, $month и $day
# По определению $day является днем месяца
$wday = Day_of_Week($year, $month, $day);
$wnum = Week_Number($year, $month, $day);
$dnum = Day_of_Year($year, $month, $day);
```

Комментарий

Функции Day_of_Week, Week_Number и Day_of_Year получают год без вычитания 1900 и месяц в нумерации, начинающейся с 1 (январь), а не с 0. Возвращаемое значение функции Day_of_Week находится в интервале 1-7 (с понедельника до воскресенья) или равняется 0 в случае ошибки (например, при неверно заданной дате).

```
use Date::Calc qw(Day_of_Week Week_Number);

$year  = 1981;
$month = 6;        # (Июнь)
$day   = 16;

@days = qw:Error Monday Tuesday Wednesday Thursday Friday Saturday Sunday:;

$wday = Day_of_Week($year, $month, $day);
print "$month/$day/$year was a $days[$wday]\n";

$wnum = Week_Number($year, $month, $day);
print "in the $wnum week.\n";
6/16/1981 was a Tuesday
in the 25 week
```

В некоторых странах существуют специальные стандарты, касающиеся первой недели года. Например, в Норвегии первая неделя должна содержать не менее 4 дней (и начинаться с понедельника). Если 1 января выпадает на неделю из 3 и менее дней, она считается 52 или 53 неделей предыдущего года. В Америке первая рабочая неделя обычно начинается с первого понедельника года. Возможно, при таких правилах вам придется написать собственный алгоритм или по крайней мере изучить форматы %G, %L, %W и %U функции UnixDate модуля Date::Manip.

Смотри также

Описание функции localtime в *perlfunc*(1); документация по стандартному модулю Date::Calc от CPAN.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

3.7. Анализ даты и времени в строках

Проблема

Спецификация даты или времени читается в произвольном формате, однако ее требуется преобразовать в отдельные компоненты (год, месяц и т. д.).

Решение

Если дата уже представлена в виде числа или имеет жесткий, легко анализируемый формат, воспользуйтесь регулярным выражением (и, возможно, хэшем, связывающим названия месяцев с номерами) для извлечения отдельных значений дня, месяца и года. Затем преобразуйте их в секунды с начала эпохи с помощью функций `timelocal` и `timegm` стандартного модуля Time::Local.

```
use Time::Local;
# $date хранится в формате "1999-06-03" (ГГГГ-ММ-ДД).
($yyyy, $mm, $dd) = ($date =^ /(\d+)-(\d+)-(\d+)/;
# Вычислить секунды с начала эпохи для полночи указанного дня
fl в текущем часовом поясе
$epoch_seconds = timelocal(0, 0, 0, $dd, $mm, $yyyy);
```

Более гибкое решение — применение функции ParseDate из модуля Date::Manip с CPAN и последующее извлечение отдельных компонентов с помощью `UnixDate`.

```
use Date::Manip qw(ParseDate UnixDate);
$date = ParseDate($STRING);
if (!$date) {
    # Неверная дата
} else {
    @VALUES = UnixDate($date, @FORMATS);
}
```

Комментарий

Универсальная функция ParseDate поддерживает различные форматы дат. Она даже преобразует такие строки, как "today" ("сегодня"), "2 weeks ago Friday" ("в пятницу две недели назад") и "2nd Sunday in 1996" ("2-е воскресенье 1996 года"), а также понимает форматы даты/времени в заголовках почты и новостей. Расшифрованная дата возвращается в собственном формате — строке вида «ГГГГММДДЧЧ:ММ:СС». Сравнение двух строк позволяет узнать, совпадают ли представленные ими даты, однако арифметические операции выполняются иначе. Поэтому мы воспользовались функцией `UnixDate` для извлечения года, месяца и дня в нужном формате.

Функция UnixDate получает дату в виде строки, возвращаемой ParseDate, и список форматов. Она последовательно применяет каждый формат к строке и возвращает результат. Формат представляет собой строку с описанием одного или нескольких элементов даты/времени и способов оформления этих элементов. Например, формат `%Y` соответствует году, состоящему из четырех цифр. Приведем пример:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Date::Manip qw(ParseDate UnixDate);

while (<>) {
    $date = ParseDate($_);
    if (!$date) {
        warn "Bad date string: $_\n";
        next;
    } else {
        ($year, $month, $day) = UnixDate($date, "%Y", "%m", "%d");
        print "Date was $month/$day/$year\n";
    }
}
```

Смотри также

Документация для модуля Date::Manip с CPAN; пример использования приведен в рецепте 3.11.

3.8. Вывод даты

Проблема

Требуется преобразовать дату и время, выраженные в секундах с начала эпохи, в более понятную длячеловека форму.

Решение

Вызовите `localtime` или `gmtime` в скалярном контексте — в этом случае функция получает количество секунд с начала эпохи и возвращает строку вида `Tue May 26 05:15:20 1998`:

```
$STRING = localtime($EPOCH_SECONDS);
```

Кроме того, функция `strftime` из стандартного модуля **POSIX** позволяет лучше настроить формат вывода и работает с отдельными компонентами полного времени:

```
use POSIX qw(strftime);
$STRING = strftime($FORMAT, $SECONDS, $MINUTES, $HOUR,
                   $DAY_OF_MONTH, $MONTH, $YEAR, $WEEKDAY,
                   $YEARDAY, $DST);
```

В модуле Date::Manip с CPAN есть функция UnixDate — нечто вроде специализированного варианта `sprintf`, предназначенного для работы с датами. Ей передается дата в формате Date::Manip. Применение Date::Manip вместо POSIX::strftime имеет дополнительное преимущество, так как для этого от системы не требуется совместимость с **POSIX.**

```
use Date::Manip qw(UnixDate);
$STRING = UnixDate($DATE, $FORMAT);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Простейшее решение — функция localtime — относится к встроенным средствам Perl. В скалярном контексте эта функция возвращает строку, отформатированную особым образом:

```
Sun Sep 21 15:33:36 1999
```

Программа получается простой, хотя формат строки сильно ограничен:

```
use Time::Local;
$time = timelocal(50, 45, 3, 18, 0, 73);
print "Scalar localtime gives: ", scalar(localtime($time)), "\n";
Scalar localtime gives: Thu Jan 18 03:45:50 1973
```

Разумеется, дата и время для localtime должны исчисляться в секундах с начала эпохи. Функция POSIX: :strftime получает набор компонентов полного времени и форматную строку, аналогичную printf, и возвращает также строку. Поля в выходной строке задаются директивами %. Полный список директив приведен в документации по strftime для вашей системы. Функция strftime ожидает, что отдельные компоненты даты/времени принадлежат тем же интервалам, что и значения, возвращаемые localtime:

```
use POSIX qw(strftime);
use Time::Local;
$time = timelocal(50, 45, 3, 18, 0, 73);
print "Scalar localtime gives; ", scalar(localtime($time)), "\n";
Scalar localtime gives: Thu Jan 18 03:45:50 1973
```

Разумеется, дата и время для localtime должны исчисляться в секундах с начала эпохи. Функция POSIX: :strftime получает набор компонентов полного времени и форматную строку, аналогичную printf, и возвращает также строку. Поля в выходной строке задаются директивами %. Полный список директив приведен в документации по strftime для вашей системы. Функция strftime ожидает, что отдельные компоненты даты/времени принадлежат тем же интервалам, что и значения, возвращаемые localtime:

```
use POSIX qw(strftime);
use Time::Local;
$time = timelocal(50, 45, 3, 18, 0, 73);
print "strftime gives: ", strftime("%A %D", localtime($time)), "\n";
strftime gives: Thursday 01/18/73
```

При использовании POSIX: :strftime все значения выводятся в соответствии с национальными стандартами. Так, во Франции ваша программа вместо "Sunday" выведет "Dimanche". Однако учтите: интерфейс Perl к функции strftime модуля POSIX всегда преобразует дату в предположении, что она относится к текущему часовому поясу.

Если функция strftime модуля POSIX недоступна, у вас всегда остается верный модуль Date::Manip, описанный в рецепте 3.6.

```
use Date::Manip qw(ParseDate UnixDate);
$date = ParseDate("18 Jan 1973, 3:45:50");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$datestr = UnixDate($date, "%a %b %e %H:%M:%S %z %Y"); # скалярный контекст
print "Date::Manip gives: $datestr\n";
Date::Manip gives: Thu Jan 18 03:45:50 GMT 1973
```

Смотри также

Описание функции gmtime и localtime в *perlfunc*(1); *perllocale*(1); man-страница *strftime*(3) вашей системы; документация по модулям POSIX и Date::Manip cCPAN.

3.9. Таймеры высокого разрешения

Проблема

Функция time возвращает время с точностью до секунды. Требуется измерить время с более высокой точностью.

Решение

Иногда эта проблема неразрешима. Если на вашем компьютере Perl поддерживает функцию syscall, а в системе имеется функция типа gettimeofday(2), вероятно, ими можно воспользоваться для измерения времени. Особенности вызова syscall зависят от конкретного компьютера. В комментарии приведен примерный вид фрагмента, однако его переносимость не гарантирована.

На некоторых компьютерах эти функциональные возможности инкапсулируются в модуле Time::HiRes (распространяется с CPAN):

```
use Time::HiRes qw(gettimeofday);
$t0 = gettimeofday;
## Ваши операции
$t1 = gettimeofday;
$elapsed = $t1 - $t0;
# $elapsed - значение с плавающей точкой, равное числу секунд
# между $t1 и $t2
```

Комментарий

В следующем фрагменте модуль Time::HiRes используется для измерения промежутка между выдачей сообщения и нажатием клавиши RETURN:

```
use Time::HiRes qw(gettimeofday);
print when ";
gettimeofday;
$line = <>;
$elapsed = gettimeofday-$before;
print "You took $elapsed seconds.\n";
return
You took 0.228149 seconds.
```

Сравните с эквивалентным фрагментом, использующим syscall:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        'sys/syscall.ph';

fl Инициализировать структуры, возвращаемые gettimeofday
$TIMEVAL_T = "LL";
$done = $start = pack($TIMEVAL_T, ());

# Вывод приглашения
print                          ";
# Прочитать время в $start
syscall(&SYS_gettimeofday, $start, 0)) != -1
              || die "gettimeofday: $!";

tt Прочитать перевод строки
$line = <>;

# Прочитать время в $done
syscall(&SYS_gettimeofday, $done, 0) != -1
              || die "gettimeofday $!";

fl Распаковать структуру
@start = unpack($TIMEVAL_T, $start);
@done  = unpack($TIMEVAL_T, $done);

# Исправить микросекунды
for ($done[1], $start[1]) { $_ /= 1_000_000 }

# Вычислить разность
$delta_time = sprintf "%.4f", ($done[0] + $done[1] )-

                              ($start[0] + $start[1] );

print "That took $delta_time seconds\n";
            when ready:
That took 0.3037 seconds
```

Программа получилась более длинной, поскольку системные функции вызываются непосредственно из Perl, а в Time::HiRes они реализованы одной функцией C. К тому же она стала сложнее — для вызова специфических функций операционной системы необходимо хорошо разбираться в структурах C, которые передаются системе и возвращаются ей. Некоторые программы, входящие в поставку Perl, пытаются автоматически определить форматы pack и unpack по заголовочному файлу C. В нашем примере *sys/syscall.ph* — библиотечный файл Perl, сгенерированный утилитой *h2ph*, которая преобразует заголовочный файл *sys/syscall.h* в *sys/syscall.ph*. В частности, в нем определена функция &SYS_gettimeofday, возвращающая номер системного вызова для gettimeofday.

Следующий пример показывает, как использовать Time::HiRes для измерения временных характеристик:

```
use Time::HiRes qw(gettimeofday);
# Вычислить среднее время сортировки
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$size = 500;
$number_of_times = 100;
$total_time = 0;

for ($i = 0; $i< number_of_times; $i++) {
    my (@array, $j, $begin, $time);
    # Заполнить массив
    @array = ();
    for ($j=0; $j<$size;$j++) { push(@array, rand) }

    # Выполнить сортировку
    $begin = gettimeofday;
    @array = sort { $a <=> $b } @array;
    $time = gettimeofday=$t1;
    $total_time += $time;
}

printf "On average, sorting %d random numbers takes %5.f seconds\n",
    $size, ($total_time/$number_of_times);
On average, sorting 500 random numbers takes 0.02821 seconds
```

Смотри также

Документация по модулям Time::HiRes и BenchMark с CPAN; описание функции syscall в perlfunc(1);man-страница syscall(2).

3.10. Короткие задержки

Проблема

Требуется сделать в программе паузу продолжительностью менее секунды.

Решение

Воспользуйтесь функцией select, если она поддерживается вашей системой:

```
select(undef, undef, undef, $time_to_sleep);
```

где `$time_to_sleep` — длительность паузы.

Некоторые системы не поддерживают select с четырьмя аргументами. В модуле Time::HiRes присутствует функция `sleep`, которая допускает длину паузы с плавающей точкой:

```
use Time::HiRes qw(sleep);
sleep($time_to_sleep);
```

Комментарий

Следующий фрагмент демонстрирует применение функции select. Он представляет собой уп рощенную версию программы из рецепта 1.5. Можете рассматривать его как эмулятор ЗОО-бодного терминала:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
while (<>) {
    select(undef, undef, undef, 0.25);
    print;
}
```

С помощью Time::HiRes это делается так:

```
use Time::HiRes qw(sleep);
while (<>) {
    sleep(0.25);
    print;
}
```

Смотри также

Документация по модулям Time::HiRes и BenchMark с CPAN; описание функций sleep и select в ***perlfunc(1)***. Функция select использована для организации короткой задержки в программе `slowcat` из рецепта 1.5.

3.11. Программа: hopdelta

Вы никогда не задавались вопросом, почему какое-нибудь важное письмо так долго добиралось до вас? Обычная почта не позволит узнать, как долго ваше письмо пылилось на полках всех промежуточных почтовых отделений. Однако в электронной почте такая возможность имеется. В заголовке сообщения присутствует строка `Received:` с информацией о том, когда сообщение было получено каждым промежуточным транспортным агентом.

Время в заголовках воспринимается плохо. Его приходится читать в обратном направлении, снизу вверх Оно записывается в разных форматах по прихоти каждого транспортного агента. Но хуже всего то, что каждое время регистрируется в своем часовом поясе. Взглянув на строки `"Tue, 26 May 1998 23:57:38 -0400"` и `"Wed, 27 May 1998 05:04:03 +0100"`, вы вряд ли сразу поймете, что эти два момента разделяют всего 6 минут 25 секунд.

На помощь приходят функции ParseDate и DateCalc модуля Date::Manip от CPAN:

```
use Date::Manip qw(ParseDate DateCalc);
$d1 = ParseDate("Tue, 26 May 1998 23:57:38 -0400");
$d2 = ParseDate("Wed, 27 May 1998 05:04:03 +0100");
print DateCalc($d1, $d2);
+0:0:0:0:0:6:25
```

Возможно, с такими данными удобно работать программе, но пользователь все же предпочтет что-нибудь более привычное. Программа hopdelta из примера 3.1 получает заголовок сообщения и пытается проанализировать дельты (разности) между промежуточными остановками. Результаты выводятся для местного часового пояса.

Пример 1.3. hopdelta

```
#!/usr/bin/perl
# hopdelta - по заголовку почтового сообщения выдает сведения
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
#            о задержке почты на каждом промежуточном участке.
use strict;
use Date::Manip qw (ParseDate UnixDate);

# Заголовок печати; из-за сложностей printf следовало
# бы использовать format/write
printf "%-20.20s %-20.20s %-20.20s  %s\n",
        "Sender", "Recipient", "Time", "Delta";
$/ = '';                # Режим абзаца
$_ = <>;                # Читать заголовок
s/\n\s+/ /g;            # Объединить строки продолжения

# Вычислить, когда и где начался маршрут сообщения
my($start_from) = /^From.*\@([^\s>]*)/m;
my($start_date) = /^Date:\s+(.*)/m;
my $then = getdate($start_date);
printf "%-20.20s %-20.20s %s\n", 'Start', $start_from, fmtdate($then);

my $prevfrom = $start_from;

# Обрабатывать строки заголовка снизу вверх
            split(/\n/)) {
    my ($delta, $now, $from, $by, $when);
    next unless /^Received:/;
    s/\bon (.*?) (id.*)/; $1/s;         # Кажется, заголовок qmail
    unless (($when) = /;\s+(.*)$/) {    #          date falls
        warn "bad          line: $_";
        next;
    }
    ($from) = /from\s+(\S+)/;
    ($from) = /\((.*?)\)/ unless $from; # Иногда встречается
    $from =~ s/\)$//;                   # Кто-то пожадничал
    ($by)   = /by\s+(\S+\.\S+)/;        # Отправитель для данного участка

    # Операции, приводящие строку к анализируемому формату
    for ($when) {
        s/ (for|via) .*$//;
        s/([+-]\d\d\d\d) \(\S+\)/$1/;
        s/id \S+;\s*//;
    }
    next unless $now = getdate($when);   # Перевести в секунды
                                         # с начала эпохи
    $delta = $now - $then;

    printf "%-20.20s %-20.20s %s  ", $from, $by, fmtdate($now);
            =
    puttime($delta);
    $then = $now;
}

exit;
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 1.3 (продолжение)

```
# Преобразовать произвольные строки времени в секунды с начала эпохи
sub getdate {
    my $string     =  shift;
    $string        =^ s/\s+\(.*\)\s*$//;        # Убрать нестандартные
                                                # терминаторы
    my $date        =  ParseDate($string);
    my $epoch_secs =  UnixDate($date,"%s");
           $epoch_secs;
}

# Преобразовать секунды с начала эпохи в строку определенного формата
sub fmtdate {
    my $epoch = shift;
    my($sec,$min,$hour,$mday,$mon,$year) = localtime($epoch);
           sprintf "%02d:%02d:%02d %04d/%02d/%02d",
           $hour, $min, $sec,
           $year + 1900, $mon + 1, $mday,
}

# Преобразовать секунды в удобочитаемый формат
sub puttime {
    my($seconds) = shift;
    my($days, $hours, $minutes);
    $days    = pull_count($seconds, 24 * 60 * 60);
    $hours   = pull_count($seconds, 60 * 60);
    $minutes = pull_count($seconds, 60);
    put_field('s',  $seconds);
    put_field('m', $minutes);
    put_field('h',  $hours);
    put_field('d', $days);
    print "\n";
}

# Применение: $count = pull_count(seconds, amount)
# Удалить из seconds величину amount, изменить версию вызывающей
# стороны и вернуть число удалений.
sub pull_count {
    my($answer) = int($_[0] / $_[1]);
    $_[0] -= $answer * $_[1];
           $answer;
}

# Применение: put_field(char, number)
# Вывести числовое поле в десятичном формате с 3 разрядами и суффиксом char
# Выводить лишь для секунд (char == 's')
sub put_field {
    my ($char, $number) = @_;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    printf  %3d%s', $number, $char if $number || $char eq 's',
>

Sender                  Recipient               Time                    Delta
 Start                  wall.org                09:17:12  1998/05/23    44s   3m
wall.org                mail.brainstorm.net     09:20:56  1998/05/23
   mail.brainstorm.net                          09:20:58  1998/05/23     2s
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

4 Массивы

Я считаю, что произведения искусства — единственные объекты материальной Вселенной, обладающие внутренним порядком. И потому, не веря в высшую ценность искусства, я все же верю в Искусство ради Искусства.

Э. М. Фостер

Введение

Если попросить вас перечислить содержимое своих карманов, назвать имена трех последних президентов или объяснить, как пройти к нужному месту, в любом случае получится список: вы называете объекты один за другим в определенном порядке. Списки являются частью нашего мировоззрения. Мощные примитивы Perl для работы со списками и массивами помогают преобразовать мировоззрение в программный код.

Термины *список* (list) и *массив* (array) трактуются в этой главе в соответствии с канонами Perl. Например, `("Reagan", "Bush", "Clinton")` — это *список* трех последних американских президентов. Чтобы сохранить его в переменной, воспользуйтесь *массивом*: `@presidents= ("Reagan", "Bush", "Clinton")`. Каждый из этих терминов относится к упорядоченной совокупности скалярных величин; отличие состоит в том, что массив представляет собой именованную переменную, размер которой можно непосредственно изменить, а список является скорее отвлеченным понятием. Можно рассматривать массив как переменную, а список — как содержащиеся в ней значения.

Отличие может показаться надуманным, но операции, изменяющие размер этой совокупности (например, `push` или `pop`), работают с массивом, а не списком. Нечто похожее происходит с `$a` и 4: в программе можно написать `$a++`, но не `4++`. А налогично, `pop(@a)` — допустимо, а `pop(1, 2, 3)` — нет.

Главное — помнить, что списки и массивы в Perl представляют собой упорядоченные совокупности скалярных величин. Операторы и функции, работающие со списками и массивами, обеспечивают более быстрый или удобный доступ к элементам по сравнению с ручным извлечением. Поскольку размер массива изменяется не так уж часто, термины "массив" и "список" обычно можно считать синонимами.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Вложенные списки не создаются простым вложением скобок. В Perl следующие строки эквивалентны:

```
@nested = ("this", "that", "the", "order");
@nested = ("this", "that", ("the", "order"));
```

Почему Perl не поддерживает вложенные списки напрямую? Отчасти по историческим причинам, но также и потому, что это позволяет многим операциям (типа `print` или `sort`) работать со списками произвольной длины и произвольного содержания.

Что делать, если требуется более сложная структура данных — например, массив массивов или массив хэшей? Вспомните, что скалярные переменные могут хранить не только числа или строки, но и ссылки. Сложные (многоуровневые) структуры данных в Perl всегда образуются с помощью ссылок. Следовательно, "двумерные массивы" или "массивы массивов" в действительности реализуются как массив *ссылок на массивы* — по аналогии с двумерными массивами C, которые могут представлять собой массивы указателей на массивы.

Для большинства рецептов этой главы содержимое массивов несущественно. Например, проблема слияния двух массивов решается одинаково для массивов строк, чисел или ссылок. Решения некоторых проблем, связанных с содержимым массивов, приведены в главе 11 "Ссылки и записи". Рецепты этой главы ограничиваются обычными массивами.

Давайте введем еще несколько терминов. Скалярные величины, входящие в массив или список, называются *элементами*. Для обращения к элементу используется его позиция, или *индекс*. Индексация в Perl начинается с 0, поэтому в следующем списке:

```
@tune = ("The", "Star-Spangled", "Banner");
```

элемент "The" находится в первой позиции, но для обращения к нему используется индекс 0: `$tune[0]`. Это объясняется как извращенностью компьютерной логики, где нумерация обычно начинается с 0, так и извращенностью разработчиков языка, которые выбрали 0 как *смещение* внутри массива, а не *порядковый номер* элемента.

4.1. Определение списка в программе

Проблема

Требуется включить в программу список — например, при инициализации массива.

Решение

Перечислите элементы, разделяя их запятыми:

```
@a = ("quick", "brown", "fox");
```

При большом количестве однословных элементов воспользуйтесь оператором `qw()`:

```
@a = qw(Why are you teasing me?);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При большом количестве многословных элементов создайте встроенный документ и последовательно извлекайте из него строки:

```
@lines = (<<"END_OF_HERE_DOC" =~ m/^\s*(.+)/gm);
    The boy stood on the  burning deck,
    It was as hot as glass.
END_OF_HERE_DOC
```

Комментарий

Наиболее распространен первый способ — в основном из-за того, что в виде литералов в программе инициализируются лишь небольшие массивы. Инициализация большого массива загромождает программу и усложняет ее чтение, поэтому такие массивы либо инициализируются в отдельном библиотечном файле (см. главу 12 "Пакеты, библиотеки и модули"), либр просто читаются из файла данных:

```
@bigarray = ();
open(DATA, "< mydatafile")  or die "Couldn't read from datafile: $!\n";
while (<DATA>) {
    chomp;
    push(@bigarray, $_);
}
```

Во втором способе используется оператор qw. Наряду с q(), qq() и qx() он предназначен для определения строковых величин в программе. Оператор q() интерпретируется по правилам для апострофов, поэтому следующие две строки эквивалентны:

```
$banner = 'The mines of Moria';
$banner = q(The mines of Moria);
```

Оператор qq() интерпретируется по правилам для кавычек:

```
$name = "Gandalf";
$banner = "Speak, $name, and enter!";
$banner = qq(Speak, $name, and welcome!);
```

А оператор qx() интерпретируется почти так же, как и обратные апострофы, — то есть выполняет команду с интерполяцией переменных и служебными символами \ через командный интерпретатор. В обратных апострофах интерполяцию отменить нельзя, а в qx — можно. Чтобы отказаться от расширения переменных Perl, используйте в qx ограничитель ':

```
$his_host   = 'www.perl.com';
$host_info  = 'nslookup $his_host'; # Переменная Perl расширяется
$perl_info  = qx(ps $$);            # Значение $$ от Perl
$shell_info = qx'ps $$';            # Значение $$ от интерпретатора
```

Если операторы q(), qq() и qx() определяют одиночные строки, то qw() определяет список однословных строк. Строка-аргумент делится по пробелам без интерполяции переменных. Следующие строки эквивалентны:

```
@banner = ('Costs', 'only', '$4.95');
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
@banner = qw(Costs only $4 95),
@banner = split(    ,   Costs only $4 95  ),
```

Во всех операторах определения строк, как и при поиске регулярных выражений, разрешается выбор символа-ограничителя, включая парные скобки. Допустимы все четыре типа скобок (угловые, квадратные, фигурные и круглые). Следовательно, вы можете без опасений использовать любые скобки при условии, что для открывающей скобки найдется закрывающая:

```
@brax   = qw! ( ) < > { } [ ] !,
@rings  = qw(Nenya Narya Vilya),
$tags   = qw<LI TABLE TR TD A IMG H1 P>,
$sample = qw(The vertical bar (|) looks and behaves like a pipe ),
```

Если ограничитель встречается в строке, а вы не хотите заменить его другим, используйте префикс \:

```
@banner = qw|The vertical bar (\|) looks and behaves like a pipe |,
```

Оператор qw( ) подходит лишь для списков, вкоторых каждый элемент является отдельным словом, ограниченным пробелами. Будьте осторожны, а то у Колумба вместо трех кораблей появится четыре:

```
$ships = qw(Nica Pinta Santa Magna),          # НЕВЕРНО
$ships = ( Nica ,  Pinta ,  Santa Magna ),    # Правильно
```

Смотри также

Раздел "List Value Constructors" *perldata*(1); раздел "Quote and Quote-Like Operators» *perlop*(1); оператор s/// описан в *perlop*(1).

4.2. Вывод списков с запятыми

Проблема

Требуется вывести список с неизвестным количеством элементов. Элементы разделяются запятыми, а перед последним элементом выводится слово and

Решение

Следующая функция возвращает строку, отформатированную требуемым образом

```
sub commify_series {
    (@_ == 0) ?  ''
    (@_ == 1) ? $_[0]
    (@_ == 2) ? join( and , @_)
                join( , , @_[0 .. ($#_-1)] and $_[-1] ),
}
```

Комментарий

При выводе содержимое массива порой выглядит довольно странно:

```
@array = ( red ,  yellow
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
print "I have ", @array, " marbles.\n";
print "I have @array roarbles\n";
I have redyellowgreen marbles.
I have red yellow
```

На самом деле вам нужна строка "I marbles". Приведенная выше функция генерирует строку именно в таком формате. Между двумя последними элементами списка вставляется "and". Если в списке больше двух элементов, все они разделяются запятыми.

Пример 4.1 демонстрирует применение этой функции с одним дополнением: если хотя бы один элемент списка содержит запятую, в качестве разделителя используется точка с запятой.

Пример 4.1. commify_series

```
#!/usr/bin/perl -w
# commify_series - демонстрирует вставку запятых при выводе списка

@lists = (
    [ 'just one thing' ],
    [ qw(Mutt Jeff) ],
    [ qw(Peter Paul Mary) ],
                        'Mother Theresa', 'God' ],
    [ 'pastrami', 'ham and cheese', 'peanut butter and jelly', 'tuna' ],
      'recycle            phrases', 'ponder big, happy thoughts' ],
      'recycle            phrases',
      'ponder big, happy thoughts',
      'sleep and        peacefully'
    );

            (@lists)
    print "The list is: " . commify_series(@$aref) . ".\n";
}

sub commify_series {
    my $sepchar =            @_)   ";"   ",";
    (@_ == 0) ? ''                                  :
    (@_ == 1) ? $_[0]                               :
    (@_ == 2) ? join(" and ", @_)                   :
                join("$sepchar ", @_[0 .. ($#_-1)], "and $_[-1]");
}
```

Результаты выглядят так:

```
The list is: Just one thing.
The list is: Mutt and Jeff.
The list is: Peter, Paul, and Mary.
The list is: To our parents, Mother
The list ls: pastrami, ham and cheese, peanut butter and jelly, and tuna.
The list is:                    phrases and ponder big, happy thoughts.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
The list is: recycle old phrases; ponder
    big, happy thoughts; and sleep and peacefully.
```

Как видите, мы отвергаем порочную практику исключения последней запятой из списка, что нередко приводит к появлению двусмысленностей.

Смотри также

Описание функции grep в *perlfunc(1);* описание тернарного оператора выбора в *perlop*(1). Синтаксис вложенных списков рассматривается в рецепте 11.1.

4.3. Изменение размера массива

Проблема

Требуется увеличить или уменьшить размер массива. Допустим, у вас имеется массив работников, отсортированный по размерам оклада, и вы хотите ограничить его пятмью самыми высокооплачиваемыми работниками. Другой пример — если окончательный размер массива точно известен, намного эффективнее выделить всю память сразу вместо того, чтобы увеличивать массив постепенно, добавляя элементы в конец.

Решение

Присвойте значение $#ARRAY:

```
# Увеличить или уменьшить @ARRAY
$#ARRAY = $NEW_LAST_ELEMENT_INDEX_NUMBER
```

Присваивание элементу, находящемуся за концом массива, автоматически увеличивает массив:

```
$ARRAY[$NEW_LAST_ELEMENT_INDEX_NUMBER] = $VALUE;
```

Комментарий

$#ARRAY — последний допустимый индекс массива @ARRAY. Если ему присваивается значение меньше текущего, массив уменьшается. Отсеченные элементы безвозвратно теряются. Если присвоенное значение больше текущего, массив увеличивается. Новые элементы получают неопределенное значение.

Однако $#ARRAY не следует путать с @ARRAY. $#ARRAY представляет собой последний допустимый индекс массива, а @ARRAY (в скалярном контексте, то есть в числовой интерпретации) — ***количество*** элементов. $#ARRAY на единицу меньше @ARRAY, поскольку нумерация индексов начинается с 0.

В следующем фрагменте использованы оба варианта:

```
sub what_about_that_array {
    print "The array now has", scalar(@people), " elements.\n";
    print "The index of the last element is $#people.\n";
    print "Element #3 is '$people[3]'.\n";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
@people = qw(Crosby Stills Nash.Young);
what_about_that_array();
```

Результат:

```
The array now has 4 elements.
The index of the last element is 3.
Element "3 is 'Young'.
```

А другой фрагмент:

```
$#people--;
what_about_that_array();
```

выводит следующий результат:

```
The array now has 3 elements.
The index of the last element is 2.
Element "3 is ''.
```

Элемент с индексом 3 пропал при уменьшении массива. Если бы программа запускалась с ключом -w, Perl также выдал бы предупреждение об использовании неинициализированной **величины**, поскольку значение $people[3] не определено.

В следующем примере:

```
$#people = 10_000;
what_about_that_array();
```

результат выглядит так:

```
The array now has 10001 elements.
The index of the last element is 10000.
Element "3 is ''.
```

Элемент "Young" безвозвратно утерян. Вместо присваивания `$#people` можно было сказать:

```
$people[10_000]=undef;
```

Массивы Perl не являются **разреженными**. Другими словами, если у вас имеется **10000-й** элемент, то должны присутствовать **и** остальные **9999** элементов. Они могут быть неопределенными, но все равно будут занимать память. Из-за этого `$array[time]` или любая другая конструкция, где в качестве индекса используется очень большое целое число, является неудачным решением. Лучше воспользуйтесь хэшем.

При вызове `print` нам пришлось написать scalar @array, поскольку Perl интерпретирует большинство аргументов в списковом контексте, а требовалось значение `@array` в скалярном контексте.

Смотри также

Описание `$#ARRAY` в *perldata*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

4.4. Выполнение операции с каждым элементом списка

Проблема

Требуется повторить некоторую операцию для каждого элемента списка.

Массивы часто используются для сбора интересующей информации - например, имен пользователей, превысивших свои дисковые квоты. Данные обрабатываются, при этом с каждым элементом массива выполняется некоторая операция. Скажем, в примере с дисковыми квотами каждому пользователю отправляется предупреждающее сообщение.

Решение

Воспользуйтесь циклом `foreach`:

```
$item (LIST) {
    # Выполнить некоторые действия с $item
}
```

Комментарий

Предположим, в массиве `@bad_users` собран список пользователей, превысивших свои дисковые квоты. В следующем фрагменте для каждого нарушителя вызывается процедура `complain()`:

```
$user (@bad_users) {
    complain($user);
}
```

Столь тривиальные случаи встречаются редко. Как правило, Для генерации списка часто используются функции

```
(sort keys %ENV) <
    print "$var=$ENV{$var}\n";
}
```

Функции sort и keys строятотсортированный список имен переменных окружения. Конечно, многократно используемые сйиски следует сохранять в массивах. Но для одноразовых задач удобнее работать со списком напрямую.

Возможности этой конструкции расширяются не только за счет построения списка в дополнительных операций в блоке кода. Одно из распространенных применений — сбор информации о каждом элементе списка и принятие некоторого решения на основании полученных данных. Вернемся к примеру с квотами:

```
$user (@all_users) {
    $disk_space = get_usage($user);    " Определить объем используемого
                                       tf дискового пространства
    if ($disk_space > $MAX_QUOTA) {    # Если он больше допустимого...
        complain($user);               # ... предупредить о нарушении.
    }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Возможны и более сложные варианты. Команда last прерывает цикл, next переходит к следующему элементу, а ращается к первой команде внутри блока. Фактически вы говорите: "Нет смысла продолжать, это не то, что мне нужно" (next), "Я нашел то, что искал, и проверять остальные элементы незачем" (last) или "Я тут кое-что изменил, так что проверки и вычисления лучше выполнить заново" (redo).

Переменная, которой последовательно присваиваются все элементы списка, называется *переменной цикла* или *итератором*. Если итератор не указан, используется глобальная переменная $_. Она используется по умолчанию во многих строковых, списковых и файловых функциях Perl. В коротких программных блоках пропуск $_ упрощает чтение программы (хотя в длинных блоках излишек неявных допущений делает программу менее понятной). Например:

```
foreach ('who') {
    if (/tchrist/) {
        print;
    }
}
```

Или в сочетании с циклом while:

```
while (<FH>) {                  # Присвоить $_ очередную прочитанную строку
    chomp;                      # Удалить из $_ юнечный символ \n,
                                # если он присутствует
            (split) {           # Разделить $_ по пропускам и получить @_
                                # Последовательно присвоить $_
                                # каждый из полученных фрагментов
        $_ = reverse,           # Переставить символы $_
                                # в противоположном порядке
        print,                  # Вывести значение $_
    }
}
```

Многочисленные применения $_ заставляют понервничать. Особенно беспокоит то, что значение $_ изменяется как в foreach, так и в while. Возникает вопрос — не будет ли полная строка, прочитанная в $_ через <FH>, навсегда потеряна после выполнения

К счастью, эти опасения необоснованны — по крайней мере, в данном случае. Pcrl не уничтожает старое значение $_, поскольку переменная-итератор ($_) существует в течение всего выполнения цикла. При входе во внутренний цикл старое значение автоматически сохраняется, а при выходе — восстанавливается.

Однако причины для беспокойства все же есть. Если цикл while будет внутренвнешним, ваши страхи в полной мере оправдаются. В отличие от конструкция while <FH> разрушает глобальное значение $_ без предварительного сохранения! Следовательно, в начале любой процедуры (или блока), где $_ используется в подобной конструкции, всегда должно присутствовать объявление local $_.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если в области действия (scope) присутствует лексическая переменная (объявленная с my), то временная переменная будет иметь лексическую область действия, ограниченную данным циклом. В противном случае она будет считаться глобальной переменной с динамической областью действия. Во избежание странных побочных эффектов версия 5.004 допускает более наглядную и понятную запись:

```
            $item (@array)
    print "i ='$item\n";
}
```

Цикл foreach обладает еще одним свойством: в цикле переменная-итератор является не копией, а скорее *синонимом* (alias) текущего элемента. Иными словами, изменение итератора приводит к изменению каждого элемента списка.

```
@array = (1,2,3);
        $item
    $item--;
}
print "@array";
0 1 2

# Умножить каждый элемент @a и @b на семь
@a = (.5, 3); @b = (0, 1);
        $item (@a, @b)
    $item *= 7;-
    print "$item ";
}
3.5 21 0 7
```

Модификация списков в цикле оказывается более понятной и быстрой, чем в эквивалентном коде с циклом for и указанием конкретных индексов. Это не ошибка; такая возможность была намеренно предусмотрена разработчиками языка. Не зная о ней, можно случайно изменить содержимое списка. Теперь вы знаете.

Например, применение s/// к элементам списка, возвращаемого функцией values, приведет к модификации только копий, но не самого хэша. Однако срез хэша @hash{keys %hash} (см. главу 5 «Хэши») дает нам нечто, что все же *можно* изменить с пользой для дела:

```
# Убрать пропуски из скалярной величины, массива и всех элементов хэша
        ($scalar, @array, @hash{keys %hash}) {
    s/^\s+//;
    s/\s+$//;
}
```

По причинам, связанным с эквивалентными конструкциями командного интерпретатора Борна для UNIX, ключевые слова for и взаимозаменяемы:

```
    $item (@array)    #                    $item (@array)
    # Сделать что-то                                  —
}

for (@array)      {  # То же, что и          (@array)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    # Сделать что-то
}
```

Подобный стиль часто показывает, что автор занимается написанием или сопровождением сценариев интерпретатора и связан с системным администрированием UNIX. Жизнь таких людей и без того сложна, поэтому не стоит судить их слишком строго.

Смотри также

Разделы "For Loops", и "Loop Control" *perlsyn*(1); раздел "Temporary Values via local() *perlsub*(1). Оператор local() рассматривается в рецепте 10.13, а my() — в рецепте 10.2.

4.5. Перебор массива по ссылке

Проблема

Имеется ссылка на массив. Вы хотите использовать обращения к каждому элементу массива.

Решение

Для перебора разыменованного массива используется цикл или for:

```
# Перебор элементов массива $ARRAYREF
        $item(@$ARRAYREF)
    # Сделать что-то с $item
}

for ($I = 0, $I <= $#$ARRAYREF, $I++) {
    # Сделать что-то с $ARARAYREF->[$i]
}
```

Комментарий

Приведенное решение предполагает, что у вас имеется скалярная переменная, содержащая ссылку на массив. Это позволяет делать следующее:

```
@fruits = ( Apple , Blackberry ),
$fruit_ref = fruits,
        $fruit (@$fruit_ref)
    print $fruit tastes good in a pie \n ,
}
Apple tastes good in a pie.
Blackberry tastes good in a pie.
```

Цикл можно переписать в цикле for следующего вида:

```
for ($i=0, $1 <= $#$fruit_ref, $i++) {
    print $fruit_ref->[$i] tastes good in a pie \n ,
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Однако ссылка на массив нередко является результатом более сложного выражения. Для превращения такого результата в массив применяется конструкция @{ EXPR }:

```
$namelist{felines} = \@rogue_cats;
foreach $cat ( @{ $namelist{felines} } ) {
    print "$cat purrs hypnotically..\n";
}
print "--More--\nYou are controlled.\n";
```

Как и прежде, цикл можно заменить эквивалентным циклом for:

```
for ($i=0; $i <= $#{ $namelist{felines}}; $i++) {
    print "$namelist{felines}[$i] purrs hypnotically.\n";
}
```

Смотри также

perllol(1); рецепты 4.4; 11.1.

4.6. Выборка уникальных элементов из списка

Проблема

Требуется удалить из списка повторяющиеся элементы — например, при построении списка из файла или на базе выходных данных некоей команды. Рецепт в равной мере относится как к удалению дубликатов при вводе, так и в уже заполненных массивах.

Решение

Хэш используется для сохранения встречавшихся ранее элементов, а функция keys — для их извлечения. Принятая в Perl концепция истинности позволит уменьшить объем программы и ускорить ее работу.

Прямолинейно

```
%seen = ();
@uniq = ();
foreach $item (@list){
    unless ($seen{$item})
        " Если мы попали сюда, значит, элемент не встречался ранее
        $seen{$item} = 1;
        push(@uniq, $item);
    }
}
```

Быстро

```
%seen = ();
        $item (@list)
    push(@uniq, $item) unless $seen{$item}++;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Аналогично, но с пользовательской функцией

```
%seen = ();
foreach $item (@list) {
    some_func($item) unless $seen{$item}++;
}
```

Быстро, но по-другому

```
%seen = ();
        $item (@list)
    $seen{$item}++,
}
@uniq = keys %seen;
```

Быстро и совсем по-другому

```
%seen = ();
@unique            $seen{$_} ++ } @list;
```

Комментарий

Суть сводится к простому вопросу — встречался ли данный элемент раньше? Хэши идеально подходят для подобного поиска. В первом варианте ("Прямолинейно") массив уникальных значений строится по мере обработки исходного списка, а для регистрации встречавшихся значений используется хэш.

Второй вариант ("Быстро") представляет собой самый естественный способ решения подобных задач в Perl. Каждый раз, когда встречается новое значение, в хэш с помощью оператора ++ добавляется новый элемент. Побочный эффект состоит в том, что в хэш попадают все повторяющиеся экземпляры. В данном случае хэш работает как множество.

Третий вариант ("Аналогично, но с пользовательской функцией") похож на второй, однако вместо сохранения значения мы вызываем некоторую пользовательскую функцию и передаем ей это значение в качестве аргумента. Если ничего больше не требуется, хранить отдельный массив уникальных значений будет излишне.

В следующем варианте ("Быстро, но по-другому") уникальные ключи извлекаются из хэша %seen лишь после того, как он будет полностью построен. Иногда это удобно, но исходный порядок элементов утрачивается.

В последнем варианте ("Быстро и совсем по-другому") построение хэша %seen объединяется с извлечением уникальных элементов. При этом сохраняется исходный порядок элементов.

Использование хэша для записи значений имеет два побочных эффекта: при обработке длинных списков расходуется много памяти, а список, возвращаемый keys, не отсортирован в алфавитном или числовом порядке и не сохраняет порядок вставки.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Ниже показано, как обрабатывать данные по мере ввода. Мы используем `'who'` для получения сведений о текущем списке пользователей, а перед обновлением хэша извлекаем из каждой строки имя пользователя:

```
# Построить список зарегистрированных пользователей с удалением дубликатов
%ucnt = ();
for ('who') {
    s/\s.*\n//;    # Стереть от первого пробела до конца строки -
                   # остается имя пользователя
    $ucnt{$ }++;   # Зафиксировать присутствие данного пользователя
}
# Извлечь и вывести уникальные ключи
@users = sort keys %ucnt;
print "users logged in: @users\n";
```

Смотри также

Раздел Loops" *perlsyn(1);* описание функции keys в *perlfunc(1).* Аналогичное применение хэшей продемонстрировано в рецептах 4.7 и 4.8.

4.7. Поиск элементов одного массива, отсутствующих в другом массиве

Проблема

Требуется найти элементы, которые присутствуют в одном массиве, но отсутствуют в другом.

Решение

Мы ищем элементы @A, которых нет в @B. Постройте хэш из ключей @B — он будет использоваться в качестве таблицы просмотра. Затем проверьте каждый элемент @A и посмотрите, присутствует ли он в @B.

Простейшая реализация

```
# Предполагается, что @A и @B уже загружены
%seen = ();          # Хэш для проверки принадлежности элемента B
@aonly = ();         # Ответ

# Построить таблицу просмотра
        $item (@B)   $seen{$item}

# Найти элементы @A, отсутствующие в @B
        $item (@A)
    unless $item (@A) {
        # Отсутствует в %seen, поэтому добавить в @aonly
        push(@aonly, $item);
    }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Идиоматическая версия

```
1my %seen;      # Таблица просмотра
my @aonly;    # Ответ

# Построить таблицу просмотра
@seen{@B} = ();

        $item (@A)
    push(@aonly, $item) unless exists $seen{$item};
}
```

Комментарий

Практически любая проблема, при которой требуется определить принадлежность скалярной величины к списку или **массиву,** решается **в** Perl с помощью **хэ**шей. Сначала мы обрабатываем @B и регистрируем **в хэше** %seen все элементы @B, присваивая соответствующему элементу хэша значение 1. Затем мы последовательно перебираем все элементы @A и проверяем, присутствует ли данный элемент в хэше %seen (то есть в @B).

В приведенном фрагменте ответ будет содержать дубликаты из массива @A. Ситуацию нетрудно исправить, для этого достаточно включать элементы @A в %seen по мере обработки:

```
        $item
    push (@aonly, $item) unless $seen{$item};
    $seen{$item} = 1;       # Пометить как уже встречавшийся
}
```

Эти решения **в** основном отличаются по способу построения хэша. **В** первом варианте перебирается содержимое @B. Во втором для инициализации хэша используется ***срез.*** Следующий пример наглядно демонстрирует срезы хэша. Фрагмент:

```
$hash{"key1"} = 1;
$hash{"key2"} = 2;
```

эквивалентен следующему:

```
@hash{"key1", "key2"} = (1,2);
```

Список в фигурных скобках содержит ключи, а список справа — значения. В первом решении %seen инициализируется перебором всех элементов @B и присваиванием соответствующим элементам %seen значения 1. Во втором мы просто говорим:

```
@seen{@B} = ();
```

В этом случае элементы @B используются в качестве ключей для %seen, а с ними ассоциируется undef, поскольку количество значений в правой части меньше количества позиций для их размещения. Показанный вариант работает,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

поскольку мы проверяем только факт существования ключа, а не его логическую истинность или определенность. Но даже если с элементами @B потребуется ассоциировать истинные значения, срез все равно позволит сократить объем кода:

```
@seen{@B} = (1) x @B;
```

Смотри также

Описание срезов хэшей в *perldata(1)*. Аналогичное применение хэшей продемонстрировано в рецептах 4.7 и 4.8.

4.8. Вычисление объединения, пересечения и разности уникальных списков

Проблема

Имеются два списка, каждый из которых содержит неповторяющиеся элементы. Требуется узнать, какие элементы присутствуют в обоих списках (*пересечение*), присутствуют в одном и отсутствуют в другом списке (*разность*) или хотя бы в одном из списков (*объединение*).

Решение

В приведенных ниже решениях списки инициализируются следующим образом:

```
@a = (1, 3, 5, 6, 7, 8);
@b = (2, 3, 5, 7, 9);

@union = @isect = @diff = ();
%union = %isect = ();
%count = ();
```

Простое решение для объединения и пересечения

```
        $e(@a)  $union{$e} = 1 }

foreach $e (@b) {
    if ( $union{$e} ) {  $isect{$e} = 1 }
    $union {$e} = 1;
}
@union = keys %union;
@isect = keys %isect;
```

Идиоматическое решение

```
            (@a, @b) { $union{$e}++ && $isect{$e}++ }

@union = keys %unions;
@isect = keys %isect;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Объединение, пересечение и симметричная разность

```
foreach $e (@a, @b) { $count{$e}++ }

foreach $e (keys %count) {
    push(@union, $e);
    if ($count{$e> == 2) {
        push @isect, $e;
    } else {
        push @diff, $e;
    }
}
```

Косвенное решение

```
@isect = @diff = @union = ();

            (@a, @b)    $count{$e}++ }

            (keys %count) {
    push(@union, $e);
    push @{ $count{$e} == 2 ? \@isect : \@diff }, $e;
}
```

Комментарий

В первом решении происходит непосредственное вычисление объединения и пересечения двух списков, ни один из которых не содержит дубликатов. Для записи элементов, принадлежащих к объединению и пересечению, используются два разных хэша. Сначала мы заносим каждый элемент первого массива в хэш объединения и ассоциируем с ним истинное значение. Затем при последовательной обработке элементов второго массива мы проверяем, присутствует ли элемент в объединении. Если присутствует, он также включается и в хэш пересечения. В любом случае элемент заНосится в хэш объединения. После завершения перебора мы извлекаем ключи обоих хэшей. Ассоциированные с ними значения не нужны.

Второе решение ("Идиоматическое") в сущности делает то же самое, однако для него потребуется хорошее знание операторов Perl (а также awk, C, C++ и Java) ++ и &&. Если ++ находится после переменной, то ее старое значение используется до приращения. Когда элемент встречается впервые, он еще отсутствует в объединении, поэтому первая часть && будет ложной, а вторая часть попросту игнорируется. Когда тот же элемент встретится во второй раз, он уже присутствует в объединении, поэтому мы заносим его и в пересечение.

В третьем решении использован всего один хэш для хранения информации о том, сколько раз встретился тот или иной элемент. Записав элементы обоих массивов в хэш, мы последовательно перебираем его ключи. Каждый ключ автоматически попадает в объединение. Ключи, с которыми ассоциировано значение 2, присутствуют в обоих массивах и потому заносятся в массив пересече-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ния. Ключи с ассоциированным значением 1 встречаются лишь в одном из двух массивов и заносятся в массив разности. В отличие от исходного решения, порядок элементов в выходных массивах не совпадает с порядком элементов входных массивов.

В последнем **решении**, как и в **предыдущем**, используется всего один хэш с количеством экземпляров каждого элемента. Однако на этот раз реализация построена на массиве в блоке `@{...}`.

Мы вычисляем не простую, а симметричную разность. Эти термины происходят из теории множеств. *Симметричная* ***разность*** представляет собой набор всех элементов, являющихся членами либо @A, либо `@B`, но не обоих сразу. *Простая разность* — набор всех элементов @A, отсутствующих в `@B` (см. рецепт 4.7).

Смотри также

Аналогичное применение хэшей продемонстрировано в рецептах 4.7 и 4.8.

4.9. Присоединение массива

Проблема

Требуется объединить два массива, дописав все элементы одного из них в конец другого.

Решение

Воспользуйтесь функцией `push`:

```
# push
push(@ARRAY1, @ARRAY2);
```

Комментарий

Функция `push` оптимизирована для записи списка в конец массива. Два массива также можно объединить посредством сглаживания (flattening) списков Perl, однако в этом случае выполняется намного больше операций копирования, чем при использовании `push`:

```
@ARRAY1 = (@ARRAY1, @ARRAY2);
```

Ниже показан пример практического использования `push`:

```
@members = ("Time", "Flies");
@initiates = ("An", "Arrow");
push(@members, @initiates);
# @members содержит элементы ("Time", "Flies", "An", "Arrow")
```

Если содержимое одного массива требуется вставить в середину другого, воспользуйтесь функцией `splice`:

```
splice(@members, 2, 0, "Like", @initiates);
print "@members\n";
splice(@members, 0, 1, "Fruit");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
splice(@members, -2, 2, "A", "Banana");
print "@members\n";
```

Результат выглядит так:

```
Time Flies Like An Arrow
Fruit Flies Like A Banana
```

Смотри также

Описание функций splice и push в *perlfunc(1);* раздел "List Value Constructors" *perldata(1).*

4.10. Обращение массива

Проблема

Требуется обратить массив (то есть переставить элементы в противоположном порядке).

Решение

Воспользуйтесь функцией reverse:

```
# Обращение @ARRAY дает @REVERSEO
@REVERSED =          @ARRAY;
```

Также можно воспользоваться циклом for:

```
for ($i = $#ARRAY; $1 >= 0; $i--) {
    # Сделать что-то с $ARRAY[$1]
}
```

Комментарий

Настоящее обращение списка выполняется функцией цикл for просто перебирает элементы в обратном порядке. Если обращенная копия списка не нужна, цикл for экономит память и время.

Если функция ользуется для обращения только что отсортированного списка, логичнее будет сразу отсортировать список в нужном порядке. Например:

```
# Два шага: сортировка, затем обращение
@ascending = sort { $a cmp $b }@users;
@descending          @ascending;

# Один шаг: сортировка с обратным сравнением
@descending = sort { $b cmp $a }@users;
```

Смотри **также**

Описание *perlfunc(1).* Она используется в рецепте 1.6.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

4.11. Обработка нескольких элементов массива

Проблема

Требуется удалить сразу несколько элементов в начале или конце массива.

Решение

Воспользуйтесь функцией `splice`:

```
# Удалить $N элементов с начала @ARRAY (shift $N)
@FRONT = splice(@ARRAY,, 0, $N);

# Удалить $N элементов с конца массива (pop $N)
@END = splice(@ARRAY, -$N);
```

Комментарий

Часто бывает удобно оформить эти операции в виде функций:

```
sub shift2 (\@) {
          splice(@{$_[0]},
}

sub pop2 (\@) {
           splice(@{$_[0]},
}                    ^                                    "
```

Использование функций делает код более наглядным:

```
@friends = qw(Peter Paul Mary JimTim);
($this, $that) = shift2(@friends);
# $this содержит Peter, $that - Paul,
# a @friends - Mary, Jim и Tim

@beverages = qw(Dew Jolt Cola Sprite
@pair = pop2(@beverages);
# $pair[0] содержит $sprite, $pair[1] -
# a @beverages - (Dew, Jolt, Cola)
```

Функция splice возвращает элементы, удаленные из массива, поэтому `shift2` заменяет первые два элемента `@ARRAY` ничем (то есть удаляет их) и возвращает два удаленных элемента. Функция pop2 удаляет и возвращает два последних элемента.

В качестве аргументов этим функциям передается ссылка на массив — это сделано для того, чтобы они лучше имитировали встроенные функции `shift`и pop. При вызове ссылка не передается явно, с использованием символа \. Вместо этого компилятор, встречая прототип со ссылкой на массив, организует передачу массива по ссылке. Преимущества такого подхода — эффективность, наглядность и проверка параметров на стадии компиляции. Недостаток — передаваемый

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

объект должен выглядеть как настоящий массив с префиксом @, а не как скалярная величина, содержащая ссылку на массив. В противном случае придется добавлять префикс вручную, что сделает функцию менее наглядной:

```
$line[5] = \@list;
@got = pop2( @{ $line[5] > );
```

Перед вами еще один пример, когда вместо простого списка должен использоваться массив. Прототип \@ требует, чтобы объект, занимающий данную позицию в списке аргументов, был массивом. $line[5] представляет собой не массив, а ссылку на него. Вот почему нам понадобился "лишний" знак @.

Смотри также

Описание функции splice в ***perlfunc(1);*** раздел "Prototypes" perlsub(l). Функция splice используется **в** рецепте **4.9.**

4.12. Поиск первого элемента списка, удовлетворяющего некоторому критерию

Проблема

Требуется найти первый элемент списка, удовлетворяющего некоторому критерию (или индекс этого элемента). Возможна и другая формулировка — определить, проходит ли проверку хотя бы один элемент. Критерий может быть как простым ("Присутствует ли элемент в списке?")[1], так и сложным ("Имеется список ботников, отсортированный в порядке убывания оклада. У кого из менеджеров самый высокий оклад?"). В простых случаях дело обычно ограничивается значением элемента, но если сам массив может изменяться, вероятно, следует определять индекс первого подходящего элемента.

Решение

Перебирайте элементы в цикле и вызовите last, как только критерий будет выполнен:

```
my($match, $found, $item);
        $item(@array)
    if ($criterion) {
        $match = $item;     # Необходимо сохранить
        $found = 1;
        last;
    }
}
if($found) {
    ## Сделать что-то с $match
} else {
```

[1] Но тогда почему бы не воспользоваться хэшем?

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    "" Неудачный поиск
}
```

Чтобы определить индекс, перебирайте все индексы массива и вызывайте last, как только критерий выполнится:

```
my($i, $match_idx);
for ($i = 0; $i < @array; $i++) {
    if ($criterion) {
        $match_idx = $i;      # Сохранить индекс
        last;
    }
}
if(defined $match_idx) {
    ## Найден элемент $array[$match_idx]
} else {
    ## Неудачный поиск
}
```

Комментарий

Стандартных механизмов для решения этой задачи не существует, поэтому мы напишем собственный код для перебора и проверки каждого элемента. В нем используются циклы foreach и for, а вызов last прекращает проверку при выполнении условия. Но перед тем, как прерывать поиск с помощью last, следует сохранить найденный индекс.

Одна из распространенных ошибок — использование функции
том, что проверяет все элементы и находит все совпадения; если вас интересует только первое совпадение, этот вариант неэффективен.

Если нас интересует значение первого найденного элемента, присвойте его переменной $match. **Мы** не можем просто проверять $item в конце цикла, потому что автоматически **локализует**[1] переменную-итератор и потому не позволяет узнать ее последнее значение после завершения цикла (см. рецепт 4.4).

Рассмотрим пример. Предположим, в массиве @employees находится список объектов с информацией о работниках, отсортированный в порядке убывания оклада. Мы хотим найти инженера с максимальным окладом; это будет первый инженер в массиве. Требуется только вывести имя инженера, поэтому нас интересует не индекс, а значение элемента.

```
        $employee (@employees)
    if ( $employee->category() eq 'engineer' ) {
        $highest_engineer = $employee;
        last;
    }
}
print "Highest paid engineer is: ", $highest_engineer->name(), "\n";
```

[1] **Термин "локализация" по отношению к переменной означает придание ей локальной области действия. —** ***Примеч. перев.***

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если нас интересует лишь значение индекса, можно сократить программу — достаточно вспомнить, что при неудачном поиске $i будет содержать недопустимый индекс. В основном экономится объем кода, а не время выполнения, поскольку затраты на присваивание невелики по сравнению с затратами на проверку элементов списка. Однако проверка условия if ($i < @ARRAY) выглядит несколько туманно по сравнению с очевидной проверкой defined из приведенного выше решения.

```
for ($i = 0; $i < @ARRAY; $i++) {
    last if $criterion;
}
if ($i < @ARRAY) {
    ## Критерий выполняется по индексу $i
> else {
    ## Неудачный поиск

}
```

Смотри также

«For Loops" и "Loop Control" *perlsyn(1);* описание функции grep в *perlfunc*(1).

4.13. Поиск всех элементов массива, удовлетворяющих определенному критерию

Проблема

Требуется найти все элементы списка, удовлетворяющие определенному критерию.

Проблема извлечения подмножества из списка остается прежней. Вопрос заключается в том, как найти всех инженеров в списке работников, всех пользователей в административной группе, все интересующие вас имена файлов и т. д.

Решение

Воспользуйтесь функцией Функция применяет критерий ко всем элементам списка и возвращает лишь те, для которых он выполняется:

```
@РЕЗУЛЬТАТ          КРИТЕРИЙ ($_) } @СПИСОК;
```

Комментарий

То же самое можно было сделать в цикле

```
@РЕЗУЛЬТАТ = ()i
        (@СПИСОК)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    push(@РЕЗУЛЬТАТ, $_) if КРИТЕРИЙ ($_),
}
```

Функция позволяет записать всю эту возню с циклами более компактно. В действительности функция сильно одноименной команды UNIX — она не имеет параметров для нумерации строк или инвертирования критерия и не ограничивается проверками регулярных выражений. Например, чтобы отфильтровать из массива очень большие числа или определить, с какими ключами хэша ассоциированы очень большие значения, применяется следующая запись:

```
@bigs = grep { $_ > 1_000_000 } @nums,
@pigs        $users{$_} > 1e7 } keys %users,
```

В следующем примере в @matching заносятся строки, полученные от команды who и начинающиеся с "gnat ":

```
@matching        /^gnat / } 'who',
```

Или другой пример:

```
@engineers = grep { $_->position() eq 'Engineer' } @employees,
```

Из массива @employees извлекаются только те объекты, для которых метод position() возвращает строку Engineer.

Grep позволяет выполнять и более сложные проверки:

```
@secondary_assistance        $_->income >= 26_000 &&
                             f $_->income < 30_000 } г
                     @applicants,
```

Однако в таких ситуациях бывает разумнее написать цикл.

Смотри также

Разделы "For Loops", "Foreach Loops" и "Loop Control" *perlsyn(1);* описание функции *в perlfunc(1);* страница руководства *who*(1) вашей системы (если есть); рецепт 4.12.

4.14. Числовая сортировка массива

Проблема

Требуется отсортировать список чисел, однако функция Pert sort (по умолчанию) выполняет алфавитную сортировку в ASCII-порядке.

Решение

Воспользуйтесь функцией Perl sort с оператором числового сравнения, оператор <=>:

```
@Sorted = sort { $a <=> $b } @Unsorted,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

При вызове функции sort можно передавать необязательный программный блок, с помощью которого принятый по умолчанию алфавитный порядок сравнения заменяется вашим собственным. Функция сравнения вызывается каждый раз, когда sort сравнивает две величины. Сравниваемые значения загружаются в специальные пакетные переменные $a и $b, которые автоматически локализуются.

Функция сравнения должна возвращать отрицательное число, если значение $a должно находиться в выходных данных перед $b; 0, если они совпадают или порядок несущественен; и положительное число, если значение $a должно находиться после $b. В Perl существуют два оператора с таким поведением: оператор <=> сортирует числа по возрастанию в числовом порядке, а cmp сортирует строки по возрастанию в алфавитном порядке. По умолчанию sort использует сравнения в стиле cmp.

Следующий фрагмент сортирует список идентификаторов процессов (PID) в массиве @pids, предлагает пользователю выбрать один PID и посылает сигнал TERM, за которым следует сигнал KILL. В необязательном программном блоке $a сравнивается с $b оператором <=>, что обеспечивает числовую сортировку.

```
# @pids - несортированный массив идентификаторов процессов
foreach my $pid (sort { $a <=> $b }@pids) {
    print "$pid\n";
}
print "Select a process ID to kill:\n";
chomp ($pid = o);
die "Exiting ... \n" unless $pid && $pid =~ /^\d=$/;
kill ('TERM',$pid);
sleep 2;
kill ('KILL',$pid);
```

При использовании условия $a<=>$b или $a cmp $b список сортируется в порядке возрастания. Чтобы сортировка выполнялась в порядке убывания, достаточно поменять местами $a и $b в функции сравнения:

```
@descending = sort { $b <=> $a } @unsorted;
```

Функции сравнения должны быть последовательными; иначе говоря, функция всегда должна возвращать один и тот же ответ для одинаковых величин. Непоследовательные функции сравнения приводят к зацикливанию программы или ее аварийному завершению, особенно в старых версиях Perl.

Также возможна конструкция вида sort ИМЯ СПИСОК, где ИМЯ — имя функции сравнения, возвращающей -1, 0 или +1. В интересах быстродействия нормальные правила вызова не соблюдаются, а сравниваемые значения, как по волшебству, появляются в глобальных пакетных переменных $a и $b. Из-за особенностей вызова этрй функции в Perl рекурсия в ней может не работать.

Предупреждение: значения $a и $b задаются в пакете, активном в момент **вызова** sort, — **а он может не** совпадать с пакетом, **в** котором была откомпилирована передаваемая sort функция сравнения (ИМЯ)! Например:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
package Sort_Subs;
sub revnum { $b <=> $a }

package Other_Pack;
@all = sort Sort_Subs::revnum 4, 19, 8, 3;
```

Такая попытка тихо заканчивается неудачей — впрочем, при наличии ключа *-w* о неудаче будет заявлено вслух. Дело в том, что вызов sort создает пакетные переменные $a и $b в своем собственном пакете, Other_Pack, а функция будет использовать версии из своего пакета. Это еще один аргумент в пользу встроенных функций сортировки:

```
@all = sort { $b <=> $a } 4, 19, 8, 3;
```

За дополнительной информацией о пакетах обращайтесь к главе **10** "Подпрограммы".

Смотри также

Описание операторов cmp и <=> *в perlop(1);* описание функций kill, sort и sleep *в perlfunc(1);* рецепт **4.15.**

4.15. Сортировка списка по вычисляемому полю

Проблема

Требуется отсортировать **список,** руководствуясь более сложным критерием, нежели простыми строковыми или числовыми сравнениям.

Такая проблема часто встречается при работе с объектами или сложными структурами данных ("отсортировать по третьему элементу массива, на который указывает данная **ссылка»).** Кроме того, она относится к сортировке по нескольким ключам — например, когда список сортируется по дню рождения, а затем по имени (когда у нескольких людей совпадают дни рождения).

Решение

Воспользуйтесь нестандартной функцией сравнения в sort:

```
@ordered = sort { compare() } @unordered;
```

Для ускорения работы значение поля можно вычислить заранее:

```
                    [compute(),$_]  @unordered;
@ordered_precomputed = sort { $a->[0] <=> $b->[0] } @precomputed;
@ordered = map { $_->[1] } @ordered_precomputed;
```

Наконец, эти три шага можно объединить:

```
@ordered = map { $_->[1] }
           sort { $a->[0] <=> $b->[0] }
           map { [compute(),$_] }
           @unordered;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

О том, как пользоваться функциями сравнения, рассказано в рецепте 4.14. Помимо использования встроенных операторов вроде <=>, можно конструировать более сложные условия:

```
@ordered = sort { $a->name cmp $b->name } @employees;
```

Функция sort часто используется подобным образом в циклах foreach:

```
        $employee (sort {$a->name cmp $b->name } @employees) {
    print $employee->name, " earns \$", $employee->salary, "\n";
}
```

Если вы собираетесь много работать с элементами, расположенными в определенном порядке, эффективнее будет сразу отсортировать их и работать с отсортированным списком:

```
@sorted_employees = sort { $a->name cmp $b->name } @employees;
        $employee (@sorted_employees) <
    print $employee->name, "earns \$", $employee->salary, "\n";
}
# Загрузить %bonus
        $employee (@sorted_employees)
    if ($bonus{ $employee->ssn > ) {
        print $employee->name, "got a bonus!\n";
    }
}
```

В функцию можно включить несколько условий и разделить их операторами ||.Оператор || возвращаст первое истинное (ненулевое) значение. Следовательно, сортировку можно выполнять по одному критерию, а при равенстве элементов (когда возвращаемое значение равно 0) сортировать по другому критерию. Получается "сортировка внутри сортировки":

```
@sorted = sort {$a->name cmp $b->name
                          ||
                $b->age  <=> $a->age} @employees;
```

Первый критерий сравнивает имена двух работников. Если они не совпадают, || прекращаетвычисления и возвращает результат cmp (сортировка в порядке возрастания имен). Но если имена совпадают, || продолжает проверку и возвращает результат <=> (сортировка в порядке убывания возраста). Полученный список будет отсортирован по именам и по возрасту в группах с одинаковыми именами.

Давайте рассмотрим реальный пример сортировки. Мы собираем информацию обо всех пользователям в виде объектовUser::pwent, после чего сортируем их по именам и выводим отсортированный список:

```
use User::pwent qw(getpwent);
@users = ();
9 Выбрать всех пользователей
while (defined($user = getpwent)) {
    push(@users, $user);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}
@users = sort { $a->name cmp $b-<name } @users,
        $user (@users) {
    print $user->name,  \n ,
}
```

Возможности не ограничиваются простыми сравнениями или комбинациями простых сравнений. В следующем примере список имен сортируется по ***второй*** букве имени. Вторая буква извлекается функцией substr:

```
@sorted = sort { substr($a,1,1) cmp substr($b,1,1) } @names,
```

А ниже список сортируется по длине строки:

```
@sorted = sort { length $a <=> length $b } @strings,
```

Функция сравнения вызывается sort каждый раз, когда требуется сравнить два элемента. Число сравнений заметно увеличивается с количеством сортируемых элементов. Сортировка **10** элементов требует (в среднем) **46** сравнений, однако при сортировке **1000** элементов выполняется **14000** сравнений. Медленные операции (например, `split` или вызов подпрограммы) при каждом сравнении тормозят работу программы.

К счастью, проблема решается однократным выполнением операции для каждого элемента перед сортировкой. Воспользуйтесь `map` для сохранения результатов операции в массиве, элементы которого являются анонимными массивами с исходным и вычисленным полем. Этот "массив массивов" сортируется по предварительно вычисленному полю, после чего map используется для получения отсортированных исходных данных. Концепция `map/sort/map` применяется часто и с пользой, поэтому ее стоит рассмотреть более подробно.

Применим ее к примеру с сортировкой по длине строки:

```
@temp = map { [ length $_, $_ ] } @strings,
@temp = sort { $a->[0] <=> $b->[0] } @temp,
@sorted = map { $_->[1] ) @temp,
```

В первой строке map создает временный массив строк с их длинами. Вторая строка сортирует временный массив, сравнивая их предварительно вычисленные длины. Третья строка превращает временный массив строк/длин в отсортированный массив строк. Таким образом, длина каждой строки вычисляется всего один раз.

Поскольку входные данные каждой строки представляют собой выходные данные предыдущей строки (массив `@temp`, созданный в строке 1, передается sort в строке 2, а результат сортировки передается map в строке 3), их можно объединить в одну команду и отказаться от временного массива:

```
@sorted = map { $_->[1] }
          sort {$a->[0] <=> $b->[0] }
          map { [ length $_, $_] }
          @strings,
```

Теперь операции перечисляются вобратном порядке. Встречая конструкцию `map/sort/map`, читайте ее снизу вверх:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

@strings: в конце указываются сортируемые данные. В данном случае это массив, но как вы вскоре убедитесь, это может быть вызов подпрограммы или даже команда в обратных апострофах. Подходит все, что возвращает список для последующей сортировки.

map: нижний вызов map строит временный список анонимных массивов. Список содержит пары **из** предварительно вычисленного поля (length $_) и исходного элемента ($_). В этой строке показано, как происходит вычисление поля.

sort: список анонимных массивов сортируется посредством сравнения предварительно вычисленных полей. По этой строке трудно о чем-то судить — разве что о том, будет ли список отсортирован в порядке возрастания или убывания.

map: вызов map в начале команды превращает отсортированный список анонимных массивов в список исходных отсортированных элементов. Как правило, во всех конструкциях map/sort/map он выглядит одинаково.

Ниже показан более сложный пример, в котором сортировка выполняется по первому **числу,** найденному в каждой строке @fields:

```
@temp = map { [ /(\d+)/, $_ ] } @fields;
@sorted_temp = sort {$a->[0] <=> $b->[0] } @temp;
@sorted_fields = map { $_->[1] } @sorted_temp;
```

Регулярное выражение в первой строке извлекает из строки, обрабатываемой map, первое число. Мы используем регулярное выражение /(\d+)/ в списковом контексте.

Из этого фрагмента можно убрать временный массив. Код принимает следующий **вид:**

```
@sorted_fields = map  { $_->[1] }
                 sort { $a->[0] <=> $b->[0] }
                 map  { [ /(\d+)/, $_ ] }
                 @fields;
```

В последнем примере выполняется компактная сортировка данных, разделенных запятыми (они взяты из файла UNIX*passwd*). Сначала выполняется числовая сортировка файла по четвертому полю (идентификатору группы), затем — числовая сортировка по третьему полю (идентификатору пользователя) **и алфавитная** сортировка по первому полю (имени пользователя).

```
print map  { $_->[0] •}              # Целая строка
      sort {
              $a->[1] <=> $b->[1]    # Идентификатор группы
                      ||
              $a->[2] <=> $b->[2]   8 Идентификатор пользователя
                      ||
              $a->[3] <=> $b->[3]   # Имя пользователя
      >
      map { [ $_, (split/:/)[3,2,0] ] }
       'cat /etc/passwd';
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Компактная конструкция map/sort/map больше напоминает программирование на Lisp и Scheme, нежели обычное наследие Perl — C и awk. Впервые она была предложена Рэндалом Шварцем (Randal Schwartz) и потому часто называется "преобразованием Шварца".

Смотри также

Описание функции sort в *perlfunc(1);* описание операторов cmp и <=> в *perlop(1);* рецепт **4.14.**

4.16. Реализация циклических списков

Проблема

Требуется создать циклический список и организовать работу с ним.

Решение

Воспользуйтесь функциями unshift и pop (или push и shift) для обычного массива.

```
unshift(@circular, pop(@circular));  # Последний становится первым
push (@circular, shift(@circular));  # И наоборот
```

Комментарий

Циклические списки обычно применяются для многократного выполнения одной и той же последовательности действий — например, обработки подключений к серверу. Приведенный выше фрагмент не является полноценной компьютерной реализацией циклических списков с указателями и настоящей цикличностью. Вместо этого мы просто перемещаем последний элемент на первую позицию, и наоборот.

```
sub grab_and_rotate (\@ ) {
                = shift;
        $element
    push(@listref, shift @$listref);
            $element;
}

@processes = ( 1, 2, 3, 4, 5 );
while (1) {
    $process = grab_and_rotate(@processes);
    print "Handling process $process\n";
    sleep 1;
}
```

Смотри также

Описание функций unshift и push *в perlfunc(1);* рецепт **13.13.**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

4.17. Случайная перестановка элементов массива

Проблема

Требуется случайным образом переставить элементы массива. Наиболее очевидное применение — тасование колоды в карточной игре, однако аналогичная задача возникает в любой ситуации, где элементы массива обрабатываются в произвольном порядке.

Решение

Каждый элемент массива меняется местом с другим, случайно выбранным элементом:

```
# fisher_yates_shuffle ( \@array ) : генерация случайной перестановки
# массива @array на месте
sub fisher_yates_shuffle {
    my $array = shift;
    my $i;
    for ($i = @$array; --$i; ) {
        my $j = int rand ($i+1);
        next if $i == $j;
        @$array[$i,$j] = @$array[$j,$i];
    }
}

fisher_yates_shuffle( \@array ); # Перестановка массива @array на месте
```

Или выберите случайную перестановку, воспользовавшись кодом из примера 4.4:

```
$permutations = factorial( scalar @array );
@shuffle = @array [ n2perm( 1+int(rand $permutations), $#array) ];
```

Комментарий

Случайные перестановки на удивление коварны. Написать плохую программу перестановки очень просто:

```
sub naive_shuffle {                          # Не делайте так!
    for (my $i = 0; $i < @_; $i++) {
        my $j = int rand @_;                 # Выбрать случайный элемент
        ($_[$i], $_[$j]) = ($_[$j], $_[$i]); # Поменять местами
    }
}
```

Такой алгоритм является смещенным — одни перестановки имеют большую вероятность, чем другие. Это нетрудно доказать: предположим, мы получили список из 3 элементов. Мы генерируем 3 случайных числа, каждое из которых может принимать 3 возможных значения — итого 27 возможных комбинаций. Однако для списка из трех элементов существует всего 6 перестановок. Поскольку 27 не делится на 6, некоторые перестановки появляются с большей вероятностью, чем другие.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В приведенном выше алгоритме Фишера—Йетса это смещение устраняется за счет изменения интервала выбираемых случайных чисел.

Смотри также

Описание функции rand *в perlfunc(1)*. Дополнительная информация о случайных числах приведена в Рецептах 2.7—2.9. В рецепте 4.19 показан другой способ построения случайных перестановок.

4.18. Программа: words

Описание

Вас когда-нибудь интересовало, каким образом программы типа /s строят столбцы отсортированных выходных данных, расположенных по столбцам, а не по строкам? Например:

```
awk       cp        ed        login     mount     rmdir     sum
basename                                mt
cat       date                          mv
chgrp     dd        grep      mkdir     ps        sort      touch
chmod     df        kill      mknod     pwd       stty      vi
chown     echo      ln        more      rm        su
```

В примере 4.2 показано, как это делается.

Пример 4.2. words

```
#!/usr/bin/perl -w
# words - вывод данных по столбцам

use strict;

my ($item, $cols, $rows, $maxlen);
my ($xpixel, $ypixel, $mask, @data);

getwinsize();

# Получить все строки входных данных
# и запомнить максимальную длину строки
$maxlen = 1;
while (<>) {
    my $mylen;
    s/\s+$//;
    $maxlen = $mylen if (($mylen = length) > $maxlen);
    push(@data, $_);
}

$maxlen += 1;                # Дополнительный пробел

# Определить границы экрана
$cols = int($cols / $maxlen) || 1;
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 4.2. (продолжение)

```
$rows = int(($#data+$cols) / $cols);

# Задать маску для ускорения вычислений
$mask = sprintf("%%-%ds ", $maxlen-1);

# Подпрограмма для обнаружения последнего элемента строки
sub EOL {  ($item+1)X $cols == 0 }

tt Обработать каждый элемент, выбирая нужный фрагмент
# на основании позиции
for ($item= 0; $item< $rows * $cbls; $item++) {
    my $target = ($item % $cols) • $rows + int($item/$cols);
    my.$piece = sprintf($mask, $target < @data ? $data[$target] : "")
    $piece =~ s/\s+$// if EOL(); # Последний элемент не выравнивать
    print $piece;
    print "\n" if EOL();
>

# Завершить при необходимости
print "\n" if EOL();

# Не переносится -- только для Linux
sub getwinsize {
    my $winsize = "\0" x 8;
    my $TIOCGWINSZ = 0x40087468;
    if (ioctl(STDOUT, $TIOCGWINSZ, $winsize)) {
        ($rows, $cols, $xpixel, $ypixel) = unpack('S4', $winsize);
    } else {
        $cols = 80;
    }
}
```

Наиболее очевидный способ вывести отсортированный список по столбцам — последовательно выводить каждый элемент списка, выравнивая его пробелами до определенной ширины. Когда вывод достигает конца строки, происходит переход на следующую строку. Но такой вариант хорош лишь тогда, когда строки читаются слева направо. Если данные должны читаться по столбцам, сверху вниз, приходится искать другое решение.

Программа *words* представляет собой фильтр, который генерирует выходные данные по столбцам. Она читает все входные данные и запоминает максимальную длину строки. После того как все данные будут прочитаны, ширина экрана делится на длину самой большой входной записи — результат равен ожидаемому количеству столбцов.

Затем программа входит в цикл, который выполняется для каждой входной записи. Однако порядок вывода неочевиден. Предположим, имеется список из девяти элементов:

Неправильно		Правильно
1 2 3		1 4 7
4 5 6	,	2 5 8
7 8 9		3 6 9

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Программа words производит все необходимые вычисления, чтобы элементы (1,4,7) выводились в одной строке, (2,5,8) — в другой и (3,6,9) — в последней строке.

Текущие размеры окна определяются вызовом ioctl. Этот вариант прекрасно работает — в той системе, для которой он был написан. В любой другой он не подойдет. Если вас это устраивает, хорошо. В рецепте 12.14 показано, как определить размер окна в вашей системе с помощью файла ioctl.pch или программы на C. Решение из рецепта 15.4 отличается большей переносимостью, однако вам придется установить модуль с CPAN.

Смотри также

Рецепт 15.4.

4.19. Программа: permute

Проблема

Вам никогда не требовалось сгенерировать все возможные перестановки массива или выполнить некоторый фрагмент для всех возможных перестановок? Например:

```
% echo man bites dog | permute
dog bites man
bites dog man
dog man bites
man dog bites
bites man dog
man bites dog
```

Количество возможных перестановок для множества равно факториалу числа элементов в этом множестве. Оно растет чрезвычайно быстро, поэтому не стоит генерировать перестановки для большого числа элементов:

Размер множества	Количество перестановок
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Соответственно, выполнение операции для всех возможных перестановок занимает много времени. Сложность факториальных алгоритмов превышает количество частиц во Вселенной даже для относительно небольших входных значений. Факториал 500 больше, чем десять в *тысячной* степени!

```
use Math::BigInt;
    sub factorial {
    my $n = shift;
    my $s = 1;
    $s *= $n-while $n > 0;

}
print factorial(Math::BigInt->new("500"));
+1220136...(1035 digits total)
```

Два решения, приведенных ниже, отличаются порядком возвращаемых перестановок.

Решение из примера 4.3 использует классический алгоритм списковых перестановок, используемый знатоками Lisp. Алгоритм относительно прямолинеен, однако в нем создаются ненужные копии. Кроме того, в решении жестко закодирован простой вывод перестановок без каких-либо дополнительных действий.

Пример 4.3. tdc-permute

```
#!/usr/bin/perl -n
# tsc_permute: вывод всех перестановок введенных слов
permute([split], []);
sub permute {
    my @items = @{ $_[0] };
    my @perms = @{ $_[1] };
    unless (@items) {
        print "@perms\n";
    } else {
        my(@newitems,@newperms,$i);
                    .. $#items)
            @newitems = @items;
            @newperms = @perms;
            unshift(@newperms, splice(@newitems, $i, 1));
            permute([@newitems], [@newperms]);
        }
    }
}
```

Решение из примера 4.4, предложенное Марком-Джейсоном Доминусом (Mark-Jason Dominus), более элегантно и работает примерно на 25 % быстрее. Вместо того чтобы рассчитывать все перестановки, программа генерирует n-ю конкретную перестановку. Элегантность проявляется в двух аспектах. Во-первых, в программе удается избежать рекурсии, кроме как при вычислении факториала (который алгоритмом перестановок обычно не используется). Во-вторых, вместо перестановки реальных данных генерируется перестановка целых чисел.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В программе для экономии времени использована методика ***запоминания.*** Ее суть заключается в том, что функция, которая всегда возвращает конкретный ответ для конкретного набора аргументов, запоминает этот ответ. При следующем вызове с теми же аргументами дальнейшие вычисления уже не потребуются. Функция factorial сохраняет ранее вычисленные значения факториала в закрытом массиве @fact (10.3).

Функция n2perm вызывается с двумя аргументами: номером генерируемой перестановки (от 0 до N!, где N — размер массива) и индексом последнего элемента массива. Функция n2perm для расчета шаблона перестановки вызывает подпрограмму n2pat. Затем шаблон преобразуется в перестановку целых чисел подпрограммой pat2perm. Шаблон представляет собой список вида (0 2 0 1 0), что означает: "Вырезать нулевой элемент, затем второй элемент оставшегося списка, затем нулевой, первый и снова нулевой".

Пример 4.4. mjd-permute

```
#!/usr/bin/perl -w
# mjd_permute: перестановка всех введенных слов
use strict;

while (o) {
    my @data = split;
    my $num_permutations = factorial(scalar @data);
    for (my $i=O; $1 < $num_permutations; $1++) <
        my @permutation = @data[n2perm($i, $#data)];
        print "@permutation\n";
    >
}

# Вспомогательная функция: факториал с запоминанием
BEGIN {
  my @fact = (1);
  sub factorial($) {
      my $n = shift;
              $fact[$n] if defined $fact[$n];
      $fact[$n] = $n * factorial($n - 1);
  }
}

# n2pat($N, $len) : построить $N-й шаблон перестановки длины $len
sub n2pat {
    my $i   = 1;
    my $N   = shift;
    my $len = shift;
    my @pat;
    while ($i <= $len + 1) {   # На самом деле просто while ($N) { ...
        push @pat, $N % $i;
        $N = int($N/$i);
        $1++;
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 4.4 (продолжение)

```
    }
            @pat;
}

# pat2perm(@pat) : превратить шаблон, возвращаемый n2pat(),
# в перестановку целых чисел.
sub pat2perm {
    my @pat    = @_;
    my @source = (0 .. $#pat);
    my @perm;
    push @perm, splice(@source, (pop @pat), 1) while @pat;
            @perm;
}

# n2perm($N, $len) : сгенерировать N-ю перестановку S объектов
sub n2perm {
    pat2perm(n2pat(@_));
}
```

Смотри также

Описание функций unshift и splice в *perlfunc(1);* рецепты 2.7; 10.3.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

5 Хэши

Выполнять линейный просмотр в ассоциативном массиве — все равно что пытаться забить кого-нибудь до смерти заряженным «Узи».

Ларри Уолл

Введение

Как люди, так и части компьютерных программ взаимодействуют между собой самым причудливым образом. Отдельные скалярные переменные похожи на отшельников, ведущих замкнутое существование в рамках собственной личности. Массив напоминает партию, где множество индивидуумов объединяется под именем харизматического предводителя. Где-то между ними расположилась удобная ниша, в которой живут совокупности связей "один-к-одному" — хэши. В старой документации по Perl хэши часто назывались *ассоциативными массивами,* но термин получается слишком длинным. Аналогичные структуры данных существуют и в других языках, где они обозначаются другими терминами — *хэш-таблицы, таблицы, словари, отображения* и даже *а-списки,* в зависимости от языка.

К сожалению, отношения хэшей являются не равными, а подчиненными — например, "Энди — начальник Ната"; "Кровяноедавление пациента — 112/62" или "Название журнала с индексом ISSN 1087-903X— The Perl Journal».Хэш всего лишь предоставляет удобные средства для получения ответов на вопросы типа: "Кто является начальником Ната?" или "Как называется журнал 1087-903X"? Вы не сможете спросить "Чьим начальником является Энди?" Впрочем, поиску ответов на подобные вопросы посвящен один из рецептов этой главы.

Однако у хэшей есть свои преимущества. В Perl хэш является встроенным типом данных. Благодаря применению хэшей многие сложные алгоритмы сводятся к простой выборке значений. Кроме того, хэши предоставляют быстрые и удобные средства для построения индексов и таблиц просмотра. Если для простой скалярной переменной применяется идентификатор типа $, а для массива — @, то для хэшей используется идентификатор %.

Префикс % относится лишь к ссылкам на хэш в целом. Значение ключа представляет собой скалярную величину, поэтому для него используется символ $ (по

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

аналогии с тем, как для ссылок на отдельный элемент массива используется префикс $). Следовательно, отношение "начальник Ната" должно записываться в виде $boss{"Nat"}.

В обычных массивах используются числовые индексы, но индексы хэшей всегда являются строковыми. Ассоциированные значения могут быть произвольными скалярными величинами, в том числе ссылками. Используя ссылки в качестве ассоциированных значений, можно создавать хэши для хранения не только строк и чисел, но и массивов, других хэшей или объектов (вернее, ссылок на массивы, хэши или объекты).

Хэши могут инициализироваться с помощью списков, содержащих пары "ключ/значение":

```
%age = ( "Nat",   24,
         "Jules", 25,
         "Josh",  17 );
```

Такая запись эквивалентна следующей:

```
$age{"Nat"}   = 24;
$age{"Jules"} = 25,
$age{"Josh"}  = 17;
```

Для упрощения инициализации хэшей был создан оператор, оператор =>. В основном он представляет собой более наглядную замену для запятой. Например, возможна следующая инициализация хэша:

```
%food_color = (
                "Apple"  => "red",
                "Banana" => "yellow",
                "Lemon"  => "yellow",
                "Carrot" => "orange"
               );
```

(хэш %food_color используется во многих примерах этой главы). Такая инициализация также является примером *списковой эквивалентности* — в некоторых отношениях хэш ведет себя так, словно он является списком пар «ключ/значение». Мы воспользуемся этим в нескольких рецептах, в частности — для объединения и инвертирования.

В отличие от обычной запятой, оператор => обладает особым свойством: любое предшествующее ему слово интерпретируется как строковое значение. Это позволяет убрать кавычки и сделать программу более понятной. Однословные ключи хэшей также автоматически интерпретируются как строки, поэтому вместо $hash{"somekey"} можно написать просто $hash{somekey}. Приведенная выше инициализация %food_color записывается в следующем виде:

```
%food_color = (
                Apple  => "red",
                Banana => "yellow",
                Lemon  => "yellow",
                Carrot => "orange"
               );
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Одно из важных свойств хэшей заключается в том, что их элементы хранятся в особой последовательности, обеспечивающей выборку. Следовательно, независимо от порядка занесения данных в хэш, порядок их хранения будет непредсказуемым.

Смотри также

Описание функций unshift и splice в perlfunc(1).

5.1. Занесение элемента в хэш

Проблема

Требуется добавить в хэш новый элемент.

Решение

Присвойте нужное значение в записи вида:

```
$ХЭШ{$КЛЮЧ} = $ЗНАЧЕНИЕ;
```

Комментарий

Процесс занесения данных в хэш весьма тривиален. В языках, где хэш не относится к встроенным типам данных, приходится беспокоиться о переполнении, изменении размеров и коллизиях в хэш-таблицах. В Perl обычное присваивание решает сразу все проблемы. Если ключ уже занят, то есть содержит предыдущее значение, память автоматически освобождается (но аналогии с присваиванием скалярной переменной).

```
# Хэш %food_color определяется во введении
$food_color{Raspberry} = "pink";
print "Known foods:\n";
foreach $food (keys %food_color) {
    print "$food\n";
}
```

```
Known foods:
Banana
Apple
Raspberry
Carrot
Lemon
```

Если в качестве ключа хэша используется неопределенная величина undef, она преобразуется в пустую строку "" (что сопровождается предупреждением при запуске с параметром -w). Вероятно, неопределенный ключ undef — это не то, что вы хотели. С другой стороны, undef является вполне допустимым *значением в* хэшах. Однако при выборке значения для ключа, отсутствующего в хэше, вы также получите undef. Это означает, что для проверки существования ключа $key в хэше %hash простая логическая проверка if ($hash{$key}) не подходит. Присутствие ключа в хэше проверяется записью вида exists($hash{$key}); определенность ассоциированного значения — defined($hash{$key}), а его истинность — if ($hash{$key}).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Во внутренних алгоритмах хэширования Perl перестановки строки попадают на одну и ту же позицию. Если в ключах хэша многократно встречаются перестановки одной строки (скажем, "sparc" и "craps"), быстродействие хэша заметно падает. На практике это происходит редко.

Смотри также

Раздел "List Value Constructors" perldata(1); рецепт 5.2.

5.2. Проверка наличия ключа в хэше

Проблема

Требуется узнать, содержит ли хэш конкретный ключ независимо от ассоциированного с ним значения.

Решение

Воспользуйтесь функцией exists:

```
# Содержит ли %ХЭШ ключ $КЛЮЧ?
if (exists($ХЭШ{$КЛЮЧ})) {
    # Ключ существует
} else {
    " Ключ не существует
}
```

Комментарий

В следующем фрагменте функция exists проверяет, присутствует ли ключ в хэше %food_color:

```
" Хэш %food_color определяется во введении
foreach $name ("Banana", "Martini") {
    if (exists $food_color{$name}) {
        print "$name is a food.\n";
    } else {
        print "$name is a drink.\n";
    }
}

Banana is a food.
Hartinl is a drink.
```

Функция exists проверяет только наличие ключа в хэше. Она не сообщает об ассоциированном значении, определено ли оно, истинно или ложно. На первый взгляд кажется, что отличия несущественны. Однако в действительности проблемы такого рода плодятся быстро, как кролики. Возьмем следующий фрагмент:

```
%age = ();
$age{"Toddler"} = 3;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$age{"Unborn"} = 0;
$age{"Phantasm"} = undef;

foreach $thing ("Toddler", "Unborn", "Phantasm", "Relic") {
    print "$thing: ";
    print "Exists " if exists $age{$thing};
    print "Defined "if defined $age{thing};
    print "True " if $age{$thing};
    print "\n";
}
```

```
Toddler:  Exists Defined  True
Unborn:  Exists Defined
Phantasm:  Exists
Relic:
```

Элемент `$age{"Toddler"}` проходит все три проверки — существования, определенности и истинности. Он существует, потому что мы присвоили ключу "Toddler" значение в хэше. Он определен, потому что значение не равно undef. Наконец, он истинен, потому что присвоенная величина не является одним из ложных значений Perl.

Элемент `$age{"Unborn"}` проходит только проверки существования и определенности. Он существует, потому что ключу "Unborn" было присвоено значение в хэше, и определен, потому что это значение не равно undef. Однако он не является *истинным*, потому что 0 интерпретируется в Perl как одна из ложных величин.

Элемент `$age{"Phantasm"}` проходит только проверку существования. Он существует, потому что ключу "Phantasm" было присвоено значение в хэше. Поскольку это значение представляет собой undef, проверка определенности не работает. Так как undef также считается в Perl одним из ложных значений, проверка истинности тоже не работает.

Наконец, `$age{"Relic"}` не проходит ни одну из проверок. Значение для "Relic" не заносилось в хэш, поэтому проверка на существование завершается неудачей. Из-за отсутствия ассоциированного значения попытка обратиться к `$age{"Relic"}` дает undef. Как мы знаем из примера с "Phantasm", undef не проходит проверки определенности и истинности.

Иногда undef полезно сохранить в хэше. Это означает: "такой ключ встречается, но с ним не связано никакого полезного значения". Например, рассмотрим программу, которая определяет размер файлов из переданного списка. Следующий фрагмент пытается пропускать файлы, которые уже встречались в списке, однако это не касается файлов нулевой длины и встречавшихся ранее несуществующих файлов:

```
%name = ();
while (<>) {
    chomp;
    next if $name{$_};     # НЕВЕРНО !
    $name{$_} = -s $ ;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Замена неправильной строки следующим вызовом exists позволяет пропускать нулевые и несуществующие файлы:

```
next if exists $name{$_},
```

В самом первом примере предполагается, что все, что не является едой (food), относится к напиткам ***(drink)***. В реальном мире подобные допущения весьма опасны.

Смотри также

Описание функций exists и defined в ***perlfunc(1)***. Концепция истинности рассматривается в разделе "Scalar Values" ***perldata(1)***.

5.3. Удаление из хэша

Проблема

Требуется удалить элемент из хэша, чтобы он не опознавался функцией keys, values или each. Например, если в хэше имена работников ассоциируются с окладами, после увольнения работника необходимо удалить сто строку из хэша.

Решение

Воспользуйтесь функцией delete:

```
# Удалить $КЛЮЧ и ассоциированное значение из хэша %ХЭШ
delete($ХЭШ{$КЛЮЧ});
```

Комментарий

Многие ошибочно пытаются удалять элементы из хэша с помощью undef — undef ${ХЭШ{$КЛЮЧ} или $ХЭШ{$КЛЮЧ} = undef. В обоих случаях в хэше будет присутствовать элемент с ключом $КЛЮЧ и значением undef.

Функция delete — единственное средство для удаления конкретных элементов из хэша. Удаленный элемент не появится ни в списке keys, ни в итерациях each; функция exists возвращает для него л ожное значение.

Следующий фрагмент демонстрирует отличия undef от delete:

```
# Хэш %food_color определяется во введении
sub print_foods {
    my @foods = keys %food_color;
    my $food;

    print "Keys: @foods\n",
    print "Values: ";

    foreach $food (@foods) {
        my $color = $food_color{$food},

        if (defined $color) {
            print "$color ",
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        } else {
            print "(undef) ";
        }
    }
    print "\n";
}

print "Initially:\n";
print_foods();

print "\nWith Banana undef\n";
undef $food_color{"Banana"};
print_foods();

print "\nWith Banana deleted\n";
delete $food_color{"Banana"},
print_foods();
```

```
Initially:
Keys: Banana Apple Carrot Lemon
Values: yellow red orange yellow

With Banana undef
Keys: Banana Apple Carrot Lemon
Values: (undef) red orange yellow

With Banana deleted
Keys: Apple Carrot Lemon
Values: red orange yellow
```

Как видите, после присвоения `$food_color{"Banana"} = undef` ключ "Banana" остается в хэше. Элемент не удаляется; Просто мы присвоили ему undef. С другой стороны, функция delete действительно удалила данные из хэша — ключ "Banana" исчезает из списка, возвращаемого функцией keys.

Функция delete также может вызываться для среза хэша, это приводит к удалению всех указанных ключей:

```
delete @food_color{"Banana", "Apple", "Cabbage"},
```

Смотри также

Описание функций delete и keys *в perlfunc(1).* Применение keys продемонстрировано в рецепте 5.4.

5.4. Перебор хэша

Проблема

Требуется выполнить некоторые действия с каждым элементом (то есть парой "ключ/значение") хэша.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь функцией each в цикле while:

```
while(($КЛЮЧ, $ЗНАЧЕНИЕ) = each(%ХЭШ)) {
    # Сделать что-то с $КЛЮЧ и $ЗНАЧЕНИЕ
}
```

Если хэш не очень велик, можно вызвать keys в цикле foreach:

```
        $КЛЮЧ (keys %ХЭШ) {
    $ЗНАЧЕНИЕ = $ХЭШ{$КЛЮЧ};
    # Сделать что-то с $КЛЮЧ и $ЗНАЧЕНИЕ
}
```

Комментарий

Следующий простой пример перебирает элементы хэша %food_color из введения:

```
# Хэш %food_color определяется во введении
while(($food, $color) = each(%food_color)) {
    print "$food is $color.\n";
}
Banana is yellow.
Apple is red.
Carrot is orange.
Lemon is yellow.
```

В примере с можно обойтись без переменной $color, поскольку она используется всего один раз. Достаточно написать:

```
print "$food is $food_color{$food}.\n".
```

При каждом вызове each для одного и того же **хэша** функция возвращает "следующую" пару ключ/значение. Слово "следующую" взято **в** кавычки, потому что пары возвращаются **в** порядке, соответствующем внутренней структуре хэша, и этот порядок почти никогда не совпадает с числовым **или** алфавитным. За последним элементом each возвращает пустой список (); результат интерпретируется как ложный, и цикл while завершается.

В примере с использована функция keys, которая строит список всех ключей из хэша еще перед началом выполнения цикла. Преимущество each заключается в том, что пары "ключ/значение" извлекаются по одной. Если хэш содержит много ключей, отказ от предварительного построения полного списка существенно экономит память и время. Однако функция each не позволяет управлять порядком обработки пар.

Применение keys для перебора списка позволяет установить свой порядок обработки. Предположим, нам понадобилось вывести содержимое хэша в алфавитном порядке ключей:

```
        $food (sort keys %food_color) {
    print "$food is $food_color{$food}.\n";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
Apple is red.
Banana is yellow.
Carrot is orange.
Lemon is yellow.
```

Подобное применение foreach встречается довольно часто. Функция keys строит список ключей в хэше, после чего перебирает их. Если хэш состоит из большого числа элементов, возникает опасность, что возвращаемый keys список займет много памяти. Приходится выбирать между затратами памяти и возможностью обработки элементов в определенном порядке. Сортировка подробнее рассматривается в рецепте 5.9.

Поскольку функции keys, values и each используют одни и те же внутренние структуры данных, следует внимательно следить за чередованием вызовов этих функций или преждевременным выходом из цикла each. При каждом вызове keys или values текущая позиция each сбрасывается. Следующий фрагмент зацикливается и бесконечно выводит первый ключ, возвращаемый each:

```
while ( ($k,$v) = each %food_color) {
    print "Processing $k\n";
    keys %food_color;          # Возврат к началу %food_color
}
```

Модификация хэша во время его перебора в each или foreach, как правило, сопряжена с опасностью. При добавлении или удалении ключей из хэша функция each ведет себя по-разному для связанных и несвязанных хэшей. Цикл перебирает заранее построенный список ключей, поэтому после начала цикла он ничего не знает о добавленных или удаленных ключах. Ключи, добавленные внутри цикла, не включаются автоматически в список перебираемых ключей, а удаленные внутри цикла ключи не удаляются из этого списка.

Программа *countfrom* из примера 5.1 читает файл почтового ящика и выводит количество сообщений от каждого отправителя. Отправитель определяется по строке From: (в этом отношении сценарий не очень интеллектуален, однако нас сейчас интересуют операции с хэшами, а не обработка почтовых файлов). Передайте имя почтового ящика в командной строке или используйте "-" для перенаправления.

Пример 5.1. countfrom

```
#!/usr/bin/perl
Я countfrom - подсчет сообщений от каждого отправителя

$filename = $ARGV[0] || "-";

open(FILE, "<$filename")        or die "Can't open $filename . $!";

while(<FILE>) {
    if (/^From: (.*)/) { $from{$1}++ }
}

    $person (sort keys %from) {
    print "$person: $from{$person}\n";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

> Смотри также

Описание функций each и keys *в perlfunc(1);* описание циклов for **и** **в** рецепте **4.5.**

5.5. Вывод содержимого хэша

Проблема

Требуется вывести содержимое хэша, однако конструкции `print "%ХЭШ"` **и** `print %ХЭШ` не работают.

Решение

Одно из возможных решений — перебрать все пары «ключ/значение» в хэше (см. рецепт 5.4) **и** вывести их:

```
while ( ($k,$v) = each %hash) {
    print "$k => $v\n";
}
```

Также можно построить **список** строк с помощью `map`:

```
print map { "$_ => $hash{$_}\n" } keys %hash;
```

Или воспользуйтесь фокусом из рецепта **1.10** и интерполируйте **хэш** как список:

```
print "@{[ %hash ]}\n";
```

Или сохраните хэш во временном массиве и выведите его:

```
{
    my @temp = %hash;
    print "@temp";
}
```

Комментарий

Все перечисленные приемы обладают различными возможностями по управлению порядком и форматированием вывода, а также различной эффективностью.

Первый способ (перебор хэша) чрезвычайно гибок и эффективен по затратам памяти. **Вы** можете как угодно форматировать выходные данные, при этом понадобятся всего две скалярные переменные — текущий ключ и значение. Использование цикла позволяет вывести **хэш** с упорядочением ключей (ценой построения отсортированного списка):

```
(sort keys %hash) {
    print "$k => $hash{$k}\n";
}
```

Функция `map` не уступает перебору по богатству возможностей. Сортировка ключей по-прежнему позволяет работать с элементами в произвольном порядке. Выходные данные можно как угодно форматировать. На этот раз создается

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

список строк (например, "КЛЮЧ=>ЗНАЧЕНИЕ", как в приведенном выше примере), передаваемый print.

Два последних приема представляют собой фокусы, связанные с интерполяцией. Интерпретация хэша каксписка не позволяет предсказать или управлять порядком вывода пар "ключ/значение". Более того, данные в этом случае выводятся в виде списка ключей и значений, элементы которого разделяются текущим содержимым переменной $". В отличие от других приемов, вам не удастся вывести каждую пару на новой строке или отделить ключи от значений символом =>.

Смотри также

Описание переменной $" в *perlvar(1);* описание функций map, keys, sort и each в *perlfunc(1)*. Строковая интерполяция рассматривается в рецепте 1.10, а перебор хэша — в рецепте 5.4.

5.6. Перебор элементов хэша в порядке вставки

Проблема

Функции keys и each извлекают элементы хэша в довольно странном порядке. Вы хотите получить элементы в порядке вставки.

Решение

Воспользуйтесь модулем Tie::IxHash.

```
use Tie::IxHash;
tie %ХЭШ, "Tie::IxHash";
# Операции с хэшем %ХЭШ
@keys = keys %ХЭШ;          # Массив @keys отсортирован в порядке вставки
```

Комментарий

Модуль Tie::IxHash заставляет функции keys, each и values возвращать элементы в порядке занесения в хэш. Это часто избавляет от необходимости заранее обрабатывать ключи хэша какой-нибудь сложной сортировкой или поддерживать отдельный массив, содержащий ключи в порядке их вставки.

Tie::IxHash также представляет объектно-ориентированный интерфейс к функциям splice, push, pop, shift, unshift, keys, values и delete, а также многим другим.

Следующий пример демонстрирует использование keys и each:

```
# Инициализировать
use Tie::IxHash;

tie %food_color, "Tie::IxHas";
$food_color{Banana} = "Yellow";
$food_color{Apple} = "Green";
$food_color{Lemon} = "Yellow";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
print "In insertion order, the foods are:\n";
foreach $food (keys %food_color) {
    print "  $food\n";
}

print "Still in insertion order, the foods' colors are:\n";
while (( $food, $color ) = each %food_color ) {
    print "$food is        $color.\n";
}

In insertion order, the
   Banana
   Apple
   Lemon
Still in insertion order, the foods' colors
Banana is          Yellow.
Apple is
Lemon is
```

Смотри также

Документация по модулю Tie::IxHash от CPAN; рецепт 13.15.

5.7. Хэши с несколькими ассоциированными значениями

Проблема

Требуется хранить в хэше несколько значений, ассоциированных с одним ключом.

Решение

Сохраните в хэше ссылку на массив для хранения ассоциированных значений.

Комментарий

В хэше могут храниться только скалярные величины. Однако ссылки являются скалярными величинами. Таким образом, проблема решается сохранением в $ХЭШ {$КЛЮЧ} ссылки на массив со значениями, ассоциированными с ключом $КЛЮЧ. Обычные операции с хэшами — вставка, удаление, перебор и проверка существования — переписываются для операций с массивами (push,

Следующий фрагмент реализует простую вставку в хэш. Он обрабатывает выходные данные команды *who*(1) на компьютере с UNIX и выводит краткий список пользователей с терминалами, на которых они зарегистрированы:

```
%ttys = ();

open(WHO, "who|")          or die "can't open who: $!";
while (<WHO>) {
    ($user, $tty) = split;
    push( @{$ttys{$user}}, $tty );
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

foreach $user (sort keys %ttys) {
    print "$user: @{$ttys{$user}}\n";
}
```

Вся суть этого фрагмента заключена в строке push, где содержится версия $tty{$user} = $"удлямногозначногохэша. Все имена терминалов интерполируются в строке print конструкцией @{$ttys{user}}. Если бы, например, нам потребовалось вывести владельца каждого терминала, мы бы организовали перебор анонимного массива:

```
        $user (sort keys %ttys) {
    print "$user: ", scalar( @{$ttys{$user}} ), "ttys.\n";
                (sort @{$ttys{$user}}) {
        @stat = stat("/dev/$tty");
        $user = @stat ? ( getpwuid($stat[4]) )[0] : "(not available)";
        print "\t$tty (owned by $user)\n";
    }
}
```

Функция exists может иметь два значения: "Существует ли в хэше хотя бы одно значение для данного ключа?" и "Существует ли данное значение для данного ключа?" Чтобы реализовать вторую интерпретацию, придется просмотреть массив в поисках нужной величины. Первая трактовка exists косвенно связана с функцией delete: если мы можем гарантировать, что ни один анонимный массив никогда не остается пустым, можно воспользоваться встроенной функцией exists. Чтобы убедиться, что анонимные массивы не остаются пустыми, их следует проверять после удаления элемента:

```
sub multihash_delete {
    my {$hash, $key, $value) = @_;
    my $i;

            unless ref( $hash->{$key} );
    for ($i = 0; $i < @{ $hash->{$key} }; $i++) {
        if ($hash->{$key}->[$i] eq $value) {
            splice( @{$hash->{$key}}, $i, 1);
            last;
        }
    >

    delete $hash->{$key} unless @{$hash->{$key}};
}
```

Альтернативная реализация многозначных хэшей приведена в главе 13 "Классы, объекты и связи", где они реализуются как связанные обычные хэши.

Смотри также

Описаниефункций splice, delete, push, foreach и exists в *perlfunc*(1); рецепт 11.1. Связи рассматриваются в рецепте 13.15.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

5.8. Инвертирование хэша

Проблема

Хэш связывает ключ с ассоциированным значением. **У** вас имеется **хэш** и значение, для которого требуется определить ключ.

Решение

Воспользуйтесь функцией создания инвертированного хэша, где ассоциированные значения исходного хэша являются ключами, и наоборот.

```
# %ХЭШ связывает ключи со значениями
%ОБРАТНЫЙ          %ХЭШ;
```

Комментарий

В этом решении используется списковая эквивалентность хэшей, о которой упоминалось во введении. **В** списковом контексте интерпретирует **%ХЭШ** как список и меняет местами составляющие его элементов. Одно из важнейших свойств списковой интерпретации хэша заключается **в** том, что элементы списка представляют собой пары "ключ/значение". После инвертирования такого списка первым элементом становится значение, а вторым — ключ. Если интерпретировать ***такой*** список как хэш, его значения будут являться ключами исходногохэша, и наоборот.

Приведем пример:

```
%surname = ( "Mickey" => "Mantle", "Babe" => "Ruth");
%first_name =          %surname;
print $first_name{"Mantle", "\n";
Hickey
```

Если интерпретировать %surname как список, мы получим следующее:

```
("Mickey", "Mantle", "Babe", "Ruth")
```

(а может быть, ("Babe", "Ruth", "Mickey", "Mantle"), поскольку порядок элементов непредсказуем). После инвертирования список выглядит так:

```
("Ruth", "Babe", "Mantle", "Mickey")
```

Интерпретация его **в** качестве хэша дает следующее:

```
("Ruth" => "Babe", "Mantle" => "Mickey")
```

В примере 5.2 приведена программа foodfind. Если передать ей название продукта, она сообщает цвет, а если передать цвет — она сообщает название.

Пример 5.2. foodfind

```
#!/usr/bin/perl -w
# foodfind - поиск продуктов по названию или цвету

$given = shift @ARGV or die "usage: foodfind food_or_color\n";

%color = (
        "Apple"  => "red",
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            "Banana" => "yellow",
            "Lemon"  => "yellow",
            "Carrot" => "orange"
        );

%food =          %color;

if (exists $color{$given}) {
        print "$given is a food with color $color{$given}.\n";
}
if (exists $food{$given}) {
        print "$food{$given} is a food with color $given.\n";
}
```

Если два ключа исходного хэша имеют одинаковые значения ("Lemon" и "Banana" в предыдущем примере), то инвертированный хэш будет содержать лишь один из них (какой именно — зависит от порядка хэширования, так что непредсказуемо). Дело в том, что хэшй в Perl по определению имеют уникальные ключи.

Чтобы инвертировать хэш с повторяющимися значениями, следует воспользоваться методикой рецепта 5.7 — то есть построить хэш, ассоциированные значения которого представляют собой списки ключей исходного хэша:

```
# Хэш %food_color определяется во введении
while (($food,$color) = each(%food_color)) {
    push(@{foods_with_color{$color}}, $food);
}

print "@{$foods_with_color{yellow}} were yellowfoods.n";
Banana Lemon were yellow foods.
```

Кроме того, это позволит модифицировать программу foodfind так, чтобы она работала с цветами, соответствующими сразу нескольким продуктам. Например, при вызове foodfind yellow будут выводиться и Banana, и Lemon.

Если какие-либо значения исходного хэша были не простыми строками и числами, а ссылками, при инвертировании возникает проблема — ссылки не могут использоваться в качестве ключей, если только вы не воспользуетесь модулем Tie::RefHash (см. рецепт 5.12).

Смотри также

Описание функций reverse в *perlfunc(1);* рецепт 13.15.

5.9. Сортировка хэша

Проблема

Требуется работать с элементами хэша в определенном порядке.

Решение

Воспользуйтесь функцией keys для построения списка ключей, а затем отсортируйте их в нужном порядке:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# %hash - сортируемый хэш
@keys = sort { criterion() } (keys %hash);
              (@keys) {
    $value = $hash{$key},
    # Сделать что-то с $key, $value
}
```

Комментарий

Хотя хранить элементы хэша в заданном порядке невозможно (без использования модуля Tie:IxHash, упомянутого в рецепте 5.6), перебирать их можно в любом порядке.

Существует множество разновидностей одного базового механизма: вы извлекаете ключи, упорядочиваете их функцией sort и обрабатываете элементы в новом порядке. Допускается применение любых хитростей сортировки, упоминавшихся в главе 4 "Массивы". Рассмотрим пару практических примеров.

В первом фрагменте sort просто используется для упорядочения ключей по алфавиту:

```
        $food (sort keys %food_color) {
    print "$food is $food_color($food) \n";
}
```

Другой фрагмент сортирует ключи по ассоциированным значениям:

```
        $food (sort { $food_color{$a} cmp $food_color{$b} } )
    keys %food_color) {
        print "$food is $food_color{$food}.\n";
}
```

Наконец, сортировка выполняется по длине ассоциированных значений:

```
@foods = sort { length($food_color{$a}) <=> length($food_color{$b}) }
    keys %food_color;
        $food (@foods) {
    print "$food is $food_color{$food}.\n";
}
```

Смотри также

Описание функций sort и keys в *perlfunc(1);* рецепт 5.6 Сортировка списков рассматривается в рецепте 4.15.

5.10. Объединение хэшей

Проблема

Требуется создать новый хэш, содержащий элементы двух существующих хэшей.

Решение

Интерпретируйте хэши как списки и объедините их **так**, как это делается со списками:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
%merged = (%A, %B);
```

Для экономии памяти можно организовать перебор элементов и построить новый хэш следующим образом:

```
%merged = ();
while ( ($k,$v) = each(%A) ) {
    $merged{$k} = $v;
}
while ( ($k,$v) = each(%B) ) {
    $merged{$k> = $v;
}
```

Комментарий

В первом варианте, как и в предыдущем рецепте инвертирования хэшей, используется списковая эквивалентность, о которой говорилось во введении. (%A, %B) интерпретируется как список пар "ключ/значение". Когда он присваивается объединенному хэшу %merged, Perl преобразует список пар снова в хэш.

Рассмотрим, как эта методика реализуется на практике:

```
# Хэш %food_color определяется во Введении
%drink_color = ( Galliano  => "yellow",
                 "Mai Tai" => "blue" );

%ingested_colors = (%drink_color, %food_color);
```

Ключи обоих входных хэшей присутствуют в выходном не более одного раза. Если в хэшах найдутся совпадающие ключи, в итоговый хэш включается тот ключ, который встретился последним.

Прямое присваивание компактно и наглядно, но при больших размерах хэшей оно приводит к большим расходам памяти. Это связано с тем, что перед выполнением присваивания итоговому хэшу Perl разворачивает оба хэша во временный список. Пошаговое объединение с помощью each, показанное ниже, избавит вас от этих затрат. Заодно вы сможете решить, как поступать с совпадающими ключами.

С применением each первый фрагмент записывается следующим образом:

```
# Хэш %food_color определяется во Введении
%drink_color = ( Galliano  => "yellow",
                 "Mai Tai" => "blue" );

%substance_color = ();
while (($k, $v) = each %food_color) {  .                    ,  .
    $substance_color{$k} = $v;
}
while (($k, ,$v) = each %drink_color) {
    $substance_color{$k} = $v;
}
```

Обратите внимание на повторяющийся код присваивания в циклах while. Проблема решается так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                    (\%food_color, \%drink_color ) {
    while (($k, $v) =
        $substance_color{$k} = $v;
    }
}
```

Если в объединяемых хэшах присутствуют одинаковые ключи, можно вставить код для обработки дубликатов:

```
        $substanceref (\%food_color, \%drink_color ) {
    while (($k, $v) = each %substanceref) {
        if (exists $substance_color{$k>) {
            print "Warning: $k seen twice. Using the first definition.\n";
            next;
        }
        $substance_color{$k} = $v;
    }
}
```

В частном случае присоединения одного хэша к другому можно воспользоваться срезом для получения более элегантной записи:

```
@all_colors{keys %new_colors} = values %new_colors;
```

Потребуется память в объеме, достаточном для хранения списков всех ключей и значений %new_colors. Как и в первом варианте, расходы памяти при большом размере списков могут сделать эту методику неприемлемой.

Смотри также

Описание функции each в ***perlfunc(1);*** рецепт **4.9.**

5.11. Поиск общих или различающихся ключей в двух хэшах

Проблема

Требуется найти в хэше ключи, присутствующие в другом хэше, — или наоборот, не входящие в другой хэш.

Решение

Организуйте перебор ключей хэша с помощью функции keys и проверяйте, присутствует ли текущий ключ в другом хэше.

Поиск общих ключей

```
my @common = ();
        (keys %hash1) {
    push(@common, $_) if exists $hash2{$_};
}
# @common содержит общие ключи
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Поиск ключей, отсутствующих в другом хэше

```
my @this_not_that = ();
        (keys %hash1) {
    push(@this_not_that, $_) unless exists $hash2{$_};
}
```

Комментарий

При поиске общих или различающихся ключей хэшей можно воспользоваться рецептами для поиска общих или различающихся элементов в массивах ключей хэшей. За подробностями обращайтесь к рецепту 4.8.

В следующем фрагменте поиск различающихся ключей применяется для нахождения продуктов, не входящих в хэш с описаниями цитрусовых:

```
# Хэш %food_color определяется во введении
# %citrus_color - хэш, связывающий названия цитрусовых плодов с их цветами
%citrus_color = (Lemon  => "yellow",
                 Orange => "orange",
                 Lime   =>

# Построить список продуктов, не входящих в хэш цитрусовых
@non-citrus = ();

        (keys %food_color) {
    push (@non_citrus, $_) unless exists $citrus_color{$_};
}
```

Смотри также

Описание функции each в *perlfunc*(1). Срезы хэшей рассматриваются в *perldata*(1).

5.12. Хэширование ссылок

Проблема

Если функция keys вызывается для хэша, ключи которого представляют собой ссылки, то возвращаемые ей ссылки не работают. Подобная ситуация часто возникает при создании перекрестных ссылок в двух хэшах.

Решение

Воспользуйтесь модулем Tie::RefHash:

```
use Tie::RefHash;
tie %hash, "Tie::RefHas";
# Теперь в качестве ключей хэша %hash можно использовать ссылки
```

Комментарий

Ключи хэшей автоматически преобразуются в строки — тоесть интерпретируются так, словно они заключены в кавычки. Для чисел и строк при этом ничего не теряется. Однако со ссылками дело обстоит иначе.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

После преобразования в строку ссылка принимает следующий вид:

```
Class::Somewhere=HASH(0x72048)
ARRAY(0x72048)
```

Преобразованную ссылку невозможно вернуть к прежнему виду, поскольку она перестала быть ссылкой и превратилась в обычную строку. Следовательно, при использовании ссылок в качестве ключей хэша они теряют свои "волшебные свойства".

Для решения этой проблемы обычно создается специальный хэш, ключами которого являются ссылки, преобразованные в строки, а значениями — настоящие ссылки. Именно это и происходит в модуле Tie::RefHash. Мы воспользуемся объектами ввода/вывода для работы с файловыми манипуляторами и покажем, что даже такие странные ссылки могут использоваться для индексации хэша, связанного с Tie::RefHash.

Приведем пример:

```
use Tie::RefHash;
use IO::File;

tie %name, "Tie::RefHash";
        $filename ("/etc/termcap","/vmunix", "/bin/cat")
    $fh = IO::File->("< $filename") or next;
    $name{$fh} = $filename;
}
print "open files: ", join(", values %name", "\n";
        $file (keys %name) {
    seek($file, 0, 2);    # Позиционирование в конец файла
    printf("%s is %d bytes long.\n", $name{$file}, tell($file));
}
```

Однако вместо применения объекта в качестве ключа хэша обычно достаточно сохранить уникальный атрибут объекта (например, имя или идентификатор).

Смотри также

Документация по стандартному модулю Tie::RefHash; раздел "Warning" *perl-*

5.13. Предварительное выделение памяти для хэша

Проблема

Требуется заранее выделить память под хэш, чтобы ускорить работу программы — в этом случае Perl не придется выделять новые блоки при каждом добавлении элемента. Окончательный размер хэша часто бывает известен в начале построения, и эта информация пригодится для повышения быстродействия.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Присвойте количество пар "ключ/значение" конструкции keys(%ХЭШ):

```
# Выделить в хэше %hash память для $num элементов.
keys(%hash) = $num;
```

Комментарий

Новая возможность, впервые появившаяся в Perl версии 5.004, может положительно повлиять на быстродействие вашей программы (хотя и не обязательно). В хэшах Perl и так применяются общие ключи, поэтому при наличии хэша с ключом "Apple" Perl уже не выделяет память под другую копию "Apple" при включении этого ключа в другой хэш.

```
# В %users резервируется место для 512 элементов.
keys(%users) = 512;
```

Внутренние структуры данных Perl требуют, чтобы количество ключей было равно степени 2. Если написать:

```
keys(%users) = 1000;
```

Perl выделит для хэша 1024 "гнезда". Количество ключей не всегда равно количеству гнезд. Совпадение обеспечивает оптимальное быстродействие, однако конкретное соответствие между ключами и гнездами зависит от ключей и внутреннего алгоритма хэширования Perl.

Смотри также

Функция keys описана в perlfunc(1). Также обращайтесь к рецепту 4.3.

5.14. Поиск самых распространенных значений

Проблема

Имеется сложная структура данных (например, массив или хэш). Требуется узнать, как часто в ней встречается каждый элемент массива (или ключ хэша). Допустим, в массиве содержатся сведения о транзакциях Web-сервера и вы хотите узнать, какой файл запрашивается чаще остальных. Или для хэша, в котором имя пользователя ассоциируется с количеством регистрации в системе, требуется определить наиболее распространенное количество регистрации.

Решение

Воспользуйтесь хэшем и подсчитайте, сколько раз встречается тот или иной элемент, ключ или значение:

```
%count = ();
        $element (@array) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $count{$element}++;
}
```

Комментарий

Каждый раз, когда возникает задача подсчета различных объектов, вероятно, стоит воспользоваться хэшем. В приведенном выше цикле `foreach` для каждого экземпляра `$element` значение `$count{$element}` увеличивается на 1.

Смотри также

Рецепты 4.6 и 4.7.

5.15. Представление отношений между данными

Проблема

Требуется представить отношения между данными — например, отношения "предок/потомок" в генеалогическом дереве или "родительский/порожденный процесс" в таблице процессов. Задача тесно связана с представлением таблиц в реляционных базах данных (отношения между записями) и графов в компьютерных технологиях (отношения между узлами графа).

Решение

Воспользуйтесь хэшем.

Комментарий

Следующий хэш представляет часть генеалогического дерева из Библии:

```
%father = ( 'Cain'      => 'Adam',
            'Abel'      => 'Adam',
            'Seth'      => 'Adam',
            'Enoch'     => 'Cain',
            'Irad'      => 'Enoch',
            'Mehujael'  => 'Irad',
            'Methusael' => 'Mehujael',
            'Lamech'    => 'Methusael',
            'Jabal'     => 'Lamech',
            'Jubal'     => 'Lamech',
            'Tubalcain' => 'Lamech',
            'Enos'      => 'Seth' );
```

Например, мы можем легко построить генеалогическое дерево любого персонажа:

```
while (o) {
    chomp;
    do {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        print "$_ ";        # Вывести текущее имя
        $_ = $father{$_};  # Присвоить $_ отца $_
    } while defined;        # Пока отцы находятся
    print "\n";
}
```

Просматривая хэш `%father`, можно отвечать на вопросы типа: "Кто родил Сета?" При инвертировании хэша отношение заменяется противоположным. Это позволяет использовать рецепт 58 для ответов на вопросы типа: "Кого родил Ламех?"

```
while ( ($k, $v) = each %father ) {
    push(
>

$" = ', ';            # Выходные данные разделяются запятыми
while (<>) <
    chomp;

        @children = @{$children{$_}};
    } else {
        @children = "nobody";
    }
    print "$_ begat @children.\n";
}
```

Хэши также могут представлять такие отношения, как директива #include языка C — A включает B, если A содержит `#include` B. Следующий фрагмент строит хэш (он не проверяет наличие файлов в /usr/include, как следовало бы, но этого можно добиться ценой минимальных изменений):

```
foreach $file (@files) {
    local *F;              # На случай, если понадобится
                           # локальный файловый манипулятор
    unless (open (F, "<$file")) {
        warn "Couldn't      file: $!; skipping.\n";
        next;
    }

    while (<F>) {
        next unless /^\s*#\s+include\s+<([^>\+)>/;
        push(@{$includes{$1}}, $file);
    }
    close F;
}
```

Другой фрагмент проверяет, какие файлы не включают других:

```
                    ();        # Список файлов, не включающих других файлов
@uniq{map { @$_ } values %includes} = undef;
        $file (sort keys %uniq) {
    push( @include_free , $flle ) unless $includes{$file};
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Результат values `%includes` представляет собой анонимный массив, поскольку один файл может включать (и часто включает) сразу несколько других файлов. Мы используем `map` для построения большого списка всех включенных файлов и удаляем дубликаты с помощью хэша.

Смотри также

Рецепт **4.6;** описание более сложных структур данных **в** рецептах **11.9—11.14.**

5.16. Прог

Программа (см. пример 5.3) преобразует выходные данные *du*:

```
% du cookbook
19      pcb/fix
20      pcb/rev/maybe/yes
10      pcb/rev/maybe/not
705     pcb/rev/maybe

1371
3       pcb/pending/mine
1016    pcb/pending
2412    pcb
```

в отсортированную иерархическую структуру с расставленными отступами:

```
2412  pcb
      1371
        |     705 maybe
              |     675 .
              |       20 yes
              |       10 not
            612.
              54 web
      1016 pending
                1013 .
                   3 mine
        19 fix
        6 .
```

Аргументы передаются программе через du. Это позволяет вызвать любым из приведенных ниже способов, а может быть, и иначе — если ваша версия du поддерживает другие параметры.

```
/usr

/bin
```

Хэш `%Dirsize` сопоставляет имена с размерами файлов. Например, значение `$Dirsize{"pcb"}` в нашем примере равно 2412. Этот хэш используется как для вывода, так и для сортировки подкаталогов каждого каталога по размерам.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Хэш %Kids представляет больший интерес. Для любого пути $path значение $Kids{path} содержит (ссылку на) массив с именами подкаталогов данного каталога. Так, элемент с ключом "pcb" содержит ссылку на анонимный массив со строками "fix", "rev" и "pending". Элемент "rev" содержит "maybe" и "web". В свою очередь, элемент "maybe" содержит "yes" и "no", которые не имеют собственных элементов, поскольку являются "листами" (конечными узлами) дерева.

Функции output передается начало дерева — последняя строка, прочитанная из выходных данных du. Сначала фун кция выводит этот каталог и его размер, затем сортирует его подкаталоги (если они имеются) так, чтобы подкаталоги наибольшего размера оказались наверху. Наконец, output вызывает саму себя, рекурсивно перебирая все подкаталоги. Дополнительные аргументы используются при форматировании.

Программа получается рекурсивной, поскольку рекурсивна сама файловая система. Однако ее структуры данных не рекурсивны — по крайней мере, не в том смысле, в котором рекурсивны циклические связанные списки. Каждое ассоциированное значение представляет собой массив ключей для дальнейшей обработки. Рекурсия заключается в обработке, а не в способе хранения.

Пример 5.3

```
#!/usr/bin/perl  -w
          печать сортированного иерархического представления
# выходных данных du
use strict;

my %Dirsize;
my %Kids;

getdots(my $topdir = input());
output($topdir);

# Запустить du, прочитать входные данные, сохранить размеры и подкаталоги
# Вернуть последний прочитанный каталог (файл?)
sub input {
    my($size, $name, $parent);
    @ARGV = ("du @ARGV |");          # Подготовить аргументы
    while (<>)
        ($size, $name) = split;
        $Dirsize{$name}` = $size;
                  $name)    s#/[^/]+$##;   # Имя каталога
        push @{ $Kids{$parent} }, $name unless eof;
    }
          $name;
}

# Рассчитать, сколько места занимают файлы каждого каталога,
# не находящиеся в подкаталогах. Добавить новый фиктивный
# подкаталог с именем ".", содержащий полученную величину.
sub getdots {
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 5.3 (продолжение)

```
    my $root = $_[0];
    my($size, $cursize);
    $size = $cursize = $Dirsize{$root};
    if ($Kids{$root}) {
        for my $kid (@< $Kids{$root} }) {
            $cursize -= $Dirsize{$kid};
            getdots($kid);
        }.
    }
    if ($size != $cursize) {
        my $dot = "$root/.";
        $Dirsize{$dot} = $cursize;
        push @{ $Kids{$root} }, $dot;
    }
}

# Рекурсивно вывести все данные,
# передавая при рекурсивных вызовах
# выравнивающие пробелы и ширину числа
sub output {
    my($root,          $width) = (shift, shift || '', shift || 0);
    my $path;
    ($path = $root) =^ s#.*/##;     # Базовое имя
    my $size = $Dirsize{$root};
    my $line = sprintf("%${width}d %s", $size, $path);
    print $prefix, $line, "\n";
                  $line) {        # Дополнительный вывод
        s/\d /| /;
        s/[^|]//g;
    }
    if ($Kids{$root}) {             # Узел имеет подузлы
        my @Kids = @{ $Kids{$root} };
        @Kids = sort { $Dirsize{$b} <=> $Dirsize{$a} ) @Kids;
        $Dirsize{$Kids[0]} =~ /(\d+)/;
        my $width = length $1;
        for my $kid (@Kids) {                        $width) }
    }
}
```

До того как в Perl появилась прямая поддержка хэшей массивов, эмуляция подобных конструкций высшего порядка требовала титанических усилий. Некоторые программисты использовали многократные вызовы split и join, но это работало чрезвычайно медленно.

В примере 5.4 приведена версия программы далеких дней. Поскольку у нас не было прямых ссылок на массивы, приходилось самостоятельно залезать в символьную таблицу Perl. Программа на ходу создавала переменные с жутковатыми именами. Удастся ли вам определить, какой хэш используется этой программой?

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Массив @{"pcb"} содержит ссылку на анонимный массив, содержащий "pcb/fix", "pcb/rev" и "pcb/pending". Массив @{"pcb/rev"} содержит "pcb/rev/maybe" и "pcb/rev/web". Массив @{"pcb/rev/maybe"} содержит "pcb/rev/maybe/yes" и "pcb/rev/maybe/not".

Когда вы присваиваете *kid что-нибудь типа "pcb/fix", строка в правой части преобразуется в тип-глоб. @kid становится синонимом для @{"pcb/fix"}, но это отнюдь не все. &kid становится синонимом для &{"pcb/fix"} и т. д.

Если эта тема покажется неинтересной, подумайте, как local использует динамическую область действия глобальных переменных, чтобы избежать передачи дополнительных аргументов. Заодно посмотрите, что происходит с переменной width в процедуре output.

Пример *5А.*

```
#!/usr/bin/perl
                    старая версия, которая появилась
# до выхода perl5 (начало 90-х)

@lines = 'du @ARGV';
chop(@lines);
&input($top = pop @lines);
&output($top);
exit;

sub input {
    local($root, "kid, $him) = @_[0,0];
    while (@lines && &childof($root, $lines[$#lines])) {
        &input($him = pop(@lines));
        push(@kid, $him);
    i}
    if (@kid) {
        local($mysize) = ($root =~ /^(\d+)/);
        for (@kid) { $mysize -= (/^(\d+)/)[0]; }
        push(@kid, "$mysize .") if $size != $mysize;
    }
    @kid = &sizesort(*kid);
}

sub output {
    local($root,                  = @_[0,0,1];
    local($size, $path) = split(' ', $root);
    $path =^ s!.*/!!;
    $line = sprintf("%${width}d %s", $size, $path);
    print          $line, "\n";
               $line;

            =- s/[^|]/ /g;
    local($width) = $kid[0] =~ /(\d+)/ && length("$1");
    for (@kid) { &output($_,          };
}
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 5.4 (продолжение)

```
sub sizesort {
    local(*list, @index) = shift;
    sub bynum { $index[$b] <=> $index[$a];
    for (@list) { push(@index, /(\d+)/); >
    @list[sort bynum 0..$#list];
}

sub childof {
    local(@pair) = @_;
    for (@pair) { s/^\d+\s+//g; s/$/\//; }
    index($pair[1], $pair[0]) >= 0;
}
```

Итак, какой же хэш используется старой программой ответ — `%main::`, то есть символьная таблица Perl. Не стоит и говорить, что эта программа не будет работать с use strict. Мы рады сообщить, что новая версия работает втрое быстрее старой. Дело в том, что старая версия постоянно ищет переменные в символьной таблице, а новая обходится без этого. Кроме того, нам удалось избежать медленных вызовов `split` для занимаемого места и имени каталога. Однако мы приводим и старую версию, поскольку она весьма поучительна.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

6 Поиск по шаблону

[Искусство — это] шаблон, наполняемый разумом.

Сэр Герберт Рид, «Значение Искусства»

Введение

В большинстве современных языков программирования существуют примитивные средства поиска по шаблону (обычно вынесенные в дополнительные библиотеки), но шаблоны Perl интегрируются на уровне самого языка. Они обладают возможностями, которыми не могут похвастаться другие языки; возможностями, которые позволяют взглянуть на данные с принципиально новой точки зрения. Подобно тому, как шахматист воспринимает расположение фигур на доске как некий образ, адепты Perl рассматривают данные с позиций шаблонов. Шаблоны записываются на языке регулярных выражений[1], богатом знаками препинания, и позволяют работать с замечательными алгоритмами, обычно доступными лишь экспертам в области компьютерных технологий.

"Если поиск по шаблону — такая потрясающая и мощная штука, — спросите вы, — то почему же эта глава не содержит сотни рецептов по применению регулярных выражений?" Да, регулярные выражения обеспечивают естественное решение многих проблем, связанных с числами, строками, датами, Web-документами, почтовыми адресами и буквально всем, что встречается в этой книге. В других главах поиск по шаблону применяется свыше 100 раз. Ав этой главе в основном представлены те рецепты, в которых шаблоны являются частью вопроса, а не ответа.

Обширная и тщательно проработанная поддержка регулярных выражений в Perl означает, что в вашем распоряжении оказываются не только те средства, которые не встречаются ни в одном другом языке, но и принципиально новые возможности их использования. Программисты, недавно познакомившиеся с Perl, часто ищут в нем функции поиска и подстановки:

Точнее, регулярные выражения в классическом смысле не содержат обратных ссылок, присутствующих в шаблонах Perl.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
match( $строка, $шаблон);
subst( $строка, $шаблон, $замена);
```

Однако поиск и подстановка — настолько распространенные задачи, что они заслуживают собственного синтаксиса:

```
$meadow =~ m/sheep/;   # Истинно, если $meadow содержит "sheep"
$meadow !~ m/sheep/;   # Истинно, если $meadow не содержит "sheep"
$meadow =" s/old/new;  # Заменить в $meadow "old" на "new"
```

Поиск по шаблону даже в упрощенном виде не похож на обычные строковые сравнения. Он больше похож на поиск строк с применением универсальных символов-мутантов, к тому же накачанных допингом. Без специального "якоря" позиция, в которой ищется совпадение, свободно перемещается по всей строке. Допустим, если вы захотите найти слово ovine или ovines и воспользуетесь выражением `$meadow =~ /ovine/`, то в каждой из следующих строк произойдет ложное совпадение:

```
Fane bovines demand fine
Muskoxen     polar ovibovine species.
Grooviness went out of fashion decades ago.
```

Иногда нужная строка находится прямо у вас перед глазами, а совпадение все равно не происходит:

```
Ovines     found typically in ovlaries.
```

Проблема в том, что вы мыслите категориями человеческого языка, а механизм поиска по шаблону — нет. Когда этот механизм получает шаблон /ovine/ и другую строку, в которой происходит поиск, он ищет в строке символ "o", за которым сразу же следует "v", затем "l", "n" и "e". Все, что находится до этой последовательности символов или после нее, не имеет значения.

Итак, выясняется, что шаблон находит совпадения там, где они не нужны, и не узнает то, что действительно нужно. Придется усовершенствовать его. Например, для поиска последовательности ovine или ovines шаблон должен выглядеть примерно так:

```
if ($meadow =^ /\bovihes?\b/i) { print      be sheep!" }
```

Шаблон начинается со ***метасимвола*** \b, который совпадает только с границей слова, **s?** обозначает необязательный символ **s** — он позволяет находить как ovine, так **и** ovines. Модификатор /i **в** конце шаблона означает, что поиск осуществляется без учета регистра.

Как видите, некоторые символы и последовательности символов имеют особый смысл для механизма поиска по шаблону. Метасимволы фиксируют шаблон **в** начале или конце строки, описывают альтернативные значения для частей шаблона, организуют повторы и позволяют запомнить часть найденной подстроки, чтобы **в** дальнейшем использовать ее **в** шаблоне или программном коде.

Освоить синтаксис поиска по шаблону не так уж сложно. Конечно, служебных символов много, но существование каждого из них объясняется вескими причинами. Регулярное выражение — это не просто беспорядочная груда знаков... это тщательно продуманная груда знаков! Если **вы** что-нибудь забыли, всегда можно

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

заглянуть в документацию. Сводка по синтаксису регулярных выражений имеется в страницах руководства *perlre*(1) и *perlop*(1), входящих в любую поставку Perl.

Три затруднения

Но синтаксис регулярных выражений — это еще цветочки по сравнению с их хитроумной семантикой. Похоже, большинство трудностей вызывают три особенности поиска по шаблону: жадность, торопливость (а так же то, как эти три аспекта взаимодействуют между собой) и возврат.

Принцип жадности: если квантификатор (например, *) может совпасть в нескольких вариантах, он всегда совпадает со строкой наибольшей длины. Объяснения приведены в рецепте 6.15.

Принцип торопливости: механизм поиска старается обнаружить совпадение как можно скорее, иногда даже раньше, чем вы ожидаете. Рассмотрим конструк- =~ /x*/. Если попросить вас объяснить еесмысл, вы, вероятно, скажете: "Содержит ли строка "Fred" символы **x?"** Вероятно, результат поиска окажется неожиданным — компьютер убежден, что символы присутствуют. Дело в том, что /x*/ означает не просто "символы x", а ***«любое*** количество символов **x".** Или более формально — иолъ ***и более*** символов. В данном случае нетерпеливый механизм поиска удовлетворяется нулем.

Приведем более содержательный пример:

```
$string = "good food";
$string =^ s/o*/e/;
```

Как вы думаете, какое из следующих значений примет $string после подстановки?

```
goof food
geod food
geed food
geed feed
ged food
ged fed
egood food
```

Правильный ответ — последний, поскольку первая точка, в которой встречается ноль и более экземпляров "o", находится прямо в начале строки. Удивлены? С регулярными выражениями "то бывает довольно часто.

А теперь попробуйте угадать, как будет выглядеть результат при добавлении модификатора /g, который делает подстановку глобальной? Строка содержит много мест, в которых встречается ноль и более экземпляров "o", — точнее, восемь. Итак, правильный ответ — "egeede efeede".

Приведем другой пример, в котором жадность уступает место торопливости:

```
% echo ababacaca | perl -ne `print "$&\n" if /(a|ba|b)+(a|ac)+/'
ababa
```

Это объясняется тем, что при поиске в Perl используются так называемые традиционные неопределенные конечные автоматы (в отличие от неопределенных конечных автоматов POSIX). Подобные механизмы поиска гарантируют возврат не самого длинного общего совпадения, а лишь самого длинного левого совпаде-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ния. Можно считать, что жадность Perl проявляется лишь слева направо, а не в глобальном контексте.

Но дело не обязательно обстоит именно так. В следующем примере используется awk — язык, от которого Perl позаимствовал немало:

```
% echo ababacaca |
  awk 'match($0,/(a|ba|b)+(a|ac)+/) { print substr($0, RSTART, RLENGTH) Г
ababacaca
```

Выбор реализации поиска по шаблону в основном зависит от двух факторов: нерегулярности выражений (то есть наличия в них обратных ссылок) и типа возвращаемой величины (логическое "да/нет", все совпадение, подвыражения). Такие инструменты, как awk, lex, используют регулярные выражения и возвращают либо логическое "да/нет", либо все совпадение. Подобные возможности поддерживаются определенными конечными автоматами; поскольку определенные конечные автоматы работают быстрее и проще, реализация в перечисленных инструментах основана именно на них. Поиск по шаблону в таких программах и ed, perl, — совсем другое дело. Обычно приходится поддерживать нерегулярные выражения и знать, какие части строки совпали с различными частями шаблона. Эта задача намного сложнее и отличается экспоненциальным ростом времени выполнения. Естественный алгоритм ее реализации основан на неопределенных конечных автоматах; в этом заключается и проблема, и возможности. Проблема — в том, что неопределенные конечные автоматы работают медленно. Возможности — в том, что формулировка шаблона с учетом особенностей конкретной реализации позволяет существенно повысить быстродействие.

Последняя и самая интересная из трех особенностей — возврат. Чтобы шаблон совпал, должно совпасть все регулярное выражение, а не лишь его отдельная часть. Следовательно, если начало щаблона с квантификатором совпадает, а одна из последующих частей шаблона — нет, механизм поиска возвращается к началу и пытается найти для него другое совпадение — отсюда и термин "во зврат". Фактически это означает, что механизм поиска должен систематически перебирать разные возможности до тех пор, пока не найдет полное совпадение. В некоторых реализациях поиска возврат используется для поиска Других совпадающих компонентов, которые могли бы увеличить длину найденного совпадения. Механизм поиска Perl этого не делает; найденное частичное совпадение используется немедленно, — если позднее другая часть шаблона сделает полное совпадение невозможным, происходит возврат и поиск другого частичного совпадения (см. рецепт 6.16).

Модификаторы

Модификаторы, используемые при поиске по шаблону, намного проще перечислить и понять, чем другие метасимволы. Ниже приведена краткая сводка:

|\ Игнорировать регистр (с учетом национальных алфавитов).

/x Игнорировать большинство пропусков в шаблонах и разрешить комментарии.

/д Глобальный модификатор — поиск/замена выполняются всюду, где это возможно.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

/gc Не сбрасывать позицию при неудачном поиске.

/s Разрешить совпадение . с переводом строки; кроме того, игнорировать устаревшее значение $*

/m Разрешить совпадение ^ и $ соответственно для начала и конца строки во внутренних переводах строк.

/o Однократная компиляция шаблонов.

/e Правая часть s/// представляет собой выполняемый код.

/ee Правая часть s/// выполняется, после чего возвращаемое значение интерпретируеся снова.

Наиболее распространены модификаторы /i и /g. Шаблон /ram/i совпадает со строками ram , RAM , Ram и т.д. При наличии этого модификатора обратные ссылки проверяются без учета регистра (пример приведен в рецепте 6.16). При вызове директивы use locale в сравнениях будет учитываться состояние текущих Локальных настроек. В текущей реализации модификатор /i замедляет поиск по шаблону, поскольку йодавляет некоторые оптимизации скорости.

Модификатор /g используется с s/// для замены всех найденных совпадений, а не только первого. Кроме того, /g используется cm// в циклах поиска (но не замены!) всех совпадений:

```
while (m/(\d+)/g) {
    print Found number $1\n ,
}
```

В списковом контексте /g извлекает все совпадения в массив:

```
@numbers = m/(\d+)/g,
```

В этом случае будут найдены только неперекрывающиеся совпадения. Длй поиска перекрывающихся совпадений придется идти на хитрость — организовать опережающую проверку нулевой ширины с помощью конструкции (?=). Раз ширина равна нулю, механизм поиска вообще не смещается вперед При этом найденные данные сохраняются внутри скобок. Однако Perl обнаруживает, что при наличии модификатора /g мы остались на прежнем месте, и перемещается на один символ вперед.

Продемонстрируем отличия на примере:

```
$digits = 123456789
@nonlap = $digits =~/(\d\d\d)/g,
@yeslap = $digits =~/(?=(\d\d\d))/g,
print Non-overlapping @nonlap\n ,
print Overlapping: @yeslap\n ;
Non-overlapping: 123 456 789
Overlapping: 123 234 345 456 567 678 789
```

Модификаторы /s и /m используются для поиска последовательностей, содержащих внутренний перевод строки При указании /s точка совпадает с \n — в обычных условиях этого не происходит. Кроме того, при поиске игнорируется значение устаревшей переменной $*. Модификатор /m приводит к тому, что ^ и $ совпадают в позициях до и после \n соответственно. Он полезен в режиме

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

поглощения файлов, о котором говорится во введении к главе 8 "Содержимое файлов" и рецепте 6.6.

При наличии модификатора /e правая часть выполняется как программный код, и затем полученное значение используется в качестве заменяющей строки. Например, подстановка `s/(\d+)/sprintf("%#x", $1)/ge` преобразует все числа в шестнадцатеричную систему счисления — скажем, 2581 превращается в 0xb23.

В разных странах существуют разные понятия об алфавите, поэтому стандарт POSIX предоставляет в распоряжение систем (а следовательно, и программ) стандартные средства для представления алфавитов, упорядочения наборов символов и т. д. Директива Perl use lo cale предоставляет доступ к некоторым из них; дополнительную информацию можно найти в странице руководства *perllocale*. При действующей директиве use locale в символьный класс \w попадают символы с диакритическими знаками и прочая экзотика. Служебные символы изменения регистра \u, \U, \l и \L (а также соответствующие функции uc, ucfirst и т. д.) также учитывают use locale, поэтому \u превратит а в **Σ**, если этого потребует локальный контекст.

Специальные переменные

В результате некоторых операций поиска по шаблону Perl устанавливаетзначения специальных переменных. Так, переменные $1, $2, $3 и т. д.*до бесконечности* (Perl не останавливается на $9)устанавливаются в том случае, если шаблон содержит обратные ссылки (то есть часть шаблона заключена в скобки). Каждая открывающая скобка, встречающаяся в шаблоне слева направо, начинает заполнение новой переменной. Переменная $+ содержит значение последней обратной ссылки для последнего успешного поиска. Это. помогает узнать, какой из альтернативных вариантов поиска был обнаружен (например, при обнаруженном совпадении для `/(x, *y) | (y. *z)/` в переменной **$+** будет находиться содержимое $1 или $2 — в зависимости от того, какая из этих переменных была заполнена). Переменная $& содержит полный текст совпадения при последнем успешном поиске. В переменных **$`** и **$'** хранятся строки соответственно до и после совпадения при успешном поиске:

```
$string = "And little lambs eat ivy";
$string =~ /l[^s]*s/;
print "($`) ($&) ($')\n";
(And ) (little lambs) ( eat ivy)
```

Переменные **$`**, $& и **$'** соблазнительны, но опасны. Само их присутствие в любом месте программы замедляет поиск по шаблону, поскольку механизм должен присваивать им значения при каждом поиске. Сказанное справедливо даже в том случае, если вы всего один раз используете лишь одну из этих переменных, — или даже если они совсем не используются, а лишь встречаются в программе. В версии 5.005 переменная $& перестала обходиться так дорого.

После всего сказанного возникает впечатление, чтошаблоны могут все. Как ни странно, это не так (во всяком случае, не совсемтак). Регулярные выражения в принципе не способны решить некоторые задачи. В этом случае на помощь при-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ходят специальные модули. Скажем, регулярные выражения не обладают средствами для работы со сбалансированным вводом, то есть любыми данными произвольной вложенности — например, парными скобками, тегами HTML и т. д. Для таких целей приходится строить настоящий анализатор наподобие HTML::Parser из рецептов главы 20 "Автоматизация в Web". Еще одна задача, не решаемая шаблонами Perl, — неформальный поиск. В рецепте 6.13 показано, как она решается с помощью специального модуля.

6.1. Копирование с подстановкой

Проблема

Вам надоело многократно использовать две разные команды для копирования и подстановки.

Решение

Замените фрагменты вида:

```
$dst = $src;
$dst =~ s/this/that/;
```

следующей командой:

```
($dst = $src) =~ s/this/that/;
```

Комментарий

Иногда подстановка должна выполняться не в исходной строке, а в ее копии, однако вам не хочется делить ее на два этапа.

Например:

```
# Выделить базовое имя
($progname = $0)       =~ s!^.*/!!;

# Начинать Все Слова С Прописной Буквы
($capword = $word)     =~ s/(\w+)/\u\L$1/g;

# /usr/man/man3/foo.1 заменяется на /usr/man/man/cat3/foo.1
($catpage = $manpage) =" s/man(?=\d)/cat/;
```

Подобная методика работает даже с массивами:

```
@bindirs = qw( /usr/bin /bin /usr/local/bin );
for (@libdirs = @bindirs) { s/bin/lib/ >
print "@libdirs\n";
/usr/lib /lib /usr/local/lib
```

Если подстановка должна выполняться для правой переменной, а в левую заносится результат, следует изменить расположение скобок. Обычно результат подстановки равен либо "" в случае неудачи, либо количеству выполненных замен. Сравните С предыдущими примерами, где в скобки заключалась сама операция присваивания. Например:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
($a = $b) =~ s/x/y/g;  # Скопировать $b и затем изменить $a
 $a = ($b =~ s/x/y/g), # Изменить $b и занести в $ количество подстановок
```

Смотри также

Раздел "Assignment Operators" *perlop(1)*.

6.2. Идентификация алфавитных символов

Проблема

Требуется узнать, состоит ли строка только из алфавитных символов.

Решение

Наиболее очевидное решение не подходит для общего случая:

```
if ($var =~ /^[A-Za-z]+$/) {
    # Только алфавитные символы
}
```

Дело в том, что такой вариант не учитывает локальный контекст пользователя. Если наряду с обычными должны идентифицироваться символы с диакритическими знаками, воспользуйтесь директивой use locale и инвертированным символьным классом:

```
use locale;
if ($var =~ /^[^\W\d_]+$/) {
        var          alphabetic\n ,
}
```

Комментарий

В Perl понятие "алфавитный символ" тесно связано с локальным контекстом, поэтому нам придется немн схитрить. Регулярное выражение \w совпадает с одним алфавитным, или цифровым симво а также символом подчеркивания. Следовательно, \W не является одним из этих символов Инвертируемый символьный класс [^\W\d_] определяет байт, который не является алфавитным символом, цифрой или подчеркиванием. После инвертирования остаются одни алфавитные символы, которые нас и интересуют.

В программе это выглядит так:

```
use locale;
use                    POSIX                    locale_h

# На вашем компьютере строка локального, контекста, может выглядеть иначе
unless (setlocale(LC_ALL, "fr_CA.IS08859-1")) {
    die "couldn't set locale, to        Canadian\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    chomp;
    if (/^[^\W\d_]+$/) {
        print "$_. alphabetic\n";
    } else {
        print "$_ line noise\n";
    }
}
__END__
silly
fa3ade
cooperate
nico
Renée
Molière
hæmoglobin
naïve
tschüß
random'stuff#here
```

Смотри также

Описание работы с локальным контекстом в *perllocale(1);* страница руководства *locale(3)* вашей системы; рецепт 6.12.

6.3. Поиск слов

Проблема

Требуется выделить из строки отдельные слова.

Решение

Хорошенько подумайте, что должно считаться словом и как одно слово отделяется от остальных. Затем напишите регулярное выражение, в котором будут воплощены ваши решения. Например:

```
/\S+/          # Максимальная серия байтов, не являющихся пропусками
/[A-Za-z'-]+/  # Максимальная серия букв, апострофов и дефисов
```

Комментарий

Концепция "слова" зависит от приложения, языка и входного потока, поэтому в Perl не существует встроенного определения слов. Слова приходится собирать вручную из символьных классов и квантификаторов, как это сделано выше. Во втором примере мы пытаемся сделать так, чтобы "shepherd's" и "sheep-sheering" воспринимались как отдельные слова.

У большинства реализаций имеются ограничения, связанные с вольностями письменного языка. Например, хотя второй шаблон успешно опознает слова "spank'd" и "counter-clockwise", он выдернет "rd" из строки "23rd Psalom". Чтобы

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

повысить точность идентификации слов в строке, можно указать то, что окружает слово. Как правило, указываются метасимволы границ[1], а не пропусков:

```
/\b([A-Za-z]+\b/        # Обычно наилучший вариант
/\s([A-Za-z]+)\s/       # Не работает в конце строки или без знаков препинания
```

В Perl существует метасимвол \w, который совпадает с одним символом, разрешенным в идентификаторах Perl. Однако идентификаторы Perl редко отвечают нашим представлениям о словах — обычно имеется в виду последовательность алфавитно-цифровых символов и подчеркиваний, но не двоеточий с апострофами. Поскольку метасимвол \b определяется через \w, он может преподнести сюрпризы при определении границ английских слов (и тем более — слов языка суахили).

И все же метасимволы \b и \B могут пригодиться. Например, шаблон /\Bis\B/ совпадает со строкой "is" только внутри слова, но не на его границах. Скажем, в "thistle" совпадение будет найдено, а в "vis-á-vis" — нет.

Смотри также

Интерпретация \b, \w и \s в шаблоны для работы со словами из рецепта 6.23.

6.4. Комментирование регулярных выражений

Проблема

Требуется сделать ваше сложное регулярное выражение более понятным и упростить его изменение в будущем.

Решение

В вашем распоряжении четыре способа: внешние комментарии, внутренние комментарии с модификатором /x, внутренние комментарии в заменяющей части s/// и альтернативные ограничители.

Комментарий

Во фрагменте из примера 6.1 использованы все четыре способа. Начальный комментарий описывает, для чего предназначено регулярное выражение. Для относительно простых шаблонов ничего больше не потребуется. В сложных шаблонах (вроде приведенного) желательно привести дополнительные комментарии.

```
#!/usr/bin/perl -p
                    е имена в стиле "foo.bar.com" во входном потоке
```

[1] Хотя метасимвол \b выше был назван "границей слова", в действительности он определяется как позиция между двумя символами, по одну сторону которой располагается \w, а по другую — \W (в любом порядке). — *Примеч. перев.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# на "foo.bar.com [204.148.40.9]" (или аналогичными)

use Socket;                  # Загрузить inet_addr
s{                           #                ,
    (                        # Сохранить имя хоста в $1
        (?:                  # Скобки только для группировки
            (?!  [-_]   )    # Ни подчеркивание, ни дефис
            [\w-]  +         # Компонент имени хоста
            \.               # и точка домена
        ) +                  # Повторяется несколько раз
        [A-Za-z]             # Следующий символ должен быть буквой
        [\w-]  +             # Завершающая часть домена
    )                        # Конец записи $1
}{                           # Заменить следующим:
    "$1 " .                  # Исходная часть плюс пробел
            ( ($addr = gethostbyname($1))   # Если имеется адрес
             ? "[" . inet_ntoa($addr) . "]" #        отформатировать
             : "[???]"                      # иначе пометить как сомнительный
            )
}gex;                # /g - глобальная замена
                     # /e - выполнение
                     # /x - улучшенное форматирование
```

Для эстетов в этом примере использованы альтернативные ограничители. Когда шаблон поиска или замены растягивается на несколько строк, наличие парных скобок делает его более понятным. Другая частая причина для использования альтернативных ограничителей — присутствие в шаблоне символов / (например, s/\/\//\/.. \//g). Альтернативные ограничители упрощают чтение такого шаблона (например, s!//!/. ./!g или s{//}{/. ./}g).

При наличии модификатора /x **Perl** игнорирует большинство пропусков в шаблоне (в символьных классах они учитываются) и интерпретирует символы # и следующий за ними текст как комментарий. Такая возможность весьма полезна, однако у вас могут возникнуть проблемы, если пропуски или символы *tt* являются частью шаблона. В таких случаях снабдите символы префиксом \, как это сделано в следующем примере:

```
s/                # Заменить
  \#              #   знак фунта
  (\w+)           #   имя переменной
  \#              #   еще один знак фунта
/${$1}/xg;        # значением глобальной переменной
```

Помните: комментарий должен пояснять программу, а не пересказывать ее. Комментарии типа "$i++ *tt* Увеличить $i на 1" станут причиной плохих оценок на курсах программирования или подорвут вашу репутацию среди коллег.

Остается модификатор /e, при котором заменяющая строка вычисляется как полноценное выражение Perl, а не как (заключенная в кавычки и интерполированная) строка. Результат выполнения этого кода используется в качестве заменяю-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

щей строки. Поскольку выражение будет интерпретировано как программный код, оно может содержать комментарии. Это несколько замедляет работу программы, но не так сильно, как может показаться (пока вы не начали писать собственные тесты, желательно представлять себе эффективность тех или иных конструкций). Дело в том, что правая сторона подстановки проверяется и компилируется на стадии компиляции вместе со всей программой. Для простой замены строк это, пожалуй, перебор, но в более сложных случаях работает просто замечательно.

Удвоение /e напоминает конструкцию eval "STRING". Это позволит применить лексические переменные вместо глобальных в предыдущем примере с заменой.

```
s/                  # Заменить
  \#                #   знак фунта
  (\w+)             #   имя переменной
  \#                #   еще один знак фунта
/'$' . $1/xeeg,  # значением "любой* переменной
```

После подстановки /ee проверьте переменную $@. Она содержит сообщения об ошибках, полученные в результате работы вашего кода, — в отличие от /e, в данном случае код действительно генерируется во время работы программы.

Смотри также

Описание модификатора /x в

6.5. Поиск N-го совпадения

Проблема

Требуется найти не первое, а N-е совпадение шаблона в строке. Допустим, вы хотите узнать, какое слово предшествует третьему экземпляру слова fish:

```
One fish two fish      fish blue fish
```

Решение

Воспользуйтесь модификатором /g и считайте совпадения в цикле while:

```
$WANT = 3;
$count = 0;
while (/(\w+)\s+fish\b/gi) {
    if (++$count == $WANT) {
        print "The third fish is a $1 one.\n";
        # Предупреждение: не выходите из этого цикла с помощью last
    }
}
The third fish is a red one.
```

Или воспользуйтесь счетчиком и шаблоном следующего вида:

```
/(?:\w+\s+fish\s+){2}(\w+)\s+fish/i;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Как объяснялось во введении к этой главе, при наличии модификатора /g в скалярном контексте происходит многократный поиск. Его удобно использовать в циклах while — например, для подсчета совпадений в строке:

```
" Простой вариант с циклом while
$count = 0;
while($string =^ /PAT/g) {
    $count++;    # Или что-нибудь другое
}
# То же с завершающим циклом while
$count = 0;
$count++ while $string =~ /PAT/g;

# С циклом for
for ($count = 0; $string =^ /PAT/g; $count++) { }

# Аналогично, но с подсчетом перекрывающихся совпадений
$count++ while $string =^ /(?=PAT)/g;
```

Чтобы найти N-й экземпляр, проще всего завести отдельный счетчик. Когда он достигнет N, сделайте то, что считаете нужным. Аналогичная методика может применяться и для поиска каждого N-го совпадения — в этом случае проверяется кратность счетчика N посредством вычисления остатка при делении. Например, проверка (++$count % 3) == 0 находит каждое третье совпадение.

Если вам не хочется брать на себя дополнительные хлопоты, всегда можно извлечь все совпадения и затем выбрать из них то, что вас интересует.

```
$pond = 'One fish two fish red fish blue fish';

# С применением временного массива
@colors = ($pond =~ /(w+)\s+fish\b\gi);  # Найти все совпадения
$color = $colors[2];                     в Выбрать одно,
                                          # интересующее нас

# Без временного массива
$color = ( $pond =^ /(\w+)\s+fish\b/gi )[2];  # Выбрать третий элемент

print "The third fish is the pond is $color.\n";
The third fish in the pond Is
```

В другом примере находятся все нечетные совпадения:

```
$count = 0;
$_ = 'One fish two fish red fish blue fish';
@evens = grep {$count++ % 2 == 1} /(\w+)\s+fish\b/gi;
print "Even fish are @evens.\n";
Even numbered fish are two blue.
```

При подстановке заменяющая строка должна представлять собой программное выражение, которое возвращает соответствующую строку. Не забывайте возвращать оригинал как заменяющую строку в том случае, если замена не нужна. В следующем примере мы ищем четвертый экземпляр "fish" и заменяем предшествующее слово другим:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$count = 0;
s{
  \b
  ( \w+)
  (
     \s+ fish \b
  )
}{
  if (++$count == 4) {
    "sushi" . $2;
  } else {
        $1 . $2;
  }
}gex;
```

```
sushi fish
```

Задача поиска последнего совпадения также встречается довольно часто. Простейшее решение — пропустить все начало строки. Например, после /.*\b(\w+)\s+fish\b/ переменная $1 будет содержать слово, предшествующее последнему экземпляру "fish".

Другой способ — глобальный поиск в списковом контексте для получения всех совпадений и последующее извлечение нужного элемента этого списка:

```
                                        fish      here.';
$color = ( $pond =^ /\b(\w+)\s+fish\b/gi )[-1];
print "Last fish is $color.\n";
Last fish is blue.
```

Если потребуется найти последнее совпадение без применения /g, то же самое можно сделать с отрицательной опережающей проверкой (?! НЕЧТО). Если вас интересует последний экземпляр произвольного шаблона **A,** вы ищете A, сопровождаемый **любым** количеством "не-A", до конца строки. Обобщенная конструкция имеет вид **A(?! .*A)*$,** однако для удобства чтения ее можно разделить:

```
m{
    A           # Найти некоторый шаблон A
    (?!         # При этом не должно находиться
        .*      # что-то другое
        A       # и A
    )
    $           # До конца строки
>x
```

В результате поиск последнего экземпляра "fish" принимает следующий вид:

```
                                        fish';
if ($pond =^ m{
                    \b  (  \w+) \s+ fish \b
                  (?!  .* \b fish \b )
              }six )
{
    print "Last fish is $1/\n";
} else {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    print "Failed!\n";
}
Last fish is blue.
```

Такой подход имеет свои преимущества — он ограничивается одним шаблоном и потому подходит для ситуаций, аналогичных описанной в рецепте 6.17. Впрочем, имеются и недостатки. Он однозначно труднее записывается и воспринимается — впрочем, если общий принцип понятен, все выглядит не так плохо. К тому же это решение медленнее работает — для протестированного набора данных быстродействие снижается примерно в два раза.

Смотри также

Поведение конструкции m//g в скалярном контексте описано в разделе "Regexp Quote-like Operators>>perlfop(l). Отрицательные опережающие проверки нулевой ширины продемонстрированы в разделе "Regular

6.6. Межстрочный поиск

Проблема

Требуется использовать регулярные выражения для последовательности, состоящей из нескольких строк. Специальные символы . (любой символ, кроме перевода строки), ^ (начало строки) и $ (конец строки), кажется, не работают. Это может произойти при одновременной чтении нескольких записей или всего содержимого файла.

Решение

Воспользуйтесь модификатором /m, /s или обоими сразу. Модификатор /s разрешает совпадение . с переводом строки (обычно этого не происходит). Если последовательность состоит из нескольких строк, шаблон `/foo.*bar/s` совпадет с `"foo"` и `"bar"`, находящимися в двух соседних строках. Это не относится к точкам в символьных классах (например, `[#%.]`), которые всегда представляют собой обычные точки.

Модификатор /m разрешает совпадение ^ и $ в переводах строк. Например, совпадение для шаблона `/^=head[1-7]$/m` возможно не только в начале записи, но и в любом из внутренних переводов строк.

Комментарий

При синтаксическом анализе документов, в которых переводы строк не имеют значения, часто используется "силовое" решение — файл читается по абзацам (а иногда даже целиком), после чего происходит последовательное извлечение лексем. Для успешного межстрочного поиска необходимо, чтобы символ . совпадал с переводом строки — обычно этого не происходит. Если в буфер читается сразу несколько строк, вероятно, вы предпочтете, чтобы символы ^ и $ совпадали с началом и концом внутренних строк, а не всего буфера.

Необходимо хорошо понимать, чем /m отличается от /s: первый заставляет ^ и $ совпадать на внутренних переводах строк, а второй заставляет . совпадать с пере-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

водом строки. Эти модификаторы можно использовать вместе, они не являются взаимоисключающими.

Фильтр из примера **6.2** удаляет теги **HTML** из всех файлов, переданных в @ARGV, и отправляет результат **в STDOUT.** Сначала мы отменяем разделение записей, чтобы при каждой операции чтения читалось содержимое всего файла. Если @ARGV содержит несколько аргументов, файлов также будет несколько. В этом случае при каждом чтении передается содержимое всего файла. Затем мы удаляем все открывающие и закрывающие угловые скобки и все, что находится между ними. Мы не можем просто воспользоваться .* по двум причинам: во-первых, этот шаблон не учитывает закрывающих угловых скобок, а во-вторых, он не поддерживает межстрочных совпадений. Проблема решается применением .*? в сочетании с модификатором /s — по крайней мере, в данном случае.

Пример *6.2.* killtags

```
#!/usr/bin/perl
# killtags - очень плохое удаление тегов HTML
undef $/;            # При каждом чтении передается весь файл
while (<>) {         # Читать по одному файлу
    s/<.*?>//gs;     # Удаление тегов (очень скверное)
    print;           # Вывод файла в STDOUT
}
```

Шаблон s/<[^>]*>//g работает намного быстрее, но такой подход наивен: он приведет к неправильной обработке тегов **в** комментариях **HTML** или угловых скобок в кавычках (<IMG SRC="here.gif" ALT="<<Ooh la la!>>">). В рецепте 20.6 показано, как решаются подобные проблемы.

Программа из примера 6.3 получает простой текстовый документ и ищет в начале абзацев строки вида "Chapter 20: Better Living Through Chemisery". Такие строки оформляются заголовками **HTML** первого уровня. Поскольку шаблон получился довольно сложным, мы воспользовались модификатором /x, который разрешает внутренние пропуски и комментарии.

Пример 6.3. headerfy

```
#!/usr/bin/perl
# headerfy: оформление заголовков глав в HTML
$/ = '';
while ( <> ) {                # Получить абзац
    s{
        \A                    # Начало записи
        (                     # Сохранить в $1
            Chapter           # Текстовая строка
          . \s+               # Обязательный пропуск
            \d+               # Десятичное число
            \s*               # Необязательный пропуск
            :                 # Двоеточие
            . *               # Все, кроме перевода строки, до конца строки
        )
    }{<H1>$1</H1>}gx;
    print;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если комментарии лишь затрудняют понимание, ниже тот жепример переписан в виде короткой командной строки:

```
% perl -OOpe 's{\A(Chapter\s+\d+\s*:.*)}{<H1>$1</H1>}gx' datafile
```

Возникает интересная проблема: **в** одном шаблоне требуется указывать как начало записи, так и конец строки. Начало записи можно было **бы** определить с помощью ^, но символ $ должен определять не только конец записи, но и конец строки. **Мы** добавляем модификатор /m, отчего изменяется смысл как ^, так и $. Начало записи вместо ^ определяется с помощью \A. Кстати говоря, метасимвол \Z (хотя **в** нашем примере он не используется) совпадает с концом записи даже при наличии модификатора /m.

Следующий пример демонстрирует совместное применение /s и /m. На этот раз мы хотим, чтобы символ ^ совпадал с началом любой строки абзаца, а точка — с переводом строки. Эти модификаторы никак не связаны, и их совместное применение ничем не ограничено. Стандартная переменная $. содержит число записей последнего прочитанного файла. Стандартная переменная $ARGV содержит файл, автоматически открываемый при обработке <ARGV>.

```
$/ = '';        # Режим чтения абзацев
while (<ARGV>) {
    while (m#^START(.*?)^END#sm) { # /s - совпадение . с переводом строки
                                   # /m - совпадение ^ с началом
                                         внутренних строк
        print "chunk $. in $ARGV has <<$1>>\n";
    }
}
```

Если **вы** уже привыкли работать с модификатором /m, то ^ и $ можно заменить на \A и \Z. Но что делать, если **вы** предпочитаете /s и хотите сохранить исходный смысл **.?** Воспользуйтесь конструкцией [^\n]. Если **вы** не намерены использовать */s,* но хотите иметь конструкцию, совпадающую с любым байтом, сконструируйте символьный класс вида [\000-\377] или даже [\d\D]. Использовать [.\n] нельзя, поскольку **в** символьных классах . не обладает особой интерпретацией.

Смотри также

Описание переменной $/ в perlvar(1); описание модификаторов /s и */m в*
Мы вернемся **к** специальной переменной $/ **в** главе **8.**

6.7. Чтение записей с разделением по шаблону

Проблема

Требуется прочитать записи, разделение которых описывается некоторым шаблоном. Perl не позволяет присвоить регулярное выражение переменной-разделителю входных записей.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Многие проблемы — в первую очередь связанные с синтаксическим анализом сложных файловых форматов, — заметно упрощаются, если у вас имеются удобные средства для чтения записей, разделенных в соответствии с определенным шаблоном.

Решение

Прочитайте весь файл и воспользуйтесь функцией split:

```
undef $/;
@chunks = split(/шаблон/,<ФАЙЛОВЫЙ_МАНИПУЛЯТОР>);
```

Комментарий

Разделитель записей Perl должен быть фиксированной строкой, а не шаблоном (ведь должен awk быть *хоть в чем-то* лучше!). Чтобы обойти это ограничение, отмените разделитель входных записей, чтобы следующая операция чтения прочитала весь файл. Иногда это называется *режимом поглощающего ввода* (slurp mode), потому что весь файл поглощается как одна большая строка. Затем разделите эту большую строку функцией split, используя шаблон разделения записей в качестве первого аргумента.

Рассмотрим пример. Допустим, входной поток представляет собой текстовый файл, содержащий строки ".Se", ".Ch" и ".Ss" — служебные коды для макросов *troff*. Эти строки представляют собой разделители. Мы хотим найти текст, расположенный между ними.

```
# .Ch, .Se и .Ss отделяют фрагменты данных STDIN
<
    local $/ = undef;
    @chunks = split(/^\.(Ch|Se|Ss)$/m, o);
}
print "I read ", scalar(@chunks), "chunks.\n";
```

Мы создаем локальную версию переменной $/, чтобы после завершения блока было восстановлено ее прежнее значение. Если шаблон содержит круглые скобки, функция split также возвращает разделители. Это означает, что данные в возвращаемом списке будут чередоваться с элементами "Se", "Ch" и "Ss".

Если разделители вам не нужны, но вы все равно хотите использовать круглые скобки, воспользуйтесь "несохраняющими" скобками в шаблоне вида /^\.(?:Ch|Se|Ss)$/m.

Чтобы записи разделялись *перед* шаблоном, но шаблон включался в возвращаемые записи, воспользуйтесь опережающей проверкой: /?(?=\'.(?:Ch|Se|Ss))/m. В этом случае каждый фрагмент будет начинаться со строки-разделителя.

Учтите, что для больших файлов такое решение потребует значительных расходов памяти. Однако для современных компьютеров и типичных текстовых файлов эта проблема уже не так серьезна. Конечно, нс стоит применять это решение для 200-мегабайтного файла журнала, не располагая достаточным местом на диске для подкачки. Впрочем, даже при избытке виртуальной памяти ничего хорошего не выйдет.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

▷ Смотри также

Описание переменной $/ *в perlvar(l)* и в главе 8; описание функции split в *perlfunc(1)*.

6.8. Извлечение строк из определенного интервала

Требуется извлечь все строки, расположенные в определенном интервале. Интервал может быть задан двумя шаблонами (начальным и конечным) или номером первой и последней строки.

Часто встречающиеся примеры — чтение первых 10 строк файла (строки с 1 по 10) или основного текста почтового сообщения (все, что следует после пустой строки).

Решение

Используйте оператор . . или . . . для шаблонов или номеров строк. В отличие от . . оператор . . . не возвращает истинное значение, если оба условия выполняются в одной строке.

```
while (o) {
    if (/НАЧАЛЬНЫЙ ШАБЛОН/ .. /КОНЕЧНЫЙ ШАБЛОН/) {
        # Строка находится между начальным
        # и конечным шаблонами включительно.
    }
}

while (<>) <
    if ($НОМЕР_НАЧАЛЬНОЙ_СТРОКИ .. $НОМЕР_КОНЕЧНОЙ_СТРОКИ) {
        # Строка находится между начальной
        # и конечной включительно.
    }
>
```

Если первое условие оказывается истинным, оператор . . . не проверяет второе условие.

```
while (<>) {
    if (/НАЧАЛЬНЫЙ ШАБЛОН/ ... /КОНЕЧНЫЙ ШАБЛОН/) {
        # Строка находится между начальным
        # и конечным шаблонами, расположенными в разных строках.
    }
}

while (o) {
    if ($НОМЕР_НАЧАЛЬНОЙ_СТРОКИ ... $НОМЕР_КОНЕЧНОЙ_СТРОКИ) {
        # Строка находится между начальной
        # и конечной, расположенными в разных строках.
    >
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Из бесчисленных операторов Perl интервальные операторы `..` и `...`, вероятно, вызывают больше всего недоразумений. Они создавались для упрощения выборки интервалов строк, чтобы программисту не приходилось сохранять информацию о состоянии. В скалярном контексте (например, в условиях операторов if и `while`) эти операторы возвращают true или false, отчасти зависящее от предыдущего состояния. Выражение `левый_операнд .. правый_операнд` возвращает false до тех пор, пока `левый_операнд` не станет истинным. Когда это условие выполняется, `левый_операнд` перестает вычисляться, а оператор возвращает `true` до тех пор, пока не станет истинным правый операнд. После этого цикл начинается заново. Другими словами, истинность первого операнда "включает" конструкцию, а истинность второго операнда "выключает" ее.

Условия могут быть абсолютно произвольными. В сущности, границы интервала могут быть заданы проверочными функциями `mytestfunc(1) .. mytestfunc(2)`, хотя на практике это происходит редко. Как правило, операндами интервальных операторов являются либо номера строк (первый пример), шаблоны (второй пример) или их комбинация.

```
# Командная строка для вывода строк с 15 по 17 включительно (см. ниже)
perl -ne 'print if 15 .. 17' datafile

# Вывод всех фрагментов <XMP> .. </XMP> из документа HTML
while (<>) {
    print if m#<XMP>#i .. m#</XMP>#i;
}

# То же, но в виде команды интерпретатора:
% perl -ne 'print if m#<XMP>#i .. m#</XMP>#i' document.html
```

Если хотя бы один из операндов задан в виде числовой константы, интервальные операторы осуществляют неявное сравнение с переменной $. ($NR или `$INPUT_LINE_NUMBER` при действующей директиве use English). Поосторожнее с неявными числовыми сравнениями! В программе необходимо указывать числовые константы, а не переменные. Это означает, что в условии можно написать `3 .. 5`, но не `$n .. $m`, даже если значения `$n` и `$m` равны 3 и 5 соответственно. Вам придется непосредственно проверить переменную `$.`.

```
# Команда не работает
perl -ne 'BEGIN { $top=3; $bottom=5 } print if $top .. $bottom' /etc/passwd
# Работает
perl -ne 'BEGIN {$top=3; $bottom=5 } \
    print if $. == $top .. $. == $bottom' /etc/passwd
# Тоже работает
perl -ne 'print if 3 .. 5' /etc/passwd
```

Операторы `..` и `...` отличаются своим поведением в том случае, если оба операнда могут оказаться истинными в одной строке. Рассмотрим два случая:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
print if /begin/ .. /end/;
print if /begin/ ... /end/;
```

Для строки "You begin" оба интервальных оператора возвращают true. Однако оператор .. не будет выводить дальнейшие строки. Дело в том, что после выполнения первого условия он проверяет второе условие в той же строке; вторая проверка сообщает о найденном конце интервала. С другой стороны, оператор ... продолжит поиск до *следующей* строки, в которой найдется /end/, — онникогда не проверяет оба операнда одновременно.

Разнотипные условия можно смешивать:

```
while (<>) {
    $in_header =   1 .. /^$/;
    $in_body   = /^$/ .. eof();
}
```

Переменная $in_header будет истинной, начиная с первой входной строки и заканчивая пустой строкой, отделяющей заголовок от основного текста, — например, в почтовых сообщениях, новостях Usenet и даже в заголовках HTTP (теоретически строки в заголовках HTTP должны завершаться комбинацией CR/LF, но на практике серверы относятся к их формату весьма либерально). Переменная $in_body становится истинной в момент обнаружения первой пустой строки и до конца файла. Поскольку интервальные операторы не перепроверяют начальное условие, остальные пустые строки (например, между абзацами) игнорируются.

Рассмотрим пример. Следующий фрагмент читает файлы с почтовыми сообщениями и выводит адреса, найденные в заголовках. Каждый адрес выводится один раз. Заголовок начинается строкой "From:" и завершается первой пустой строкой. Хотя это определение и не соответствует RFC-822, оно легко формулируется.

```
%seen = ();
while (o) {
    next unless /^From:?\s/i .. /"$/;
    while (/([^<>(),;\s]+\@[^<>(),;\s]+)/g) {
        print "$1\n" unless $seen{$1}++;
    }
```

Если интервальные операторы Perl покажутся вам странными, записывайтесь в команды поддержки *s2p* и *a2p* — трансляторов для переноса кода sed и awk в Perl. В обоих языках есть свои интервальные операторы, которые должны работать в Perl.

Смотри также

Описание операторов .. и ... в разделе "Range Operator" *perlop(1);* описание переменной $NR в *perlvar(1).*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

6.9. Работа с универсальными символами командных интерпретаторов

Проблема

Вы хотите, чтобы вместо регулярных выражений Perl пользователи могли выполнять поиск с помощью традиционныхуниверсальных символов командного интерпретатора. В тривиальных случаях шаблоны с универсальными символами выглядят проще, нежели полноценные регулярные выражения.

Решение

Следующая подпрограмма преобразует четыре универсальных символа командного интерпретатора в эквивалентные регулярные выражения; все остальные символы интерпретируются как строки.

```
sub glob2pat {
    my $globstr = shift;
    my %patmap = (
         '*' => '.*',
         '?' => '.',
        '[' => T,
         ']' => ']'
    );
    $globstr =^ s{(.)} { $patmap{$1} || "\Q$1" }ge;
                $globstr . '$';
}
```

Комментарий

Шаблоны Perl отличаются от применяемых в командных интерпретаторах конструкций с универсальными символами. Конструкция *.* интерпретатора не является допустимым регулярным выражением. Она соответствует шаблону /^. *\. . *$/, который совершенно не хочется вводить с клавиатуры.

Функция, приведенная в решении, выполняет все преобразования за вас. При этом используются стандартные правила встроенной функции glob.

Интерпретатор	Perl
list.?	^list\..$
project.*	^project\..*$
*old	".*old$
type*,[ch]	^type.*\.[ch]$
	^.*\..*$
_*	^.*$

В интерпретаторе действуют другие правила. Шаблон неявно закрепляется на концах строки. Вопросительный знак соответствует любому символу, звездочка —

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

произвольному количеству любых символов, а квадратные скобки определяют интервалы. Все остальное, как обычно.

Большинство интерпретаторов не ограничивается простыми обобщениями в одном каталоге. Например, конструкция */* означает: "все файлы во всех подкаталогах текущего каталога". Более того, большинство интерпретаторов не выводит имена файлов, начинающиеся с точки, если точка не была явно включена в шаблон поиска. Функция glob2pat такими возможностями не обладает, если они нужны — воспользуйтесь модулем File::KGlob с CPAN.

Смотри также

Страницы руководства *csh*(1) и *ksh*(1) вашей системы; описание функции glob в *perlfunc(1);* документация по модулю Glob::DosGlob от CPAN; раздел "I/O Operators" *perlop(1);* рецепт 9.6.

6.10. Ускорение интерполированного поиска

Проблема

Требуется, чтобы одно или несколько регулярных выражений передавались в качестве аргументов функции или программы. Однако такой вариант работает медленнее, чем при использовании литералов.

Решение

Если имеется всего один шаблон, который не изменяется в течение всей работы программы, сохраните его в строке и воспользуйтесь шаблоном `/$pattern/o`:

```
while ($line = o) {
    if ($line =^ /$pattern/o) {
        # Сделать что-то
    }
}
```

Однако для нескольких шаблонов это решение не работает. Три приема, описанные в комментарии, позволяют ускорить поиск на порядок или около того.

Комментарий

Во время компиляции программы Perl преобразует шаблоны во внутреннее представление. На стадии компиляции преобразуются шаблоны, не содержащие переменных, однако преобразование шаблонов с переменными происходит во время выполнения. В результате интерполяция переменных в шаблонах (например, `/$pattern/`) замедляет работу программы. Это особенно заметно при частых изменениях `$pattern`.

Применяя модификатор /o, автор сценария гарантирует, что значения интерполируемых в шаблоне переменных остаются неизменными, а если они все же

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

изменятся, Perl будет использовать прежние значения. Получив такие гарантии, Perl интерполирует переменную и компилирует шаблон лишь при первом поиске. Но если интерполированная переменная изменится, Perl этого не заметит. Применение модификатора к изменяющимся переменным дастневерный результат.

Модификатор /o в шаблонах без интерполированных переменных не дает никакого выигрыша в скорости. Кроме того, он бесполезен в ситуации, когда у вас имеется неизвестное количество регулярных выражений и строка должна поочередно сравниваться со всеми шаблонами. Не поможет он и тогда, когда интерполируемая переменная является аргументом функции, поскольку при каждом вызове функции ей присваивается новое значение.

В примере 6.4показана медленная, но очень простая методика многострочного поиска для неск ольких шаблонов. Массив @popstates содержит стандартные сокращенные названия тех штатов, в которых безалкогольные газированные напитки обозначаются словом *pop*. Задача — вывести все строки входного потока, в которых хотя бы одно из этих сокращений присутствует в виде отдельного слова. Модификатор /o не подходит,поскольку переменная, содержащая шаблон, постоянно изменяется.

```
            ➡ поиск строк с названиями штатов
# версия 1: медленная, но понятная
@popstates = qw(CO ON MI WI MN);
LINE: while (defined($line = <>)) {
    for $state (@popstates) {
        if ($line =~ /\b$state\b/) {
            print, next LINE,
        }
    }
}
```

Столь примитивное, убогое, "силовое" решение оказывается ужасно медленным — для каждой входной строки все шаблоны приходится перекомпилировать заново. Мы рассмотрим три варианта решения этой проблемы. Первый вариант генерирует строку кода Perl и вычисляет ее с помощью eval; второй кэширует внутренние представления регулярных выражений в замыканиях; третий использует модуль Regexp с CPAN для хранения откомпилированных регулярных выражений.

Традиционный подход к ускорению многократного поиска в Perl — построение строки, содержащей нужный код, и последующий вызов eval "$code". Подобная методика использована в примере 6.5.

```
#'/usr/bin/perl
                    строк с названиями штатов
# версия 2: eval; быстрая, но сложная в написании
@popstates = qw(CO ON MI WI MN);
$code = 'while (defined($line = <>)) {';
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
for $state (@popstates) {
    $code .= "\tif (\$line =^ /\\b$state\\b/) { print \$line; next; }\n";
}
$code .= '}';
print "CODE IS\n    \n$code\n   \n" if 0;  # Отладочный вывод
eval $code;
die if $@;
```

генерирует строки следующего вида:

```
while (defined($line = <>) {
    if ($line =^ /bCO\b/) { print $line; next; }
    if ($line =^ /bON\b/) { print $line; next; }
    if ($line =^ /bMI\b/) { print $line; next; }
    if ($line =^ /bWI\b/) { print $line; next; }
    if ($line =~ /bMN\b/) { print $line; next; }
}
```

Как видите, получается что-то вроде строковых констант, вычисляемых eval. В текст включен весь цикл вместе с поиском по шаблону, что ускоряет работу программы.

Самое неприятное в таком решении — то, что правильно записать все строки и служебные символы довольно трудно. Функция dequote из рецепта 1.11 может упростить чтение программы, но проблема с конструированием переменных, используемых позже, остается насущной. Кроме того, в строках нельзя использовать символ /, поскольку он служит ограничителем в операторе m//.

Существует изящный выход, впервые предложенный Джеффри Фридлом (JeffreyFriedl). Он сводится к построению анонимной функции, которая кэширует откомпилированные шаблоны в созданном ей замыкании. Для этого функция eval вызывается для строки, содержащей определение анонимной функции, которая проверяет совпадения с передаваемыми ей шаблонами. Perl компилирует шаблон всего только при определении анонимной функции. После вызова eval появляется возможность относительно быстрого поиска.

В примере 6.6 приведена очередная версия программы ро которой используется данный прием.

6.6.

```
#!/usr/bin/perl
                поиск строк с названиями штатов
# версия 3: алгоритм с построением вспомогательной функции
@popstates = qw(CO ON MI WI MN);
    $expr = join('||', map { "m/\\b\$popstates[$_]\\b/o" } 0..$#popstates);
$match_any = eval "sub { $expr }";
die if $@;
while (<>) {
    print if &$match_any;
}
```

В результате функции eval передается следующая строка (за вычетом форматирования):

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub {
    m/\b$popstates[0]\b/o || m/\b$popstates[1]\b/o ||
    m/\b$popstates[2]\b/o || m/\b$popstates[3]\b/o ||
    m/\b$popstates[4]\b/o
}
```

Ссылка на массив @popstates находится внутри замыкания. Применение модификатора /o в данном случае безопасно.

Пример 6.7 представляет собой обобщенный вариант этой методики. Создаваемые в нем функции возвращают true, если происходит совпадение хотя бы с одним (и более) шаблоном.

Пример 6.7.

```
#!/usr/bin/perl
            вывод строк, в которых присутствуют Tom и Nat

$multimatch = build_match_all(q/Tom/, q/Nat/),
while (<>) {
    print if &$multimatch,
}
exit,

sub build_match_any { build_match_func( || , @_) }
sub build_match_all { build_match_func( && , @_) }
sub build_match_func {
    my $condition = shift,
    my @pattern = @_   # Переменная должна быть лексической,
                       # а не динамической
    my $expr = join $condition => map { m/\$pattern[$_]/o } (0 $#pattern),
    my $match_func = eval sub { local \$_ = shift if \@_, $expr } ,
    die if $@   # Проверить $@, переменная должна быть пустой!
            $match_func,
}
```

Конечно, вызов eval для интерполированных строк (см. представляет собой фокус, кое-как но работающий. Зато применениелексических переменных в замыканиях, как в функциях build_match_*, — это уже высший пилотаж. Даже матерый программист Perl не сразу поверит, что такое решение действительно работает Впрочем, программа будет работать независимо от того, поверили в нее или нет.

На самом деле нам хотелось бы, чтобы Perl один раз компилировал каждый шаблон и позволял позднее ссылаться на него в откомпилированном виде. Такая возможность появилась в версии 5 005 в виде оператора определения регулярных выражений qr//. В предыдущих версиях для этого был разработан экспериментальный модуль Regexp с CPAN Объекты, создаваемые этим модулем, представляют откомпилированные регулярные выражения. При вызове метода match объекты выполняют поиск по шаблону в строковом аргументе. Существуют специальные методы для извлечения обратных ссылок, определения позиции совпадения и передачи флагов, соответствующих определенным модификаторам — например, /i.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В примере 6.8 приведена версия программы демонстрирующая простейшее применение этого модуля.

Пример 6.8.

```
#!/usr/bin/perl
                поиск строк с названиями штатов
# версия 4: применение модуля Regexp
use Regexp;
@popstates = qw(CO ON MI WIMN);
@poppats   = map { Regexp->new( '\b' . $_ . '\b') } @popstates;
while(defined($line = <>)) {
    for $patobj (@poppats) {
        print $line if $patobj->match($line);
    }
}
```

Возможно, вам захочется сравнить эти решения по скорости. Текстовый файл, состоящий из 22 000 строк ("файл Жаргона"), был обработан версией 1 за 7,92 секунды, версией 2 — всего за 0,53 секунды, версией 3 — за 0,79 секунды и версией 4 — за 1,74 секунды. Последний вариант намного понятнее других, хотя и работает несколько медленнее. Кроме того, он более универсален.

Смотри также

Описание интерполяции в разделе "Scalar Value Constructors" *perldata(1);* описание модификатора /o в по модулю Regexp с CPAN.

6.11. Проверка правильности шаблона

Проблема

Требуется, чтобы пользователь мог ввести свой собственный шаблон. Однако первая же попытка применить неправильный шаблон приведет к аварийному завершению программы.

Решение

Сначала проверьте шаблон с помощью конструкции eval {} для какой-нибудь фиктивной строки. Если переменная $@ не устанавливается, следовательно, исключение не произошло и шаблон был успешно откомпилирован. Следующий цикл работает до тех пор, пока пользователь не введет правильный шаблон.

```
do {
    print "Pattern?";
    chomp($pat = <>);
    eval { "" =~ /$pat/ };
    warn "INVALID PATTERN $@" if $@;
} while $@;
```

Отдельная функция для проверки шаблона выглядит так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub is_valid_pattern {
    my $pat = shift;
        eval { "" =^ /$pat/; 1 } || 0;
}
```

Работа функции основана на том, что при успешном завершении блока возвращается 1. При возникновении исключения этого никогда не произойдет. '

Комментарий

Некомпилируемые шаблоны встречаются сплошь и рядом. Пользователь может по ошибке ввести "<Is*[^>", "*** GET RICH ***" или "+5-i". Если слепо воспользоваться введенным шаблоном в программе, возникнет исключение — как правило, это приводит к аварийному завершению программы.

Крошечная программа из примера 6.9 показывает, как проверяются шаблоны.

Пример 6.9.

```
#!/usr/bin/perl
#           простейший поиск
die "usage: $0 pat [files]\n" unless @ARGV;
$/ = '';
$pat = shift;
eval { "" =~ /$pat/; 1 }     or die "$0: Bad pattern $pat: $@\n";
while (o) {
    print "$ARGV $.: $_" if /$pat/o;
}
```

Модификатор /o обещает Perl, что значение интерполируемой переменной останется постоянным во время всей работы программы — это фокус для повышения быстродействия. Даже если значение $pat изменится, Perl этого не заметит.

Проверку можно инкапсулиро вать в функции, которая возвращает 1 при успешном завершении блока и 0 в противном случае (см. выше функцию is_valid_pattern). Хотяисключение можно также перехватить с помощью eval "/$pat/", у такого решения есть два недостатка. Во-первых, во введенной пользователем строке не должно быть символов / (или других выбранных ограничителей). Во-вторых, в системе безопасности открывается зияющая брешь, которую было бы крайне желательно избежать. Некоторые строки могут сильно испортить настроение:

```
$pat = "You lose @{[ system('rm -rf *')]} big here";
```

Если вы не желаете предоставлять пользователю настоящие шаблоны, сначала всегда можно выполнить метапреобразование строки:

```
$safe_pat = quotemeta($pat);
something() if /$safe_pat/;
```

Или еще проще:

```
something() if AQ$pat/;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Но если вы делаете нечто подобное, зачем вообще связываться с поиском по шаблону? В таких случаях достаточно простого применения index.

Разрешая пользователю вводить настоящие шаблоны, вы открываете перед ним много интересных и полезных возможностей. Это, конечно, хорошо. Просто придется проявить некоторую осторожность, вот и все. Допустим, пользователь желает выполнять поиск без учета регистра, а вы не предусмотрели в своей программе параметр вроде -i в Работая с полными шаблонами, пользователь сможет ввести внутренний модификатор /i в виде (?i) — например, /(?i)stuff/.

Что произойдет, если в результате интерполяции получается пустая строка? Если $pat — пустая строка, с чем совпадет /$pat/ — иначе говоря, что произойдет при пустом поиске //? С началом любой возможной строки? Неправильно. Как ни странно, при поиске по пустому шаблону повторно используется шаблон предыдущего успешного поиска. Подобная семантика выглядит сомнительно, и ее практическое использование в Perl затруднительно.

Даже если шаблон проверяется с помощью eval, учтите: время поиска по некоторым шаблонам связано с длиной строки экспоненциальной зависимостью. Надежно идентифицировать такие шаблоны не удается. Если пользователь введет один из них, программа надолго задумается и покажется "зависшей". Возможно, из тупика можно выйти с помощью установленного таймера, однако в версии 5.004 прерывание работы Perl в неподходящий момент может привести к аварийному завершению.

Смотри также

Описание функции eval в *perlfunc(1)*.

6.12. Локальный контекст в регулярных выражениях

Проблема

Требуется преобразовать регистр в другом локальном контексте или заставить метасимвол \w совпадать с символами национальных алфавитов — например, *José* или *déjá vu*.

Предположим, у вас имеется полгигабайта текста на немецком языке, для которого необходимо составить предметный указатель. Вы хотите извлекать слова (с помощью \w+) и преобразовывать их в нижний регистр (с помощью lc или \L). Однако обычные версии \w и lc не находят слова немецкого языка и не изменяют регистр символов с диакритическими знаками.

Решение

Регулярные выражения и функции обработки текста Perl имеют доступ к локальному контексту POSIX. Если включить в программу директиву use locale, Perl

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

позаботится о символах национальных алфавитов — конечно, при наличии разумной спецификации LC_CTYPE и системной поддержки.

```
use locale,
```

Комментарий

По умолчанию \w+ и функции преобразования регистра работают с буквами верхнего и нижнего регистров, цифрами и подчеркиваниями. Преобразуются лишь простейшие английские слова, и даже в очень распространенных заимствованных словах происходят сбои. Директива use locale помогает справиться с затруднениями.

Пример 6.10 показывает, чем отличаются выходные данные для английского (en) и немецкого (de) локальных контекстов.

Пример 6.10. localeg

```
#!/usr/bin/perl -w
# localeg - выбор локального контекста

use locale,
use POSIX  locale_h

$name            k\xF6nig
@locale{qw(German English)} = qw(de_DE ISO_8859-1 us-ascii),
setlocale(LC_CTYPE, $locale{English})
  or die Invalid locale $locale{English} ,
@english_names = (),
while ($name =~ /\b(\w+)\b/g) {
        push(@english_names, ucfirst($1)),
}
setlocale(LC_CTYPE,  $locale{German})
  or die Invalid locale $locale{German} ,
@german_names = (),
while ($name =^ /\b(\w+)\b/g) {
        push(@german_names, ucfirst($1))
}
print English names @english_names\n ,
print German names  @german_names\n ,
English names:
German names:           König
```

Решение основано на поддержке локальных контекстов **в POSIX.** Ваша система может обладать, а может и не обладать такой поддержкой. Но даже если система заявляет о поддержке локальных контекстов **POSIX, в** стандарте не определены имена локальных контекстов. Разумеется, переносимость такого решения не гарантирована.

Смотри также

Описание метасимволов \b, \w и \s описание локальных контекстов Perl в *perllocale*(1) и странице руководства *locale*(3) вашей системы; рецепт 6.2.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

6.13. Неформальный поиск

Проблема

Требуется выполнить неформальный поиск по шаблону.

Задача часто возникает в ситуации, когда пользовательский ввод может быть неточным или содержащим ошибки.

Решение

Воспользуйтесь модулем String::Approx от CPAN:

```
use Strin::Approx qw(amatch);

if (amatch("ШАБЛОН", @list)) {
    # Совпадение
}

@matches = amatch("ШАБЛОН", @list);
```

Комментарий

Модуль String::Approx вычисляет, насколько шаблон отличается от каждой строки списка. Если количество односимвольных вставок, удалений или замен для получения строки из шаблона не превышает определенного числа (по умолчанию 10 процентов длины шаблона), строка "совпадает" с шаблоном. В скалярном контексте `amatch` возвращает количество успешных совпадений. В списковом контексте возвращаются совпавшие строки.

```
use String::Approx qw(amatch);
open(DICT, "/usr/dict/words"          or die "Can't open dict: $!";
while(<DICT>) {
    print if amatch("balast");
}

ballast
ballustrade
blast
blastula
sandblast
```

Функции `amatch` также можно передать параметры, управляющие учетом регистра и количеством допустимых вставок, удалений и подстановок. Параметры передаются в виде ссылки на список. Они полностью описаны в документации по String::Approx.

Следует заметить, что поисковые функции модуля работают в 10-40 раз медленнее встроенных функций Perl. Используйте String::Approx лишь в том случае, если регулярные выражения Perl не справляются с неформальным поиском.

Смотри также

Документация по модулю String::Approx от CPAN; рецепт 1.16.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

6.14. Поиск от последнего совпадения

Проблема

Требуется возобновить поиск с того места, где было найдено последнее совпадение.

Такая возможность пригодится при многократном извлечении фрагментов данных из строки.

Решение

Воспользуйтесь комбинацией модификатора /g, метасимвола \G и функции pos.

Комментарий

При наличии модификатора /g механизм поиска запоминает текущую позицию в строке. При следующем поиске с /g совпадения ищутся, начиная с сохраненной позиции. Это позволяет создать цикл while для извлечения необходимой информации из строки:

```
while (/(\d+)/g) {
    print "Found $1\n";
}
```

Присутствие \G в шаблоне привязывает поиск к концу предыдущего совпадения. Например, если число хранится в строке с начальными пробелами, замена каждого пробела нулем может выполняться так:

```
$n = "   49 here";
$n =- s/\G /0/g;
print $n;
00049
```

\G часто применяется в циклах while. Например, в следующем примере анализируется список чисел, разделенных запятыми:

```
while (/\G,?(\d+)/g) {
    print "Found number $1\n";
}
```

Если поиск закончился неудачей (например, если в последнем примере кончились числа), сохраненная позиция по умолчанию перемещается в начало строки. Если это нежелательно (например, требуется продолжить поиски с текущей позиции, но с другим шаблоном), воспользуйтесь модификатором /c в сочетании с /g:

```
$_ = "The year 1752 lost 10 days on the 3rd of September";

while (/(\d+)/gc) {
    print "Found number $1\n";
}

if (/\G(\S+)/g) {
    print "Found $1 after the last number.\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
Found numeral 1752
Found numeral 10
Found numeral 3
Found rd after the last number.
```

Как видите, при последовательном применении шаблонов можно изменять позицию начала поиска с помощью модификатора /g. Позиция последнего совпадения связывается со скалярной величиной, в которой происходит поиск, а не с шаблоном. Позиция не копируется вместе со строкойи не сохраняется оператором local.

Позиция последнего совпадения читается и задается функцией pos. Аргументом функции является строка, для которой читается или задается позиция последнего совпадения. Если аргумент не указан, pos работает с переменной $_:

```
print "The position in \$a is ", pos($a);
pos($a) = 30;
print "The position in \$_ is ", pos;
pos = 30;
```

Смотри также

Описание модификатора /g в *perlre*(1).

6.15. Максимальный и минимальный поиск

Проблема

Имеется шаблон с максимальным квантификатором — *, +, ? или {}. Требуется перейти от максимального поиска к минимальному.

Классический пример — наивная подстановка для удаления тегов из HTML-документа. Хотя `s#<TT>.*</TT>##gsi` выглядит соблазнительно, в действительности будет удален весь текст от первого открывающего до последнего закрывающего тега TT. От строки "Even <TT>vi</TT> can edit <TT>troff</TT> effectively." остается лишь "Even effectively" — смысл полностью изменился!

Решение

Замените максимальный квантификатор соответствующим минимальным. Другими словами, *, +, ? или {} соответственно заменяются *?, +?, ?? и {}?.

Комментарий

В Perl существуют два набора квантификаторов: *максимальные* (*, +, ? и {}) и *минимальные*[1] (*?, +?, ?? и {}?). Например, для строки "Perl is a Swiss Army Chainsaw!" шаблон /(r.*s)/ совпадет с "rl is a Swiss Army Chains", а шаблон /(r.*?s)/ — с "rl is".

[1] Также часто называемые "жадными" (greedy) и "скупыми" (stingy) квантификаторами. — *Примеч. перев.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Предположим, шаблон содержит максимальный квантификатор. При поиске подстроки, которая может встречаться переменное число раз (например, 0 и более раз для * или 1 и более раз для +), механизм поиска всегда предпочитает "и более". Следовательно, шаблон /foo. *bar/ совпадает от первого "foo"до последнего "bar", а не до следующего "bar", как можно ожидать. Чтобы при поиске предпочтение отдавалось минимальным, а не максимальнымсовпадениям, поставьте после квантификатора вопросительный знак. Таким образом, *?, как и *, соответствует 0 и более повторений, но при этом выбирается совпадение минимальной, а не максимальной длины.

```
# Максимальный поиск
s/<.*>//gs;        # Неудачная попытка удаления тегов

# Минимальный поиск
s/<.*?>//gs;       # Неудачная попытка удаления тегов
```

Показанное решение не обеспечивает правильного удаления тегов из HTML-документа, поскольку отдельное регулярное выражение не заменит полноценного анализатора. Правильное решение этой проблемы продемонстрировано в рецепте 20.6.

Впрочем, с минимальными совпадениями дело обстоит не так просто. Не стоит ошибочно полагать, что BEGIN.*?END в шаблоне всегда соответствует самому короткому текстовому фрагменту между соседними экземплярами BEGIN и END. Возьмем шаблон /BEGIN(.*?)END/. После поиска в строке "BEGIN and BEGIN and END" переменная $1 будет содержать "and BEGIN and". Вероятно, вы рассчитывали на другой результат.

Представьте, что мы хотим извлечь из HTML-документа весь текст, оформленный полужирным и курсивным шрифтом одновременно:

```
<b><i>this</i> and <i>that</i> are important</b> Oh, <b><i>me too!</i></b>
```

Может показаться, что шаблон для поиска текста, находящегося *между* парами тегов HTML (то есть не включающий теги), должен выглядеть так:

```
m{ <b><i>(.*?)</i></b> }sx;
```

Как ни странно, шаблон этого не делает. Многие ошибочно полагают, что он сначала находит последовательность "", затем нечто отличное от "", а затем — "", оставляя промежуточный текст в $1. Хотя по отношению к входным данным он часто работает именно так, в действительности делается совершенно иное. Шаблон просто находит левую строку минимальной длины, которая соответствует *всему шаблону*. В данном примере это вся строка. Если вы хотели ограничиться текстом между "" и "", не включающим другие теги полужирного или курсивного начертания, результат окажется неверным.

Если искомая строка состоит всего из одного символа, инвертированный класс (например, /X[^X]*)X/) заметно превосходит минимальный поиск по эффективности. Однако обобщенный шаблон, который находит "сначала BEGIN, затем не-BEGIN, затем END» для произвольных BEGIN и END и сохраняет промежуточный текст в $1, выглядит следующим образом:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
/BEGIN((?:(?!BEGIN).)*)END/
```

Наш пример с тегами HTML выглядит примерно так:

```
m{ <b><i>( (?: (?!</b>|</i>). )* ) </i></b> >sx;
```

или так:

```
m{ <b><i>( (?: (?!</[ib]>). )* ) </i></b> }sx;
```

Как замечает Джеффри Фридл, это скороспелое решение не очень эффективно. В ситуациях, гдескорость действительно важна, он предлагает воспользоваться более сложным шаблоном:

```
m{
    <b><i>
    [^<]*   # Заведомо допустимо
    (?:
# Символ '<' возможен, если он не входит в недопустимую конструкцию
     (?! </?[ib]> )  # Недопустимо
     <               # Все нормально, найти <
     t^<]*           # и продолжить
    ) *
    </i></b>
}sx
```

Смотри также

Описание минимальных квантификаторов в разделе "Regular

6.16. Поиск повторяющихся слов

Проблема

Требуется найти в документе повторяющиеся слова.

Решение

Воспользуйтесь обратными ссылками в регулярных выражениях.

Комментарий

Механизм поиска запоминаетчасть строки, которая совпала с частью шаблона, заключенной в круглые скобки. Позднее в шаблоне обозначение \1 ссылается на первый совпавший фрагмент, \2 — на второй и т. д. Не используйте обозначение $1 — оно интерпретируется как переменная и интерполируется до начала поиска. Шаблон /([A-Z])\1/ совпадает с символом верхнего регистра, за которым следует не просто другой символ верхнего регистра, а именно тот, что был сохранен в первой паре скобок.

Следующий фрагмент читает входной файл по абзацам. При этом используется принятое в Perl определение абзаца как фрагмента, заканчивающегося двумя и более смежными переводами строк. Внутри каждого абзаца находятся все по-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

вторяющиеся слова. Программа не учитывает регистр и допускает межстрочные совпадения.

Модификатор /x разрешает внутренние пропуски и комментарии, упрощающие чтение регулярных выражений. Модификатор /i позволяет найти оба экземпляра "ls" в предложении "Is is this ok?". Модификатор /g в цикле while продолжает поиск повторяющихся слов до конца текста. Внутри шаблона метасимволы \b (граница слова) и \s (пропуск) обеспечивают выборку целых слов.

```
$/ = '';
while (<>) {
    while ( m{
                \b
                (\S+)
                \b
                (
                  \s+
                  \1
                  \b
                ) +
              }xig
            )
    {
        print "dup word '$1' at paragraph $.\n";
    }
}
```

Приведенный фрагмент найдет удвоенное test в следующем примере:

```
This is a test
test of the duplicate word funder.
```

Проверка \S+ между двумя границами слов обычно нежелательна, поскольку граница слова определяется как переход между \w (алфавитно-цифровым символом или подчеркиванием) и либо концом строки, либо не-\w. Между двумя \b обычный смысл \S+ (один и более символов, не являющихся пропусками) распространяется до последовательности символов, не являющихся пропусками, первый и последний символ которой должны быть алфавитно-цифровыми символами или подчеркиваниями.

Рассмотрим другой интересный пример использования обратных ссылок. Представьте себе два слова, причем конец первого совпадает с началом второго — например, "nobody" и "bodysnatcher". Требуется найти подобные "перекрытия" и сформировать строку вида "nobodysnatcher". Это вариация на тему нашей основной проблемы — повторяющихся слов.

Чтобы решить эту задачу, программисту на С, привыкшему к традиционной последовательной обработке байтов, придется написать длинную и запутанную программу. Но благодаря обратным ссылкам задача сводится к одному простому поиску:

```
$a = 'nobody';
$b = 'bodysnatcher';
if ("$a $b" =- /^(\w+)(\w+) \2(\w+)$/) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    print "$2 overlaps in $1-$2-$3\n";
}
body overlaps in no-body-snatcher
```

Казалось бы, из-за наличия максимального квантификатора переменная $1 должна захватывать все содержимое "nobody". В действительности так и происходит — на некоторое время. Но после этого не остается ни одного символа, который можно было бы занести в $2. Механизм поиска дает задний ход, и $1 неохотно уступает один символ переменной $2. Пробел успешно совпадает, но далее в шаблоне следует переменная \2, которая в настоящий момент содержит просто "y". Следующий символ в строке — не "y", а "b". Механизм поиска делает следующий шаг назад; через некоторое время $1 уступит $2 достаточно символов, чтобы шаблон нашел фрагмент, пробел и затем тот же самый фрагмент.

Этот прием не работает, если само перекрытие содержит повторяющиеся фрагменты — как, например, для строк "rococo" и "cocoon". Приведенный выше алгоритм решит, что перекрываются символы "co", а не "coco". Однако мы хотим получить не "rococococoon", а "rococoon". Задача решается включением минимального квантификатора в $1:

```
/^(\w+?)(\w+) \2(\w+)$/
```

Трудно представить, насколько мощными возможностями обладают обратные ссылки. Пример 6.11 демонстрирует принципиально новый подход к проблеме разложения числа на простые множители (см. главу 2 "Числа).

Пример 6.11. prime-pattern

```
#!/usr/bin/perl
# prime_pattern - разложение аргумента на простые множители по шаблону
for ($N = ('o' x shift); $N =~ /^(oo+?)\1+$/; $N =^ s/$1/o/g) {
    print length($1), " ";
}
print length ($N), "\n";
```

Несмотря на свою непрактичность, этот подход отлично демонстрирует возможности обратных ссылок и потому весьма поучителен.

Приведем другой пример. Гениальная идея, предложенная Дугом Макилроем (Doug McIlroy) — во всяком случае, так утверждает Эндрю Хьюм (Andrew Hume), — позволяет решать диофантовы уравнения первого порядка с помощью регулярных выражений. Рассмотрим уравнение 12x + 15y + 16z = 281. Сможете ли вы найти возможные значения x, y и z? А вот Perl может!

```
# Решение 12x + 15y + 16z = 281 для максимального x
if (($X, $Y, $Z) =
    (('o' x 281) =- /^(o*)\1{11}(o*)\2{14}(o*)\3{15}$/))
{
    ($x, $y, $z) = (length($X), length($Y), length($Z));
    print "One solution is: x=$x; y=$y; z=$z.\n";
} else {
    print "No solution.\n";
}
One solution is: x=17; y=3; z=2.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Поскольку для первого o* ищется максимальное совпадение, x растет до максимума. Замена одного или нескольких квантификаторов * на *?, + или +?дает другие решения:

```
(('o' x 281) =- /^(o+)\1{11}(o+)\2{14}(o+)\3{15}$/))
One solution is: x=17; y=3; z=2.
(('o' x 281) =- /^(o*?)\1{11}(o*)\2{14}(o*)\3{15}$/))
One solution is: x=0; y=17; z=11.
(('o' x 281) =- /^(o+?)\1{11}(o*)\2{14}(o*)\3{15}$/))
One solution is: x=1; y=3;  z=14.
```

Подобные демонстрации математических возможностей выглядят потрясающе, но из них следует вынести один важный урок: механизм поиска по шаблону (особенно с применением обратных ссылок) всей душой желает предоставить вам ответ и будет трудиться с феноменальным усердием. Однако обратные ссылки в регулярных выражениях могут привести к экспоненциальному росту времени выполнения. Для любых нетривиальных данных программа будет работать так медленно, что даже дрейф континентов по сравнению с ней покажется быстрым.

Смотри также

Описание обратных ссылок в разделе "Regular

6.17. Логические AND, OR и NOT в одном шаблоне

Проблема

Имеется готовая программа, которой в качестве аргумента или входных данных передается шаблон. В нее невозможно включить дополнительную логику — например, параметры для управления учетом регистра при поиске, AND и NOT. Следовательно, вы должны написать один шаблон, который будет совпадать с любым из двух разных шаблонов (OR), двумя шаблонами сразу (AND) или менять смысл поиска на противоположный (NOT).

Подобная задача часто возникает при получении данных из конфигурационных файлов, Web-форм или аргументов командной строки. Пусть у вас имеется программа, в которой присутствует следующий фрагмент:

```
chomp($pattern = <CONFIG_FH>);
if ( $data =" /$pattern/ ) { ... }
```

Если вы отвечаете за содержимое CONFIG_FH, вам понадобятся средства для передачи программе поиска логических условий через один-единственный шаблон.

Решение

Выражение истинно при совпадении /ALPHA/ или /BETA/ (аналогично /ALPHA/ || /BETA/):

```
/ALPHA|BETA/
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Выражение истинно, если и /ALPHA/, и /BETA/ совпадают при разрешенных перекрытиях (то есть когда подходит строка "BETALPHA"). Аналогично /ALPHA/ && /BETA/:

```
/^(?=.*ALPHA)(?=.*BETA)/s
```

Выражение истинно, если и /ALPHA/, и /BETA/ совпадают при запрещенных перекрытиях (то есть когда "BETALPHA" не подходит):

```
/ALPHA.*BETA|BETA.*ALPHA/s
```

Выражение истинно, если шаблон /PAT/ не совпадает (аналогично $var !^ /PAT/):

```
/^(?:(?!PAT).)*$/s
```

Выражение истинно, если шаблон BAD не совпадает, а шаблон GOOD — совпадает:

```
/(?=^(?:(?!BAD).)*$)GOOD/s
```

Комментарий

Предположим, вы пишете программу и хотите проверить некоторый шаблон на несовпадение. Воспользуйтесь одним из вариантов:

```
if (!($string =~ /pattern/)) {something() }  # Некрасиво
if (  $string !^ /pattern/)  {something() }  # Рекомендуется
```

Если потребовалось убедиться в совпадении обоих шаблонов, примените следующую запись:

```
if ($string =~ /pat1/ && $string =^ /pat2/ ) { something() }
```

Проверка совпадения хотя бы одного из двух шаблонов выполняется так:

```
if ($string =~ /pat1/ || $string =^ /pat2/ ) { something() }
```

Короче говоря, нормальные логические связки Perl позволяют комбинировать логические выражения вместо того, чтобы объединять их в одном шаблоне. Но давайте рассмотрим программу minigrep из примера 6.12, которая в качестве аргумента получает всего один шаблон.

Пример 6.12.

```
#!/usr/bin/perl
            - тривиальный поиск
$pat = shift;
while (<>) {
    print if /$pat/o;
}
```

Если потребуется сообщить minigrep, что некоторый шаблон не должен совпадать или что должны совпасть оба мини-шаблона в произвольном порядке, вы оказываетесь в тупике. Программа просто не предусматривает подобных конструкций. Как сделать все в одном шаблоне? Другими словами, вы хотите выполнить программу параметром PAT, который не совпадает или содержит несколько логически связанных шаблонов. Такая задача нередко возникает в программах, читающих шаблоны из конфигурационных файлов.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Проблема с OR решается просто благодаря символу альтернативного выбора |. Однако AND и OR потребуют особого кодирования.

В случае с AND придется различать перекрывающиеся и неперекрывающиеся совпадения. Допустим, вы хотите узнать, совпадают ли в некоторой строке шаблоны "bell" и "lab". Если разрешить перекрытия, слово "labelled" пройдет проверку, а если отказаться отперекрытий — нет. Случай с перекрытиямипотребует двух опережающих проверок:

```
"labelled" =^ /^(?=.*bell)(?=.*lab)/s
```

Помните: в нормальной программе подобные извращения не нужны. Достаточно сказать:

```
$string =^ /bell/ && $string =' /lab/
```

Мы воспользуемся модификатором /x с комментариями. Развернутая версия шаблона выглядит так:

```
if ($murray_hill =^ m{
                        # Начало строки
            (?=         # Опережающая проверка нулевой ширины
                .*      # Любое количество промежуточных символов
                bell    # Искомая строка bell
            )           # Вернуться, мы лишь проверяем
            (?=         # Повторить
                .*      # Любое количество промежуточных символов
                lab     # Искомая строка labs
            )                                                    }
          }sx )         # /s разрешает совпадение . с переводом строки
{
    print "Looks like Bell Labs might be in Murray Hill!\n";
}
```

Мы не воспользовались .*? для раннего завершения поиска, поскольку минимальный поиск обходится дороже максимального. Поэтому для произвольных входных данных, где совпадение с равной вероятностью может произойти как в начале, так и в конце строки, .* будет эффективнее нашего решения. Разумеется, выбор между .* и .*? иногда определяется правильностью программы, а не эффективностью, но не в данном случае.

Для обработки перекрывающихся совпадений шаблон будет состоять из двух частей, разделенных OR. В первой части "lab" следует после "bell", а во второй — наоборот:

```
"labelled" =~ /(?:^.*bell.*lab)|(?:^.*lab.*bell)/
```

или в развернутой форме:

```
$brand = "labelled";
if ($brand =~ m{
        (?:         # Группировка без сохранения
            ^.*?    в Любое количество начальных символов
            bell    # Искомая строка bell
            .*?     # Любое количество промежуточных символов
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                lab      # Искомая строка lab
            )            # Конец группировки
        |                # Или попробовать другой порядок
            (?:          # Группировка без сохранения
                ^.*?     # Любое количество начальных символов
                lab      # Искомая строка lab
                .*?      # Любое количество промежуточных символов
                bell     # Искомая строка bell
            )            # Конец группировки
        }sx )            # /s разрешает совпадение . с переводом строки
{
    print "Our brand has bell and lab separate.\n";
}
```

Такие шаблоны не всегда работают быстрее. `$murray_hill =" /bell/ && $murray_hille =~/lab/` сканирует строку не более двух раз, однако для `(?=^.*?bell)(?=^.*?lab)` механизм поиска ищет `"lab"` для каждого экземпляра `"bell"`, что в наихудшем случае приводит к квадратичному времени выполнения.

Тем, кто внимательно рассмотрел эти два случае, шаблон NOT покажется тривиальным. Обобщенная форма выглядит так:

```
$map =~ /^(?:(?!waldo).)*$/s
```

То же в развернутой форме:

```
if ($map =~ m{
                        # Начало строки
        (?:             # Группировка без сохранения
            (?!         # Опережающая отрицательная проверка
                waldo   9 Нашли впереди?
            )           tt Если да, отрицание не выполняется
                        # Любой символ (благодаря /s)
        ) *             # Повторить группировку 0 и более раз
        $               # До конца строки
    }sx )               # /s разрешает совпадение . с переводом строки
{
    print               waldo here!\n";
}
```

Как объединить в одном шаблоне AND, OR и NOT? Результат выглядит отвратительно, и в обычных программах делать нечто подобное практически никогда не следует. Однако при обработке конфигурационных файлов или командных строк, где вводится всего один шаблон, у вас нет выбора. Объедините все изложенное выше. Будьте осторожны.

Предположим, вы хотите запустить программу UNIX w и узнать, зарегистрировался ли пользователь `tchrist` с любого терминала, имя которого начинается не с ttyp; иначе говоря, шаблон `"tchrist"` должен совпадать, а `"ttyp"` — нет.

Примерный вывод w в моей системе Linux выглядит так:

```
 7:15am  up 206 days, 13:30,   4 users,   load average: 1.04, 1,07, 1.04
USER     TTY      FROM              LOGIN"  IDLE   JCPU   PCPU  WHAT
tchrist  tty1                      5:16pm 36days 24:43   0.03s xinit
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
tchrist  tty2                    5:19pm  6days  0.43s  0.43s -tcsh
tchrist  ttypO   chthon            7:58am  3days 23:44s  0.44s -tcsh
gnat     ttys4   coprolith         2:01pm 13:36m  0.30s  0.30s -tcsh
```

Посмотрим, как поставленная задача решается с помощью приведенной выше или программы **tcgrep**, приведенной в конце главы:

```
% w          '^(?!.*ttyp).*tchrist'
```

Расшифруем структуру шаблона:

```
m {
                    fl Привязка к началу строки
    (?!            # Опережающая проверка нулевой ширины
        .*          # Любое количество любых символов (быстрее .*?)
       ttyp        # Строка, которая не должна находиться
    )              # Опережающая отрицательная проверка; возврат к началу
    .*              # # Любое количество любых символов (быстрее .*?)
    tchrist        # Пытаемся найти пользователя tchrist
}x
```

Неважно, что любой нормальный человек в такой ситуации дважды вызывает g rep (из них один - *с* параметром *-v,* чтобы отобрать несовпадения):

```
% w | grep tchrist |
```

Главное — что логические конъюнкции и отрицания *можно* закодировать в одном шаблоне. Однако подобные вещи следует снабжать комментариями — пожалейте тех, кто займется ими после вас.

Как внедрить модификатор /s в шаблон, передаваемый программе из командной строки? По аналогии с /i, который в шаблоне превращается в `(?i)`. Модификаторы /s и /m также безболезненно внедряются в шаблоны в виде `/(?s)` или `/(?m)`. Их даже можно группировать — например, `/(?smi)`. Следующие две строки фактически эквивалентны:

```
% grep -i  'ШАБЛОН' ФАЙЛЫ
          '(?i)ШАБЛОН' ФАЙЛЫ
```

Смотри также

Описание опережающих проверок в разделе "Regular man-страницы и w*(1)* вашей системы. Работа с конфигурационными файлами рассматривается в рецепте 8.16.

6.18. Поиск многобайтовых символов

Проблема

Требуется выполнить поиск регулярных выражений для строк с многобайтовой кодировкой символов.

Кодировка определяет соответствие между символами и их числовыми представлениями. В кодировке ASCII каждый символ соответствует ровно одному

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

байту, однако языки с иероглифической письменностью (китайский, японский и корейский) содержат так много символов, что в их кодировках символы приходится представлять несколькими байтами.

Perl исходит из предположения, что один байт соответствует одному символу. В ASCII все работает нормально, но поиск по шаблону в строках, содержащих многобайтовые символы, — задача по меньшей мере нетривиальная. Механизм поиска не понимает, где в последовательности байтов расположены границы символов, и может вернуть "совпадения" от середины одного символа до середины другого.

Решение

Воспользуйтесь кодировкой и преобразуйте шаблон в последовательность байтов, образующих многобайтовые символы. Основная мысль заключается в построении шаблона, который совпадает с одним (многобайтовым) символом кодировки, а затем применить этот шаблон "любого символа" в более сложных шаблонах.

Комментарий

В качестве примера мы рассмотрим одну из кодировок японского языка, EUC-JP, и разберемся, как воспользоваться ей для решения многих проблем, связанных с многобайтовыми символами. В EUC-JP можно представить тысячи символов, но в сущности эта кодировка является надмножеством ASCII. Байты с 0 по 127 (0x00 - Ox7F)почти точно совпадают с ASCII-аналогами и соответствуют однобайтовым символам. Некоторые символы представляются двумя байтами; первый байт равен 0x8E, а второй принимает значения из интервала 0xA0-0x0F. Другие символы представляются тремя байтами; первый байт равен 0x8F, а остальные принадлежат интервалу 0xA1-0xFE. Наконец, часть символов представляется двумя байтами, каждый из которых принадлежит интервалу 0xA1–0xFE.

Исходя из этих данных, можно построить регулярное выражение. Для удобства последующего применения мы определим строку $euc]p с регулярным выражением, которое совпадает с одним символом кодировки EUC-JP:

```
my $eucjp = q{                  # Компоненты кодировки EUC-JP
    [\x00-\x7F]                  # ASCII/JIS-Roman (один байт/символ)
  | \x8E[\xA0-\xDF]             # катакана половинной ширины (два байта/символ)
  | \x8F[\xA1-xFE][\xA1-\xFE]  # JIS X 0212-1990 (три байта/символ)
  | [\xA1-\xFE][\xA1-\xFE]      # JIS X 0208 1997 (два б айта/символ)
},
```

(строка содержит комментарии и пропуски, поэтому при ее использовании для поиска или замены необходимо указывать модификатор /x).

Располагая этим шаблоном, мы расскажем, как:

- Выполнить обычный поиск без "ложных" совпадений.
- Подсчитать, преобразовать (в другую кодировку) и/или отфильтровать символы.
- Убедиться в том, что проверяемый текст содержит символы данной кодировки.
- Узнать, какая кодировка используется в некотором тексте.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Во всех приведенных примерах используется кодировка EUC-JP, однако они будут работать и в большинстве других распространенных многобайтовых кодировок, встречающихся при обработке текстов — например, Unicode, Big-5 и т. д.

Страховка от ложных совпадений

Ложное совпадение происходит, когда найденное совпадение приходится на середину многобайтового представления одного символа. Чтобы избежать ложных совпадений, необходимо контролировать процесс поиска и следить, чтобы механизм поиска синхронизировался с границами символов.

Для этого можно связать шаблон с началом строки и вручную пропустить байты, для которых в текущей позиции не может произойти нормальное совпадение. В примере с EUC-JP за "пропуск символов" отвечает часть шаблона `/(?:$eucjp)*?/`. `$eucjp` совпадает с любым допустимым символом. Поскольку он применяется с минимальным квантификатором `*?`, совпадение возможно лишь в том случае, если не совпадает то, что идет после него (искомый текст). Рассмотрим реальный пример:

```
/^ (?: $eucjp )*? \xC5\xEC\xB5\xFE/ox # Пытаемся найти Токио
```

В кодировке EUC-JP японское название Токио записывается двумя символами — первый кодируется двумя байтами `\xC5\xEC`, а второй — двумя байтами `\xB5\xFE`. С точки зрения Perl мы имеем дело с обычной 4-байтовой последовательностью `\xC5\xEC\xB5\xFE`. Однако, поскольку использование `(?:$eucjp)*?` обеспечивает перемещение в строке только по символам целевой кодировки, мы знаем, что синхронизация сохраняется.

Не забывайте о модификаторах /ox. Модификатор /x особенно важен из-за наличия пропусков в шаблоне `$eucjp`. Модификатор /o повышает эффективность, поскольку значение `$eucjp` заведомо остается неизменным.

Аналогично выполняется и замена, но поскольку текст перед настоящим совпадением также является частью общего совпадения, мы должны заключить его в круглые скобки и включить в заменяющую строку. Предположим, переменным $Tokyo и $Osaka были присвоены последовательности байтов с названиями городов Токио и Осака в кодировке EUC-JP. Замена Токио на Осаку происходит следующим образом:

```
/^ ( (?:eucjp)*? ) $Tokyo/$1Osaka/ox
```

При использовании модификатора /д поиск должен быть привязан не к началу строки, а к концу предыдущего совпадения. Для этого достаточно заменить ^ на \G:

```
/\G ( (?:eucjp)*? ) $Tokyo/$10saka/gox
```

Разделение строк в многобайтовой кодировке

Другая распространенная задача — разбивка входной строки на символы. Для однобайтовой кодировки достаточно вызвать функцию split//, но для многобайтовых конструкция будет выглядеть так:

```
@chars = /$eucjp/gox; # По одному символу на каждый элемент списка
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Теперь каждый элемент @chars содержит один символ строки. В следующем фрагменте этот прием используется для создания фильтра:

```
while (<>) {
  my @chars = /$eucjp/gox;  # Каждый элемент списка содержит один символ
  for my $char (@chars) {
    if (length($char) == 1) {
      # Сделать что-то интересное с однобайтовым символом
    } else {
      # Сделать что-то интересное с многобайтовым символом
    >
  }
  my $line = join("",@chars); # Объединить символы списка в строке
  print $line;
}
```

Любые изменения $char в двух фрагментах, где происходит "что-то интересное", отражаются на выходных данных при объединении символов @chars.

Проверка многобайтовых строк

Успешная работа приемов, подобных /$eucjp/gox, существенно зависит от правильного форматирования входных строк в предполагаемой кодировке (EUC-JP). Если кодировка не соблюдается, шаблон /$eucjp/ не будет работать, что приведет к пропуску байтов.

Одно из возможных решений — использование /\G$eucjp/gox. Этот шаблон запрещает механизму поиска пропускать байты при поиске совпадений (модификатор \G означает, что новое совпадение должно находиться сразу же после предыдущего). Но и такой подход не идеален, потому что он просто прекращает выдавать совпадения для входных данных неправильного формата.

Более удачный способ убедиться в правильности кодировки строки — воспользоваться конструкцией следующего вида:

```
$is_eucjp = m/^(?:$eucjp)*$/xo;
```

Если строка от начала до конца состоит только из допустимых символов, значит, она имеет правильную кодировку.

И все же существует потенциальная проблема, связанная с особенностями работы метасимвола конца строки **$:** совпадения возможны как **в** конце строки (что нам и требуется), так **и** перед символом перевода строки **в** ее конце. Следовательно, успешное совпадение возможно даже **в** том случае, если символ перевода строки не является допустимым **в** кодировке. Проблема решается заменой **$** более сложной конструкцией (?!\n).

Базовая методика проверки позволяет определить кодировку. Например, японский текст обычно кодируется либо **в** EUC-JP, либо **в** другой кодировке, которая называется **Shift-JIS.** Имея шаблоны $eucjp и $sjis, можно определить кодировку следующим образом:

```
$is_eucjp = m/^(?:$eucjp)*$/xo;
$is_sjis  = m/^(?:$sjis)*$/xo;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если обе проверки дают истинный результат, вероятно, мы имеем дело с ASCII-текстом (поскольку ASCII, в сущности, является подмножеством обеих кодировок). Однако такое решение не дает стопроцентной гарантии, поскольку некоторые строки с многобайтовыми символами могут оказаться допустимыми в обеих кодировках. В таких случаях автоматическое распознавание становится невозможным, хотя по относительным частотам символов можно выдвинуть разумное предположение.

Преобразование кодировок

Преобразование может сводиться к простому расширению описанного выше процесса перебора символов. Для некоторых взаимосвязанных кодировок достаточно тривиальных математических операций с байтами, в других случаях потребуются огромные таблицы соответствия. В любом случае код вставляется в те фрагменты, где происходит "что-то интересное" (см. выше).

Следующий пример преобразует строки из EUC-JP в Unicode, при этом в качестве таблицы соответствия используется хэш %euc2uni:

```
while (o) {
  my @chars = /$eucjp/gox;   # Каждый элемент списка содержит один символ
  for roy $char (@chars) {
    my $uni = $euc2uni{$char};
    if (defined $uni) {
      $euc = $uni;
    > else {
      Я Обработать неизвестное преобразование из EUC в Unicode
    >
  }
  my $line = join("",@chars);
  print $line;
}
```

Поиск и обработка многобайтовых символов играет особенно важную роль в Unicode, имеющей несколько разновидностей. В UCS-2 и UCS-4 символы кодируются фиксированным числом байтов. UTF-8 использует от одного до шести байтов на символ. UTF-16, наиболее распространенный вариант Unicode, представляет собой 16-битную кодировку переменной длины.

6.19. Проверка адресов электронной почты

Проблема

Требуется построить шаблон для проверки адресов электронной почты.

Решение

Задача в принципе неразрешима, проверка адреса электронной почты в реальном времени невозможна. Приходится выбирать один из возможных компромиссов.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Многие шаблоны, предлагаемые для решения этой проблемы, попросту неверны. Допустим, адрес `fred&barney@stonehedge.com` правилен и по нему возможна доставка почты (на момент написания книги), однако большинство шаблонов, претендующих на проверку почтовых адресов, бесславно споткнутся на нем.

Документы RFC-822 содержат формальную спецификацию синтаксически правильного почтового адреса. Однако полная обработка требует рекурсивного анализа вложенных комментариев — задача, с которой одно регулярное выражение не справится. Если предварительно удалить комментарии:

```
1 while $addr =~ s/\([^()]*\)//g;
```

тогда теоретически можно воспользоваться довольно длинным шаблоном для проверки соответствия стандарту RFC, но и это недостаточно хорошо по трем причинам.

Во-первых, не по всем адресам, соответствующим спецификации RFC, возможна доставка. Например, адрес `foo@foo.foo.foo.foo` теоретически правилен, но на практике доставить на него почту невозможно. Некоторые программисты пытаются искать записи **MX** на серверах **DNS** или даже проверяют адрес на хосте, обрабатывающем его почту. Такой подход неудачен, поскольку большинство узлов не может напрямую подключиться к любому другому узлу, но даже если **бы** это было возможно, получающие почту узлы обычно либо игнорируют команду **SMTP** VRFY, либо откровенно врут.

Во-вторых, почта может прекрасно доставляться по адресам, не соответствующим RFC. Например, сообщение по адресу `postmaster` почти наверняка будет доставлено, но этот адрес не соответствует канонам RFC — **в** нем нет символа **@**.

В-третьих (самая важная причина), даже если адрес правилен и по нему возможна доставка, это еще не означает, что он вам подойдет. Например, адрес `president@whitehouse.gov` соответствует стандартам RFC и обеспечивает доставку. И все же крайне маловероятно, чтобы этот адресат стал поставлять информацию для вашего сценария CGI.

Отважная (хотя и далеко не безупречная) попытка приведена **в** сценарии по адресу *http://www.perl.com/CPAN/authors/Tom_Christiansen/scripts/ckaddr.gz*. Эта программа выкидывает множество фортелей, среди которых — проверка регулярного выражения на соответствие RFC-822, просмотр записей **MX DNS** и стоп-списки для ругательств **и** имен знаменитостей. Но и такой подход оказывается откровенно слабым.

При проверке почтового адреса мы рекомендуем организовать его повторный ввод, как это часто делается при проверке пароля. При этом обычно исключаются опечатки. Если обе версии совпадут, отправьте на этот адрес личное сообщение следующего содержания:

```
Дорогой someuser@host.com,

Просим подтвердить почтовый адрес, сообщенный вами в 09:38:41 6 мая 1999 года. Для
этого достаточно ответить на настоящее сообщение. Включите в ответ строку
"Rumpelstiltskin", но в обратном порядке (то есть начиная с "Nik...")). После этого
ваш подтвержденный адрес будет занесен в нашу базу данных.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если вы получите ответное сообщение и ваши указания будут выполнены, можно с достаточной уверенностью предположить, что адрес правилен.

Возможна и другая стратегия, которая обеспечивает лучшую защиту от подделок, — присвойте своему адресату личный идентификатор (желательно случайный) и сохраните его вместе с адресом для последующей обработки. В отправленном сообщении попросите адресата включать личный идентификатор в свои ответы. Однако идентификатор будет присутствовать и при возврате недоставленного сообщения, и при включении рассылки в сценарий. Поэтому попросите адресата слегка изменить идентификатор — например, поменять порядок символов, прибавить или вычесть 1 из каждой цифры и т. д.

Смотри также

Рецепт 18.9.

6.20. Поиск сокращений

Проблема

Предположим, у вас имеется список команд — например, "send", "abort", "list" и "edit". Пользователь вводит лишь часть имени команды, и вы не хотите заставлять его вводить всю команду доконца.

Решение

Воспользуйтесь следующим решением, если все строки начинаются с разных символов или если одни совпадения имеют более высокий приоритет по сравнению с другими (например, если "SEND" отдается предпочтение перед "STOP"):

```
chomp ($answer = <>);
if    ("SEND"  =^ /^\Q$answer\i) { print "Action is send\n"  }
elsif ("STOP"  =^ /^\Q$answer\i) { print "Action is stop\n"  }
elsif ("ABORT" =" /^\Q$answer\i) { print "Action is abort\n" }
elsif ("LIST"  =^ /^\Q$answer\i) { print "Action is list\n"  }
elsif ("EDIT"  =' /^\Q$answer\i) { print "Action is edit\n"  }
```

Кроме того, можно воспользоваться модулем

```
use Text::Abbrev;
      =        qw(send abort list edit);                <
for (print "Action: "; o; print "Action: ") {
    chomp;
    my $action = $href->{ lc($_) };
    print "Action is $action\n";
}
```

Комментарий

В первом решении изменяется стандартный порядок поиска; обычно слева указывается переменная, а справа — шаблон. Мы бы также могли попытаться опре-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

делить, какое действие выбрал пользователь, с помощью конструкции `$answer= =' /^ABORT/i`. Выражение будет истинным, если `$answer` начинается со строки `"ABORT"`. Однако совпадение произойдет и в случае, если после `"ABORT"` в `$answer` следует что-то еще — скажем, для строки `"ABORT LATER"`. Обработка сокращений обычно выглядит весьма уродливо: `$answer =~ /^A(B(O(R(T)?)?)?)?$/i`.

Сравните классическую конструкцию "переменная =^ шаблон" с `"ABORT" =~ /^\Q$answer/i`. `\Q` подавляет интерпретацию метасимволов, чтобы ваша программа не "рухнула" при вводе пользователем неверного шаблона. Когда пользователь вводит что-нибудь типа `"ab"`, после замены переменной шаблон принимает вид `"ABORT" =~ /^ab/i`. Происходит совпадение.

работает иначе. Вы передаете ему список слов и получаете ссылку на хэш, ключи которого представляют собой все однозначные сокращения, а значения — полные строки. Если ссылка создается так, как показано в решении, возвращает строку `"abort"`.

Подобная методика часто используется для вызова функции по имени, вводимому пользователем. При этом применяется символическая ссылка:

```
$name = 'send';
&$name();
```

Впрочем, это небезопасно — пользователь сможет выполнить любую функцию нашей программы, если он знает ее имя. Кроме того, такое решение противоречит директиве `use`

Ниже приведена часть программы, создающая хэш, в котором ключ представляет собой имя команды, а значение — ссылку на функцию, вызываемую этой командой:

```
# Предполагается, что &invoke_editor, &deliver_message,
# $file и $PAGER определяются в другом месте.
use Text::Abbrev;
my($href, %actions, $errors);
%actions = (
     "edit"  => \&invoke_editor,
     "send"  => \&deliver_message,
     "list"  => sub {  system($PAGER, $file) >,
     "abort" => sub {
                    print "See ya!\n";
                    exit;
                },
             => sub {
                    print "Unknown command: $cmd\n";
                    $errors++;
                >,
);

                    %actions);
local $_;
for (print "Action; "; <>; print "Action; ") {
     s/^\s+//;
     s/\s+$//;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    next unless $_;
    $actions->{ $href->{ lc($_) } }->();
}
```

Если вы не любите слишком кратких выражений или хотите приобрести навыки машинистки, последнюю команду можно записать так:

```
                lc($_);
$expansion     =
                $actions->{$expansion};
&$coderef();
```

Смотри также

Документация по стандартному модулю Интерполяция рассматривается в разделе "Scalar Value Constructors" *perldata(1)*.

6.21. Программа: urlify

Программа *urlify* оформляет URL-адреса, найденные в файлах, в виде ссылок HTML. Она работает не для всех возможных URL, но справляется с наиболее распространенными. Программа старается избежать включения знаков препинания, завершающих предложения, в помеченный URL.

Программа является типичным фильтром Perl и потому может использоваться для перенаправленного ввода:

```
% gunzip -c ~/mail/archive.gz | urlify > archive.urlified
```

Исходный текст программы приведен в примере6.13.

Пример **6.13.** urlify

```
#!/usr/bin/perl
# urlify - оформление URL-подобных конструкций в виде ссылок HTML

$urls = '(http|telnet|gopher|file|wais|ftp)';
$ltrs = '\w';
$gunk = '/#~:.?+=&%@!\-';
$punc = '.:?\-';
$any  = "${ltrs}${gunk}${punc}";

while (o) {
    s{
      \b                    # Начать с границы слова
      (                     " Начать сохранение $1  {
       $urls     :          # Искать имя ресурса и двоеточие,
        [$any] +?           # за которыми следует один или более
                             # любых допустимых символов, но
                             #  проявлять умеренность и брать лишь то,
                             #  что действительно необходимо ....
      )                     # Завершить сохранение $1  }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        (?=                    # Опережающая проверка без смещения
         [$punc]*              # либо 0, либо знак препинания,
         [^$any]               #   за которыми следует символ, не входящий в url,
          |                    # или
         $                     #  конец строки
        )
      }{<A HREF="$1">$1</A>}igox;
    print;
}
```

6.22. Программа:

Ниже приведена программа UNIX grep, написанная на Perl. Хотя она работает медленнее версий, написанных на C (особенно GNU-версии g обладает многими усовершенствованиями.

Первая и самая важная особенность — эта программа работает везде, где работает Perl. Имеется ряд дополнительных возможностей — t игнорирует все файлы, кроме простых текстовых; распаковывает сжатые или обработанные утилитой gzip файлы; выполняет просмотр в подкаталогах; ищет полные абзацы или записи, определенные пользователем; ищет более свежие версии файлов, а также подчеркивает или выделяет найденные совпадения. Кроме того, параметр -c выводит количество найденных записей, а параметр -C — число найденных совпадений, которые могут содержать несколько записей.

Распаковка сжатых файлов выполняется утилитами gzcat или zcat, поэтому данная возможность отсутствует в системах, где эти программы недоступны, а также в системах, не позволяющих запускать внешние программы (нацример, Macintosh).

При запуске программы без аргументов на экран выводится краткая справка по ее использованию (см. процедуру usage в программе). Следующая командная строка рекурсивно и без учета регистра ищет во всех файлах почтового ящика ~/mail сообщения с отправителем "kate" и выводит имена найденных файлов:

```
.*kate' ~/mail
```

Исходный текст программы приведен в примере 6.14.

Пример 6.14.

```
#!/usr/bin/perl  -w
                    написанная на Perl
# версия 1.0: 30 сентября 1993 года
# версия 1.1: 1 октября 1993 года
# версия 1.2: 26 июля 1996 года
# версия 1.3: 30 августа 1997 года
# версия 1.4: 18 мая 1998 года

use strict;
```

9 Глобальные переменные

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 6.14 (продолжение)

```
use vars qw($Me $Errors $Grand_Total $Mult          $Matches);

my ($matcher, $opt);              # matcher - анонимная функция
                                   # для поиска совпадений
                                   # opt - ссылка на хэш, содержащий
                                   й параметры командной строки
init();                           Я Инициализировать глобальные переменные

($opt, $matcher) = parse_args();  # Получить параметры командной строки
                                   tt и шаблоны

matchfile($opt, $matcher, @ARGV); # Обработать файлы

exit(2) if $Errors;
exit(0) if $Grand_Total;
exit(1);



sub init {
    ($Me = $0) =~ s!../!!;          # Получить базовое имя программы, "tcgrep"
    $Errors = $Grand_Total = 0;     # Инициализировать глобальные счетчики
    $Mult = "";                     # Флаг для нескольких файлов в @ARGV
    $1 = 1;                         # Автоматическая очистка выходного буфера

                                    8 Расширения и имена программ
        z  => 'gzcat',              # для распаковки
        gz => 'gzcat',
        Z  => 'zcat',
    );
}



sub usage {
        die "EOF
usage: $Me [flags] [files]

Standard      options:
        i   case insensitive
        n   number lines
        c   give count of lines matching
        C   ditto, but >1 match per line possible
        w   word boundaries only
        s   silent mode
        x   exact matches only
        v   invert search sense (lines that DON'T match)
        h   hide filenames
                    (for exprs beginning with -)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        f    file with
        1    list filenames matching

Specials:
        1    1 match per file
        H    highlight matches
        u    underline matches

        t    process                        order
        p    paragraph mode (default: line mode)
        P    ditto, but specify separator, e.g. -P '%%\\n'
        a    all files, not just  plain text files
        q    quiet about failed file and dir  opens
        T    trace files as opened

May use a TCGREP environment variable to set default options.
EOF
}

##################################

sub parse_args {
    use Getopt::Std;

    my ($optstring, $zeros, $nulls, %opt, $pattern, @patterns, $match_code);
    my ($SO,  $SE);

    if ($_ = $ENV{TCGREP}) {      # Получить переменную окружения TCGREP
        s/^([^\-])/-$1/;          # Если начальный - отсутствует, добавить
        unshift(@ARGV, $_);       " Включить строку TCGREP в @ARGV
    }

    $optstring = "incCwsxvhe:f:l1HurtpP:aqT";

    $zeros = 'inCwxvhelut';        " Параметры, инициализируемые 0
                                    В (для отмены предупреждений)
    $nulls = 'pP';                # Параметры, инициализируемые ""
                                    * (для отмены предупреждений)

    @opt{ split //, $zeros } = ( 0 )  x  length($zeros);
    @opt{ split //, $nulls } = (  ''  ) x length($nulls);

    getopts($optstring, \%opt)              or usage();

    if ($opt{f}) {                  # -f  файл с шаблонами
        open(PATFILE, $opt{f}) or die qq($Me: Can't open '$opt{f}': $!);

                                    # Проверить каждый шаблон в файле
        while ( defined($pattern = <PATFILE>) ) {
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 6.14 (продолжение)

```
            chomp $pattern;
            eval { 'foo' =" /$pattern/, 1 } or
                die "$Me: $opt{f}:$.: bad pattern: $@"
            push @patterns, $pattern;
        }
        close PATFILE;
    }
    else {                          # Проверить шаблон
        $pattern = $opt{e} || shift(@ARGV) || usage();
        eval { 'foo' =~ /$pattern/, 1 } or
            die "$Me: bad pattern: $@";
        @patterns = ($pattern);
    }

    if ($opt{H} || $opt{u>) {      # Выделить или подчеркнуть
        my $term = $ENV{TERM} || 'vt100';
        my $terminal;

        eval {                      # Попытаться найти служебные
                                    # последовательности для выделения
                    POSIX;         # или подчеркнуть через Term::Cap
            use Term::Cap;

            my $termios = POSIX::Termios->new();
            $termios->getattr;
            my $ospeed = $termios->getospeed;

            $terminal = Tgetent Term::Cap { TERM=>undef, OSPEED=>$ospeed }
        };

        unless ($@) {               # Если успешно, получить служебные
                                    # последовательности для выделения (-H)
            local $~W = 0;         # или подчеркивания (-u)
            ($SO, $SE) = $opt{H}
                ? ($terminal->Tputs('so'), $terminal->Tputs('se'))
                : ($terminal->Tputs('us'), $terminal->Tputs('ue'));
        }
        else {                     # Если попытка использования Term::Cap
                                    # заканчивается неудачей, получить
            ($SO, $SE) = $opt{H}   # служебные последовательности
                                    # командой tput
                ? ('tput -T $term smso', 'tput -T $term rmso')
                : ('tput -T $term smul', 'tput -T $term rmul')
        }
    }

    if ($opt{i}) {
        @patterns = map {"(?i)$_"} @patterns;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

if ($opt{p} || $opt{P}) {
    @patterns = map {"(?m)$_"} @patterns;
}

$opt{p}    && ($/ = '');
$opt{P}    && ($/ = eval(qq("$opt{P}")));     Я for -P  '%%\n'
$opt{w}    && (@patterns = map {'\b' . $_ . '\b'}@patterns);
$opt{'x'} && (@patterns = map {"^$_\$"}@patterns);
if (@ARGV) {
    $Mult = 1 if ($opt{r} || (@ARGV > 1) || -d$ARGV[0]) && !$opt{h};
}
$opt{1}   += $opt{l};                  # Единица и буква 1
$opt{H}   += $opt{u};
$opt{c}   +=  $opt{C};
$opt{'s'} += $opt{c};
$opt<1}   += $opt{'s'} && !$opt{c};   fl   Единица

@ARGV = ($opt{r} ?  '.' :  '-') unless @ARGV;
$opt{r} = 1 if !$opt{r} &&          @ARGV)    @ARGV;

$match_code  = '';
$match_code .= 'study;' if @patterns > 5;  # Может немного
                                           # ускорить работу

        (@patterns) { s(/)(\\/)g }

if ($opt{H}) {
                    (@patterns)
        $match_code .= "\$Matches += s/($pattern)/${SO}\$1${SE}/g;";
    }
>
elsif ($opt{v}) {
            $pattern (@patterns) {
        $match_code .= "\$Matches += !/$pattern/;";
    }
}
elsif ($opt{C}) •{
                    (@patterns)
        $match_code .= "\$Matches++ while /$pattern/g;";
    }
}
else {
                    (@patterns)
        $match_code .= "\$Matches++ if /$pattern/;";
    }
>
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 6.14 (продолжение)

```
    $matcher = eval "sub { $match_code }";
    die if $@;

            (\%opt, $matcher);
}



sub matchfile {
    $opt = shift;                  # Ссылка на хэш параметров
    $matcher = shift;              # Ссылка на функцию поиска совпадений

    my ($file, @list, $total, $name);
    local($_);
    $total = 0;

FILE: while (defined ($file = shift(@_))) {

        if (-d $file) {
            if (-1 $file && @ARGV != 1) {
                warn "$Me: \"$file\" is a symlinkto a directory\n"
                    if $opt->{T};
                next FILE;
            }
            if (!$opt->{r}) {
                warn "$Me: \"$file\" is a                    given\n"
                    if $opt->{T};
                next FILE;
            }
            unless (opendir(DIR, $file)) {
                unless ($opt->{'q'}) {
                    warn "$Me: can't opendir $file: $!\n";
                    $Errors++;
                }
                next FILE;
            }
            elist = ();

                push(@list, "$file/$_") unless /^\.{1,2}$/;
            }
            closedir(DIR);
            if ($opt->{t}) {
                my (@dates); .
                for (elist) { push(@dates, -M) }
                elist = elist[sort { $dates[$a] <=> $dates[$b] } 0..$#dates];
            >
            else {
                elist = sort @list;
            }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            matchfile($opt, $matcher, @list);    # process files
            next FILE;
        }

        if ($file eq '-') {
            warn "$Me:         from stdin\n" if -t STDIN && !$opt->{'q'};
            $name = '<STDIN>';
        }
        else {
            $name = $file;
            unless (-e $file) {
                warn qq($Me: file "$file" does not exist\n)
                    unless $opt->{'q'};
                $Errors++;
                next FILE;
            }
            unless (-f $file || $opt->{a}) {
                warn qq($Me: skipping non-plain file "$file"\n)
                    if $opt->{T};
                next FILE;
            }

            my ($ext) = $file =~ /\.([^.]+)$/;
            if (defined $ext && exists $Compress{$ext}) {
                $file = "$Compress{$ext} <$file |";
            }
            elsif (! (-T $file || $opt->{a})) {
                warn qq($Me; skipping binary file "$file"\n) if $opt->{T};
                next FILE;
            }
        }

        warn "$Me: checking $file\n" if $opt->{T};

        unless (open(FILE, $file)) {
            unless ($opt->{'q'}) {
                warn "$Me: $file: $!\n";
                $Errors++;
            }
            next FILE;
        }

        $total = 0;

        $Matches = 0;

LINE:   while (<FILE>) {
            $Matches = 0;
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 6.14 (продолжение)

```
            ##############
            &{$matcher}();        # Поиск совпадений


            next LINE unless $Matches;

            $total += $Matches;

            if ($opt->{p} || $opt->{P}) {
                s/\n{2,}$/\n/ if $opt->{p};
                chomp         if $opt->{P};
            }

            print("$name\n"), next FILE if $opt->{l};

            $opt->{'s'} || print $Mult && "$name:",
                $opt->{n} ? "$.." : "",
                $_,
                ($opt->{p} || $opt->{P}) && ('-' x 20) . "\n";

            next FILE if $opt->{l};                  # Единица
        }
    >
    continue {
        print $Mult && "$name.", $total, "\n" if $opt->{c};
    }
    $Grand_Total += $total;
}
```

6.23. Копилка регулярных выражений

Следующие регулярные выражения показались нам особенно полезными или интересными.

Римские цифры

```
m/^m*(d?c{0,3}|c[dm])(l?x{0,3}|x[lc])(v?i{0,3}|i[vx])$/i
```

Перестановка двух первых слов

```
s/(\S+))\s+)(\S+)/$3$2$1/
```

Ключевое слово = значение

```
m/(\w+)\s*=\s*(.*)\s*$/   # Ключевое слово в $1, значение - в $2
```

Строка содержит не менее 80 символов

```
ra/{80,}/
```

ММ/ДД/ГГ ЧЧ:ММ:СС

```
m|(\d+)/(\d+)/\d+) (\d+):(\d+):(\d+)|
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смена каталога

```
s(/usr/bin)(/usr/local/bin)g
```

Расширение служебных последовательностей %7E(шестн.)

```
s/%([0-9A-Fa-f][0-9A-Fa-f])/chr hex 41/ge
```

Удаление комментариев С (не идеальное)

```
s{
    /\*     # Начальный ограничитель
    .*?     # Минимальное количество символов
    \*/     # Конечный ограничитель
} []gsx;
```

Удаление начальных и конечных пропусков

```
s/^\s+//;
s/\s+$//;   .
```

Преобразование символа \ и следующего за ним n в символ перевода строки

```
s/\\n/\n/g;
```

Удаление пакетных префиксов из полностью определенных символов

```
s/^.*::/
```

IP-адрес

```
m/^[01]?\d\d|2[0-4]\d|25[0-5])\.([01]?\d\d|2[0-4]\d|25[0-5])\.
 ([01]?\d\d|2[0-4]\d|25[0-5])\.([01]?\d\d|2[0-4]\d|25[0-5])$/;
```

Удаление пути из полного имени файла

```
s(^.*/)()
```

Определение ширины строки с помощью TERMCAP

```
$cols = ( ($ENV{TERMCAP} || " ") =^ m/:co#(\d+):/ ) ? $1 : 80;
```

Удаление компонентов каталогов из имени программы и аргументов

```
($name = join(" ", map { s,^\S+/,,; $_ } ($0 @ARGV));
```

Проверка операционной системы

```
die "This isn't Linux" unless $^0 =^m/linux/i;
```

Объединение строк в многострочных последовательностях

```
s/\n\s+/ /g;
```

Извлечение всех чисел из строки

```
@nums = m/(\d+\.?\d*|\.\d+)/g;
```

Поиск всех слов, записанных символами верхнего регистра

```
@capwords = m/(\b[^\Wa-z0-9_]+\b)/g;
```

Поиск всех слов, записанных символами нижнего регистра

```
@capwords = m/(\b[^\WA-Z0-9_]+\b)/g;
```

Поиск всех слов, начинающихся с буквы верхнего регистра

```
@icwords = m/(\b[^\Wa-z0-9_][^\WA-Z0-9_]*\b)/;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Поиск ссылок в HTML-документах

```
@links = m/<A[^>]+?HREF\s*=\s*["']?([^'" >]+?)[ '"]?>/sig;
```

Поиск среднего инициала в $_

```
$initial = m/"\S+\s+(\s)\S*\s+\S/ ? $1 : "";
```

Замена кавычек апострофами

```
s/"([^"]*)"/''$1''/g
```

Выборка предложений (разделитель — два пробела)

```
{ local $/ ="";
  while (<>) {
    s/\n/ /g;
    s/ {3,>/  /g;
    push @sentences, m/(\S.*?[!?.])(?= |\Z)/g;
  }
}
```

ГГГГ-ММ-ДД

```
m/(\d{4})-(\d\d)-(\d\d)/  # ГГГГ в $1, ММ в $2 и ДД в $3
```

Выборка строк независимо от терминатора (завершающего символа)

```
push(@lines, $1)
  while ($input =- s/^([^\012\015]*)(\012\015?|\015\012?)//);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7 Доступ к файлам

Я — вечности наследник
В анналах времени...

А. Теннисон,
"Локсли-Холл"

Введение

Файлы занимают центральное место в обработке данных. Как и во всем остальном в Perl, простые операции с файлами выполняются **просто,** а сложные... как-нибудь да выполняются. Стандартные задачи (открытие файлов, чтение данных, запись данных) используют простые функции ввода/вывода и операторы, а более экзотические функции способны даже на асинхронный ввод/вывод и блокировку (locking) файлов.

В этой главе рассматривается механика *доступа* к файлам: открытие файлов, передача сведений о том, с какими файлами вы собираетесь работать, блокировка и т. д. Глава 8 "Содержимое файлов" посвящена работе с ***содержимым*** файлов: чтению, записи, перестановке строк и другим операциям, которые становятся возможными после получения доступа к файлу.

Следующий фрагмент выводит все строки **файла** ***/usr/local/widgets/data,*** содержащие слово `"blue"`:

```
open (INPUT, "< /usr/local/widgets/data")
    or die "Couldn't open /usr/local/widgets/data for          $!\n";

while (<INPUT>) {
    print if /blue/;
}
close(INPUT);
```

Получение файлового манипулятора

Доступ к файлам в Perl организуется при помощи *файловых манипуляторов* **(filehandle)** - таких, как `INPUT` из предыдущего примера. Манипулятор - это символическое имя, которое представляет файл в операциях чтения/записи. **Файло-**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

вые манипуляторы не являются переменными. В их именах отсутствуют префиксы $, @ или %, однако они наряду с функциями и переменными попадают в символьную таблицу Perl. По этой причине не всегда удается сохранить файловый манипулятор в переменной или передать его функции. Приходится использовать префикс *, который является признаком тип-глоба — базовой единицы символьной таблицы Perl:

```
$var = *STDIN;
mysub($var, *LOGFILE);
```

Файловые манипуляторы, сохраняемые в переменных подобным образом, не используются напрямую. Они называются *косвенными файловыми манипуляторами* (indirect filehandle), поскольку косвенно ссылаются на настоящие манипуляторы. Два модуля, IO::File (стал стандартным, начиная с версии 5.004) и FileHandle (стандартный с версии 5.000), могут создавать анонимные файловые манипуляторы.

Когда в наших примерах используются модули IO::File или IO::Handle, аналогичные результаты можно получить с применением модуля FileHandle, поскольку сейчас он является интерфейсным модулем (wrapper).

Ниже показано, как выглядит программа для поиска "blue" с применением модуля IO::File в чисто объектной записи:

```
use IO::File;

$input = IO::File->new("< /usr/local/widgets/data")
    or die "Couldn't open /usr/local/widgets/data for $!\n";

while (defined($line = $input->getline())) {
   chomp($line);
   STDOUT->print($line) if $line =~ /blue/;
}
$input->close();
```

Как видите, без прямого использования файловых манипуляторов программа читается намного легче. Кроме того, она гораздо быстрее работает.

Но поделимся одним секретом: из этой программы можно выкинуть все стрелки и вызовы методов. В отличие от большинства объектов, объекты IO::File *не обязательно* использовать объектно-ориентированным способом. В сущности, они представляют собой анонимные файловые манипуляторы и потому могут использоваться везде, где допускаются обычные косвенные манипуляторы. В рецепте 7.16 рассматриваются эти модули и префикс *. Модуль IO::File и символические файловые манипуляторы неоднократно встречаются в этой главе.

Стандартные файловые манипуляторы

Каждая программа при запуске получает три открытых глобальных файловых манипулятора: STDIN, STDOUT и STDERR. STDIN *(стандартный ввод)* является источником входных данных по умолчанию. В STDOUT *(стандартный вывод)* по умолчанию направляются выходные данные. В STDERR *(стандартный поток ошибок)* по умолчанию направляются предупреждения и ошибки. В интер-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

активных программах STDIN соответствует клавиатуре, а STDOUT и STDERR — экрану монитора:

```
while(<STDIN>)  {                    # Чтение из STDIN
    unless (/\d/) {
        warn "No digit found.\n";    # Вывод в STDERR
    }
    print "Read: ", $_;              И Запись в STDOUT
}
END { close(STDOUT)    or die "couldn't close STDOUT: $!"  >
```

Файловые манипуляторы существуют на уровне пакетов. Это позволяет двум пакетам иметь разные файловые манипуляторы с одинаковыми именами (по аналогии с функциями и переменными). Функция open связывает файловый манипулятор с файлом или программой, после чего его можно использовать для ввода/вывода. После завершения работы вызовите для манипулятора функцию close, чтобы разорвать установленную связь.

Операционная система работает с файлами через файловые дескрипторы, значение которых определяется функцией fileno. Для большинства файловых операций хватает манипуляторов Perl, однако в рецепте 7.19 показано, как файловый дескриптор преобразуется в файловый манипулятор, используемый в программе.

Операции ввода/вывода

Основные функции для работы с файлами в Perl — open, print, <...> (чтение записи) и close. Они представляют собой интерфейсные функции для процедур буферизованной библиотеки ввода/вывода C stdio. Функции ввода/вывода Perl документированы *в perlfunc(1)* и страницах руководства *stdio*(3S) вашей системы. В следующей главе операции ввода/вывода — такие, как оператор> <>, print, seek и tell — рассматриваются более подробно.

Важнейшей функцией ввода/вывода является функция open. Она получает два аргумента — файловый манипулятор и строку с именем файла и режимом доступа. Например, открытие файла */tmp/log* для записи и его связывание с манипулятором LOGFILE выполняется следующей командой:

```
open(LOGFILE, "> /tmp/log")  or die "Can't write /tmp/log: $!";
```

Три основных режима доступа — < (чтение), > (запись) и " (добавление). Дополнительные сведения о функции open приведены в рецепте 7.1.

При открытии файла или вызове практически любой системной функции[1] необходимо проверять возвращаемое значение. Не каждый вызов open заканчивается успешно; не каждый файл удается прочитать; не каждый фрагмент данных, выводимый функцией print, достигает места назначения. Многие программисты для повышения устойчивости своих программ проверяют результаты open, seek, tell и close. Иногда приходится вызывать и другие функции. В документации Perl описаны возвращаемые значения всех функций и операторов. При неудачном завершении системная функция возвращает undef (кроме функций wait, waitpid и

[1] Системной функцией называется обращение к сервису операционной системы. Термин не имеет отношения к функции system в языках C и Perl.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

syscall, возвращающих -1). Системное сообщение или код ошибки хранится в переменной $1 и часто используется в die или сообщениях warn.

Для чтения записей в Perl применяется оператор <МАНИПУЛЯТОР>, также часто дублируемый функцией Обычнозапись представляет собой одну строку, однако разделитель записей можно изменить (см. главу 8). Если МАНИПУЛЯТОР не указывается, Perl открывает и читает файлы из @ARGV, а если они не указаны — из STDIN. Нестандартные и просто любопытные применения этого факта описаны в рецепте 7.7.

С абстрактной точки зрения файл представляет собой обычный поток байтов. Каждый файловый манипулятор ассоциируется с числом, определяющим текущую позицию внутри файла. Текущая позиция возвращается функцией tell и устанавливается функцией seek. В рецепте 7.10 мы перезаписываем файл, обходясь без закрытия и повторного открытия, — для этого мы возвращаемся к началу файла функцией seek.

Когда надобность в файловом манипуляторе отпадает, закройте его функцией close. Функция получает один аргумент (файловый манипулятор) и возвращает true, если буфер был успешно очищен, а файл — закрыт, и false в противном случае. Закрывать все манипуляторы функцией close необязательно. При открытии файла, который был открыт ранее, Perl сначала неявно закрывает его. Кроме того, все открытые файловые манипуляторы закрываются при завершении программы.

Неявное закрытие файлов реализовано для удобства, а не для повышения надежности, поскольку вы не узнаете, успешно ли завершилась системная функция. Не все попытки закрытия завершаются успешно. Даже если файл открыт только для чтения, вызов close может завершиться неудачей — например, если доступ к устройству был утрачен из-за сбоя сети. Еще важнее проверять результат close, если файл был открыт для записи, иначе можно просто не заметить переполнения диска:

```
close(FH)    or die "FH didn't close: $!";
```

Усердный программист даже проверяет результат вызова close для STDOUT в конце программы на случай, если выходные данные были перенаправлены в командной строке, а выходная файловая система оказалась переполнена. Вообще-то об этом должна заботиться runtime-система, но она этого не делает.

Впрочем, проверка STDERR выглядит сомнительно. Даже если этот поток не закроется, как вы собираетесь на это реагировать?

Манипулятор STDOUT по умолчанию используется для вывода данных функциями print, printf и write. Его можно заменить функцией select, которая получает новый и возвращает предыдущий выходной манипулятор, используемый по умолчанию. Перед вызовом select должен быть открыт новый манипулятор вывода:

```
$old_fh = select(LOGFILE);     " Переключить вывод на LOGFILE
print "Countdown initiated ...\n";
select($old_fh);               # Вернуться к выводу на прежний манипулятор
print "You have 30 seconds to       minumum safety distance.\n";
```

Некоторые специальные переменные Perl изменяют поведение текущего файлового манипулятора вывода. Особенно важна переменная $|, которая управляет

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

буферизацией вывода для файловых манипуляторов. Буферизация рассматривается врецепте7.12.

Функции ввода/вывода в Perl делятся на буферизованные и небуферизованные (табл. 7.1). Несм отря на отдельные исключения, не следует чередовать их вызовы в программе. Связь между функциями, находящимися в одной строке таблицы, весьма условна Например, посемантике функция sys ичается от <. . .>, однако они находятся в одной строке, поскольку выполняют общую задачу — получение входных данных из файлового манипулятора.

Таблица 7.1 Функции ввода/вывода в Perl

Действие	Буферизованные функции	Небуферизованные функции
Открытие	open, sysopen	sysopen
Закрытие	close	close
Ввод	<.. >, readline	sysread
Вывод	print	syswrite
Позиционирование	seek, tell	sysseek

Позиционирование рассматривается в главе 8, однако мы также воспользуемся им в рецепте 7.10.

7.1. Открытие файла

Проблема

Известно имя файла. Требуется открыть его для чтения или записи **в Perl.**

Решение

Функция open отличается удобством, sysopen — точностью, а модуль IO::File позволяет работать с анонимным файловым манипулятором.

Функция open получает два аргумента: открываемый файловый манипулятор и строку с именем файла и специальными символами, определяющими режим открытия:

```
open(SOURCE, "< $path")
    or die "Couldn't open $path for          $!\n";

open(SINK, "> $path")
    or die "Couldn't open $path for writing: $!\n";
```

где SOURCE — файловый манипулятор для ввода, а SINK — для вывода.

Функции sysopen передаются три **или** четыре аргумента: файловый манипулятор, имя файла, режим и необязательный параметр, определяющий права доступа. **Режим** представляет собой **число,** конструируемое **из** констант модуля Fcntl:

```
use Fcntl;

sysopen(SOURCE, $path, O_RDONLY)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    or die "Couldn't open $path for          $!\n";

sysopen(SINK, $path, O_WRONLY)
    or die "Couldn't open $path for writing: $!\n";
```

Аргументы метода new модуля IO::File могут задаваться в стиле как open, так и sysopen. Метод возвращает анонимный файловый манипулятор. Кроме того, также возможно задание режима открытия в стиле fopen(3):

```
use IO::File;

# По аналогии с open
$sink = IO::File->new("> $filename")
    or die "Couldn't open $filename for writing: $!\n";

И По аналогии с sysopen
$fh = IO::File->new($filename, O_WRONLY|O_CREAT)
    or die "Couldn't open $filename for          $!\n";

# По аналогии с fopen(3) библиотеки stdio
$fh = IO::File->new($filename, "r+")
    or die "Couldn't open $filename for          write: $!\n";
```

Комментарий

Все операции ввода/вывода осуществляются через файловые манипуляторы независимо от того, упоминаются манипуляторы в программе или нет. Файловые манипуляторы не всегда связаны с конкретными файлами — они также применяются для взаимодействия с другими программами (см. главу 16 «Управление процессами и межпроцессные взаимодействия") и в сетевых коммуникациях (см. главу 17 «Сокеты»). Функция open также применяется для работы с файловыми дескрипторами, данная возможность рассматривается в рецепте 7.19.

Функция open позволяет быстро и удобно связать файловый манипулятор с файлом. Вместе с именем файла передаются сокращенные обозначения стандартных режимов (чтение, запись, чтение/запись, присоединение). Функция не позволяет задать права доступа для создаваемых файлов и вообще решить, нужно ли создавать файл. Если вам потребуются подобные возможности, воспользуйтесь функцией sysopen, которая использует константы модуля Fcntl для управления отдельными компонентами режима (чтение, запись, создание и усечение).

Большинство программистов начинает работать с open задолго до первого использования sysopen. В таблице показано соответствие между режимами функции open ("Файл"), константами sysopen ("Флаги") и строками fopen(3), передаваемыми IO::File->new («Символы»). Столбцы "Чтение" и "Запись" показывают, возможно ли чтение или запись для данного файлового манипулятора. "Присоединение" означает, что выходные данные всегда направляются в конец файла независимо от текущей позиции (в большинстве систем). В режиме усечения функция open уничтожает все существующие данные в открываемом файле.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Файл	Чтение	Запись	Присоединение	Создание	Очистка содержимого	Флаги O_	Символы
<файл	Да	Нет	Нет	Нет	Нет	RDONLY	"r"
>файл, режим открытия>	Нет	Да	Нет	Да	Да	WRONLY TRUNC CREAT	"w"
" файл>, режим открытия>	Нет	Да	Да	Да	Нет	WRONLY APPEND CREAT	"a"
+<файл	Да	Да	Нет	Нет	Нет	RDWR	"r+"
+>файл, режим открытия>	Да	Да	Нет	Да	Да	RDWR TRUNC CREAT	"w+"
+" файл>, режим открытия>	Да	Да	Да	Да	Нет	RDWR APPEND CREAT	"a+"

Подсказка: режимы +> и +" почти никогда не используются. В первом случае файл уничтожается еще до того, как он будет прочитан, а во втором часто возникают затруднения, связанные с тем, что указатель чтения может находиться в произвольной позиции, но при записи на многих системах почти всегда происходит переход в конец файла.

Функция sysopen получает три или четыре аргумента:

```
sysopen(FILEHANDLE, $name, $flags)        or die "Can't open $name : $!";
sysopen(FILEHANDLE, $name, $flags, $perms) or die "Can't open $name : $!";
```

Здесь $name — имя файла без "довесков" в виде < или +; $flags — число, полученное объединением констант режимов O_CREAT, O_WRONLY, O_TRUNC и т. д. операцией OR. Конкретный состав доступных констант O_ зависит от операционной системы. Дополнительные сведения можно найти в электронной документации (обычно open(2), но не всегда) или в файле /usr/include/fcntl.h. Обычно встречаются следующие константы;

O_RDONLY	Только чтение.
O_WRONLY	Только запись.
O_RDWR	Чтение и запись.
O_CREAT	Создание файла, если он не существует.
O_EXCL	Неудачное завершение, если файл уже существует.
O_APPEND	Присоединение к файлу.
O_TRUNC	Очистка содержимого файла.
O_NONBLOCK	Асинхронный доступ,

К числу менее распространенных констант принадлежат O_SHLOCK, O_EXLOCK, O_BINARY, O_NOCTTY и O_SYNC. Обращайтесь к странице руководства open(2) или к ее эквиваленту.

Если функции sysopen не передается аргумент $perms, Perl использует восьмеричное число 0666. Права доступа задаются в восьмеричной системе и учитыва-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ют текущее значение маски доступа (задаваемой функцией umask) процесса. В маске доступа сброшенные биты соответствуют запрещенным правам. Например, если маска равна 027 (группа не может записывать; прочие не могут читать, записывать или выполнять), то вызов sysopen с параметром **066** создает файл с правами **0640** (0666&~027 - 0640).

Если у вас возникнут затруднения с масками доступа, воспользуйтесь простым советом: передавайте значение **0666** для обычных файлов **и 0777** для каталогов **и** исполняемых файлов. **У** пользователя появляется выбор: если ему понадобятся защищенные файлы, то может выбрать маску **022, 027** или антиобщественную маску **077.** Как правило, решения из области распределения прав должны приниматься не программой, а пользователем. Исключения возникают при записи **в** файлы, доступ к которым ограничен: почтовые файлы, cookies **в** Web-броузерах, файлы *.rhosts* и т. д. Короче говоря, функция sysopen почти никогда не вызывается с аргументом **0644,** так как у пользователя пропадает возможность выбрать более либеральную маску.

Приведем примеры практического использования open и sysopen.

Открытие файла для чтения:

```
open(FH, "< $path")                                or die$!;
sysopen(FH, $path, O_RDONLY)                       or die$!;
```

Открытие файла для записи (если файл не существует, он создается, а если существует — усекается):

```
open(FH, "> $path")                                or die$!
sysopen(FH, $path, O_WRONLY|O_TRUNC|O_CREAT)       or die$!
sysopen(FH, $path, O_WRONLY|O_TRUNC|O_CREAT, 0600) or die$!
```

Открытие файла для записи с созданием нового файла (файл не должен существовать):

```
sysopen(FH, $path, O_WRONLY|O_EXCL|O_CREAT)        or die$!;
sysopen(FH, $path, O_WRONLY|O_EXCL|O_CREAT, 0600)  or die$!;
```

Открытие файла для присоединения (в случае необходимости файл создается):

```
open(FH, ">> $path")                               or die$!;
sysopen(FH, $path, O_WRONLY|O_APPEND|O_CREAT)      or die$!;
sysopen(FH, $path, O_WRONLY|O_APPEND|O_CREAT, 0600) or die$!;
```

Открытие файла для присоединения (файл должен существовать):

```
sysopen(FH, $path, O_WRONLY|O_APPEND)              or die$!;
```

Открытие файла для обновления (файл должен существовать):

```
open(FH, "+< $path")                               or die$!;
sysopen(FH, $path, O_RDWR)                         or die$!;
```

Открытие файла для обновления **(в** случае необходимости файл создается):

```
sysopen(FH, $path, O_RDWR|O_CREAT)                 or die$!;
sysopen(FH, $path, O_RDWR|O_CREAT, 0600)           or die$!;
```

Открытие файла для обновления (файл не должен существовать):

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sysopen(FH, $path, O_RDWR|O_EXCL|O_CREAT)          or die$!;
sysopen(FH, $path, O_RDWR|O_EXCL|O_CREAT, 0600)    or die$!;
```

Маска **0600** всего **лишь** поясняет, как создаются файлы с ограниченным доступом. Обычно этот аргумент пропускается.

Смотри также

Описание функций open, sysopen и umask в *perlfunc*(1); документация по стандартным модулям IO::File и Fcntl; страницы руководства open(2), fopen(3) и umask(2); рецепт **7.2.**

7.2. Открытие файлов с нестандартными именами

Проблема

Требуется открыть файл с нестандартным именем — например, "-"; начинающимся с символа <, > или |; содержащим начальные или конечные пропуски; заканчивающимся символом |. Функция open не должна принимать эти функции за служебные, поскольку вам нужно совершенно иное.

Решение

Выполните предварительное преобразование:

```
$filename =^ s#^(\s)#./$1#;
open(HANDLE, "< $filename\0")        or die "cannot open $filename : $!\n";
```

Или просто воспользуйтесь функцией sysopen:

```
sysopen(HANDLE, $filename, O_RDONLY) or die "cannot open $filename : $!\n";
```

Комментарий

Функция open определяет имя файла и режим открытия по одному строковому аргументу. Если имя файла начинается с символа, обозначающего один из режимов, open вполне может сделать что-нибудь неожиданное. Рассмотрим следующий фрагмент:

```
$fllename = shift @ARGV;
open(INPUT, $filename)                or die "cannot open $filename : $!\n";
```

Если пользователь указывает в командной строке файл ">/etc/passwd", программа попытается открыть */etc/passwd* для записи — со всеми вытекающими последствиями! Режим можно задать и явно (например, для записи):

```
open(OUTPUT, ">$filename")
    or die "Couldn't open $filename for writing: $!\n";
```

но даже в этом случае пользователь может ввести имя ">data", после чего программа **будет дописывать** данные **в** конец файла data вместо того, чтобы стереть прежнее содержимое.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Самое простое решение — воспользоваться функцией sysopen, у которой режим и имя файла передаются в разных аргументах:

```
use Fcntl;                # Для файловых констант

sysopen(OUTPUT, $filename, O_WRONLY|O_TRUNC)
    or die "Couldn't open $filename for writing: $!\n";
```

А вот как добиться того же эффекта с функцией open для имен файлов, содержащих начальные или конечные пропуски:

```
$file =- s#^(\s)#./$1#;
open(HANDLE, "> $file\0")
        or die "Could't open $file for OUTPUT : $!\n";
```

Такая подстановка защищает исходные пропуски, но не в абсолютных именах типа " /etc/passwd", а лишь в относительных (" passwd"). Функция open не считает нуль-байт ("\0") частью имени файла, но благодаря ему не игнорируются конечные пропуски.

Волшебная интерпретация файловых имен в функции open почти всегда оказывается удобной. Вам никогда не приходится обозначать ввод или вывод с помощью особой формы "-". Если написать фильтр и воспользоваться простой функцией open, пользователь сможет передать вместо имени файла строку "gzip -dc bible.gz|" — фильтр автоматически запустит программу распаковки.

Вопросы безопасности open актуальны лишь для программ, работающих в особых условиях. Если программа должна работать под управлением чего-то другого — например, сценариев CGI или со сменой идентификатора пользователя, — добросовестный программист всегда учтет возможность ввода пользователем собственного имени файла, при котором вызов open для простого чтения превратится в перезапись файла или даже запуск другой программы. Параметр командной строки Perl -T обеспечивает проверку ошибок.

Смотри также

Описание функций open и sysopen в *perlfunc*(1); рецепты 7.1, 7.7, 16.2, 19.4 и 19.6.

7.3. Тильды в именах файлов

Проблема

Имя файла начинается с тильды (например, *~username/blah*), однако функция open не интерпретирует его как обозначение

Решение

Выполните ручное расширение с помощью следующей подстановки:

```
$filename =^ s{ ^ ~ ( [^/]* ) }
               < $1
                    ? (getpwnam($1))[7]
                    : ( $ENV{HOME} ||$ENV{LOGDIR}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                    || (getpwuid($>))[7]
                )
}ex;
```

Комментарий

Нас интересуют следующие применения тильды:

~user
~user/blah

~/blah

где ***user*** — имя пользователя.

Если ~ не сопровождается никаким именем, используется домашний каталог текущего пользователя.

В данной подстановке использован параметр /e, чтобы заменяющее выражение интерпретировалось как программный код Perl. Если за тильдой указано имя пользователя, оно сохраняется в $1 и используется `getpwnam` для выбора домашнего каталога пользователя из возвращаемого списка. Найденный каталог образует заменяющую строку. Если за тильдой не указано имя пользователя, подставляется либо текущее значение переменной окружения `HOME` или `LOGDIR`. Если эти переменные не определены, задается домашний каталог текущего пользователя.

Смотри также

Описание функции `getpwnam` в *perlfunc*(1); man-страница `getpwnam`(2) вашей системы; рецепт **9.6.**

7.4. Имена файлов в сообщениях об ошибках

Проблема

Программа работает с файлами, однако в предупреждения и сообщения об ошибках Perl включается только последний использованный файловый манипулятор, а не имя файла.

Решение

Воспользуйтесь именем файла вместо манипулятора:

```
open($path, "< $path")
    or die "Couldn't open $path for $!\n";
```

Комментарий

Стандартное сообщение об ошибке выглядит так:

```
Argument "3\n" isn't numeric in multiply at tallyweb line 16, <LOG> chunk 17.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Манипулятор LOG не несет полезной информации, поскольку вы не знаете, с каким файлом он был связан. Если файловый манипулятор косвенно передается через имя файла, предупреждения и сообщения об ошибках Perl становятся более содержательными:

```
Argument "3\n" isn't numeric in multiply at tallyweb
      line 16, </usr/local/data/mylog3.dat> chunk 17.
```

К сожалению, этот вариант не работает при включенной директиве strict поскольку переменная $path в действительности содержит не файловый манипулятор, а всего лишь строку, которая иногда ведет себя как манипулятор. Фрагмент (chunk), упоминаемый в предупреждениях и сообщениях об ошибках, представляет собой текущее значение переменной $..

Смотри также

Описание функции open в *perlfunc*(1)рецепт 7.1.

7.5. Создание временных файлов

Проблема

Требуется создать временный файл и автоматически удалить его при завершении программы. Допустим, вы хотите записать временный конфигурационный файл, который будет передаваться запускаемой программе. Его имя должно быть известно заранее. В других ситуациях нужен временный файл для чтения и записи данных, причем его имя вас не интересует.

Решение

Если имя файла не существенно, воспользуйтесь методом класса new_tmpfile модуля IO::File для получения файлового манипулятора, открытого для чтения и записи:

```
use IO::File;

$fh = IO::File->new_tmpfile
    or die "Unable to make new temporary file: $!";
```

Если имя файла должно быть известно, получите его функцией tmpnam из модуля POSIX и откройте файл самостоятельно:

```
use IO::File;
use POSIX qw(tmpnam);

# Пытаться получить временное имя файла до тех пор,
# пока не будет найдено несуществующее имя
do { $name = tmpnam() }
    until $fh = IO::File->new($name, O_RDWR|O_CREAT|O_EXCL);

# Установить обработчик, который удаляет временный файл
# при нормальном или аварийном завершении программы
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
END { unlink($name) or die "Couldn't unlink $name : $!" }

й Перейти к использованию файла...
```

Комментарий

Если все, что вам нужно, — область для временного хранения данных, воспользуйтесь методом `new_tmpfile` модуля IO::File. Он возвращает файловый манипулятор для временного файла, открытого в режиме чтения/записи фрагментом следующего вида:

```
for (;;) {
    $name = tmpnam();
    sysopen(TMP, $tmpnam, O_RDWR | O_CREAT | O_EXC) && last;
}
unlink $tmpnam;
```

Файл автоматически удаляется при нормальном или аварийном завершении программы. Вам не удастся определить имя файла и передать другому процессу, потому что у него нет имени. В системах с поддержкой подобной семантики имя удаляется еще до завершения метода. Впрочем, открытый файловый манипулятор может наследоваться производными процессами[1].

Ниже показан пример практического применения `new_tmpfile`. Мы создаем временный файл, выполняем запись, возвращаемся к началу и выводим записанные данные:

```
use IO::File;

$fh = IO::File->new_tmpfile      or die "IO::File->new_tmpfile: $!";
$fh->autoflush(1);
print( $fh "$i\n" while $i++ < 10;
seek($fh, 0, 0);
print "Tmp file has: ", <$fh>;
```

Во втором варианте создается временный файл, имя которого можно передать другому процессу. Мы вызываем функцию `POSIX::tmpnam`, самостоятельно открываем файл и удаляем его после завершения работы. Перед открытием файла мы не проверяем, существует ли файл стаким именем, поскольку при этом может произойти подмена — кто-нибудь создаст файл между проверкой и созданием[2]. Вместо этого `tmpnam` вызывается в цикле, что гарантирует создание нового файла и предотвращает случайное удаление существующих файлов. Теоретически метод `new_tmpfile` не должен возвращать одинаковые имена разным процессам.

Смотри также

Документация по стандартным модулям IO::File и **POSIX;** рецепт **7.19;** страница руководства *tmpnam*(3) вашей системы.

[1] Но перед вызовом exec следует присвоить `$^F` хотя бы fileno($fh).

[2] См. рецепт 19.4.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.6. Хранение данных в тексте программы

Проблема

Некоторые данные должны распространяться вместе с программой и интерпретироваться как файл, но при этом они не должны находиться в отдельном файле.

Решение

Лексемы DATA и END после исходного текста программы отмечают начало блока данных, который может быть прочитан программой или модулем через файловый манипулятор DATA.

В модулях используется лексема DATA :

```
while (<DATA>) {
    # Обработать строку
}
__DATA__
# Данные
```

Аналогично используется __END__ в главном файле программы:

```
while (<main::DATA>) {
    # Обработать строку
}
__END__
# Данные
```

Комментарий

Лексемы DATA и END обозначают логическое завершение модуля или сценария перед физическим концом файла. Текст, находящийся после DATA или END , может быть прочитан через файловый манипулятор DATA уровня пакета. Предположим, у нас имеется гипотетический модуль Primes; текст после __DATA__ в файле *Primes.pm* может быть прочитан через файловый манипулятор Primes::DATA.

Лексема END представляет собой синоним DATA в главном пакете. Текст, следующий после лексем END в модулях, недоступен.

Появляется возможность отказаться от хранения данных в отдельном файле й перейти к построению автономных программ. Такая возможность нередко используется для документирования. Иногда в программах хранятся конфигурационные или старые тестовые данные, использованные при разработке программ, — они могут пригодиться в процессе отладки.

Манипулятор DATA также применяется для определения размера или даты последней модификации текущей программы или модуля. В большинстве систем переменная $0 содержит полное имя файла для работающего сценария. В тех системах, где значение $0 оказывается неверным, можно воспользоваться манипулятором DATA для определения размера, даты модификации и т. д. Вставьте в конец файла специальную лексему __DATA__ (и предупреждение о том, что __DATA__ не следует удалять), и файловый манипулятор DATA будет связан с файлом сценария.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use POSIX qw(strftime);

$raw_time = (stat(DATA))[9];
$size     = -s DATA;
$kilosize = int($size / 1024) . 'k';

print "<P>Script size is $kilosize\n";
print strftime("<P>Last script update: %c (%Z)\n", localtime($raw_time));

__DATA__
DO NOT REMOVE THE PRECEDING LINE '
Everything else in this file will be
```

Смотри также

Раздел "Scalar Value Constructors» *perldata*(1).

7.7. Создание фильтра

Проблема

Вы хотите написать программу, которая получает из командной строки список файлов. Если файлы не заданы, входные данные читаются из STDIN. При этом пользователь должен иметь возможность передать программе "-" для обозначения STDIN или "someprogram |"для пол учения выходных данных другой программы. Программа может непосредственно модифицировать файлы или выводить результаты на основании входных данных.

Решение

Читайте строки оператором, оператор> <>:

```
while (<>) {
    # Сделать что-то со строкой
}
```

Комментарий

Встречая конструкцию:

```
while (<>) {
    B ...
}
```

Perl преобразует ее к следующему виду[1]:

```
unshift(@ARGV, '-') unless @ARGV;
while($ARGV = shift @ARGV) {
    unless (open(ARGV, $ARGV)) {
```

[1] В программе показанный фрагмент не будет работать из-за внутренней специфики ARGV.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        warn "Can't open $ARGV: $!\n";
        next;
    }
    while (defined($_ = <ARGV>)) {
        #...
    }
}
```

Внутри цикла с помощью ARGV и $ARGV можно получить дополнительные данные или узнать имя текущего обрабатываемого файла. Давайте посмотрим, как это делается.

Общие принципы

Если пользователь не передает аргументы, Perl заносит в @ARGV единственную строку, "-". Это сокращенное обозначение соответствует STDIN при открытии для чтения и STDOUT — для записи. Кроме того, пользователь может передать "-" в командной строке вместо имени файла для получения входных данных из STDIN.

Далее в цикле из @ARGV последовательно извлекаются аргументы, а имена файлов копируются в глобальную переменную $ARGV. Если файл не удается открыть, Perl переходит к следующему файлу. В противном случае начинается циклическая обработка строк открытого файла. После завершения обработки открывается следующий файл, и процесс повторяется до тех пор, пока не будет исчерпано все содержимое @ARGV.

При вызове open не используется форма open(ARGV, "> $ARGV"). Это позволяет добиться интересных эффектов — например, передать в качестве аргумента строку "gzip -dc file.gz |", чтобы программа получила в качестве входных данных результаты команды "gzip -dc file.gz". Такое применение open рассматривается в рецепте **16.15.**

Массив @ARGV может изменяться перед циклом или внутри него. Предположим, вы хотите, чтобы при отсутствии аргументов входные данные читались не из **STDIN,** а из всех программных и заголовочных файлов C и C++. Вставьте следующую строку перед началом обработки <ARGV>:

```
@ARGV = glob("*.[Cch]") unless @ARGV;
```

Перед началом цикла следуетобработать аргументы командной строки — либо с помощью модулей Getopt (см. главу 15 "Пользовательские интерфейсы"), либо вручную:

```
Я Аргументы 1: Обработка необязательного флага -c
if (@ARGV && $ARGV[0] eq '-c') {
    $chop_first++;
    shift;
}

# Аргументы 2: Обработка необязательного флага -NUMBER
if (@ARGV && $ARGV[0] =^ /^-(\d+)$/) {
    $columns = $1;
    shift;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

# Аргументы 3: Обработка сгруппированных флагов -a, -i -n, и -u
while (@ARGV && $ARGV[0] =~ /^-(.+)/ & (shift, ($_ = $1), 1)) {
    next if /^$/;
    s/a// && (++$append,
    s/i// &&

    s/u// && (++$unbuffer,
    die "usage: $0 [-ainu] [filenames] ...\n";
}
```

Если не считать неявного перебора аргументов командной строки, о не выделяется ничем особенным. Продолжают действовать все специальные переменные, управляющие процессом ввода/вывода (см. главу 8). Переменная $/ определяет разделитель записей, а $. содержит номер текущей строки (записи). Если $/ присваивается неопределенное значение, то при каждой операции чтения будет получено не объединенное содержимое всех файлов, а полное содержимое одного файла:

```
undef $/;
while (o) {
    # Теперь в $_ находится полное содержимое файла,
    # имя которого хранится в $ARGV
}
```

Если значение $/ локализовано, старое значение автоматически восстанавливается при выходе из блока:

```
{                # Блок для local
    local $/;    # Разделитель записей становится неопределенным
    while (<>) {
        # Сделать что-то; в вызываемых функциях
        # значение $/ остается неопределенным
    }
}                # Восстановить $/
```

Поскольку при обработке <ARGV> файловые манипуляторы никогда не закрываются явно, номер записи $. не сбрасывается. Если вас это не устраивает, самостоятельно организуйте явное закрытие файлов для сброса $.:

```
while (<>) {
    print "$ARGV:$.:$_";
    close ARGV if eof;
}
```

Функция eof проверяет достижение конца файла при последней операции чтения. Поскольку последнее чтение выполнялось через манипулятор ARGV, eof сообщает, что мы находимся в конце текущего файла. В этом случае файл закрывается, а переменная $. сбрасывается. С другой стороны, специальная запись eof() с круглыми скобками, но без аргументов проверяет достижение конца всех файлов при обработке <ARGV>.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Параметры командной строки

В Perl предусмотрены специальные параметры командной строки — **-n**, -p и -i, упрощающие написание фильтров и однострочных программ.

Параметр -n помещает исходный текст программы внутрь цикла `while(<>)`. Обычно он используется в фильтрах типа grep или программах, которые накапливают статистику по прочитанным данным.

Пример 7.1. findlogin1

```
#!/usr/bin/perl
# findlogin1 - вывести все строки, содержащие подстроку "login"
while (<>) {        # Перебор файлов в командной строке
    print if /login/;
}
```

Программу из примера 7.1 можно записать так, как показано в примере 7.2.

Пример 7.2. findlogin2

```
#!/usr/bin/perl -n
# findlogin2 - вывести все строки, содержащие подстроку "login"
print if /login/;
```

Параметр -n может объединяться с -e для выполнения кода Perl из командной строки:

```
% perl -ne 'print if /login/'
```

Параметр -p аналогичен -n, однако он добавляет `print` в конец цикла. Обычно он используется в программах для преобразования входных данных.

Пример 7.3. lowercase1

```
#!/usr/bin/perl
# lowercase - преобразование всех строк в нижний регистр

use locale;
while (<>) {                # Перебор в командной строке
    s/([^\W0-9_])/\l$1/g;   # Перевод всех букв в нижний регистр
    print;
}
```

Программу из примера 7.3 можно записать так, как показано в примере 7.4.

Пример 7.4. lowercase2

```
#!/usr/bin/perl -p
# lowercase - преобразование всех строк в нижний регистр
use locale;
s/([^\W0-9_])/\l$1/g;   # Перевод всех букв в нижний регистр
```

Или непосредственно в командной строке следующего вида:

```
% perl -Mlocale -pe 's/([^\W0-9_])/\l$1/g'
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При использовании *-n* или *-p* для неявного перебора входных данных для всего цикла негласно создается специальная метка LINE:. Это означает, что из внутреннего цикла можно перейти к следующей входной записи командой next LINE (аналог next в awk). При закрытии ARGV происходит переход к следующему файлу (аналог nextfile в awk). Обе возможности продемонстрированы в примере 7.5.

Пример 7.5. countchunks

```
#!/usr/bin/perl -n
№ countchunks - подсчет использованных слов
# с пропуском комментариев. При обнаружении   END   или   DATA
# происходит переход к следующему файлу.
for (split /\W+/) {
    next LINE if /^#/;
    close ARGV if /__(DATA|END)__/;
    $chunks++;
}
END { print "Found $chunks chunks\n" }
```

В файле *.history*, создаваемым командным интерпретатором *tcsh,* перед каждой строкой указывается время, измеряемое **в** секундах с начала эпохи:

```
#+0894382237
less /etc/motd
#+0894382239
vi ~/.exrc
#+0894382242
date
#+0894382239
who
#+0894382288
telnet home
```

Простейшая однострочная программа приводит его к удобному формату:

```
%perl -pe 's/^#\+(\d+)\n/localtime($1) . " "/e'
Tue May  5 09:30:37 1998     less /etc/motd
Tue May  5 09:30:39 1998     vi V. exrc
Tue May  5 09:30:42 1998     date
Tue May  5 09:30:42 1998     who
Тие May  5 09:30:28 1998     telnet home
```

Параметр **-i** изменяет каждый файл **в** командной строке. Он описан **в** рецепте **7.9** и обычно применяется **в** сочетании с **-p.**

Для работы с национальными наборами символов используется директива use locale.

Смотри также

perlrun(1); рецепты 7.9; 16.6.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.8. Непосредственная модификация файла с применением временной копии

Проблема

Требуется обновить содержимое файла на месте. При этом допускается применение временного файла.

Решение

Прочитайте данные из исходного файла, запишите изменения во временный файл и затем переименуйте временный файл в исходный:

```
open(OLD, "< $old")          or die "can't open $old: $!";
open(NEW, "< $new")           or die "can't open $new: $!";
select(NEW);                 # Новый файловый манипулятор,
                             # используемый print по умолчанию
while (<OLD>) {
    # Изменить $_, затем...
    print NEW $_             or die "can't write $new: $!";
}
close(OLD)                   or die "can't close $old: $!";
close(NEW)                   or die "can't close $new: $!";
rename($old, "$old.orig")    or die "can't        $old to $old.orig: $!";
            $old)            or die "can't                $old: $!";
```

Такой способ лучше всего приходит для обновления файлов "на месте".

Комментарий

Этот метод требует меньше памяти, чем другие подходы, не использующие временных файлов. Есть и другие преимущества — наличие резервной копии файла, надежность и простота программирования.

Показанная методика позволяет внести в файл те же изменения, что и другие версии, не использующие временных файлов. Например, можно вставить новые строки перед 20-й строкой файла:

```
while (<OLD>) {
    if ($. == 20) {
        print NEW "Extra line 1\n";
        print NEW "Extra line 2\n";
    }
    print NEW $_;
}
```

Или удалить строки с 20 по 30:

```
while (<OLD>) {
    next if 20 .. 30;
    print NEW $_;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Обратите внимание: функция rename работает лишь в пределах одного каталога, поэтому временный файл должен находиться в одном каталоге с модифицируемым.

Программист-перестраховщик непременно заблокирует файл на время обновления.

Смотри также

Рецепты 7.1; 7.9–7.10

7.9. Непосредственная модификация файла с помощью параметра -i

Проблема

Требуется обновить файл на месте из командной строки, но вам лень[1] возиться с файловыми операциями из рецепта 7.8.

Решение

Воспользуйтесь параметрами -i и -p командной строки Perl. Запишите свою программу в виде строки:

```
% perl -i.orig -p 'ФИЛЬТР' файл1 файл2 файл3 ...
```

Или воспользуйтесь параметрами в самой программе:

```
#!/usr/bin/perl -i.orig -p
# Фильтры
```

Комментарий

Параметр командной строки -i осуществляет непосредственную модификацию файлов. Он создает временный файл, как и в предыдущем рецепте, однако Perl берет на себя все утомительные хлопоты с файлами. Используйте -i в сочетании с -p (см. рецепт 7.7), чтобы превратить:

```
% perl -pi.orig -e 's/DATE/localtime/e'
```

в следующий фрагмент:

```
while (<>) {
    if ($ARGV ne $oldargv) {        # Мы перешли к следующему файлу?
        rename($ARGV, $ARGV . '.orig');
        open(ARGVOUT, ">$ARGV");    # Плюс проверка ошибок
        select(ARGVOUT);
        $oldargv = $ARGV;
    }
    s/DATE/localtime/e;
}
continue{
```

[1] Конечно, имеется в виду лень творческая, а не греховная.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    print;
}
select (STDOUT);                    # Восстановить стандартный вывод
```

Параметр -i заботится о создании резервных копий (если вы не желаете сохранять исходное содержимое файлов, используйте -i вместо **-i.orig**), а -p заставляет Perl перебирать содержимое файлов, указанных в командной строке (или STDIN при их отсутствии).

Приведенная **выше** однострочная программа приводит данные:

```
Dear Sir/Madam/Ravenous Beast,
    As of DATE, our              your account
is overdue.  Please settle by the end of, the month.
Yours in cheerful usury,
    --A. Moneylender
```

к следующему виду:

```
Dear Sir/Madam/Ravenous Beast,
    As of Sat Apr 25 12:28:33 1998, our              your account
is overdue.  Please settle by the end of the month.
Yours in cheerful usury,
    --A. Moneylender
```

Этот параметр заметно упрощает разработку и чтение программ-трансляторов. Например, следующий фрагмент заменяет все изолированные экземпляры "hisvar" на "hervar" во всех файлах C, C++ и *yacc*:

```
%perl -i.old -pe 's{\bhisvar\b}{hervar}g' *.[Cchy]
```

```
%perl -i.old -ne 'print unless /^START$/ .. /^END$/' bigfile.text
```

Действие -i может включаться и выключаться с помощью специальной переменной $^I. Инициализируйте @ARGV и затем примените <> так, как применили бы -i для командной строки:

```
# Организовать перебор файлов *.c в текущем каталоге,
# редактирование на месте и сохранение старого файла с расширением .orig
local $^I   = '.orig';          # Эмулировать -i.orig
local @ARGV = glob("*.c");      # Инициализировать список файлов
while (<>) {
    if ($. == 1) {
        print "This line should appear at the top of each file\n";
    }
    s/\b(p)earl\b/{1}erl/ig;     # Исправить опечатки с сохранением регистра
    print;
} continue {close ARGV if eof}
```

Учтите, что при создании резервной копии предыдущая резервная копия уничтожается.

Смотри также

Описание переменных **$^I** и **$.** *в perlvar(\);* описание оператора . . **в** разделе "Range Operator" *perlop(1); perlrun*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.10. Непосредственная модификация файла без применения временного файла

Проблема

Требуется вставить, удалить или изменить одну или несколько строк файла. При этом вы не хотите (или не можете) создавать временный файл.

Решение

Откройте файл в режиме обновления ("+<"), прочитайте все его содержимое в массив строк, внесите необходимые изменения в массиве, после чего перезапишите файл и выполните усечение до текущей позиции.

```
open(FH, "+< FILE"          or die "Opening: $!";
@ARRAY = <FH>;
# Модификация массива ARRAY
seek(FH,0,0)                or die "Seeking: $!";
print FH @ARRAY             or die "Printing: $!";
truncate(FH,tell(FH))       or die "Truncating: $!";
close(FH)                   or die "Closing: $!";
```

Комментарий

Как сказано во введении, операционная система интерпретирует файлы как неструктурированные потоки байтов. Из-за этого вставка, непосредственная модификация или изменение отдельных битов невозможны (кроме особого случая, рассматриваемого в рецепте 8.13 — файлов с записями фиксированной длины). Для хранения промежуточных данных можно воспользоваться временным файлом. Другой вариант — прочитать файл в память, модифицировать его и записать обратно.

Чтение в память всего содержимого подходит для небольших файлов, но с большими возникают сложности. Попытка применить его для 800-мегабайтных файлов журналов на Web-сервере приведет либо к переполнению виртуальной памяти, либо общему сбою системы виртуальной памяти вашего компьютера. Однако для файлов малого объема подойдет такое решение:

```
open(F, "+< $infile")       or die "can't read $infile: $!";
$out = '';
while (<F>) {
    s/DATE/localtime/eg;
    $out .= $_;
}
seek(F, 0, 0)               or die "Seeking: $!";
print F $out                or die "Printing: $!";
truncate(F, tell(F))        or die "Truncating: $!";
close(F)                    or die "Closing: $!";
```

Другие примеры операций, которые могут выполняться на месте, приведены в рецептах главы 8.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Этот вариант подходит лишь для самых решительных. Он сложен в написании, расходует больше памяти (теоретически — *намного* больше), не сохраняет резервной копии и может озадачить других программистов, которые попытаются читать данные из обновляемогофайла. Как правило, он не оправдывает затраченных усилий.

Если вы особо мнительны, не забудьте заблокировать файл.

Смотри также

Описаниефункцийseek,truncate,openиsysopen*perlfunc(1);* рецепты7.8—

7.11. Блокировка файла

Проблема

Несколько процессов одновременно пытаются обновить один и тот же файл.

Решение

Организуйте условную блокировку с помощью функции flock:

```
open(FH, "+< $path")            or die "can't open $path: $!";
flock(FH,2)                     or die "can't flock $path: $!";
# Обновить файл, затем...
close(FH)                       or die "can't close $path: $!";
```

Комментарий

Операционные системы сильно..отличаются по типу и степени надежности используемых механизмов блокировки. Perl старается предоставить программисту рабочее решение даже в том случае, если операционная система использует другой базовый механизм. Функция flock получает два аргумента: файловый манипулятор и число, определяющее возможные действия с данным манипулятором. Числа обычно представлены символьными константами типа LOCK_EX, имена которых можно получить из модуля Fcntl или IO::File.

Символические константы LOCK_SH, LOCK_EX, LOCK_UN и LOCK_NB появились в модуле Fcntl лишь начиная с версии 5.004, но даже теперь они доступны лишь по специальному запросу с тегом :flock. Онираven соответственно 1, 2, 4и 8, и эти значения можно использовать вместо символическихконстант. Нередко встречается следующая запись:

```
sub LOCK_SH()   { 1 }   # Совместная блокировка (для чтения)
sub LOCK_EX()   { 2 }   # Монопольная блокировка (для записи)
sub LOCK_NB()   { 4 } tf Асинхронный запрос блокировки
sub LOCK_UN()   { 8 >   # Снятие блокировки (осторожно!)
```

Блокировки делятся на две категории монопольные (exclusive). Термин "монопольный" может ввести вас в заблуждение, поскольку процессы не обязаны соблюдать блокировку файлов. Иногда говорят, что flock реализует *условную блокировку,* чтобы операционная система могла приостано-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

вить все операции записи в файл до того момента, когда с ним закончит работу последний процесс чтения.

Условная блокировка напоминает светофор на перекрестке. Светофор работает лишь в том случае, если люди обращают внимание на цвет сигнала: красный или зеленый — или желтый для условной блокировки. Красный цвет не останавливает движение; он всего лишь сообщает, что движение следует прекратить. Отчаянный, невежественный или просто наглый водитель проедет через перекресток независимо от Сигнала светофора. Аналогично работает и функция flock — она тоже блокирует другие вызовы flock, а не процессы, выполняющие ввод/вывод. Правила должны соблюдаться всеми, иначе могут произойти (и непременно произойдут) несчастные случаи.

Добропорядочный процесс сообщает о своем намерении прочитать данные из файла, запрашивая блокировку LOCK_SH. Совместная блокировка файла может быть установлена сразу несколькими процессами, поскольку они (предположительно) не будут изменять данные. Если процесс собирается произвести запись в файл, он должен запросить монопольную блокировку с помощью LOCK_EX. Затем операционная система приостанавливает этот процесс до снятия блокировок остальными процессами, после чего приостановленный процесс получает блокировку и продолжает работу. Можно быть уверенным в том, что на время сохранения блокировки никакой другой процесс не сможет выполнить flock(FH, LOCK_EX) для того же файла. Это похоже на другое утверждение — "в любой момент для файла может быть установлена лишь одна монопольная блокировка", но не совсем эквивалентно ему. В некоторых системах дочерние процессы, созданные функцией fork, наследуют от своих родителей не только открытые файлы, но и установленные блокировки. Следовательно, при наличии монопольной блокировки и вызове fork без exec производный процесс может унаследовать монопольную блокировку файла.

Функция flock по умолчанию приостанавливает процесс. Указывая флаг LOCK_NB, при запросе можно получить блокировку без приостановки. Благодаря этому можно предупредить пользователя об ожидании снятия блокировок другими процессами:

```
unless (flock(FH, LOCK_EX|LOCK_NB)) {
    warn "can't immediately write-lock the file ($!), blocking ...";
    unless (flock(FH, LOCK_EX)) {
        die "can't get write-lock on numfile: $!";
    }
}
```

Если при использовании LOCK_NB вам было отказано в совместной блокировке, следовательно, кто-то другой получил LOCK_EX и обновляет файл. Отказ в монопольной блокировке означает, что другой процесс установил совместную или монопольную блокировку, поэтому пытаться обновлять файл не следует.

Блокировки исчезают с закрытием файла, что может произойти лишь после завершения процесса. Ручное снятие блокировки без закрытия файла — дело рискованное. Это связано с буферизацией. Если между снятием блокировки и очисткой буфера проходит некоторое время, то данные, заменяемые содержимым буфера, могут быть прочитаны другим процессом. Более надежный путь выглядит так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
if ($] < 5.004) {                    # Проверить версию Perl
    my $old_fh = select(FH);
    local $| = 1;                     # Разрешить буферизацию команд
    local $\ = '';                    # Очистить разделитель выходных записей
    print "";                         # Вызвать очистку буфера
    select($old_fh);                  # Восстановить предыдущий манипулятор
}
flock(FH, LOCK_UN);
```

До появления Perl версии **5.004** очистку буфера приходилось выполнять принудительно. Программисты часто забывали об этом, поэтому **в 5.004** снятие блокировки изменилось так, чтобы несохраненные буферы очищались непосредственно перед снятием блокировки.

А вот как увеличитьчисло в файле с применением flock:

```
use Fcntl qw(:DEFAULT :flock);

sysopen(FH, "numfile", O_RDWR|O_CREAT)
                                    or die "can't open numfile: $!";
flock(FH, LOCK_EX)                  or die "can't write-lock numfile: $!";
# Блокировка получена, можно выполнять ввод/вывод
$num = <FH> || 0;                   # НЕ ИСПОЛЬЗУЙТЕ "or" !!
seek(FH, 0, 0)                      or die "can't numfile : $!";
truncate(FH, 0)                     or die "can't truncate numfile: $!";
print FH $num+1, "\n"               or die "can't write numfile: $!";
close(FH)                           or die "can't close numfile: $!";
```

Закрытие файлового манипулятора приводит к очистке буферов и снятию блокировки с файла. Функция truncate описана в главе 8.

С блокировкой файлов дело обстоит сложнее, чем можно подумать — и чем нам хотелось бы. Блокировка имеет условный характер, поэтому если один процесс использует ее, а другой — нет, все идет прахом. Никогда не используйте факт существования файла в качестве признака блокировки, поскольку между проверкой существования и созданием файла может произойти вмешательство извне. Более того, блокировка файлов подразумевает концепцию состояния и потому не соответствует моделям некоторых сетевых файловых систем — например, NFS. Хотя некоторые разработчики утверждают, что fcntl решает эти проблемы, практический опыт говорит об обратном.

В блокировках NFS участвует как сервер, так и клиент. Соответственно, нам не известен общий механизм, гарантирующий надежную блокировку в NFS. Это возможно в том случае, если некоторые операции заведомо имеют атомарный характер в реализации сервера или клиента. Это возможно, если и сервер, и клиент поддерживают flock или fcntl;большинство не поддерживает. На практике вам не удастся написать код, работающий в любой системе.

Не путайте функцию Perl flock с функцией SysV lockf. В отличие от lockf flock блокирует сразу весь файл. Perl не обладает непосредственной поддержкой lockf. Чтобы заблокировать часть файла, необходимо использовать функцию fcntl (см. программу конце главы).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Описание функций flock и fcntl в *perlfunc(1);* документация по стандартным модулям Fcntl и DB_File; рецепт **7.21—7.22.**

7.12. Очистка буфера

Проблема

Операция вывода через файловый манипулятор выполняется не сразу. Из-за этого могут возникнуть проблемы в сценариях CGI на некоторых Web-серверах, враждебных по отношению к программисту. Если Web-сервер получит предупреждение от Perl до того, как увидит (буферизованный) вывод вашего сценария, он передает броузеру малосодержательное сообщение 500 Server Error. Проблемы буферизации возникают при одновременном доступе к файлам со стороны нескольких программ и при взаимодействии с устройствами или сокетами.

Решение

Запретите буферизацию, присвоив истинное значение (обычно 1) переменной $| на уровне файлового манипулятора:

```
$old_fh = select(OUTPUT_HANDLE);
$| = 1:
select($old_fh);
```

Или, если вас не пугают последствия, вообще запретите буферизацию вызовом метода autoflush из модулей **IO:**

```
use IO·:Handle;
OUTPUT_HANOLE->autoflush(1);
```

Комментарий

В большинстве реализаций stdio буферизация определяется типом выходного устройства. Для дисковых файлов применяется блочная буферизация **с** размером буфера, превышающим **2 Кб.** Для каналов (pipes) и сокетов часто применяется буфер размера от **0,5** до **2 Кб.** Последовательные устройства, **к** числу которых относятся терминалы, модемы, мыши и джойстики, обычно буферизуются построчно; stdio передает всю строку лишь при получении неревода строки.

Функция Perl print не поддерживает по-настоящему небуферизованного вывода — физической записи каждого отдельного символа. Вместо этого поддерживается ***командная буферизация,*** при которой физическая запись выполняется после каждой отдельной команды вывода. По сравнению с полным отсутствием буферизации обеспечивается более высокое быстродействие, при этом выходные данные получаются сразу же после вывода.

Для управления буферизацией вывода используется специальная переменная $|. Присваивая ей true, вы темсамым разрешаете командную буферизацию.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

На ввод она не влияет (небуферизованный ввод рассматривается в рецептах 15.6 и 15.8). Если $| присваивается false, будет использоваться стандартная буферизация stdio. Отличия продемонстрированы в примере 7.6.

Пример 7.6. seeme

```
fl!/usr/bin/perl  -w
# seeme - буферизация вывода в stdio
$| = (@ARGV > 0);       # Командная буферизация при наличии аргументов
print "Now you don't see it...";
sleep 2;
print "now you do\n";
```

Если программа запускается без аргументов, STDOUT не использует командную буферизацию. Терминал (консоль, окно, сеанс *telnet* и т. д.) получит вывод лишь после завершения всей строки, поэтому вы ничего не увидите в течение 2 секунд, после чего будет выведена полная строка "Now you don't see it...now you do".

В сомнительном стремлении к компактности кода программисты включают возвращаемое значение select (файловый манипулятор, который *был* выбран в настоящий момент) в другой вызов select:

```
select((select(OUTPUT_HANDLE), $| = 1)[0]);
```

Существует и другой выход. Модули FileHandle и IO содержат метод autoflush. Его вызов с аргументом true или false (по умолчанию используется true) управляет автоматической очисткой буфера для конкретного выходного манипулятора:

```
use FileHandle;

STDERR->autoflush;        # Уже небуферизован в stdio
$filehandle->autoflush(0);
```

Если вас не пугают странности косвенной записи (см. главу 13 "Классы, объекты и связи"), можно написать нечто похожее на обычный английский текст:

```
use IO::Handle;
8 REMOTE_CONN - манипулятор интерактивного сокета,
# a DISK_FILE - манипулятор обычного файла.
autoflush REMOTE_CONN 1;      # Отказаться от буферизации для ясности
autoflush DISK_FILE   O;    # Буферизовать для повышения быстродействия
```

Мы избегаем жутких конструкций select, и программа становится более понятной. К сожалению, при этом увеличивается время компиляции, поскольку включение модуля IO::Handle требует чтения и компиляции тысяч строк кода. Научитесь напрямую работать с $|, этого будет вполне достаточно.

Чтобы выходные данные оказались в нужном месте в нужное время, необходимо позаботиться о своевременной очистке буфера. Это особенно важно для сокетов, каналов и устройств, поскольку они нередко участвуют в интерактивном вводе/выводе, а также из-за того, что вы не сможете полагаться на построчную буферизацию. Рассмотрим программу из примера 7.7.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 7.7. getcomidx

```
ft!/usr/bin/perl
# getpcomidx - получить документ index.html с www.perl.com
use IO::Socket;
$sock = new IO::Socket::INET (PeerAddr => 'www.perl.com',
                              PeerPort => 'http(80)');
die "Couldn't socket: $@" unless $sock;
# Библиотека не поддерживает $!; в ней используется $@

$sock->autoflush(1);

# На Mac \n\n "об язательно* заменяется последовательностью \015\012\015\012.
# Спецификация рекомендует это ,и длядругих систем,
# однако в реализациях рекомендуется поддерживать и "\cJ\cJ".
# Наш опыт показывает, что именно так и получается.
$sock->print("GET /index.html http/1.1\n\n");
$document = join('', $sock->getlines());
print "DOC IS: $document\n";
```

Ни один из рассмотренных нами типов буферизации не позволяет управлять буферизацией ввода. Для этого обращайтесь к рецептам 15.6 и 15.8.

Смотри также

Описание переменной $ | в *perlvar*(1);описание функции select в *perlfunc*(1); документация по стандартным модулям FileHandle и IO::Handle.

7.13. Асинхронное чтение из нескольких манипуляторов

Проблема

Вы хотите узнавать о наличии данных для чтения, вместо того чтобы приостанавливать процесс в ожидании ввода, как это делает <>. Такая возможность пригодится при получении данных от каналов, сокетов, устройств и других программ.

Решение

Если вас не смущают операции с битовыми векторами, представляющими наборы файловых дескрипторов, воспользуйтесь функцией select с нулевым таймаутом:

```
$rin = '';
# Следующая строка повторяется для всех опрашиваемых манипуляторов
vec($rin, fileno(FH1), 1) = 1;
vec($rin, fileno(FH2), 1) = 1;
vec($rin, fileno(FH3), 1) = 1;
```

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

```
$nfound = select($rout=$rin, undef, undef, 0);
if ($nfound) {
  # На одном или нескольких манипуляторах имеются входные данные
  if (vec($r,fileno(FH1),1)) {
      # Сделать что-то с FH1
  }
  if (vec($r,fileno(FH2),1)) {
      # Сделать что-то с FH2
  }
  if (vec($r,fileno(FH3),1)) {
      # Сделать что-то с FH3
  }
}
```

Модуль IO::Select позволяет абстрагироваться от операций с битовыми векторами:

```
use IO::Select;

$select = IO::Select->new();
# Следующая строка повторяется для всех опрашиваемых манипуляторов
$select->add(*FILEHANDLE);
    (@ready
    # Имеются данные на манипуляторах из массива @ready
}
```

Комментарий

Функция select в действительности объединяет сразу две функции. Вызванная с одним аргументом, она изменяет текущий манипулятор вывода по умолчанию (см. рецепт 7.12). При вызове с четырьмя аргументами она сообщает, какие файловые манипуляторы имеют входные данные или готовы получить вывод. В данном рецепте рассматривается только 4-аргументный вариант select.

Первые три аргумента select представляют собой строки, содержащие битовые векторы. Онй определяют состояние файловых дескрипторов, ожидающих ввода, вывода или сообщений об ошибках (например, сведений о выходе данных за пределы диапазона для срочной передачи сокету). Четвертый аргумент определяет тайм-аут — интервал, в течение которого select ожидает изменения состояния. Нулевой тайм-аут означает немедленный опрос. Тайм-аут также равен вещественному числу секунд или undef. В последнем варианте select ждет, пока состояние изменится:

```
$rin = '';
vec($rin, fileno(FILEHANDLE), 1) = 1;
$nfound = select($rin, undef, undef, 0);    # Обычная проверка
if ($nfound) {
    $line = <FILEHANDLE>;
    print "I      $line";
}
```

Однако такое решение не идеально. Если среди передаваемых символов не встретится символ перевода строки, программа переходит в ожидание в <FILEHANDLE>.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Чтобы справиться с этой проблемой, мы последовательно читаем по одному символу и обрабатываем готовую строку при получении "\n". При этом отпадает необходимость в синхронном вызове <FILEHANDLE>. Другое решение (без проверки файлов) описано в рецепте 7.15.

Модуль IO::Select скрывает от вас операции с битовыми векторами. Метод IO: :Select->new() возвращает новый объект, для которого можно вызвать метод add, чтобы дополнить набор новыми файловыми манипуляторами. После включениях всех интересующих вас манипуляторов вызываются функции can_write и can_exception. Функции возвращают список манипуляторов, ожидающих чтения, записи или непрочитанных срочных данных (например, информации о нарушении диапазона TCP).

Вызовы 4-аргументной версии select не должны чередоваться с вызовами каких-либо функций буферизованного вывода, перечисленных во введении о, seek, tell и т. д.). Вместо этого следует использовать — вместе с sysseek, если вы хотите изменить позицию внутри файла для данного манипулятора.

Чтение данных из сокета или канала с немедленным продолжением работы описано в рецепте 17.13. Асинхронному чтению с терминала посвящены рецепты 15.6 и 15.8.

Смотри также

Описание функции select *в perlfunc(1);* документация по стандартному модулю IO::Select; рецепт 7.14.

7.14. Асинхронный ввод/вывод

Проблема

Требуется прочитать или записать данные через файловый манипулятор так, чтобы система не приостанавливала процесс до наступления готовности программы, файла, сокета или устройства надругом конце. Такая задача чаще возникает для специальных, нежели для обычных файлов.

Решение

Откройте файл функцией sysopen с параметром O_NOCBLOCK:

```
use Fcntl;

sysopen(MODEM, "/dev/cua0", O_NONBLOCK|O_RDWR)
    or die "Can't openmodem: $!\n";
```

Если у вас уже есть файловый манипулятор, измените флаги с помощью функции fcntl:

```
use Fcntl;
                                                t
$flags = '';
fcntl(HANDLE, F_GETFL, $flags)
    or die "Couldn't get flags for HANDLE : $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$flags |= O_NONBLOCK;
fcntl(HANDLE, F_SETFL, $flags)
    or die "Couldn't set flags for HANDLE: $!\n";
```

После того как файловый манипулятор будет открыт для асинхронного ввода/вывода, измените флаги с помощью функции fcntl:

```
use POSIX qw(:errno_h);

$rv = syswrite(HANDLE, $buffer, length $buffer);
if (!defined($rv) && $! == EAGAIN) {
    # Ожидание
> elsif ($rv != length $buffer) {
    # Незавершенная запись
} else {
    # Успешная запись
}

                        $buffer, $BUFSIZ);
    or die "sysread: $!";
if (!defined($rv) && $! == EAGAIN) {
    " Ожидание
} else {
    # Успешно прочитано $rv байт из HANDLE
}
```

Комментарий

Константа O_NONBLOCK входит в стандарт **POSIX** и потому поддерживается большинством компьютеров. Мы используем модуль **POSIX** для получения числового значения ошибки EAGAIN.

Смотри также

Описание функций sysopen и fcntl в *perlfunc*(1); документация по стандартному модулю POSIX; страницы руководства *open*(2) и *fcntl*(2); рецепты 7.13 и 7.15.

7.15. Определение количества читаемых байтов

Проблема

Требуется узнать, сколько байтов может быть прочитано через файловый манипулятор функцией read или sysread.

Решение

Воспользуйтесь функцией ioctl в режиме **FIONREAD:**

```
$size = pack("L", 0);
ioctl(FH, $FIONREAD, $size)     or die "Couldn't call ioctl: $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$size = unpack("L", $size);

# Могут быть прочитаны $size байт
```

Комментарий

Функция Perl ioctl предоставляет прямой интерфейс к системной функции ioctl(2). Если вашкомпьютер не поддерживает запросы FIONREAD при вызове ioctl(2), вам неудастся использовать этот рецепт. FIONREAD и другие запросы ioctl(2) соответствуют числовым значениям, которые обычно хранятся в заголовочных файлах C.

Вам может понадобиться утилита Perl *h2ph*, преобразующая заголовочные файлы C в код Perl. FIONREAD в конечном счете определяется как функция в файле *sys/ioctl.ph*:

```
          'sys/ioctl,ph';

$size = pack("L", 0);
ioctl(FH, FIONREAD(), $size)     or die "Couldn't call ioctl: $!\n";
$size = unpack("L", $size);
```

Если утилита *h2ph* не установлена или не подходит вам, найдите нужное место в заголовочном файле с помощью

```
%grep FIONREAD /usr/include/*/*
/usr/include/asm/ioctls.h:#define FIONREAD        0x541B
```

Также можно написать небольшую программу на C в "редакторе настоящего программиста":

```
"include   <sys/ioctl.h>
main() {
    printf("%#08x\n", FIONREAO);
}
^D


0x4004667f
```

Затем жестко закодируйте полученное значение в программе. С переносимостью пускай возится ваш преемник:

```
$FIONREAD = 0x4004667f;          # XXX: зависит от операционной системы

$size = pack("L", 0);
ioctl(FH, $FIONR EAD, $size)     or die "Couldn't call ioctl: $!\n";
$size = unpack("L", $size);
```

FIONREAD требует, чтобы файловый манипулятор был подключен к потоку. Следовательно, сокеты, каналы и терминальные устройства будут работать, а файлы — нет.

Если вам это покажется чем-то вроде системного программирования, взгляните на проблему под другим угЛом. Выполните асинхронное чтение данных из ма-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

нипулятора (см. рецепт 7.14). Если вам удастся что-нибудь прочитать, вы узнаете, столько байтов ожидало чтения, а если не удастся — значит, и читать нечего.

Смотри также

Рецепт 7.14; страница руководства *ioctl*(2) вашей системы; описание функции ioctl в *perlfunc*(1).

7.16. Хранение файловых манипуляторов в переменных

Проблема

Вы собираетесь использовать файловый манипулятор как обычную переменную, чтобы его можно было передать или вернуть из функции, сохранить в структуре данных и т. д.

Решение

Если у вас уже имеется символьный файловый манипулятор (например, STDIN или LOGFILE), воспользуйтесь записью тип-глоба, *FH. Такой подход является самым эффективным.

```
$variable = "FILEHANDLE;        # Сохранить в переменной
subroutine(*FILEHANDLE);        Я или передать функции

sub subroutine {
    my $fh = shift;
    print $fh "Hello, filehandle!\n";
}
```

Если вы хотите работать с анонимным файловым манипулятором, воспользуйтесь функцией ниже) или новыми методами модулей IO::File или IO::Handle, сохраните его в скалярной переменной и используйте так, словно это обычный файловый манипулятор:

```
use FileHandle;                 # Анонимные манипуляторы
$fh = FileHandle->new();

use IO::File;                   # 5.004 и выше
$fh = IO::File->new();
```

Комментарий

Существует немало способов передать файловый манипулятор функции или сохранить его в структуре данных. Самое простое и быстрое решение заключается в применении тип-глоба, *FH. Рассматривайте запись *FH как обозначение типа файлового манипулятора, подобно тому, как представляли молекулы на уроках химии в виде цветных шариков — не совсем точно, зато удобно.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Когда вы начнете понимать недостатки этой модели, она вам уже не понадобится.

Конечно, в простых ситуациях этого вполне достаточно, но что если вам потребовался массив файловых манипуляторов с неизвестными именами? Как показано в главе 11 "Ссылки и записи", построение анонимных массивов, хэшей и даже функций во время выполнения программы оказывается исключительно удобным приемом. Нам хотелось бы иметь аналогичную возможность и для файловых манипуляторов. На помощь приходят модули IO.

Метод new модуля IO::Handle или IO::File генерирует анонимный файловый манипулятор. Его можно передать функции, сохранить в массиве и вообще применять везде, где используются именованные тип-глобы файловых манипуляторов — и не только. Эти модули также могут использоваться в иерархии наследования, поскольку конструктор new возвращает полноценные объекты, для которых могут вызываться методы.

Объекты могут косвенно использоваться в качестве файловых манипуляторов, что избавляет вас от необходимости придумывать для них имена.

Чтобы получить тип-глоб из именованного файлового манипулятора, снабдите его префиксом *:

```
$fh_a = IO::File->new("< /etc/motd")   or die "open /etc/motd: $!";
$fh_b = "STDIN;
some_sub($fh_a,  $fh_b);
```

Существуют и другие способы, но этот проще и удобнее всех остальных. Единственное ограничение — в том, что его нельзя превратить в объект вызовом bless. Bless вызывается для *ссылки* на тип-глоб — именно это и происходит в IO::Handle. Ссылки на тип-глоб, как и сами тип-глобы, можно косвенно использовать в качестве файловых манипуляторов, с приведением посредством bless или без него.

Создание и возврат нового файлового манипулятора из функции происходит следующим образом:

```
                                Создание анонимных файловых манипуляторов
    local *FH;                # Должны быть local, не my
    # now open it if you want to, then...
          *FH;
}

$handle =
```

Функция, получающая файловый манипулятор в качестве аргумента, может либо сохранить его в переменной (желательно лексической) и затем косвенно использовать его:

```
sub accept_fh {
    my $fh = shift;
    print $fh "Sending to          filehandle\n";
}
```

либо локализовать тип-глоб и использовать файловый манипулятор напрямую:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub accept_fh {
    local *FH = shift;
    print  FH "Sending tolocalized filehandle\n";
}
```

Оба варианта работают как с объектами IO::Handle, так и с тип-глобами и настоящими файловыми манипуляторами:

```
accept_fh(*STDOUT);
accept_fh($handle);
```

Perl позволяет использовать **строки, тип-глобы** и ссылки на **тип-глобы** в качестве косвенных файловых манипуляторов, но без передачи тип-глобов или объектов **IO::Handle** можно нарваться на неприятности. Применение строк (`"LOGFILE"` вместо `*LOGFILE`) между пакетами потребует специальных усилий, а функции не могут возвращать ссылки на тип-глобы.

В предыдущих примерах файловый манипулятор перед использованием присваивался скалярной переменной. Дело в том, что во встроенных функциях (`print` или `printf`) или в операторе `o` могут использоваться только простые скалярные переменные, но не выражения или элементы хэшей и массивов. Следующие строки даже не пройдут компиляцию:

```
@fd = (*STDIN, *STDOUT, *STDERR);
print $fd[1] "Type it: ";                          # НЕВЕРНО
$got = <$fd[0]>                                    # НЕВЕРНО
print $fd[2] "What was that: $got";                # НЕВЕРНО
```

В `print` и `printf` это ограничение удается обойти — воспользуйтесь блоком и выражением, в котором находится файловый манипулятор:

```
print  { $fd[1] } "funny stuff\n";
printf { $fd[1] } "Pity the poor %x.\n", 3_735_928_559;
Pity the poor deadbeef.
```

Внутри блока может находиться и более сложный код. Следующий фрагмент отправляет сообщение в один из двух адресов:

```
$ok = -x "/bin/cat";
print { $ok ? $fd[1] : $fd[2] } "cat stat $ok\n";
print { $fd[ 1 + ($ok || 0) ] } "cat stat $ok\n";
```

Подход, при котором `print` и `printf` интерпретируются как вызовы методов объекта, не работает для оператора `o`, поскольку это настоящий оператор, а не вызов функции с аргументом без запятых. Если тип-глобы сохранены в структуре, как это было сделано выше, то для чтения записей можно воспользоваться встроенной функцией работающей аналогично <>:

```
$got =
```

Смотри также

Рецепт 7.1; документация по стандартному модулю **FileHandle**; описание функции `open` в *perlfunc*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.17. Кэширование открытых файловых манипуляторов

Проблема

Требуется одновременно открыть больше файлов, чем позволяет ваша система.

Решение

Воспользуйтесь стандартным модулем FileCache:

```
use FileCache;
cacheout ($path);          # При каждом применении манипулятора
print $path "output";
```

Комментарий

Функция cacheout модуля FileCache позволяет одновременно открывать больше файлов, чем позволяет операционная система. Если воспользоваться ей для открытия существующего файла, который FileCache видит впервые, этот файл без лишних вопросов усекается до нулевой длины. Однако во время фонового открытия и закрытия файлов cacheout следит за открывавшимися ранее файлами и не стирает их, а присоединяет к ним данные. Она не умеет создавать каталоги, поэтому, если попытаться открыть файл */usr/local/dates/merino.ewe* в несуществующем каталоге */usr/local/dates*, из cacheout будет вызвана die.

Функция cacheout() проверяет значение константы NOFILE уровня C из стандартного заголовочного файла *sys/params.h,* чтобы определить, сколько файлов разрешается открывать одновременно. В некоторых системах это значение может быть неверным или вовсе отсутствовать (например, там, где максимальное количество дескрипторов является лимитом ресурса процесса и устанавливается командой limit или ulimit). Если cacheout() не может получить значение NOFILE, достаточно присвоить $FileCache::maxopen значение, на 4 меньше правильного, или подобрать разумное число методом проб и ошибок.

В примере 7.8 файл *xferlog*, создаваемый популярным FTP-сервером wuftpd, разбивается на файлы, имена которых соответствуют именам пользователей. Поля файла *xferlog* разделяются пробелами; имя пользователя хранится в четвертом поле с конца.

Пример 7.8. splitwulog

```
ft!/usr/bin/perl
# splitwulog - разделение журнала wuftpd по именам пользователей
use FileCache;
$outdir  =  '/var/log/ftp/by-user';
while (<>) {
    unless (defined ($user = (split)[-4])) {
       warn "Invalid line: $.\n";
       next;
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 7.8 (продолжение)

```
    }
    $path = "$outdir/$user";
    cacheout $path;
    print $path $_;
}
```

Смотри также

Документация по стандартному модулю FileCache; описание функции `open` в *perlfunc(1)*.

7.18. Одновременный вывод через несколько файловых манипуляторов

Проблема

Одни и те же данные требуется вывести через несколько разных файловых манипуляторов.

Решение

Если **вы** предпочитаете обходиться без создания новых процессов, напишите цикл `foreach` для перебора файловых манипуляторов:

```
        $filehandle (@FILEHANDLES) {
    print $filehandle $stuff_to_print;
}
```

Если новые процессы вас не пугают, откройте файловый манипулятор, связав его с программой **tee:**

```
open(MANY, "| tee file1 file2 file3 > /dev/null")  or die $!
print MANY "data\n"                                 or die $!
close(MANY)                                         or die $!
```

Комментарий

Файловый манипулятор передает выходные данные **лишь** одному файлу или программе. Чтобы дублировать вывод, следует многократно вызвать `print` или связать манипулятор с программой распределения выходных данных (например, tee). В первом варианте проще всего занести файловые манипуляторы в список или массив и организовать их перебор:

```
# 'use strict' пожалуется на эту команду:
for $fh ('FH1', 'FH2', 'FH3')   { print $fh "whatever\n"  }
tt но не возразит против этой:
for $fh (*FH1, *FH2, -FH3)      { print  $fh "whatever\n"}
```

Но если ваша система включает программу tee или **вы** установили Perl-версию tee из рецепта **8.19,** можно открыть канал к **tee** и поручить ей всю работу по копированию файла **в** несколько приемников. Не забывайте, что **tee** обычно ко-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

пирует выходные данные в STDOUT; если лишняя копия данных вам не нужна, перенаправьте стандартный вывод tee в */dev/null*:

```
open (FH, "| tee file1 file2 file3 >/dev/null");
print FH "whatever\n";
```

Вы даже можете перенаправить процессу *tee* свой собственный STDOUT и использовать при выводе обычную функцию print:

```
# Продублировать STDOUT в трех файлах с сохранением исходного STDOUT
open (STDOUT, "| tee file1 file2 file3") or die "Teeing off: $!\n";
print "whatever\n"                       or die "Writing: $!\n",
close(STDOUT)                             or die "Closing: $!\n";
```

Смотри также

Описание функции print в *perlfunc*(1)Аналогичная методика используется в рецептах 8.19 и 13.15.

7.19. Открытие и закрытие числовых файловых дескрипторов

Проблема

Вам известны файловые **дескрипторы**, через которые должен выполняться ввод/вывод, но Perl вместо числовых дескрипторов требует манипуляторы.

Решение

Для открытия файлового дескриптора воспользуйтесь режимами "<&=" и "<&" или методом fdopen модуля IO::Handle:

```
open(FH, "<&=$FDNUM");      # FH открывается для дескриптора
open(FH, "<&$FDNUM");       # FH открывается для копии дескриптора

use IO::Handle;

$fh->fdopen($FDNUM, "r");   # Открыть дескриптор 3 для чтения
```

Чтобы закрыть дескриптор, воспользуйтесь функцией POSIX::close или откройте его описанным выше способом.

Комментарий

Иногда вам известен файловой дескриптор, а Не манипулятор. В системе ввода/вывода Perl вместо дескрипторов используются манипуляторы, поэтому для уже открытого файлового дескриптора придется создать новый манипулятор. Режимы open "<&", ">&" и "+<&" решают эту задачу соответственно для чтения, записи и обновления. Режимы со знаком равенства ("<&=", ">&=" и "+<&=") работают с дескрипторами более экономно, при этом почти всегда делается именно то,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

что нужно. Дело в том, что они используют лишь функцию fdopen уровня C без системной функции dup2.

Если у вас установлена версия Perl 5.004 и выше, воспользуйтесь методом объекта IO::Handle:

```
use IO::Handle;
$fh = IO::Handle->new();

$fh->fdopen(3, "r");            # Открыть fd 3 для чтения
```

Закрытие числовых файловых дескрипторов встречается еще реже. Задача напрямую решается функцией POSIX::close. Если в вашей системе нет библиотеки POSIX, но зато имеется работающая функция syscall (и вашсистемный администратор установил файл *sys/syscall.ph*,созданный транслятором *h2ph*), можно воспользоваться функцией syscall, хотя переносимость в этом случае оставляет желать лучшего. Вам придется добавлять 0 к числам и присоединять "" к строкам, чтобы обеспечить правильность типов C; кроме того, код ошибки равен -1, а не false, как вдругих функциях Perl. Не стоит и говорить, что syscall применяется лишь в крайнем случае.

Ниже показано, как открыть файловые дескрипторы, которые почтовая система MH передает своим дочерним процессам. Дескрипторы идентифицируются по переменной окружения MHCONTEXTFD:

```
$fd = $ENV{MHCONTEXTFD};
open(MHCONTEXT, "<&=$fd")   or die "couldn't fdopen $fd: $!";
# after processing
close(MHCONTEXT)            or die "couldn't close context file: $!";
```

Чтобы закрыть дескриптор по числовому значению, достаточно сначала вызвать для него open.

Смотри также

Документация по стандартным модулям POSIX и IO::Handle; страница руководства *fdopen*(3)вашейсистемы;описаниефункцииopen*perlfunc(1)*.

7.20. Копирование файловых манипуляторов

Проблема

Требуется создать копию файлового манипулятора.

Решение

Синоним файлового манипулятора создается следующей командой:

```
"СИНОНИМ = *ОРИГИНАЛ;
```

Чтобы создать независимую копию файлового дескриптора для существующего манипулятора, воспользуйтесь функцией open в режиме &:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
open(OUTCOPY, ">&STDOUT")    or die "Couldn't dup STDOUT: $!";
open(INCOPY, "<&STDIN" )    or die "Couldn't dup STDIN : $!";
```

Чтобы создать синоним файлового дескриптора для существующего манипулятора, воспользуйтесь функцией open в режиме &=:

```
open(OUTALIAS, ">&=STDOUT") or die "Couldn't alias STDOUT: $!";
open(INALIAS, "<&=STDIN") or die "Couldn't alias STDIN : $!";
open(BYNUMBER, ">&=5")      or die "Couldn't alias file descriptor 5: $!";
```

Комментарий

Если синоним манипулятора создан с помощью тип-глоба, программа по-прежнему работает лишь с одним объектом ввода/вывода Perl. При закрытии манипулятора-синонима закрывается и объект ввода/вывода. Все последующие попытки использования копий этого манипулятора лишь приводят к выдаче сообщений rana"print on closed filehandle". Чередование записи через разные синонимы не вызывает проблем, поскольку при этом не создаются дублирующиеся структуры данных, способные вызвать десинхронизацию.

При копировании дескриптора командой open(КОПИЯ, ">&МАНИПУЛЯТОР") вызывается системная функция dup(2). Вы получаете два независимых дескриптора с общей текущей позицией, блокировкой и флагами, но разными буферами ввода/вывода. Закрытие одного дескриптора не отражается на его копии. Одновременная работа с файлом через оба дескриптора — верный путь к катастрофе. Обычно этот прием используется для сохранения и восстановления STDOUT и STDERR:

```
# Получить копии дескрипторов
open(OLDOUT, ">&STDOUT");
open(OLDERR, ">&STDERR");

# Перенаправить stdout и stderr
open(STDOUT, "> /tmp/program.out")  or die "Can't          stdout: $!";
open(STpERR, ">&STDOUT")            or die "Can't dup stdout: $!";

# Запустить программу
system($joe_random_program);

# Закрыть измененные манипуляторы
close(STDOUT)                       or die "Can't close STDOUT: $!";
close(STDERR)                       or die "Can't close STDERR: $!";

# Восстановить stdout и stderr
open(STDERR, ">&OLDERR")            or die "Can't         stderr: $!";
open(STDOUT, ">&OLDOUT")            or die "Can't         stdout: $!";

8 Для надежности закрыть независимые копии
close(OLDOUT)                       or die "Can't close OLDOUT: $!";
close(OLDERR)                       or die "Can't close OLDERR: $!";
```

Если синоним дескриптора создается командой open(СИНОНИМ, ">&=МАНИПУЛЯТОР"), в действительности вызывается системная функция ввода/вывода fdopen(3).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Вы получаете один файловый дескриптор с двумя буферами, доступ к которым осуществляется через два манипулятора. Закрытие одного манипулятора закрывает дескрипторы синонимов, но не манипуляторы — если вы попытаетесь вызвать print для манипулятора с закрытым синонимом, Perl не выдаст предупреждения "print on closed filehandle", даже если вызов print закончится неудачей. Короче говоря, попытки работать с файлом через оба манипулятора тоже наверняка приведут к катастрофе. Такая методика используется только для открытия файлового дескриптора по известному числовому значению (см. рецепт 7.19).

Смотри также

Страница руководства *dup*(2) вашей системы; описание функции open в *perlfunc(1)*.

7.21. Программа: netlock

При блокировке файлов мы рекомендуем по возможности использовать функцию flock. К сожалению, в некоторых системах блокировка через flock ненадежна. Допустим, функция flock может быть настроена на вариант блокировки без поддержки сети или вы работаете в одной из редких систем, в которой вообще не существует эмуляции flock.

Приведенная ниже программа и модуль содержат базовую реализацию механизма блокировки файлов. В отличиё от обычной функции flock, данный модуль блокирует файлы по именам, а не по дескрипторам.

Следовательно, он может применяться для блокировки каталогов, сокетов и других нестандартных файлов. Более того, вы даже сможете блокировать несуществующие файлы. При этом используется каталог, созданный в иерархии на одном уровне с блокируемым файлом, поэтому вы должны иметь право записи в каталог, содержащий его. Файл в каталоге блокировки содержит сведения о владельце блокировки. Это пригодится в рецепте 7.8, поскольку блокировка сохраняется, несмотря на изменение файла, которому принадлежит данное имя.

Функция nflock вызывается с одним или двумя аргументами. Первый определяет имя блокируемого файла; второй, необязательный — промежуток времени, в течение которого происходит ожидание. Функция возвращает true при успешном предоставлении блокировки и false при истечении времени ожидания. При возникновении различных маловероятных событий (например, при невозможности записи в каталог) инициируется исключение.

Присвойте true переменной $File: :LockDir: :Debug, чтобы модуль выдавал сообщения при неудачном ожидании. Если вы забудете снять блокировку, при выходе из программы модуль снимет ее за вас. Этого не произойдет, если ваша программа получит неперехваченный сигнал.

Вспомогательная программа из примера 7.9 демонстрирует применение модуля File::LockDir.

Пример 7.9. drivelock

```
#!/usr/bin/perl -w
# drivelock - демонстрация модуля File::LockDir
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use strict;
use File::LockDir;
$SIG{INT} = sub { die "outta here\n" };
$File::LockDir::Debug = 1;
my $path = shift                    or die "usage: $0 <path>\n";
unless (nflock($path, 2)) {
    die "couldn't lock $path in 2 seconds\n";
}
sleep 100;
nunflock($path);
```

Исходный текст модуля приведен в примере 7.10. За дополнительными сведениями о построении модулей обращайтесь к главе 12 "Пакеты, библиотеки и модули".

Пример 7.10. File::LockDir

```
package File::LockDir;
# Модуль, обеспечивающий простейшую блокировку
# на уровне имен файлов без применения хитрых системных функций.
# Теоретически информация о каталогах синхронизируется в NFS.
# Стрессовое тестирование не проводилось.

use strict;

use Exporter;
use vars qw(@ISA @EXPORT);
@ISA      = qw(Exporter);
@EXPORT   = qw(nflock nunflock);

use vars qw($Debug $Check);
$Debug  ||= 0; # Может определяться заранее
$Check  ||= 5; # Может определяться заранее

use Cwd;
use Fcntl;
use Sys::Hostname;
use File::Basename;
use File::stat;
use Carp;

my %Locked_Files = ();

# Применение: nflock(ФАЙЛ; ТАЙМАУТ)
sub nflock($;$) {
    my $pathname = shift;
    my $naptime  = shift || 0;
    my $lockname = name2lock($pathname);
    my $whosegot = "$lockname/owner";
    my $start    = time();
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 7.10 (продолжение)

```
    my $missed   = 0;
    local *OWNER;

    # Если блокировка ужеустановлена, вернуться
    if ($Locked_Files{$pathname}) {
            "$pathname        locked";

    }

    if (!-w dirname($pathname)) {
        croak "can't write to              $pathname";
    }

    while (1) {
        last if mkdir($lockname, 0777);
        confess "can't get $lockname: $!" if $missed++ > 10
                        && !-d $lockname;
        if ($Debug) {<
            open(OWNER, "< $whosegot") || last; # exit "if"!
            my $lockee = <OWNER>;
            chomp($lockee);
            printf STDERR "%s $0\[$$]: lock on %s held by %s\n",
                scalar(localtime), $pathname, $lockee;
            close OWNER;
        }}
        sleep $Check;                          /
                if $naptime && time > $start+$naptime;
    }
    sysopen(OWNER, $whosegot, O_WRONLY|O_CREAT|O_EXCL)
                            or croak "can't         whosegot; $!";
    printf OWNER "$0\[$$] on %s since %s\n",
            hostname(), scalar(localtime);
    close(OWNER)
      or croak "close $whosegot: $!";
    $Locked_Files{$pathname}++;

}

# Освободить заблокированный файл
sub nunflock($) {
    my $pathname = shift;
    my $lockname = name2lock($pathname);
    my $whosegot = "$lockname/owner";
    unlink($whosegot);
    carp            lock on $lockname" if $Debug;
    delete $Locked_Files{$pathname};
            rmdir($lockname);
}

# Вспомогательная функция
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub name2lock($) {
    my $pathname = shift;
    my $dir  = dirname($pathname);
    my $file = basename($pathname);
    $dir = getcwd() if $dir eq '.';
    my $lockname = "$dir/$file.LOCKDIR";
            $lockname;
}

fl Ничего не забыли?
END {
    for my $pathname (keys %Locked_Files) {
        my $lockname = name2lock($pathname);
        my $whosegot = "$lockname/owner";
        carp            forgotten $lockname";
        unlink($whosegot);
                rmdir($lockname);
    }
}

1;
```

7.22, Программа:

Функция Perl flock блокирует только целые файлы, но не отдельные их области. Хотя fcntl поддерживает частичную блокировку файлов, из Perl с ней работать трудно — в основном из-за отсутствия модуля XS, который бы обеспечивал переносимую упаковку необходимой структуры данных.

Программа из примера 7.11 реализует fcntl, но лишь для трех конкретных архитектур: SunOS, BSD и Linux. Если вы работаете в другой системе, придется узнать формат структуры flock. Для этого мы просмотрели заголовочный файл C *sys/fcntl.h* и запустили программу c2ph, чтобы получить информацию о выравнивании и типах. Эта программа, распространяемая с Perl, работает только в системах с сильным влиянием Беркли (как те, что перечислены выше). Вы *не обязаны* использовать c2ph, но эта программа несомненно облегчит ваше существование.

Функция struct_flock в программе выполняет упаковку и распаковку структуры, руководствуясь переменной $^O с именем операционной системы. Объявления функции struct_flock не существует, мы просто создаем синоним для версии, относящейся к конкретной архитектуре. Синонимы функций рассматриваются в рецепте 10.14.

Программа lockarea открывает временный файл, уничтожая его текущее содержимое, и записывает в него полный экран (80x23) пробелов. Все строки имеют одинаковую длину.

Затем программа создает производные процессы и предоставляет им возможность одновременного обновления файла. Первый аргумент, N, определяет количество порождаемых процессов (2**N). Следовательно, 1 порождает два процесса, четыре, восемь, шестнадцать

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

и т. д. С увеличением числа потомков возрастает конкуренция за блокировку участков файла.

Каждый процесс выбирает из файла случайную строку, блокирует и обновляет ее. Он записывает в строку свой идентификатор процесса с префиксом — количеством обновлений данной строки:

```
' 4: 18584 was ]ust here
```

Если в момент запроса блокировки строка уже была заблокирована, то после предоставления блокировки в сообщение включается идентификатор предыдущего процесса:

```
29: 24652 ZAPPED 24656
```

Попробуйте запустить программу фоновом режиме и отображайте изменения файла с помощью программы rep из главы 15. Получается видеоигра для системных программистов.

Если работа основной программы прерывается клавишами Ctrl+C или сигналом SIGINT из командной строки, она уничтожает всех своих потомков, посылая сигнал всей группе процессов.

Пример 7.11.

```
#!/usr/bin/perl -w
#          - частичная блокировка с использованием fcntl

use strict;

my $FORKS = shift || 1;
my $SLEEP = shift || 1;

use Fcntl;                                    >
use POSIX qw(:unistd_h :errno_h);

my $COLS = 80;
my $ROWS = 23;

# Когда вы в последний раз видели *этот* режим правильно работающим?
open(FH, "+> /tmp/lkscreen")          or die $!;

select(FH);
$1 = 1;
select STDOUT;

# Очистить экран
for (1 .. $ROWS) {
        print FH " " x $COLS, "\n";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

7.22. Программа:

```
my $progenitor = $$;
fork while $FORKS-- > 0;

print "hello from $$\n";

if ($progenitor == $$) {
        $SIG{INT} = \&genocide;
} else {
        $SIG{INT} = sub { die  "goodbye from $$" };
}

while (1) {
        my $line_num = int rand($ROWS);
        my $line;
        my $n;

        S Перейти к строке
        seek(FH, $n = $line_num * ($COLS+1), SEEK_SET)        or next;

        # Получить блокировку
        my $place = tell(FH);
        my $him;
        next unless defined($him = lock(*FH, $place, $COLS));

        # Прочитать строку
                $line, $COLS) == $COLS                        or next;
        my $count = ($line =^ /(\d+)/) ? $1 : 0;
        $count++;

        # Обновить строку
        seek(FH, $place, 0)                                   or die $!;
        my $update = sprintf($him
                            ? "%6d: %d ZAPPED %d"
                              "%6d:
                        $count, $$, $him);
        my $start = int(rand($COLS - length($update)));
        die "XXX" if $start + length($update) > $COLS;
        printf FH "%*.*s\n", -$COLS, $COLS, " " x $start . $update;

        # Снять блокировку и сделать паузу
        unlock(*FH, $place, $COLS);
        sleep $SLEEP if $SLEEP;
}
die "NOT REACHED";                # На всякий случай

# lock($handle, $offset, $timeout) - get an fcntl lock
sub lock {
        my ($fh, $start, $till) = @_;
        ##print "$$: Locking $start. $till\n";
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 7.11 (продолжение)

```
        my $lock = struct_flock(F_WRLCK, SEEK_SET, $start, $till, 0);
        my $blocker = 0;
        unless (fcntl($fh, F_SETLK, $lock)) {
            die "F_SETLK $$ @_: $!" unless $! == EAGAIN || $! == EDEADLK;
            fcntl($fh, F_GETLK, $lock)          or die "F_GETLK $$ @_: $!'
            $blocker = (struct_flock($lock))[-1];
            ##print "lock $$ @_: waiting for $blocker\n";
            $lock = struct_flock(F_WRLCK, SEEK_SET, $start, $till, 0);
            unless (fcntl($fh, F_SETLKW, $lock)) {
                warn "F_SETLKW $$ @_: $!\n";
                            undef
            }
        }
               $blocker;
}

# unlock($handle, $offset, $timeout) - снять блокировку fcntl
sub unlock {
        my ($fh, $start, $till) = @_;
        ##print "$$: Unlocking $start, $till\n";
        my $lock = struct_flock(F_UNLCK, SEEK_SET, $start, $till, 0);
        fcntl($fh, F SETLK, $lock) or die "F_UNLCK $$ @ : $!";
>

# Структуры flock для разных ОС

Я Структура flock для Linux
#   short l_type;
#   short l_whence;
#   off_t l_start;
#   off_t l_len;
#   pid_t l_pid;
BEGIN {
        # По данным c2ph: typedef='s2 12 i', sizeof=16
        my $FLOCK_STRUCT = 's s 1 1 i';

        sub linux_flock {
            if (wantarray) {
                my ($type, $whence, $start, $len, $pid) =
                    unpack($FLOCK_STRUCT, $_[0]);
                        ($type, $whence, $start, $len, $pid);
            } else {
                my ($type, $whence, $start, $len, $pid) = @_
                        pack($FLOCK_STRUCT,
                         $type, $whence, $start, $len, $pid);
            }
        }

>
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Структура flock для  SunOS
#   short   l_type;          /* F_RDLCK, F_WRLCK или F_UNLCK */
#   short   l_whence;        /* Флаг выбора начального смещения */
Я   long    l_start;         /* Относительное смещение в байтах */
#   long    l_len;   ,       /* Длина в байтах;
                                0 - блокировка до EOF */
#   short   l_pid;           /* Возвращается F_GETLK "/
#   short   l_xxx;           /* Зарезервировано на будущее */
BEGIN {
        # По данным c2ph: typedef='s2 12 s2', sizeof=16
        my $FLOCK_STRUCT = 's s 1 1 s s';

        sub sunos_flock {
            if (wantarray) {
                my ($type, $whence, $start, $len, $pid, $xxx) =
                    unpack($FLOCK_STRUCT, $_[0]);
                        ($type, $whence, $start, $len, $pid);
            } else <
                my ($type, $whence, $start, $len, $pid) = @_;
                        pack($FLOCK_STRUCT,
                         $type, $whence, $start, $len, $pid, 0);
            }
        }
}

# Структура flock для
#   off_t   l_start;         /* Начальное смещение */
#   off_t   l_len;           /* len = 0 означает блокировку до конца файла */
#   pid_t   l_pid;           /* Владелец блокировки */
#   short   l_type;          /* Тип блокировки: чтение/запись и т. д. "/
#   short   l_whence;        /* Тип l_start */
BEGIN {
        # По данным c2ph: typedef="q2 i s2", size=24
        my $FLOCK_STRUCT = 'll 11 i s s';

        sub bsd_flock {
            if (wantarray) {
                my ($xxstart, $start, $xxlen, $len, $pid, $type, $whence) =
                    unpack($FLOCK_STRUCT, $_[0]);
                        ($type, $whence, $start, $len, $pid);
            } else {
                my ($type, $whence, $start, $len, $pid) = @_;
                my ($xxstart, $xxlen) = (0,0);
                        pack($FLOCK_STRUCT,
                    $xxstart,. $start, $xxlen, $len, $pid, $type, $whence);
            }
        }
}
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 7.11 (продолжение)

```
# Синоним структуры fcntl на стадии компиляции
BEGIN {
        for ($^O) <
            *struct_flock =                 do
                                  /bsd/     &&   \&bsd_flock
                                            ||
                                /linux/     &&    \&linux_flock
                                            ||
                              /sunos/       &&       \&sunos_flock
                                            ||
                        die "unknown operating system $^O, bailing out";
            };
        }
}

# Установить обработчик сигнала для потомков
BEGIN {
        my $called = 0;

        sub genocide {
            exit if $called++;
            print "$$: Time to die, kiddies.\n"  if $$ == $progenitor;
            my $job = getpgrp();
            $SIG{INT} = 'IGNORE';
            kill -2, $job if $job;  # killpg(SIGINT, ]ob)
            1 while wait > 0;
            print "$$: My turn\n" if $$ == $progenitor;
            exit;
        }
}

END { &genocide }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8 Содержимое файлов

Из всех решений UNIX самым гениальным был выбор одного символа для перевода строки.

Майк О'Делл, лишь с долей шутки

Введение

До революции UNIX всевозможные источники и приемники данных не имели ничего общего. Чтобы две программы пообщались друг с другом, приходилось идти на невероятные ухищрения и отправлять в мусор целые горы перфокарт. При виде этой компьютерной Вавилонской башни порой хотелось бросить программирование и подыскать себе менее болезненное хобби — например, податься в секту флаггелантов.

В наши дни этот жестокий и нестандартный стиль программирования в основном ушел в прошлое. Современные операционные системы всячески стараются создать иллюзию, будто устройства ввода/вывода, сетевые подключения, управляющие данные процессов, другие программы, системные консоли и даже терминалы пользователей представляют собой абстрактные потоки байтов, именуемые *файлами*. Теперь можно легко написать программу, которая нисколько не заботится о том, откуда взялись ее входные данные и куда отправятся результаты.

Поскольку чтение и запись данных осуществляется через простые байтовые потоки, любая программа может общаться с любой другой программой. Трудно переоценить всю элегантность и мощь такого подхода. Пользователи перестают зависеть от сборников магических заклинаний JCL (или COM) и могут собирать собственные нестандартные инструменты, используя простейшее перенаправление ввода/вывода и конвейерную обработку.

Интерпретация файлов как неструктурированных байтовых потоков однозначно определяет круг возможных операций. Вы можете читать и записывать последовательные блоки данных фиксированного размера в любом месте файла, увеличивая его размер при достижении конца. Чтение/запись блоков переменной длины (например, строк, абзацев и слов) реализуется в Perl на базе стандартной библиотеки ввода/вывода C.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Что нельзя сделать с неструктурированным файлом? Поскольку вставка и удаление байтов возможны лишь в конце файла, вы не сможете вставить или удалить записи, а также изменить их длину. Исключение составляет последняя запись, которая удаляется простым усечением файла до конца предыдущей записи. В остальных случаях приходится использовать временный файл или копию файла в памяти. Если вам приходится часто заниматься этим, вместо обычных файлов лучше подойдет база данных (см. главу 14 "Базы данных»).

Самый распространенный тип файлов — текстовые файлы, а самый распространенный тип операций с ними — построчное чтение и запись. Для чтения строк используется оператор <> (или его внутренняя реализация, си — функция print. Эти способы также могут применяться для чтения или записи любых блоков с конкретным разделителем. Строка представляет собой запись с разделителем "\n".

При достижении конца файла оператор <> возвращает undef или ошибку, поэтому его следует использовать в цикле следующего вида:

```
while (defined ($line = <DATAFILE>)) {
    chomp $line;
    $size = length $line;
    print "$size\n";                # Вывести длину строки
}
```

Поскольку эта операция встречается довольно часто, в Perl для нее предусмотрена сокращенная запись, при которой строки читаются в $_ вместо $line. Переменная $_ используется по умолчанию и в других строковых операциях и вообще куда удобнее, чем может показаться на первый взгляд:

```
while (<DATAFILE>) {
    chomp;
    print length, "\n";             # Вывести длину строки
}
```

В скалярном контексте оператор <> читает следующую строку. В списковом контексте он читает оставшиеся строки:

```
@lines = <DATAFILE>;
```

При чтении очередной записи через файловый манипулятор о увеличивает значение специальной переменной $. (текущий номер входной записи). Переменная сбрасывается лишь при явном вызове close и сохраняет значение при повторном открытии уже открытого манипулятора.

Заслуживает внимания и другая специальная переменная — $/, разделитель входных записей. По умолчанию ей присваивается "\n", маркер конца строки. Ей можно присвоить любое желаемое значение — например, "\0" для чтения записей, разделяемых нуль-байтами. Для чтения целых абзацев следует присвоить $/ пустую строку, "". Это похоже на присваивание "\n\n", поскольку для разделения записей используются пустые строки, однако "" интерпретирует две и более смежных пустых строки как один разделитель, а "\n\n" в таких случаях возвращает пустые записи. Присвойте $/ неопределенное значение, чтобы прочитать остаток файла как одну скалярную величину:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
undef $/;
$whole_file = <FILE>;                    S Режим поглощения
```

Запуск Perl с флагом **-0** позволяет задать $/ из командной строки:

```
% perl -040 -e '$word = <>; print "First word is $word\n";'
```

Цифры после **-0** определяют восьмеричное значение отдельного символа, который будет присвоен $/. Если задать недопустимое значение (например, ***-0777),*** **Perl** присваивает $/ неопределенное значение `undef`. Если задать *-00,* $/ присваивается "". Ограничение в один восьмеричный символ означает, что вы не сможете присвоить $/ многобайтовую строку — например, "%%\n" для чтения файлов программы fortune. Вместо этого следует воспользоваться блоком BEGIN:

```
% perl -ne 'BEGIN { $/="%%\n" } chomp; print if /Unix/i' fortune.dat
```

Запись строк и других данных выполняется функцией print. Она записывает свои аргументы в порядке указания и по умолчанию не добавляет к ним разделители строк или записей:

```
print HANDLE "One", "two",          # "Onetwothree"
print "Baa baa black sheep.\n";     tt Передается выходному манипулятору
                                    # по умолчанию
```

Между манипулятором и выводимыми данными не должно быть запятых. **Если поставить запятую, Perl выдает сообщение об ошибке** "No comma allowed after filehandle". По умолчанию для вывода используется манипулятор STDOUT. Для выбора другого манипулятора применяется функция select (см. главу 7 "Доступ к **файлам»).**

. Во всех системах строки разделяются виртуальным разделителем "\n", который называется ***переводом строки*** (newline). Не существует такого понятия, как символ перевода строки. Это всего лишь иллюзия, которая по общему сговору поддерживается операционной системой, драйверами устройств, библиотеками C **и** Perl, Иногда это приводит к изменению количества символов **в** прочитанных **или** записываемых строках. Подробности заговора изложены **в** рецепте **8.11.**

Записи фиксированной длины читаются функцией Функция получает три аргумента: файловый манипулятор, скалярную переменную **и** количество читаемых байт. Возвращается количество прочитанных байт, а **в** случае ошибки — undef. Для записи используется функция print:

```
(HANDLE, $buffer, 4096)
    "Couldn't          HANDLE : $!\n";
tt $rv - количество прочитанных байт,
# $buffer содержит прочитанные данные
```

Функция truncate изменяет длину файла, который задается с помощью манипулятора или по имени. Функция **возвращает** true, если усечение прошло успешно, и false **в** противном случае:

```
truncate(HANDLE, $length)
    or die "Couldn't truncate: $!\n";
truncate("/tmp/$$.pid", $length)
    or die "Couldn't truncate: $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Для каждого файлового манипулятора отслеживается текущая позиция в файле. Операции чтения/записи выполняются именно в этой позиции, если при открытии не был указан флаг O_APPEND (см. рецепт 7.1). Чтобы узнать текущую позицию файлового манипулятора, воспользуйтесь функцией tell, а чтобы задать ее — функцией seek. Поскольку стандартная библиотека ввода/вывода стремится сохранить иллюзию того, что "\n" является разделителем строк, вы не сможете обеспечить переносимый вызов seek для смещений, вычисляемых посредством подсчета символов. Вместо этого seek следует вызывать только для смещений, возвращаемых tell:

```
$pos = tell(DATAFILE);
print "I'm $pos bytes from the start of DATAFILE.\n";
```

Функция seek получает три аргумента: файловый манипулятор, новое смещение (в байтах) и число, определяющее интерпретацию смещения. Если оно равно 0, смещение отсчитывается от начала файла (в соответствии со значениями, возвращаемыми tell); 1 — от текущей позиции (положительное число означает прямое перемещение в файле, а отрицательное — обратное); 2 — от конца файла.

```
seek(LOGFILE, 0, 2)          or die "Couldn't seek to the end: $!\n";
seek(DATAFILE, $pos, 0)      or die "Couldn't seek to $pos: $!\n";
seek(OUT, -20, 1)            or die "Couldn't seek back 20 bytes: $!\n";
```

Все сказанное выше относится к буферизованному вводу/выводу. Другими словами, операции <>, print, seek и tell используют буферы для повышения скорости. В Perl также предусмотрены небуферизованные операции ввода/вывода: sysopen, sysread, syswrite, sysseek и close. Буферизация, sysopen и close рассматриваются в главе 7.

Функции отличаются от своих аналогов, о и print. Они получают одинаковые аргументы — файловый манипулятор; скалярную переменную, с которой выполняется чтение или запись; и количество читаемых или записываемых байт. Кроме того, они могут получать необязательный четвертый аргумент — смещение внутри скалярной переменной:

```
$written = syswrite(DATAFILE, $mystring, length($mystring));
die "syswrite failed: $!\n" unless $written == length($mystring);
                          $block, 256, 5);
warn                                  if
```

Функция syswrite посылает содержимое $mystring в DATAFILE. При вызове из INFILE читаются 256 символов, сохраняемых с шестого символа в $block, при этом первые пять символов остаются без изменений. И syswrite возвращают фактическое количество переданных байт; оно может не совпадать с тем, которое пытались передать вы. Например, файл содержал меньше данных, чем вырассчитывали, и чтение получилось укороченным. Может быть, произошло переполнение носителя, на котором находился файл. А может быть, процесс был прерван на середине записи. Stdio заботится о завершении записи в случае прерывания, но при вызовах syswrite этим придется заняться вам. Пример приведен в рецепте 9.3.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Функция sysseek является небуферизованной заменой для seek и tell. Она получает те же аргументы, что и seek, но возвращает новую позицию при успешном вызове или undef в случае ошибки. Текущая позиция внутри файла определяется следующим образом:

```
$pos = sysseek(HANDLE, 0, 1);          # Не изменять позицию
die "Couldn't sysseek: $!\n" unless defined $pos;
```

Мы описали базовые операции с файлами, которые находятся в вашем распоряжении. Искусство программирования как раз и заключается в применении простейших операций для решения сложных проблем — например, определения количества строк в файле, перестановки строк, случайного выбора строки из файла, построения индексов и т. д.

8.1. Чтение строк с символами продолжения

Проблема

Имеется файл с длинными строками, которые делятся на две и более строки. Символ \ означает, что данная строка продолжается на следующей. Вы хотите объединить разделенные строки. Подобное разделение длинных строк на короткие встречается в make-файлах, сценариях командного интерпретатора, конфигурационных файлах и многих языках сценариев.

Решение

Последовательно объединяйте прочитанные строки, пока не встретится строка без символа продолжения:

```
while (defined($line = <FH>) ) {
    chomp $line;
    if ($line =- s/\\$//) {
        $line .= <FH>;
        redo unless eof(FH);
    }
    # Обработать полную запись в $line
}
```

Комментарий

Рассмотрим пример входного файла:

```
DISTFILES = $(DIST_COMMON) $(SOURCES) $(HEADERS) \
        $(TEXINFOS) $(INFOS) $(MANS) $(DATA)
DEP_DISTFILES = $(DIST_COMMON) $(SOURCES) $(HEADERS) \
        $(TEXINFOS) $(INFO_DEPS) $(MANS) $(DATA) \
        $(EXTRA_DIST)
```

Вы хотите обработать текст, игнорируя внутренние разрывы строк. В приведенном примере первая запись занимает две строки, вторая — три строки и т. д.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Алгоритм работает следующим образом. **Цикл** while читает строки, которые могут быть, а могут и не быть полными записями, — они могут заканчиваться символом \ (и переводом строки). Оператор подстановки s/// пытается удалить \ в конце строки. Если подстановка заканчивается неудачей, значит, мы нашли строку без \. В противном случае мы читаем следующую запись, приписываем ее к накапливаемой переменной $line и возвращаемся к началу цикла while с помощью

Затем выполняется команда chomp.

У файлов такого формата имеется одна распространенная проблема — невидимые пробелы между \ и концом строки. Менее строгий вариант подстановки выглядит так:

```
if ($line =- s/\\\s*$//) {
    # Как и прежде
}
```

К сожалению, даже если ваша программа прощает мелкие погрешности, существуют и другие, которые этого не делают. Будьте снисходительны к входным данным и строги — к выходным.

Смотри также

Описание функции chomp в *perlfunc*(1); описание ключевого слова в разделе "Loop Control" *perlsyn*(1).

8.2. Подсчет строк (абзацев, записей) в файле

Проблема

Требуется подсчитать количество строк в файле.

Решение

Во многих системах существует программа wc, подсчитывающая строки в файле:

```
$count = 'wc -1 < $file';
die "wc failed: $?" if $?;
chomp($count);
```

Кроме того, можно открыть файл и последовательно читать строки до конца, увеличивая значение счетчика:

```
open(FILE, "< $file") or die "can't орел $file: $!";
$count++ while <FILE>;
# $count содержит число прочитанных строк
```

Самое быстрое решение предполагает, что строки ***действительно*** завершаются "\n":

```
$count += tr/\n/\n/ while                    **
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Хотя размер файла в байтах можно определить с помощью -s $file, обычно полученная цифра никак не связана с количеством строк. Оператор -s рассматривается в главе 9 "Каталоги".

Если вы не хотите или не можете перепоручить черную работу другой программе, имитируйте работу wc — самостоятельно откройте и прочитайте файл:

```
open(FILE, "< $file") or die "can't open $file: $!";
$count++ while <FILE>;
# $count содержит число прочитанных строк
```

Другой вариант выглядит так:

```
open(FILE, "< $file") or die "can't open $file: $!";
for ($count=0; <FILE>; $count++) { }
```

Если вы не читаете из других файлов, можно обойтись без переменной $count. Специальная переменная $. содержит количество прочитанных строк с момента последнего явного вызова close для файлового манипулятора:

```
1 while <FILE>;
$count = $.;
```

В этом варианте все записи файла последовательно читаются без использования временных переменных.

Чтобы подсчитать абзацы, присвойте перед чтением глобальному разделителю входных записей $/ пустую строку ("""), и тогда оператор <> будет считывать не строки, а целые абзацы:

```
$/ = '';              # Включить режим чтения абзацев
open(FILE, $file) or die "can't open $file: $!";
1 while <FILE>;
$para_count = $.;
```

Смотри также

Описание специальной переменной $/ в *perlvar*(1); введение главы 9; страница руководства *wc*(1).

8.3. Обработка каждого слова в файле

Проблема

Требуется выполнить некоторую операцию с каждым словом файла, по аналогии с функцией

Решение

Разделите каждую строку по пропускам с помощью функций split:

```
while (o) {
    for $chunk (split) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        # Сделать что-то с $chunk
    }
}
```

Или воспользуйтесь оператором m//g для последовательного извлечения фрагментов строки:

```
while (<>) {
    while ( /(\w[\w'-]*)/g ) {
        # Сделать что-то с $1
    }
}
```

Комментарий

Сначала необходимо решить, что же подразумевается под "словом". Иногда это любые последовательности символов, кроме пропусков; иногда — идентификаторы программ, а иногда — слова английского языка. От определения зависит и используемое регулярное выражение.

Два варианта решения, приведенные выше, работают по-разному. В первом варианте шаблон определяет, что *не является* словом. Во втором варианте все наоборот — шаблон решает, что им *является*.

На основе этой методики нетрудна подсчитать относительные частоты всех слов в файле. Количество экземпляров каждого слова сохраняется в хэше:

```
# Подсчет экземпляров слов в файле
%seen = ();
while (<>) {
    while ( /(\w['\w-]*)/g ) {
        $seen{lc $1}++;
    }
}

# Отсортировать выходной хэш по убыванию значений
foreach $word ( sort { $seen{$b} <=> $seen{$a} } keys %seen) {
    printf "%5d %s\n", $seen{$word>, $word;
}
```

Чтобы программа подсчитывала количество строк вместо слов, уберите второй цикл while и замените его на $seen{lc $_}++:

```
# Подсчет экземпляров строк в файле
%seen = ();
while (o) {
    $seen{lc $_}++;
}
        $line ( sort { $seen{$b} <=> $seen{$a} } keys %seen ) {
    printf "%5d %s", $seen{$line}, $line;
}
```

Порой слова могут выглядеть довольно странно — например, «M.I.T», "Micro$oft», "o'clock", "49ers", «street-wise», "and/or", "&", "c/o", "St.", «Tschüß» или

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

«Niño». Помните об этом при выборе шаблона. В двух последних примерах вам придется включить в программу директиву use locale и использовать метасимвол \w в текущем локальном контексте.

Смотри также

Описание функции split в *perlfunc(1);* рецепты 6.3;6.23.

8.4. Чтение файла по строкам или абзацам в обратном направлении

Проблема

Требуется обработать каждую строку или абзац файла в обратном направлении.

Решение

Прочитайте все строки в массив и организуйте обработку элементов массива от конца к началу:

```
@lines = <FILE>;
while ($line = pop @lines) {
    # Сделать что-то с $line
}
```

Или занесите строки в массив в обратном порядке:

```
@lines          <FILE>;
        $line (@lines) {
    # Сделать что-то с $line
}
```

Комментарий

Ограничения, связанные с доступом к файлам (см. введение), не позволяют последовательно читать строки с конца файла. Приходится читать строки в память и обрабатывать их в обратном порядке. Конечно, расходы памяти при этом будут по крайней мере не меньше размера файла.

В первом варианте массив строк перебирается в обратном порядке. Такая обработка является *деструктивной,* поскольку при каждой итерации из массива выталкивается последний элемент. Впрочем, то же самое можно сделать и недеструктивно:

```
for ($i = $#lines; $i != -1; $i--) {
    $line = $lines[$i];
}
```

Во втором варианте генерируется массив строк, изначально расположенных в обратном порядке. Его тоже можно обработать недеструктивно. Мы получаем массив с обратным порядком строк, поскольку присваивание @lines обеспечива-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ет вызов списковом контексте, что, в свою очередь, обеспечивает списковый контекст для оператора <FILE>. В списковом контексте <> возвращает список всех строк файла.

Показанные решения легко распространяются на чтение абзацев, достаточно изменить значение $/:

```
# Внешний блок обеспечивает существование временной локальной копии $/
{
    local $/ = '';
    @paragraphs          <FILE>;.
}

        $paragraph  (@paragraphs)  {
    # Сделать что-то
}
```

Смотри также

Описание функции *в perlfunc(1);* описание специальной переменной $/ *в perlvar(1);* рецепты **4.10; 1.6.**

8.5. Чтение из дополняемого файла

Проблема

Требуется читать данные из непрерывно растущего файла, однако при достижении конца файла (текущего) следующие попытки чтения завершаются неудачей.

Решение

Читайте данные, пока не будет достигнут конец файла. Сделайте паузу, сбросьте флаг EOF и прочитайте новую порцию данных. Повторяйте, пока процесс не прервется. Флаг EOF сбрасывается либ® функцией seek:

```
for (;;) {
    while (<FH>) { .... >
    sleep $SOMETIME;
    seek(FH, 0, 1);
}
```

либо методом модуля IO::Handle:

```
use IO::Seekable;

for (;;) {
    while (<FH>) { .... >
    sleep $SOMETIME;
    FH->clearerr();
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

При достижении конца файла во время чтения устанавливается внутренний флаг, который препятствует дальнейшему чтению. Для сброса этого флага проще всего воспользоваться методом если он поддерживается (присутствует в модулях IO::Handle и FileHandle). Кроме того, можно вызвать метод POSIX:

```
$naptime = 1;

use IO::Handle;
open (LOGFILE, "/tmp/logfile") or die "can't open /tmp/logfile: $!";
for (;;) {
    while (<LO6FILE>) { print }      в Или другая операция
    sleep $naptime;
    LOGFILE->clearerr();            # Сбросить флаг ошибки ввода/вывода
›
```

Если простейший вариант в вашей системе не **работает,** воспользуйтесь функцией seek. Приведенный выше фрагмент с seek пытается переместиться на 0 байт от текущей **позиции,** что почти всегда завершается успехом. Текущая позиция при этом не изменяется, но зато для манипулятора сбрасывается признак конца файла, благодаря чему при следующем вызове <LOGFILE> будут прочитаны новые данные.

Если и этот вариант не работает (например, из-за того, что он полагается на так называемую "стандартную" реализацию ввода/вывода библиотек **C),** попробуйте следующий фрагмент — он явно запоминает старую позицию в файле и напрямую возвращается к ней:

```
for (;;) {
    for ($curpos = tell(LOGFILE); <LOGFILE>; $curpos = tell(LOGFILE)) {
        # Обработать $_
    }
    sleep $naptime;                \
    seek(LOGFILE, $curpos, 0);.  # Вернуться к прежней позиции
}
```

Некоторые файловые системы позволяют удалить файл во время чтения из него. Вероятно, в таких случаях нет смысла продолжать работу с файлом. Чтобы программа в подобных ситуациях завершалась, вызовите stat для манипулятора и убедитесь в том, что количество ссылок на него (третье поле возвращаемого списка) не стало равным нулю:

```
exit if (stat(LOGFILE))[3] == 0
```

Модуль File::stat позволяет записать то же самое в более понятном виде:

```
use File::stat;
exit if stat(*LOGFILE)->nlink == 0;
```

Смотри также

Описание функции seek *в perlfunc(1);* документация по стандартным модулям POSIX и **IO::Seekable;** страницы руководства fru7(l) и ***stdio*(3).**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8.6. Выбор случайной строки из файла

Проблема

Требуется прочитать из файла случайную строку.

Решение

Воспользуйтесь функцией rand и переменной $. (текущим номером строки):

```
srand;
rand($.) < 1 && ($line= $_) while <>;
# $line - случайно выбранная строка
```

Комментарий

Перед вами — изящный и красивый пример неочевидного решения. Мы читаем все строки файла, но не сохраняем их в памяти. Это особенно важно для больших файлов. Вероятность выбора каждой строки равна 1/N (где N — количество прочитанных строк).

Следующий фрагмент заменяет хорошо известную программу fortune:

```
$/ = "%%\n";
$data = `/usr/share/games/fortunes`;
srand;
rand($.) < 1 && ($adage = $_) while<>;
print $adage;
```

Если вам известны смещения строк (например, при наличии индекса) и их общее количество, можно выбрать случайную строку и перейти непосредственно к ее смещению в файле. Впрочем, индекс доступен далеко не всегда.

Приведем более формальное пояснение работы данного алгоритма. Функция rand ($.) выбирает случайное число от 0 до текущего номера строки. Строка с номером N сохраняется в возвращаемой переменной с вероятностью 1/N. Таким образом, первая строка сохраняется с вероятностью 100 %, вторая — с вероятностью 50 %, третья — 33 % и т. д. Вопрос лишь в том, насколько это честно для любого положительного целого N.

Начнем с конкретных примеров, а затем перейдем к абстрактным.

Разумеется, для файла из одной строки (N=1) все предельно честно: первая строка сохраняется всегда, поскольку 1/1 = 100%. Для файла из двух строк N = 2. Первая строка сохраняется всегда; когда вы достигаете второй строки, она с вероятностью 50 % заменяет первую. Следовательно, обе строки выбираются с одинаковой вероятностью, и для N = 2 алгоритм тоже работает корректно. Для файла из трех строк N = 3. Третья строка сохраняется с вероятностью 1/3 (33%). Вероятность выбора одной из двух первых строк равна 2/3 (66%). Но как показано выше, две строки имеют одинаковую вероятность выбора (50 %). Пятьдесят процентов от 2/3 равны 1/3. Таким образом, каждая из трех строк файла выбирается с вероятностью 1/3.

В общем случае для файла из N+1 строк последняя строка выбирается с вероятностью 1/(N+1), а одна из предыдущих строк — N/(N+1). Деление N/(N+1) на

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

N дает вероятность 1/(N+1) для каждой из N первых строк и те же 1/(N+1) для строки с номером N+1. Следовательно, алгоритм корректно работает для любого положительного целого N.

Нам удалось случайным образом выбрать из файла строку со скоростью, пропорциональной количеству строк в файле. При этом максимальный объем используемой памяти даже в худшем случае равен размеру самой длинной строки.

Смотри также

Описание специальной переменной $. в *perlvar*(1); рецепты 2.7—2.8.

8.7. Случайная перестановка строк

Проблема

Требуется скопировать файл и случайным образом переставить строки копии.

Решение

Прочитайте все строки в массив, перетасуйте элементы массива (см. рецепт 4.17) и запишите полученную перестановку:

```
# Используется функция shuffle из главы 4
while (<INPUT>) {
    push(@lines, $_);
}
            shuffle(@lines);
        (@reordered)
    print OUTPUT $_;
}
```

Комментарий

Самое простое решение — прочитать все строки файла и переставить их в памяти. Смещения строк в файле неизвестны, поэтому нельзя перетасовать список с номерами строк и затем извлечь строки в порядке их появления в файле. Впрочем, даже при известных смещениях такое решение, вероятно, будет работать медленнее, поскольку придется многократно перемещаться по файлу функцией seek вместо простого последовательного чтения.

Смотри также

Рецепты 2.7-2.8; 4.17.

8.8. Чтение строки с конкретным номером

Проблема

Требуется извлечь из файла строку с известным номером.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Простейший выход — читать строки до обнаружения нужной:

```
# Выборка строки с номером $DESIRED_LINE_NUMBER
$. = 0;
do { $LINE = <HANDLE> } until $. == $DESIRED_LINE_NUMBER || eof;
```

Если подобная операция должна выполняться многократно, а файл занимает не слишком много места в памяти, прочитайте его в массив:

```
@lines = <HANDLE>;
$LINE = $lines[$DESIRED_LINE_NUMBER];
```

Если вы собираетесь многократно извлекать строки по номеру, а файл не помещается в памяти, постройте индекс смещений для отдельных строк и переходите к началу строки функцией seek:

```
" Применение : build_index(*МАНИПУЛЯТОР_ДАННЫХ, *МАНИПУЛЯТОР_ИНДЕКСА)
sub build_index {
    my $data_file  = shift;
    my $index_file = shift;
    my $offset     = 0;

    while (<$data_file>) {
        print $index_file pack("N", $offset);
        $offset = tell($data_file);
    }
>

# Применение : line_with_index(*МАНИПУЛЯТОР_ДАННЫХ, *МАНИПУЛЯТОР_ИНДЕКСА,
                                $НОМЕР_СТРОКИ)
в Возвращает строку или undef, если НОМЕР_СТРОКИ выходит за пределы файла
sub line_with_index {
    my $data_file   = shift;
    my $index_file  = shift;
    my $line_number = shift;

    my $size;              # Размер элемента индекса
    my $i_offset;           # Смещение элемента в индексе
    my $entry;              # Элемент индекса
    my $d_offset;           # Смещение в файле данных

    $size = length(pack("N", 0));
    $i_offset = $size * ($line_number-1);
    seek($index_file, $i_offset, 0) or
                      $entry, $size);
    $d_offset = unpack("N", $entry);
    seek($data_file, $d_offset, 0);
          scalar(<$data_file>);
}

# Применение:
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
open(FILE, "< $file")          or die "Can't open $file for          $!\n";
open(INDEX, "+>$file.idx")
        or die "Can't open $file.idx for              $!\n";
build_index(*FILE,   *INDEX);
$line = line_with_index(*FILE, *INDEX, $seeking);
```

При наличии модуля DB_File можно воспользоваться методом DB_RECNO, который связывает массив с файлом (по строке на элемент массива):

```
use DB_File;
use Fcntl;

$tie = tie(@lines, $FILE, O_RDWR, 0666, $DB_RECNO) or die
    "Cannot open file $FILE: $!\n";
" Извлечь строку                    "
$line = $lines[$sought-1];
```

Комментарий

Каждый вариант имеет свои особенности и может пригодиться в конкретной ситуации. Линейное чтение легко программируется и идеально подходит для коротких файлов. Индексный метод обеспечивает ускоренную выборку, но требует предварительного построения индекса. Он применяется в случаях, когда индексируемый файл редко изменяется по сравнению с количеством просмотров. Механизму DB_File присущи некоторые начальные издержки, зато последующая выборка строк выполняется намного быстрее, чем при линейном чтении. Обычно он применяется для многократных обращений к большим файлам.

Необходимо знать, с какого числа начинается нумерация строк — с 0 или 1. Переменной $. присваивается 1 после чтения первой строки, поэтому при линейном чтении нумерацию желательно начинать с 1. В индексном механизме широко применяются смещения, и нумерацию лучше начать с 0. DB_File интерпретирует записи файла как элементы массива, индексируемого с 0, поэтому строки также следует нумеровать с 0.

Ниже показаны три реализации одной и той же программы, print_line. Программа получает два аргумента — имя файла и номер извлекаемой строки.

Версия print_line из примера 8.1 просто читает строки файла до тех пор, пока не найдет нужную.

Пример 8.1. print_line-v1

```
ft!/usr/bin/perl   -w
" print_line-v1 - линейное чтение

@ARGV == 2 or die "usage: p rint_line FILENAME LINE_NUMBER\n";

($filename, $line_number) = @ARGV;
open(INFILE, "< $filename") or die "Can't open  $filenamefor          $!\n";
while (<INFILE>) {
    $line = $_;
    last if $. == $line_number;
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 8.1 (продолжение)

```
if ($. != $line_number) {
    die "Didn't find line $line_number in $filename\n";
}
print;
```

Версия из примера 8.2 сначала строит индекс. При большом количестве обращений индекс строится один раз, а затем используется во всех последующих чтениях.

Пример 8.2. print_line-v2

```
#!/usr/bin/perl -w
# print_line-v2 - построение индекса
# Функции build_index и line_with_index приведены выше.
@ARGV == 2 or
    die "usage: print_line FILENAME LINE_NUMBER";

($filename, $line_number) = @ARGV;
open(ORIG, "< $filename")
        or die "Can't open $filename for $!";

# Открыть индекс и при необходимости построить его
# Если две копии программы замечают, что индекс не существует,
# они могут одновременно попытаться построить его.
# Проблема легко решается с применением блокировки.
$indexname = "$filename.index";
sysopen(IDX, $indexname, O_CREAT|O_RDWR)
        or die "Can't open $indexname for $!";
build_index(*ORIG, *IDX) if -z $indexname;

$line = line_with_index(*ORIG, *IDX, $line_number);
die "Didn't find line $line_number in $filename" unless defined $line;
' print $line;
```

Версия с модулем DB_File из примера 8,3 похожа на волшебство.

Пример 8.3. print_line-v3

```
#!/usr/bin/perl -w
# print_line-v3 - решение с применением DB_File
use DB_File;
use Fcntl;

@ARGV == 2 or
    die "usage: print_line FILENAME LINE_NUMBER\n";

($filename, $line_number) = @ARGV;
$tie = tie(@lines, "DB_File", $filename, O_RDWR, 0666, $DB_RECNO)
        or die "Cannot open file $filename: $!\n";

unless ($line_number < $tie->length) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    die "Didn't find line $line_number in $filename\n"
}

print $lines[$line_number-1];                    # Легко, правда?
```

Смотри также

Описание функции tie в *perlfunc*(1); описание специальной переменной **$.** в *perlvar*(1); документация по стандартному модулю DB_File.

8.9. Обработка текстовых полей переменной длины

Проблема

Требуется извлечь из входных данных поля переменной длины.

Решение

Воспользуйтесь функцией split с шаблоном, совпадающим с разделителями полей:

```
# Имеется $ЗАПИСЬ с полями, разделенными шаблоном ШАБЛОН.
# Из записи извлекаются @ПОЛЯ.
ШЛЯ = split(/ШАБЛОН/, $ЗАПИСЬ);
```

Комментарий

Функция split вызывается с тремя аргументами: шаблон, выражение и лимит (максимальное количество извлекаемых полей). Если количество полей во входных данных превышает лимит, лишние поля возвращаются неразделенными в последнем элементе списка. Если лимит не указан, возвращаются все поля (кроме завершающих пустых полей). Выражение содержит разделяемую строковую величину. Если выражение не указано, разделяется переменная $_. Шаблон совпадает с разделителем полей. Если шаблон не указан, в качестве разделителей используются смежные последовательности пропусков, а начальные пустые поля отбрасываются.

Если разделитель входных полей не является фиксированной строкой, можно вызвать split так, чтобы функция возвращала разделители полей вместе с данными, — для этого в шаблон включаются круглые скобки. Например:

```
split(/([+-])/, "3+5-2");
```

возвращает список:

```
(3, '+', 5, '-', 2)
```

Поля, разделенные двоеточиями (в стиле файла */etc/passwd*), извлекаются следующим образом:

```
@fields = split(/:/,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Классическое применение функции split — извлечение данных, разделенных пропусками:

```
@fields = split(/\s+/,
```

Если $ЗАПИСЬ начинается с пропуска, в последнем варианте первому элементу списка будет присвоена пустая строка, поскольку split сочтет, что запись имеет начальное пустое поле. Если это не подходит, используйте особую форму split:

```
@fields = split(" ", $ЗАПИСЬ);
```

В этом случае split ведет себя так же, как и с шаблоном /\s+/> но игнорирует начальный пропуск.

Если разделитель может присутствовать внутри самих полей, возникает проблема. Стандартное решение — снабжать экземпляры разделителя в полях префиксом \. См. рецепт 1.13.

Смотри также

Описание функции split в *perlfunc*(1).

8.10. Удаление последней строки файла

Проблема

Требуется удалить из файла последнюю строку.

Решение

Читайте файл по одной строке и запоминайте байтовое смещение последней прочитанной строки. Когда файл будет исчерпан, обрежьте файл по последнему сохраненному смещению:

```
open (FH, "+< $file")          or die "can't update $file: $!";
while ( <FH> ) {
    $addr = tell(FH) unless eof(FH);
}
truncate(FH, $addr)            or die "can't truncate $file: $!";
```

Комментарий

Такое решение намного эффективнее загрузки всего файла, поскольку в любой момент времени в памяти хранится всего одна строка. Хотя вам все равно приходится читать весь файл, программу можно использовать и для больших файлов, размер которых превышает объем доступной памяти.

Смотри также

Описание функций open и binmode в *perlfunc*(1); man-страницы *open*(2) и *fopen*(3) вашей системы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8.11. Обработка двоичных файлов

Проблема

Операционная система отличает текстовые файлы от двоичных. Как это сделать в программе?

Решение

Вызовите функцию `binmode` для файлового манипулятора:

```
binmode(МАНИПУЛЯТОР);
```

Комментарий

Не существует единого мнения по поводу того, что является строкой текстового файла; текстовые символы одного компьютера могут превратиться в двоичную белиберду на другом. Но даже если все станут пользоваться кодировкой ASCII вместо EBCDIC, Rad50 или Unicode, могут возникнуть затруднения.

Как говорилось во введении, конкретного символа перевода строки не существует. Это чисто абстрактное понятие, которое поддерживается операционной системой, стандартными библиотеками, драйверами устройств и Perl.

В Unix или Plan9 "\n" представляет физическую последовательность "\cJ" (служебная последовательность Perl, соответствующая Ctrl+J). Однако на терминале, не работающем в "чистом" (raw) режиме, нажатие на клавишу Enter генерирует код "\cM" (возврат курсора), транслируемый в "\cJ", а выходной код "\cJ" транслируется в "\cM\cJ". Подобные странности характерны не для обычных файлов, а лишь для терминальных устройств, и обрабатываются строго на уровне драйвера устройства.

На Mac код "\n" обычно представляется "\cM"; чтобы жизнь была интереснее (а также из-за стандартов, требующих различий между "\n" и "\r"), "\r" соответствует "\cJ". Такая интерпретация в точности противоположна стандартам UNIX, Plan9, VMS, CP/M... словом, почти всем. Следовательно, программисты Mac, которые пишут файлы для других систем или общаются с ними по сети, должны проявлять осторожность. Если отправить "\n", вы получите "\cM", а "\cJ" исчезнет. Многие сетевые службы предпочитают отправлять и принимать в качестве разделителя строк последовательность "\cM\cJ", однако большинство позволяет ограничиться простым "\cJ".

В VMS, DOS и их производных "\n" также представляет "\cJ", по аналогии с Unix и Plan9. С терминальной точки зрения UNIX и DOS ведут себя одинаково: при нажатии пользователем клавиши Enter генерируется "\cM", однако в программу поступает уже "\n", то есть "\cJ". Код "\n", переданный терминалу, превращается в "\cM\cJ".

Эти странные преобразования выполняются и с файлами Windows. В текстовых файлах DOS каждая строка завершается двумя символами, "\cM\cJ". Последний блок файла содержит код "\cZ", определяющий окончание текста. В таких системах при записи строки "bad news\n" файл будет содержать "bad news\cM\cJ", как при выводе на терминал.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Но при чтении строк в таких системах происходят еще более странные вещи. Файл содержит "bad news\cM\cJ" — строку, состоящую из 10 байт. При чтении ваша программа не получит ничего, кроме "bad news\n", где "\n" — виртуальный символ перевода строки, то есть "\cJ". Следовательно, от него можно избавиться одним вызовом chop или chomp. Однако при этом приходится обманывать бедную программу и внушать ей, что из файла было прочитано всего 9 байт. Если прочитать 10 таких строк, она будет полагать, что из файла было прочитано 90 байт, хотя в действительности смещение будет равно 100. Из-за этого для определения текущей позиции всегда следует использовать функцию tell. Простой подсчет прочитанных байтов не подходит.

Такое наследие старой файловой системы CP/M, в которой хранились лишь сведения о количестве блоков, но не о размере файлов, бесит программистов уже несколько десятилетий, и конца-края этому не видно. Ведь DOS была совместима с файловым форматом CP/M, Windows — с форматом DOS, а NT — с форматом Windows. Грехи отцов преследуют потомков в четвертом поколении.

Впрочем, проблему одиночного "\n" можно обойти — достаточно сообщить Perl (и операционной системе), что вы работаете с двоичными данными. Функция binmode означает, что прочитанные или записанные через конкретный манипулятор данные не должны преобразовываться по правилам, установленным в системе для текстовых файлов.

```
$gifname = "picture.gif";
open(GIF, $gifname)          or die "can't open $gifname: $!";

binmode(GIF);                # Теперь DOS не преобразует двоичные
                             # входные данные GIF
binmode(STDOUT);             # Теперь DOS не преобразует двоичные
                             # выходные данные STDOUT

while            $buff, 8 * 2**10)) {
    print STDOUT $buff;
}
```

Вызов binmode всистемах, где отличия между текстовыми и двоичнымифайлами несущественны (в том числе UNIX, Mac и Plan9), непринесет никакого вреда. Однако несвоевременный вызов функции в других системах (включая MVS, VMS и всех разновидностей DOS) может исказить содержимое файлов.

Если функция binmode *не используется*, в данных, прочитанных с помощью <>, строковый терминатор системы заменяется на "\n", даже если $/ было присвоено другое значение. Аналогично, любой "\n", выводимый через манипулятор функцией print, превращается в строковый терминатор данной системы. Дополнительные сведения приведены во введении.

Если вы хотите, чтобы прочитанные данные совпадали с содержимым файла байт в байт, и при этом работаете в одной из перечисленных странных систем, — вызовите binmode. Конечно, если вы захотите использовать их с <>, вам придется присвоить $/ настоящий разделитель записей.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Описание функций open и binmode в *perlfunc(1);* страницы руководства *open(2)* и *fopen*(3) вашей системы.

8.12. Ввод/вывод с произвольным доступом

Проблема

Нужно прочитать двоичную запись из середины большого файла, но вам не хочется добираться до нее, последовательно читая все предыдущие записи.

Решение

Определите размер записи и умножьте его на номер записи, чтобы получить смещение в байтах. Затем вызовите seek для полученного смещения и прочитайте запись:

```
$АДРЕС = $РАЗМЕР - $НОМЕР;
seek(FH, $АДРЕС, 0) or die "seek:$!";
         $БУФЕР, $РАЗМЕР);
```

Комментарий

В решении предполагается, что $НОМЕР первой записи равен нулю. Если нумерация начинается с единицы, измените первую строку фрагмента:

```
$АДРЕС = $РАЗМЕР * ($НОМЕР-1);
```

Для текстовых файлов это решение не работает — только строки не имеют одинаковую длину. Но такие ситуации встречаются очень редко.

Смотри также

Описание функции seek в *perlfunc*(1); рецепт **8.13.**

8.13. Обновление файла с произвольным доступом

Проблема

Требуется прочитать старую запись из двоичного файла, изменить ее содержимое и записать обратно.

Решение

Прочитайте старую запись, упакуйте (pack) обновленное содержимое и запишите обратно.

```
use Fcntl;                          # Для SEEK_SET и SEEK_CUR

$ADDRESS = $RECSIZE * $RECNO;
seek(FH, $ADDRESS, SEEK_SET)        ordie "Seeking: $!";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        $BUFFER, $RECSIZE) == $RECSIZE
                                    or die "Reading: $!";
@FIELDS = unpack($FORMAT, $BUFFER);
tt Обновить содержимое, затем
$BUFFER = pack($FORMAT, @FIELDS);
seek(FH, -$RECSIZE, SEEK_CUR)       ordie "Seeking: $!";
print FH $BUFFER;
close FH                            or die "Closing: $!";
```

Комментарий

Для вывода записей в Perl не потребуется **ничего**, кроме функции print. Помните, что антиподом read является print, а не write, хотя, как ни странно, антиподом является syswrite.

В примере 8.4 приведен исходный текст программы weekearly, которой передается один аргумент — имя пользователя. Программа смещает дату регистрации этого пользователя на неделю в прошлое. Конечно, на практике с системными файлами экспериментировать не следует — впрочем, из этого все равно ничего не выйдет! Программа должна иметь право записи для **файла**, поскольку тот открывается в режиме обновления. После выборки и изменения записи программа упаковывает **данные**, возвращается на одну запись назад и записывает буфер.

Пример 8.4. weekearly

```
#!/usr/bin/perl
# weekearly - смещение даты регистрации на неделю назад
use User::pwent;
use IO::Seekable;

$typedef = 'L A12 A16';          # Формат linux ; в sunos - "L A8 A16"
$sizeof  = length(pack($typedef, ()));
$user    = shift(@ARGV) || $ENV{USER} || $ENV{LOGNAME};

          getpwnam($user)->uid    $sizeof;

open (LASTLOG, "+</var/log/lastlog")
    or die "can't update /usr/adm/lastlog: $!";
seek(LASTLOG,
    or die "seek failed: $!";
                        $sizeof) == $sizeof
                failed: $!";

($time, $line, $host) = unpack($typedef, $buffer);
$time -= 24 • 7 * 60 * 60;          " На неделю назад
$buffer = pack($typedef, $time, $line, $time);

seek(LASTLOG, -$sizeof, SEEK_CUR)   # Вернуться на одну запись
    or die "seek failed: $!";
print LASTLO

close(LASTLOG)
    or die "close failed: $!";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Описание функций open, pack и unpack в *perlfunc*(1); рецепты **8.12; 8.14.**

8.14. Чтение строки из двоичного файла

Проблема

Требуется прочитать из файла строку, завершенную нуль-символом, начиная с определенного адреса.

Решение

Присвойте $/ нуль-символ ASCII и прочитайте строку с помощью <>:

```
$old_rs = $/;                    # Сохранить старое значение $/
$/ = "\0";                       # Нуль-символ
seek(FH, $addr, SEEK_SET)        or die "Seek error: $!\n";
$string = <FH>;                  # Прочитать строку
chomp $string;                   # Удалить нуль-символ
$/ = $old_rs;            #          Восстановить старое значение $/
```

При желании сохранение и восстановление $/ можно реализовать с помощью local:

```
{
    local $/ = "\0";
    " ...
}            #            $/ восстанавливается автоматически
```

Комментарий

Программа bgets **из** примера 8.5 получает в качестве аргументов имя файла и одно или несколько байтовых смещений. Допускается десятичная, восьмеричная или **шестнадцатеричная** запись смещений. Для каждого смещения программа читает и выводит строку, которая начинается в данной позиции и завершается нуль-символом или концом файла:

Пример 8.5. bgets

```
#!/usr/bin/perl
# bgets - вывод строк по смещениям в двоичном файле
use IO::Seekable;
($file, @addrs) = @ARGV          or die "usage: $0 addr ...";
open(FH, $file)                  or die "cannot open $file: $!";
$/ = "\000";

foreach $addr (@addrs) {
    $addr = oct $addr if $addr =^ /^0/;
    seek(FH, $addr, SEEK_SET)
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 8.5 (продолжение)

```
        or die "can't seek to $addr in $file: $!";
    printf qq{%#x %#o %d "%s"\n}, $addr, $addr, $addr, scalar <>;
}
```

Приведем простейшую реализацию программы UNIX strings:

Пример 8.6. strings

```
#!/usr/bin/perl
# strings - извлечение строк из двоичного файла
$/ = "\0";
while (0) {
    while (/([\040-\176\s]{4,})/g) {
        print $1, "\n";
```

Смотри также

Описание функций seek, getc и ord в *perlfunc*(1), описание qq// в разделе "Quote and Quote-like Operators" man-страницы *perlop*(1).

8.15. Чтение записей фиксированной длины

Проблема

Требуется прочитать файл с записями фиксированной длины.

Решение

Воспользуйтесь функциями pack и unpack:

```
# $RECORDSIZE - длина записи в байтах.
# $TEMPLATE - шаблон распаковки для записи
# FILE - файл, из которого читаются данные
# @FIELDS - массив для хранения полей

until ( eof(FILE) ) {
                $record, $RECORDSIZE)    $RECORDSIZE
        or die "short read\n";
    @FIELDS = unpack($TEMPLATE, $record);
}
```

Комментарий

Поскольку мы работаем не с текстовым, а с двоичным файлом, для чтения записей нельзя воспользоваться оператором <...> или методом getline модулей IO::. Вместо этого приходится считывать конкретное количество байт в буфер

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

функцией read. После этого буфер содержит данные одной записи, которые декодируются функцией unpack с правильным форматом.

При работе с двоичными данными трудности часто начинаются как раз с правильного выбора формата. Если данные были записаны программой на C, приходится просматривать заголовочные файлы C или страницы руководства с описанием структур, для чего необходимо знание языка C. Заодно вы должны близко подружиться с компилятором C, поскольку без этого вам будет трудно разобраться с выравниванием полей (например, x2 в формате из рецепта 8.18). Если вам посчастливилось работать в Berkeley UNIX или в системе с поддержкой дсс, вы сможете воспользоваться утилитой c2ph, распространяемой с Perl, и заставить компилятор C помочь вам в этом.

Программа tailwtmp в конце этой главы использует формат, описанный в *utmp*(5) системы Linux, и работает с файлами */var/log/wtmp* и */var/run/utmp*. Но стоит вам привыкнуть к работе с двоичными данными, как возникает другая напасть — особенности конкретных компьютеров. Вероятно, программа не будет работать в вашей системе без изменений, но выглядит она поучительно. Приведем соответствующую структуру из заголовочного файла C для Linux:

```
"define UT_LINESIZE          12
"define UT_NAMESIZE          8
"define UT_HOSTSIZE          16

struct utmp {                        /* Коды для шаблона распаковки      */
    short ut_type;                   /* s - short, должно быть дополнено */
    pid_t ut_pid;                    /* i для integer                    */
    char ut_line[UT_LINESIZE];       /* A12 - 12-символьная строка       */
    char ut_id[2];                   /* A2, но для выравнивания
                                        необходимо x2      '             */
    time_t ut_time;                  /* 1 - long                         */
    char ut_user[UT_NAMESIZE];       /* A8 - 8-символьная строка         "/
    char ut_host[UT_HOSTSIZE];       /* A16 - 16-символьная строка       */
    long ut_addr;                    /* 1 - long                         */
};
```

Вычисленная двоичная структура (в нашем примере — "s x2 *i* A12 A2 x2 1 A8 A16 1") передается pack с пустым списком полей для определения размера записи. Не забудьте проверить код возврата чтении записи, чтобы убедиться в том, что вы получили запрошенное количество байт.

Если записи представляют собой текстовые стр оки, используйте шаблон распаковки "a" или "A".

Записи фиксированной длины хороши тем, что n-я запись начинается в файле со смещения SIZE*(n-1), где SIZE — размер одной записи. Пример приведен в программе с построением индекса из рецепта 8.8.

Смотри также

Описание функций unpack, pack и в *perlfunc*(1); рецепт 1.1.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8.16. Чтение конфигурационных файлов

Проблема

Вы хотите, чтобы пользователи вашей программы могли изменить ее поведение с помощью конфигурационного файла.

Решение

Организуйте обработку файла в тривиальном формате ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, создавая для каждого параметра элемент хэша "ключ/значение":

```
while (<CONFIG>) {
    chomp;                  # Убрать перевод строки
    s/#.*//;                # Убрать комментарии
    s/^\s+//;               # Убрать начальные пропуски
    s/\s+$//;               # Убрать конечные пропуски
    next unless length;       # Что-нибудь осталось?
    my ($var, $value) = split(/\s*=\s*/, $_, 2);
    $User_Preferences{$var} = $value;
}
```

Существует другой более изящный вариант — интерпретировать конфигурационный файл какполноценный код Perl:

```
do "$ENV{HOME}/.progrc";
```

Комментарий

В первом решении конфигурационный файл интерпретируется в тривиальном формате следующего вида (допускаются комментарии и пустые строки):

```
8 Сеть класса C
NETMASK = 255.255.255.0
MTU     = 296

DEVICE  = cua1
RATE    = 115200
MODE    = adaptive
```

После этого можно легко получить значение нужных параметров — например, `$User_Preferences{"RATE"}` дает значение 115200. Если вы хотите, чтобы конфигурационный файл непосредственно устанавливал значения переменных в программе вместо заполнения хэша, включите в программу следующий фрагмент:

```
no strict 'refs';
$$var = $value;
```

и переменная `$RATE` будет содержать значение 115200.

Во втором решении do организует непосредственное выполнение кода Perl. Если вместо блока используется выражение, do интерпретирует его как имя файла. Это практически идентично применению риска фатальных

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

исключений. В формате второго решения конфигурационный файл принимает следующий вид:

```
# Сеть класса C
$NETMASK = '255.255.255.0';
$MTU     = 0x128;
$DEVICE  = 'cua1';
$RATE    = 115_200;
$MODE    = 'adaptive';
```

Если вам непонятно, зачем включать в файл лишние знаки препинания, задумайтесь — в вашем распоряжении оказывается весь синтаксис Perl. Теперь простые присваивания можно дополнить логикой и проверкой условий:

```
if ($DEVICE =~ /1$/) {
    $RATE = 28_800;
} else {
    $RATE = 115 200;
}
```

Во многих программах предусмотрены системные и личные конфигурационные файлы. Если вы хотите, чтобы предпочтения пользователя отменяли действия системных параметров, загрузите личный файл после системного:

```
$APPDFLT = "/usr/local/share/myprog";

do "$APPDFLT/sysconfig.pl";
do "$ENV{HOME}/.myprogrc";
```

Если при существующем личном файле системный файл должен игнорироваться, проверьте возвращаемое значение do:

```
do "$APPDFLT/sysconfig.pl"
    or
do "$ENV{HOME}/.myprogrc";
```

Возможно, вас интересует, в каком контексте должны выполняться эти файлы. Они будут принадлежать пакету, в котором была откомпилирована команда do. Обычно пользователи устанавливают значения конкретных переменных, которые представляют собой неуточненные глобальные величины и потому принадлежат текущему пакету. Если вы предпочитаете, чтобы неуточненные переменные относились к конкретному пакету, воспользуйтесь записью вида:

```
{ package Settings; do "$ENV{HOME}/.myprogrc" }
```

Файл, прочитанный с помощью do (а также `require` и use), представляет собой отдельную, самостоятельную область действия. Это означает как то, что конфигурационный файл не может обратиться к лексическим (ту) переменным вызывающей стороны, так и то, что вызывающая сторона не сможет найти такие переменные, заданные в файле. Кроме того, пользовательский код не подчиняется директивам типа use strict или use `integer`, способным воздействовать на вызывающую сторону.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если столь четкое разграничение видимости переменных нежелательно, вы можете заставить код конфигурационного файла выполняться в вашей лексической области действия. Имея под рукой программу cat или ее эквивалент, можно написать доморощенный аналог do:

```
eval `cat $ENV{HOME}/.myprogrc`;
```

Мы еще не видели, чтобы кто-нибудь (кроме Ларри) использовал такой подход в рабочем коде.

Во-первых, do проще вводится. Кроме того, do учитывает @INC, который обычно просматривается при отсутствии полностью указанного пути, но в отличие выполняется неявная проверка ошибок. Следовательно, вам не придется заворачивать do в eval для перехвата исключений, от которых ваша программа может скончаться, поскольку do уже работает как eval.

При желании можно организовать собственную проверку ошибок:

```
$file = "someprog.pl";
unless ($return = do $file) {
    warn "couldn't parse $file: $@"          if $@;
    warn "couldn't do $file: $!"             unless defined $return;
    warn "couldn't run $file"                unless $return;
}
```

Программисту намного проще отследить это в исходном тексте, чем изобретать новый, сложный синтаксис. Проще будет и пользователю, которому не придется изучать правила синтаксиса очередного конфигурационного файла. Приятно и то, что пользователь получает доступ к мощному алгоритмическому языку программирования.

Однако не следует забывать о безопасности. Как убедиться в том, что файл не модифицировался никем, кроме пользователя? Традиционный подход — не делать ничего, полагаясь исключительно на права доступа каталогов и файлов. В девяти случаях из десяти такое решение оказывается правильным, поскольку большинство проектов попросту не оправдывает подобной паранойи. А если все же оправдывает, загляните в следующий рецепт.

Смотри также

Описание функций eval и в *perlfunc*(1); рецепты 8.17; 10.12.

8.17. Проверка достоверности файла

Проблема

Требуется прочитать файл (например, содержащий данные о конфигурации). Вы хотите использовать файл лишь в том случае, если правом записи в него (а возможно, даже правом чтения) не обладает никто, кроме его владельца.

Решение

Получите данные о владельце и правах доступа с помощью функции stat. Можно воспользоваться встроенной версией, которая возвращает список:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
( $dev, $ino, $mode, $nlink,
  $uid, $gid, $rdev, $size,
  $atime, $mtime, $ctime,
  $blksize, $blocks )       = stat($filename)
        or die "no $filename: $!";

$mode &= 07777;                # Отбросить информацию о типе файла
```

Или воспользуйтесь интерфейсом с именованными полями:

```
$info = stat($filename)     or die "no $filename: $!";
if ($info->uid == 0) {
    print "Superuser owns $filename\n";
}
if ($info->atime > $info->mtime) {
    print "$filename has been        since it was written.\n";
}
```

Комментарий

Обычно мы доверяем пользователям и позволяем им устанавливать права доступа по своему усмотрению. Если они захотят, чтобы другие могли читать или даже записывать данные в их личные файлы — это их дело. Однако многие приложения (редакторы, почтовые программы, интерпретаторы) часто отказываются выполнять код конфигурационных файлов, если запись в них осуществлялась кем-то, кроме владельца. Это помогает избежать нападений "троянских" программ. Программы, следящие за безопасностью — например, ftp или rlogin, — могут даже отвергнуть конфигурационные файлы, прочитанные кем-то, кроме владельца.

Если файл может быть записан кем-то, кроме владельца, или принадлежит кому-то, отличному от текущего или привилегированного пользователя; он не признается достоверным. Информация о владельце и правах доступа может быть получена с помощью функции stat. Следующая функция возвращает true для достоверных файлов и false для всех остальных. Если вызов stat завершается неудачей, возвращается undef.

```
use File::stat;

sub is_safe {
    my $path = shift;
    my $info = stat($path);
            unless $info;

    # Проверить владельца (привилегированный или текущий пользователь)
    # Настоящий идентификатор пользователя хранится в переменной $<.
    if (($info->uid != 0) && ($info->uid != $<)) {


    # Проверить, может ли группа или остальные пользователи
    # записывать в файл.
    в Для проверки чтения/записи используйте константу 066
    if ($info->mode & 022) {    # Если другие имеют право записи
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                unless -d _;  " Не-каталоги недостоверны .
            " но каталоги с битом запрета (01000) - достоверны
                unless $info->mode & 01000;
        }

    }
```

Каталог считается достоверным даже в том случае, если другие имеют право записи в него — при условии, что для него установлен бит **01000** (разрешающий удаление только владельцу каталога).

Осторожный программист также проследит, чтобы запись была запрещена и для всех каталогов верхнего уровня. Это связано с известной "проблемой chown", при которой любой пользователь может передать принадлежащий ему файл и сделать его владельцем кого-то другого. Приведенная ниже функция is_very_safe обращается к функции POSIX::sysconf, чтобы выяснить, существует ли "проблема chown" всистеме. Если проблема существует, далее функцией проверяются is_safe всекаталоги верхнего уровня вплоть до корневого. Если в вашей системе установлена ограниченная версия chown, функция is_very_safe ограничивается простым вызовом is_safe.

```
use Cwd;
use POSIX qw(sysconf _PC_CHOWN_RESTRICTED);
sub is_verysafe {
    my $path = shift;
            is_safe($path)    sysconf(_PC_CHOWN_RESTRICTED);
    $path = getcwd() .  '/' • $path if $path !~ m{^/};
    do {
                unless is_safe($path);
        $path =" s#([^/]+|/)$##;        #        Имя каталога
        $path =" s#/$#  if length($path) > 1;  # Последний символ /
    > while length $path;

          1;
}
```

В программе эта функция используется примерно так:

```
$file = "$ENV{HOME}/.myprogrc";
                    is_safe($file);
```

При этом возникает потенциальная опасность перехвата, поскольку предполагается, что файл открывается гипотетической функцией readconfig. Между получением сведений о файле (is_safe) и егооткрытием функцией readconfig теоретически может случиться что-нибудь плохое. Чтобы избежать перехвата, передавайте is_ safe уже открытый файловый манипулятор:

```
$file = "$ENV{HOME}/.myprogrc";
if (open(FILE, "< $file")) {
    readconfig(*FILE) if is safe(*FILE);
>
```

Впрочем, вам также придется позаботиться о том, чтобы функция readconfig принимала файловый манипулятор вместо имени.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8.18. Программа: tailwtmp

В начале и конце рабочего сеанса пользователя в системе UNIX в файл *wtmp* добавляется новая запись. Вам не удастся получить ее с помощью обычной программы tail, поскольку файл хранится в двоичном формате. Программа tailwtmp из примера 8.7 умеет работать с двоичными файлами и выводит новые записи по мере их появления. Формат pack придется изменить для конкретной системы.

Пример 8.7. tailwtmp

```
#!/usr/bin/perl
# tailwtmp - отслеживание начала/конца сеанса
# Использует структуру linux utmp, см. utmp(5)
$typedef = 's x2 i A12 A4 1 A8 A16 l';
$sizeof = length pack($typedef, ());
use IO::File;
open(WTMP, '/var/log/wtmp') or die. "can't open /var/log/wtmp: $!";
seek(WTMP, 0, SEEK_END);
for (;;) {
        while (read(WTMP, $buffer, $sizeof) == $sizeof) {
        ($type, $pid, $line, $id, $time, $user, $host, $addr)
            = unpack($typedef, $buffer);
        next unless $user && ord($user) && $time;
        printf "%1d %-8s %-12s %2s %-24s %-16s %5d %08x\n",
            $type,$user,$line,$id,scalar(localtime($time)),
     ,      $host,$pid,$addr;
    >
    for ($size = -s WTMP; $size == -s WTMP; sleep 1) {}
    WTMP->clearerr();
```

8.19. Программа: tctee

Во многих системах существует классическая программа tee для направления выходных данных в несколько приемников. Например, следующая команда передает выходные данные someprog в */tmp/output* и через конвейер — в Почтовую систему:

```
% someprog | tee /tmp/output | Mail -s 'check this' user@host.org
```

Программа tctee пригодится не только тем пользователям, которые работают вне UNIX и не имеют tee. Она обладает некоторыми возможностями, отсутствующими в стандартной версии tee.

При запуске программа может получать четыре флага:

-i — игнорировать прерывания,

-a — дописывать данные в конец выходных файлов,

-u — выполнять небуферизованный вывод,

-n — отменить копирование выходных данных в стандартный вывод.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Поскольку в программе используется "волшебная" функция open, вместо файлов можно передавать каналы:

```
% someprog | tctee f1  "|cat -n"f2  ">>f3"
```

В примере 8.8 приведена программа-ветеран, написанная на Perl почти **10** лет назад и работающая до сих пор. Если бы нам пришлось писать ее заново, вероятно, мы бы использовали strict, предупреждения и модули с десятками тысяч строк. Но как известно, "лучшее — враг хорошего".

Пример 8.8. tctee

```
#!/usr/bin/perl
# tctee - клон tee
# Программа совместима с perl версии 3 и выше.

while ($ARGV[0] =^ /^-(.+)/ && (shift, ($_ = $1), 1)) {
    next if /^$/;
    s/i// &&
    s/a// &&
    s/u// && (++$unbuffer,'
    s/n// && (++$nostdout,
    die "usage tee [-aiun] [filenames] ...\n";
}


    for $sig ('INT', 'TERM', 'HUP', 'QUIT') { $SIG{$sig} = 'IGNORE'; }

}
$SIG{'PIPE'} = 'PLUMBER';
$mode = $append ? "" : '>';
$fh = 'FHOOO';

unless ($nostdout) {
    %fh = ('STDOUT', 'standard output'); # Направить в STDOUT
    }

$| = 1 if $unbuffer;

for (@ARGV) {
    if (!open($fh, (/^[^>|]/ && $mode) . $_)) {
        warn "$0: cannot open $_: $!\n"; # Как в sun; я предпочитаю die
        $status++;
        next;
    }
    select((select($fh), $| = 1)[0]) if $unbuffer;
    $fh{$fh++> = $_;
}

while (<STDIN>) {
    for $fh (keys %fh) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        print $fh $_,
    >
}

for $fh (keys %fh) {
    next if close($fh) || !defined $fh{$fh},
    warn "$0: couldn't close $fh{$fh}: $!\n",
    $status++,
}

exit $status,

sub PLUMBER {
    warn "$0: pipe to \$fh{$fh}\: broke!\n"
    $status++
    delete $fh{$fh}
}
```

8.20. Программа: laston

Во время регистрации в системе UNIX на экран выводятся сведения о времени последней регистрации. Эта информация хранится в двоичном файле с именем *lastlog*. Каждый пользователь имеет собственную запись в этом файле; данные пользователя с UID 8 хранятся в записи 8, UID 239 — в записи 239 и т. д. Чтобы узнать, когда пользователь с заданным UID регистрировался в последний раз, преобразуйте имя пользователя в числовое значение UID, найдите соответствующую запись в файле, прочитайте и распакуйте данные. Средствами интерпретатора это сделать очень сложно, зато в программе laston все очень легко. Приведем пример:

```
% laston gnat
gnat  UID 314 at Mon May 25 08:32:52 1998 on ttyp0 from below.perl.com
```

Программа из примера 8 9 была написана гораздо позже программы tctee из примера 8.8, однако она менее переносима, поскольку в ней используется двоичная структура файла *lastlog* системы UNIX. Для других систем ее необходимо изменить.

Пример 8.9. laston

```
#!/usr/bin/perl
# laston - определение времени последней регистрации пользователя
use User::pwent,
use IO::Seekable qw(SEEK_SET),

open (LASTLOG, "/var/log/lastlog") or die "can't open /usr/adm/lastlog: $!",

$typedef = "L A12 A16",  # Формат linux, для SunOS - "L A8 A16"
$sizeof = length(pack($typedef, ())),
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 8.9 (продолжение)

```
for $user (@ARGV) {
    $U = ($user =' /^\d+$/) ? getpwuid($user) : getpwnam($user);
    unless ($U) { warn "no such uid $user\n"; next; }
    seek(LASTLOG, $U->uid * $sizeof, SEEK_SET) or die "seek failed: $!'
    read(LASTLOG, $buffer, $sizeof) == $sizeof or next;
    ($time, $line, $host) = unpack($typedef, $buffer);
    printf "%-8s UID %5d %s%s%s\n", $U->name, $U->uid,
          $time ? ("at " . localtime($time)) : "never logged in",
          $line && " on $line",
          $host && " from $host";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9 Каталоги

У UNIX есть свои недостатки, но файловая система к ним не относится.

Крис ***Торек***

Введение

Для полноценного понимания работы с каталогами необходимо понимать механизмы, заложенные в ее основу. Наш материал ориентирован на файловую систему UNIX, поскольку функции каталогов Perl разрабатывались для системных функций и особенностей именно этой системы, однако в определенной степени он относится и к большинству других платформ.

Файловая система состоит из двух компонентов: набора блоков данных, где хранится содержимое файлов и каталогов, и индекса к этим блокам. Каждому объекту файловой системы, будь то обычный файл, каталог, ссылка или специальный файл (вроде файлов из каталога */dev*), соответствует определенный элемент индекса. Элементы индекса называются *индексными узлами* (inode). Поскольку индекс является одномерным, индексные узлы определяются по номерам.

Каталог представляет собой файл специального формата, помеченный в индексном узле как каталог. Блоки данных каталога содержат множество пар. Каждая пара содержит имя объекта каталога и соответствующий ему индексный узел. Блоки данных каталога */usr/bin* могут содержать следующую информацию:

Имя	Индексный узел
bc	17
du	29
nvi	8
pine	55
vi	8

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Подобную структуру имеют все каталоги, включая корневой (/). Чтобы прочитать файл */usr/bin/vi*,операционная система читает индексный узел /, находит в его блоках данных информацию о */usr*,читает индексный узел */usr*, находит в его блоках данных информацию о */usr/bin*, читает индексный узел */usr/bin,* находит в его блоках данных информацию о */usr/bin/vi*,читает индексный узел */usr/bin/vi*, после чего читает данные из блока данных.

Имена, хранящиеся в каталогах, не являются полными. Файл */usr/bin/vi*хранится в каталоге */usr/bin* под именем *vi.* Если открыть каталог */usr/bin* и последовательно читать его элементы, вы увидите имена файлов *tyatch, login* и *vi)* вместо полных имен */usr/bin/patch,/usr/bin/rlogin* и */usr/bin/vi.*

Однако индексный узел — больше, чем просто указатель на блоки данных. Каждый индексный узел также содержит информацию о типё представляемого объекта (каталог, обычный файл и т. д.) и его размере, набор битов доступа, информацию о владельце и группе, время последней модификации объекта, количество элементов каталога, ссылающихся на данный узел, и т. д.

Одни операции с файлами изменяют содержимое блоков данных файла; другие ограничиваются изменением индексного узла. Например, при дополнении или усечении файла в его индексном узле изменяется информация о размере. Некоторые операции изменяют элемент каталога, содержащий ссылку на индексный узел файла. Изменение имени файла влияет только на элемент каталога; ни данные файла, ни его индексный узел не изменяются.

В трех полях структуры индексного узла хранится время последнего обращения, изменения и модификации: `atime`, `ctime` и `mtime`. Поле atime обновляется при каждом чтении данныхфайла через указатель на его блоки данных. Поле `mtime` обновляется при каждом изменении содержимого файла. Поле ctime обновляется при каждом изменении индексного узла файла. Ctime не *является* временем создания; в стандартных версиях UNIX время создания файла определить невозможно.

При чтении файла изменяется только значение `atime`. Переименование файла не отражается на atime, ctime или `mtime`, поскольку изменяется лишь элемент каталога (хотя при этом меняются atime и mtime для каталога, в котором находится файл). Усечение файла не влияет на atime (поскольку мы не читаем, а лишь изменяем поле размера в элементе каталога), но изменяет ctime (из-за изменения поля размера) и mtime (из-за изменения содержимого, хотя бы и косвенного).

Чтобы получить индексный узел по имени файла или каталога, можно воспользоваться встроенной функцией stat. Например, индексный узел файла */usr/bin/vi* может быть получен следующим образом:

```
@entry = stat("/usr/bin/vi") or die "Couldn't stat /usr/bin/vi : $!";
```

Следующий фрагмент получает индексный узел для каталога */usr/bin*:

```
@entry = stat("/usr/bin")     or die "Couldn't stat /usr/bin : $!";
```

Функция stat также вызывается и для файловых манипуляторов:

```
@entry = stat(INFILE)        or die "Couldn't stat INFILE : $!";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Функция stat возвращает список значений, хранящихся в полях элемента каталога. Если получить информацию не удалось (например, если файл не существует), функция возвращает пустой список. В приведенных примерах пустой список проверялся конструкцией `or die`. Не путайте с конструкцией `|| die`, поскольку выражение будет преобразовано в скалярный контекст и функция stat сообщит лишь о том, успешно ли она была вызвана. Список при этом не возвращается. Впрочем, кэш _ (см. ниже) все жебудет обновлен.

Элементы списка, возвращаемые функцией stat, перечислены в следующей таблице.

Элемент	Обозначение	Описание
0	dev	Номерустройствавфайловой системе
1	ino	Номер индексного узла
2	mode	Режим файла (тип и права доступа)
3	nlink	Количество (прямых) ссылок на файл
4	uid	Числовой идентификатор пользователя владельца файла
5	gid	Числовой идентификатор группы владельца файла
6	rdev	Идентификатор устройства (только для специальных файлов)
7	size	Общий размер файла в байтах
8	atime	Время последнего обращения (в секундах с начала эпохи)
9	mtime	Время последней модификации (в секундах с начала эпохи)
10	ctime	Время изменения индексного узла (в секундах с начала эпохи)
11	blksize	Предпочтительный размер блока для операций ввода/вывода в файловой системе
12	blocks	Фактическое количество выделенных блоков

Стандартный модуль File::stat предоставляет именованный интерфейс к этим значениям. Он переопределяет функцию stat, поэтому вместо массива, описанного выше, функция возвращает объект с методами для получения каждого атрибута:

```
use File::stat;

$inode = stat("/usr/bin/vi");
$ctime = $inode->ctime;
$size  = $inode->size;
```

Кроме того, в Perl предусмотрен набор операторов, вызывающих функцию stat и возвращающих лишь один атрибут. Эти операторы совокупно называются "операторами -X>>, поскольку их имена состоят из дефиса, за которым следует ОДИН СИМВОЛ. Они построены по образцу операторов test командного интерпретатора.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

-X	Поле stat	Значение
-r	mode	Файл может читаться текущими UID/GID
-w	mode	Файлможет записываться текущими UID/GID
-x	mode	Файл может исполняться текущими UID/GID
-o	mode	Владельцем файла является текущий UID
-R	mode	Файл может читаться фактическими UID/GID
-W	mode	Файл может записываться фактическими UID/GID
-X	mode	Файл может исполняться фактическими UID/GID
-0	mode	Владельцем файла является фактический UID
-e		Файл существует
-z	size	Размер файла равен нулю
-s	size	Размер файла отличен от нуля (возвращает размер)
-f	mode, rdev	Файл является обычным файлом
-d	mode, rdev	Файл является каталогом
-1	mode	Файл является символической ссылкой
-p	mode	Файл является именованным каналом (FIFO)
-S	mode	Файл является сокетом
-b	rdev	Файл является блочным специальным файлом
-c	rdev	Файл является символьным специальным файлом
-t	rdev	Файловый манипулятор открыт для терминала
-u	mode	У файла установлен бит setuid
-g	mode	У файла установлен бит setgid
-k	mode	У файла установлен бит запрета
-T	-	Файл является текстовым
-B	-	Файл является двоичным (противоположность -T)
-M	mtime	Возраст файла в днях на момент запуска сценария
-Д	atime	То же для времени последнего обращения

Функция stat и операторы -X кэшируют значения, полученные при вызове системной функции stat(2). Если stat или оператор -X вызывается для специального файлового манипулятора _ (один символ подчеркивания), то вместо повторного вызова stat будет использована информация, хранящаяся в кэше. Это позволяет проверять различные атрибуты файла без многократного вызова stat(2) или возникновения опасности перехвата:

```
open( F, "< $filename" )
    or die "Opening $filename: $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
unless (-s F && -T _) {
    die "$filename doesn't have text in it.\n";
}
```

Однако отдельный вызов stat возвращает информацию лишь об одном индексном узле. Как же получить список содержимого каталога? Для этой цели в Perl предусмотрены функции opendir, closedir:

```
opendir(DIRHANDLE, "/usr/bin") or die "couldn't open /usr/bin : $!";
while ( defined ($filename = readdir(DIRHANDLE)) ) {
    print "Inside /usr/bin is something called $filename\n";
}
closedir(DIRHANDLE);
```

Функции чтения каталога намеренно разрабатывались по аналогии с функциями открытия и закрытия файлов. Однако если функция open вызывается для манипулятора файла, то opendir получает манипулятор каталога. Внешне они похожи, но работают по-разному: в программе могут соседствовать вызовы open(BIN, "/a/file") и opendir(BIN, "/a/dir"), и Perl не запутается. Вы — возможно, но Perl точно не запутается. Поскольку манипуляторы файлов отличаются от манипуляторов каталогов, вы не сможете использовать оператор о для чтения из манипулятора каталога.

Имена файлов в каталоге не обязательно хранятся в алфавитном порядке. Чтобы получить алфавитный список файлов, прочитайте все содержимое каталога и отсортируйте его самостоятельно.

Отделение информации каталога от информации индексного узла может быть связано с некоторыми странностями. Операции, изменяющие каталог, требуют права записи для каталога, но не для файла. Большинство операций, изменяющих содержимое файла, требует права записи в файл. Операции, изменяющие права доступа к файлу, требуют, чтобы вызов осуществлялся владельцем файла или привилегированным пользователем. Могут возникнуть странные ситуации — например, появляется возможность удаления файла, который нельзя прочитать, или записи в файл, который нельзя удалить.

Хотя из-за подобных ситуаций файловая система на первый взгляд кажется нелогичной, в действительности они способствуют широте возможностей UNIX. Реализация ссылок (два имени, ссылающиеся на один файл) становится чрезвычайно простой — в двух элементах каталога просто указывается один номер индексного узла. Структура индексного узла содержит количество элементов каталога, ссылающихся на данный файл (nlink в списке значений, возвращаемых stat), что позволяет операционной системе хранить и поддерживать лишь одну копию времени модификации, размера и других атрибутов файла. При уничтожении ссылки на элемент каталога блоки данных удаляются лишь в том случае, если это была последняя ссылка для индексного узла данного файла, а сам файл не остается открытым ни в одном процессе. Можно вызвать unlink и для открытого файла, но дисковое пространство будет освобождено лишь после его закрытия последним процессом.

Ссылки делятся на два типа. Тип, описанный выше (два элемента каталога, в которых указан один номер индексного узла), называется *прямой* (или *жесткой)*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ссылкой (hard link). Операционная система не может отличить первый элемент каталога, соответствующий файлу (созданный при создании файла), от всех последующих ссылок на него. Со ссылками другого типа — ***символическими ссылками*** — дело обстоит совершенно иначе. Символические ссылки представляют собой файлы особого типа: в блоке данных хранится имя файла, на который указывает ссылка. Символические ссылки имеют особое значение `mode`, отличающее их от обычных файлов. При вызове `open` для символической ссылки операционная система открывает файл, имя которого указано в блоке данных.

Резюме

Имена файлов хранятся в каталогах отдельно от размера, атрибутов защиты и прочих метаданных, хранящихся в индексномузле.

Функция stat возвращает информацию индексного узла (метаданные).

Функции `opendir`, спутники обеспечивают доступ к именам файлов в каталоге с помощью *манипулятора каталога*.

Манипулятор каталога похож на файловый манипулятор, но не идентичен ему. В частности, для манипулятора каталога нельзя вызвать <>.

Права доступа к каталогу определяют, можете ли вы прочитать или записать список имен файлов. Права доступа к файлу определяют, можете ли вы изменить метаданные или содержимое файла.

В индексном узле хранятся три атрибута времени. Ни один из них не определяет время создания файла.

9.1. Получение и установка атрибутов времени

Проблема

Требуется получить или изменить время последней модификации (записи или изменения) или обращения (чтения) для файла.

Решение

Функция stat получает атрибуты времени, а функция `utime` устанавливает их значения. Обе функции являются встроенными в Perl:

```
($READTIME, $WRITETIME) = (stat($filename))[8,9];

utime($NEWREADTIME, $NEWWRITETIME, $filename);
```

Комментарий

Как говорилось во введении, в традиционной файловой системе UNIX с каждым индексным узлом связываются три атрибута времени. Любой пользователь может установить значения `atime` и `mtime` функцией `utime`, если он имеет право записи в каталог, содержащий файл. Изменить `ctime` практически невозможно. Следующий пример демонстрирует вызов функции `utime`:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$SECONDS_PER_DAY = 60 * 60 * 24;
($atime, $mtime) = (stat($file))[8,9];
$atime -= 7 * $SECONDS_PER_DAY;
$mtime -= 7 * $SECONDS_PER_DAY;

utime($atime, $mtime, $file)
    or die "couldn't backdate $file by a week w/ utime: $!";
```

Функция utime должна вызываться для обоих атрибутов, atime и mtime. Если вы хотите задать лишь одно из этих значений, необходимо предварительно получить другое с помощью функции stat:

```
$mtime = (stat $file)[9];
utime(time, $mtime, $file);
```

Применение модуля File::stat упрощает этот фрагмент:

```
use File::stat;
utime(time, stat($file)->mtime, $file);
```

Функция utime позволяет сделать вид, будто к файлу вообще никто не притрагивался (если не считать обновления ctime). Например, для редактирования файла можно воспользоваться программой из примера 9.1.

Пример 9.1. uvi

```
#!/usr/bin/perl -w
# uvi - редактирование файла в vi без изменения атрибутов времени

$file = shift or die "usage: uvi filename\n";
($atime, $mtime) = (stat($file))[8,9];
system($ENV{EDITOR} || "vi", $file);
utime($atime, $mtime, $file)
    or die "couldn't          $file to orig times: $!";
```

Смотри также

Описание функций stat и utime в *perlfunc(1)*; стандартный модуль File::stat и страница руководства *utime*(3).

9.2. Удаление файла

Проблема

Требуется удалить файл. Функция Perl delete вам не подходит.

Решение

Воспользуйтесь функцией Perl unlink:

```
unlink($FILENAME)                  or die "Can't delete $FILENAME: $!\n";
unlink(@FILENAMES) == @FILENAMES   or die
                                   "Couldn't unlink all of @FILENAMES: $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Функция unlink была названа по имени системной функции UNIX. В Perl она получает список имен файлов и возвращает количество успешно удаленных файлов. Возвращаемое значение можно проверить с помощью || или or:

```
unlink($file) or die "Can't unlink $file: $!";
```

Функция unlink не сообщает, какие файлы не были удалены — лишь ихобщее количество. Следующий фрагмент проверяет", успешно ли состоялось удаление нескольких файлов, и выводит количество удаленных файлов:

```
unless (($count = unlink(@filelist)) == @filelist) {
    warn "could only delete $count of "
            . (@filelist) . " files";
}
```

Перебор @fil elist в цикле выводить отдельные сообщения об ошибках.

В UNIX удаление файла из каталога требует права записи для **каталога**[1], а не для файла, поскольку изменяется именно каталог. В некоторых ситуациях появляется возможность удаления файла, в который запрещена запись, или записи в файл, который нельзя удалить.

Если удаляемый файл открыт некоторым процессом, операционная система удаляет элемент каталога, но не освобождает блоки данных до закрытия файла во всех процессах. Именно так работает функция new_tmpfile в IO::File (см. рецепт 7.5).

Смотри также

Описание функции unlink в *perlfunc*(1); страница руководства *unlink*(2). Идея с удаленным файлом, который продолжает оставаться доступным, применяется в рецепте 7.5.

9.3. Копирование или перемещение файла

Проблема

Необходимо скопировать файл, однако в Perl не существует встроенной команды копирования.

Решение

Воспользуйтесь функцией copy стандартного модуля File::Copy:

```
use File::Copy;
copy($oldfile, $newfile);
```

Если для каталога не был установлен бит запрета 010000, который разрешает удаление только владельцу. В общих каталогах типа /tmp по соображениям безопасности обычно используется режим 01777.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

То же самое делается и вручную:

```
open(IN,  "< $oldfile")          or die "can't open $oldfile: $!";
open(OUT, "> $newfile")           or die "can't open $newfile: $!";

$blksize = (stat IN)[11] || 16384;        # Желательный размер блока?
while ($len =          IN, $buf, $blksize) {
    if (!defined $len) {
        next if $! =~ /^Interrupted/;
        die "System       error: $!\n";
    }
    $offset = 0;
    while ($len) {           # Частичные операции записи
        defined($written = syswrite OUT, $buf, $len, $offset)
            or die "System write error: $!\en";
        $len    -= $written;
        $offset += $written;
    };
}

close(IN);
close(OUT);
```

Также можно воспользоваться программой copy вашей системы:

```
system("cp $oldfile $newfile");        # unix
system("copy $oldfile $newfile");      # dos, vms
```

Комментарий

Модуль File::Copy содержит функции copy и move. Они удобнее низкоуровневых функций ввода/вывода и обладают большей переносимостью по сравнению с вызовом system. Функция move допускает перемещение между каталогами, а стандартная функция Perl rename — нет(обычно).

```
use File::Copy;

copy("datafile.dat", "datafile.bak")
    or die "copy failed: $!";

move("datafile.new", "datafile.dat")
    or die "move failed: $!";
```

Поскольку обе функции возвращают лишь простой признак успешного завершения, вы не сможете легко определить, какой файл помешал успешному копированию или перемещению. При ручном копировании файлов можно узнать, какие файлы не были скопированы, но в этом случае ваша программа забивается сложными вызовами syswrite.

Смотри также

Описание функций и syswrite в *perlfunc*(1); документация по стандартному модулю File::Copy.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9.4. Распознавание двух имен одного файла

Проблема

Требуется узнать, соответствуют ли два имени файла из списка одному и тому же файлу на диске (благодаря жестким и символическим ссылкам два имени могут ссылаться на один файл). Такая информация поможет предотвратить модификацию файла, с которым вы уже работаете.

Решение

Создайте хэш, **кэшируемый** по номеру устройства и индексного узла для уже встречавшихся файлов. В качестве значений хэша используются имена файлов:

```
%seen = ();

sub do_my_thing {
    my $filename = shift;
    my ($dev, $ino) = stat $filename;

    unless (! $seen{$dev, $ino}++) {
        # Сделать что-то с $filename, поскольку это имя
        # нам еще не встречалось
    }
}
```

Комментарий

Ключ %seen образуется объединением номеров устройства ($dev) и индексного узла ($ino) каждого файла. Для одного файла номера устройства и индексного узла совпадут, поэтому им будут соответствовать одинаковые ключи.

Если вы хотите вести список всех файлов с одинаковыми именами, то вместо подсчета экземпляров сохраните имя файла в анонимном массиве:

```
foreach $filename (@files) {
    ($dev, $ino) = stat $filename;
    push( @{ $seen{$dev,$ino} }, $filename);
}

                (sort keys %seen) {
    ($dev, $ino) = split(/$;/o, $devino);
    if (@{$seen{$devino}} > 1) {
        # @{$seen{$devino}} - список имен одного файла
    }
}
```

Переменная $; содержит строку-разделитель и использует старый синтаксис эмуляции многомерных массивов, $hash{$x,$y,$z}. Хэш остается одномерным,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

однако он имеет составной ключ. В действительности ключ представляет собой join($; =>$x, $y, $z). Функция split снова разделяет составляющие. Хотя многоуровневый хэш можно использовать и напрямую, здесь в этом нет необходимости и дешевле будет обойтись без него.

Смотри **также**

Описание переменной **$;** в *perlvar*(1);описание функции stat в *perlfunc*(1).

9.5. Обработка всех файлов каталога

Проблема

Требуется выполнить некоторые действия с каждым файлом данного каталога.

Решение

Откройте каталог функцией opendir и последовательно читайте имена файлов функцией readdir:

```
opendir(DIR, $dirname) or die "can't opendir $dirname: $!";
while                    ddir(DIR))) {
    # Сделать что-то с "$dirname/$file"
}
closedir(DIR);
```

Комментарий

Функции opendir, closedir работают с каталогами по аналогии с функциями open, close, работающими с файлами. В обоих случаях используются манипуляторы, однако манипуляторы каталогов, используемые opendir и другими функциями этого семейства, отличаются от файловых манипуляторов функции ppen и других. В частности, для манипулятора каталога нельзя использовать оператор <>.

В скалярном контексте возвращает следующее имя файла в каталоге, пока не будет достигнут конец каталога — в этом случае возвращается undef. В списковом контексте возвращаются остальные имена файлов каталога или пустой список, если файлов больше нет. Как объяснялось во введении, имена файлов, возвраща- содержат имя каталога. При работе с именами, полученными от необходимо либо заранее перейти в нужный каталог, либо вручную присоединить его к имени.

Ручное присоединение может выглядеть так:

```
$dir = "/usr/local/bin";
print "Text files in $dir are:\n";
opendir(BIN, $dir) or die "Can't open $dir: $!";
while( defined ($file = readdir BIN) ) {
    print "$file\n" if -T "$dir/$file";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
closedir(BIN);
```

Мы проверяем $file с помощью defined, поскольку простое условие while ($file BIN) проверяет истинность, а не определенность. Хотя наш цикл завершается после перебора всех файлов, возвращаемых readdir, он также завершится преждевременно при наличии файла с именем "0".

Функция также возвращает специальные каталоги "." (текущий каталог) и ".." (родительский каталог). Обычно они пропускаются фрагментом следующего вида:

```
while ( defined ($file =
    next if $file =" /^\.\.?$/;     # Пропустить . и ..
    # ...
}
```

Манипуляторы каталогов, как и файловые манипуляторы, существуют на уровне пакетов. Более того, локальный манипулятор каталога можно получить двумя способами: с помощью local *DIRHANDLE или модуля (см. рецепт 7.16). В данном случае нужен модуль DirHandle. Следующий фрагмент использует DirHandle для получения отсортированного списка обычных файлов, которые не являются "скрытыми" (имена которых не начинаются с "."):

```
use DirHandle;

sub plainfiles {
    my $dir = shift;
    my $dh = DirHandle->new($dir)    or die "can't opendir $dir: $!";
                                    #                    имена
                                     # Выбрать "обычные" файлы
            map  { "$dir/$_" }       # Построить полные пути
                 {  !/^\./   }       # Отфильтровать скрытые файлы
            $dh->read();             # Прочитать все элементы
}
```

Метод read модуля DirHandle работает так же, как и ащает остальные имена файлов. Нижний вызов оставляет лишь те имена, которые не начинаются с точки. Вызов map преобразует имена файлов, полученные от полные, а верхний вызов q'er отфильтровывает каталоги, ссылки и т. д. Полученный список сортируется и возвращается.

В дополнение к также существуют функции rewinddir (перемещает манипулятор каталога к началу списка файлов), seekdir (переходит к конкретному смещению в списке) и telldir (определяет смещение от начала списка).

Смотри также

Описание функций closedir, opendir, readdi seekdir и telldir в perlf/wnc(1); документация по стандартному модулю DirHandle.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9.6. Получение списка файлов по шаблону

Проблема

Требуется получить список файлов по **шаблону**, аналогичному конструкциям `*.*` (MS-DOS) и `*.h` (UNIX).

Решение

Семантика командного интерпретатора C shell системы UNIX поддерживается в Perl с помощью ключевого слова glob и оператора <>:

```
@list = <*.c>;
@list = glob("*.c");
```

Для ручного извлечения имен файлов можно воспользоваться функцией

```
opendir(OIR, $path);
@files          /\.c$/ }
closedir(DIR);
```

Модуль File::KGlob от CPAN получает список файлов без ограничений длины:

```
use File::KGlob;

@files = glob("*.c");
```

Комментарий

Встроенная функция Perl glob и запись <ШАБЛОН> (не путать с записью <МАНИПУЛЯТОР>!) в настоящее время на большинстве платформ используют внешнюю программу для получения спискафайлов. В UNIX это программа ***csh***[1], а в Windows — ***dosglob.exe***. На Macintosh и в VMS это реализуется на внутреннем уровне, без внешних программ. Предполагается, что шаблоны обеспечивают семантику C shell во всех системах, отличных от UNIX, и улучшают переносимость. Из-за использования интерпретатора в UNIX такое решение не подходит для сценариев с атрибутом setuid.

Чтобы справиться с затруднениями, можно реализовать собственный механизм отбора с применением встроенного оператора opendir илимодуля File::KGlob от CPAN — в обоих случаях внешние программы не используются. File::KGlob обеспечивает семантику отбора по типу интерпретаторов UNIX, тогда как opendir позволяет отбирать файлы с помощью регулярных выражений Perl.

В простейшем решении с opendir список, возвращаемый `readdir`, фильтруется с помощью

```
@files          /\.[ch]$/i   readdir(DH);
```

[1] Обычно при наличии установленного интерпретатора tcsh Perl использует его, поскольку он надежнее. Если не установлен ни один из этих интерпретаторов, используется *ft>\n/sh>*.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

То же самое можно сделать и с помощью модуля DirHandle:

```
use DirHandle;

$dh = DirHandle->new($path)    or die "Can't open $path  : $!\n";
@files          /\.[ch]$/i
```

Как обычно, возвращаемые имена файлов не содержат каталога. При использовании имени каталог приходится присоединять вручную:

```
opendir(DH, $dir)          or die "Couldn't open $dir for            $!";

@files = ();
while( defined ($file =                ) {
    next unless /\.[ch]$/i;

    my $filename ="$dir/$file";
    push(@files, $filename) if -T $fle;
```

В следующем прийере чтение каталога ифильтрация для повышения эффективности объединяются с преобразованием Шварца (см. главу 4 «Массивы»). В массив `@dirs` заносится отсортированный список подкаталогов, имена которых представляют собой числа:

```
@dirs = map  { $_->[1] }                    # Извлечение имен
        sort { $a->[0] <=> $b->[0] }        # Числовая сортировка имен
                $_->[1] }                   # Каталоги
        map  { [ $_, "$path/$_" ] }         # Сформировать (имя, путь)
             /^\d+$/                        # Только числа
                                                файлы
```

В рецепте 4.14 показано, как читать подобные странные конструкции. Как обычно, форматирование и документирование кода заметно упрощает его чтение и понимание.

Смотри также

Описание функций closedir, opendir, seekdir и telldir в *perlfunc*(1);документация по стандартному модулю DirHandle; раздел "I/O Operators" *perlop*(1); рецепты 6.9; 9.7.

9.7. Рекурсивная обработка всех файлов каталога

Проблема

Требуется выполнить некоторую операцию с каждым файлом и подкаталогом некоторого каталога.

Решение

Воспользуйтесь стандартным модулем File::Find.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use File Find,
sub process_file {
    Я Делаем то, что хотели
}
find(\&process_file, @DIRLIST),
```

Комментарий

Модуль File::Find обеспечивает удобные средства рекурсивной обработки файлов. Просмотр каталога и рекурсия организуются без вашего участия. Достаточно передать find ссылку на функцию и список каталогов. Для каждого файла в этих каталогах find вызовет заданную функцию.

Перед вызовом функции find переходит в указанный каталог, имя которого по отношению к начальному каталогу хранится в переменной $File • Find • dir. Переменной $_ присваивается базовое имя файла, а полный путь к этому файлу находится в переменной $File Find name. Ваша программа может присвоить $File Find prune истинное значение, чтобы функция find не спускалась в только что просмотренный каталог.

Использование File::Find демонстрируется следующим простым примером. Мы передаемfind анонимнуюподпрограмму, которая выводит имя каждого обнаруженного файла и добавляет к именам каталогов /:

```
@ARGV = qw( ) unless @ARGV,
use File Find,
find sub { print $File Find name, -d && / , \n }, @ARGV,
```

Для вывода / после имен каталогов используется оператор проверки -d, который при отрицательном результате возвращает пустую строку ''.

Следующая программа выводит суммарный размер всего содержимого каталога. Она передает find анонимную подпрограмму для накопления текущей суммы всех рассмотренных ей файлов. Сюда входят не только обычные файлы, но и все типы индексных узлов, включая размеры каталогов и символических ссылок. После выхода из функции find программа выводит накопленную сумму.

```
use File Find,
@ARGV = (' ') unless @ARGV,
my $sum = 0,
find sub { $sum += -s }, @ARGV,
print @ARGV contains $sum bytes\n ,
```

Следующий фрагмент ищет самый большой файл в нескольких каталогах:

```
use File Find,
@ARGV = ( ) unless @ARGV,
my ($saved_size, $saved_name) = (-1, ''),
sub biggest {
        unless -f && -s _ > $saved_size,
    $saved_size = -s _,
    $saved_name = $File Find name,
}
find(\&biggest, @ARGV),
print Biggest file $saved_name in @ARGV is $saved_size bytes long \n ,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Переменные $saved_size и $saved_name используются для хранения имени и размера самого большого файла. Если мы находим файл, размер которого превышает размер самого большого из просмотренного до настоящего момента, сохраненное имя и размер заменяются новыми значениями. После завершения работы find выводится имя и размер самого большого файла в весьма подробном виде. Вероятно, более практичная программа ограничится выводом имени файла, его размера или и того и другого. На этот раз мы воспользовались именованной функцией вместо анонимной, поскольку она получилась относительно большой.

Программу нетрудно изменить так, чтобы она находила файл, который изменялся последним:

```
use File::Find;
@ARGV = ('.') unless @ARGV;
my ($age, $name);
sub youngest {
                defined $age && $age > -M;
    $age = (stat(_))[9];
    $name = $File::Find::name;
}
find(\&youngest, @ARGV);
print "$name " . scalar(localtime($age)) . "\n";
```

Модуль File::Find не экспортирует имя переменной $name, поэтому на нее следует ссылаться по полному имени. Пример 9.2 демонстрирует скорее работу с пространствами имен, нежели рекурсивный перебор в каталогах. Он делает переменную $name текущего пакета синонимом переменной File::Find (в сущности, именно на этом основана работа модуля Exporter). Затем мы объявляем собственную версию find с прототипом, обеспечивающим более удобный вызов.

Пример 9.2. fdirs

```
#!/usr/bin/perl -lw
# fdirs - поиск всех каталогов
@ARGV = qw(.) unless @ARGV;
use File::Find ();
sub find(&@) { &File::Find::find >
*name = *File::Find::name;
find { print $name if -d } @ARGV;
```

Наша версия find вызывает File::Find, импортирование которой предотвращается включением пустого списка () в команду use. Вместо записи вида:

```
find sub { print $File::Find::name if -d >, @ARGV;
```

можно написать более приятное

```
find { print $name if -d } @ARGV;
```

Смотри также

Man-страница *find*(1); рецепт 9.6; документация по стандартным модулям File::Find и Exporter.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9.8. Удаление каталога вместе с содержимым

Проблема

Требуется рекурсивно удалить ветвь дерева каталога без применения *rm -r*.

Решение

Воспользуйтесь функцией `finddepth` модуля File::Find (см. пример 9.3).

Пример 9.3.

```
#!/usr/bin/perl
9 rmtree1 - удаление ветви дерева каталогов (по аналогии с  rm -r)
use  File::Find  qw(finddepth);
die "usage: $0 dir ..\n" unless @ARGV;
*name = *File::Find::name;
finddepth \&zap,  @ARGV;
sub zap {
    if (!-l && -d _) {                                          /
        print "rmdir $name\n";
        rmdir($name)  or warn  "couldn't rmdir $name: $!";
    } else {
        print "unlink $name";
        unlink($name) or warn "couldn't unlink $name: $!";
    }
}
```

Или воспользуйтесь функцией `rmtree` модуля File::Path (см. пример 9.4).

Пример 9

```
#!/usr/bin/perl
          - удаление ветви дерева каталогов (по аналогии с  rm -r)
use File::Path;
die "usage: $0 dir ..\n" unless @ARGV;
            $dir (@ARGV) {
    rmtree($dir);
}
```

> **Предупреждение**

Эти программы удаляют целые ветви дерева каталогов. Применяйте крайне осторожно!

Комментарий

Модуль **File::Find** экспортирует функцию `find`, которая перебирает содержимое каталога практически в случайном порядке следования файлов, и функцию `finddepth`, гарантирующую перебор всех внутреннихфайлов перед посещением самого каталога. Именно этот вариант поведения использован нами для удаления каталога вместе с содержимым.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

У нас есть две функции, rmdir и unlink. Функция unlink удаляет только файлы, а rmdir —только пустые каталоги. Мы должны использовать finddepth, чтобы содержимое каталога заведомо удалялось раньше самого каталога.

Перед тем как проверять, является ли файл каталогом, необходимо узнать, не является ли он символической ссылкой. -d возвращает true и для каталога, и для символической ссылки на каталог. Функции stat, lstat и операторы проверки (типа -d) используют системную функцию stat(2), которая возвращает всю информацию о файле, хранящуюся в индексном узле. Эти функции и операторы сохраняют полученную информацию и позволяют выполнить дополнительные проверки того же файла с помощью специального манипулятора _. При этом удается избежать лишних вызовов системных функций, возвращающих старую информацию и замедляющих работу программы.

Смотри также

Описание функций unlink, rmdir, lstat и stat в *perlfunc*(1); документация по стандартному модулю File::Find; man-страницы *rm*(1) и *stat*(2); раздел *perlfunc(l),* посвященный операторам -X.

9.9. Переименование файлов

Проблема

Требуется переименовать файлы, входящие в некое множество.

Решение

Воспользуйтесь циклом функцией rename:

```
foreach $file (@NAMES) {
    my $newname = $file;
    # change $file
    rename($file, $newname) or
        warn "Couldn't rename $file to $newname: $!\n";
}
```

Комментарий

Программа вполне тривиальна. Функция rename получает два аргумента — старое и новое имя. Функция предоставляет интерфейс к системной функции переименования, которая обычно позволяет переименовывать файлы только в том случае, если старое и новое имена находятся в одной файловой системе.

После небольших изменений программа превращается в универсальный сценарий переименования вроде написанного Ларри Уоллом (см. пример 9.5).

```
#!/usr/bin/perl -w
        переименование файлов от Ларри
$op = shift or die "Usage: expr [files]\n";
chomp(@ARGV = <STDIN>) unless @ARGV;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
for (@ARGV) {
    $was = $_;
    eval $op;
    die $@ if $@;
    rename($was,$_) unless $was eq $ ;
}
```

Первый аргумент сценария — код Perl, который изменяет имя файла, хранящееся в $_, и определяет алгоритм переименования. Вся черная работа поручается функции eval. Кроме того, сценарий пропускает вызов rename в том случае, если имя осталось прежним. Этопозволяет просто использовать универсальные символы (rename EXPR *) вместо составления длинных списков имен.

Приведем п'ять примеров вызова программы командного интерпретатора:

```
        's/\.orıg$//'  *.orig
        'tr/A-Z/a-z/ unless /^Make/'  *
              ".bad"'
        'prınt "$_: "; s/foo/bar/ if <STDIN> =^ /^y/i'  *
% fınd /tmp -name '*^' -print          's/^(.+)~$/.#$1/'
```

Первая команда удаляет из имен файлов суффикс *.orig*.

Вторая команда преобразует символы верхнего регистра в символы нижнего регистра. Поскольку вместо функции **lc** используется прямая трансляция, такое преобразование не учитывает локальный контекст. Проблема решается следующим образом:

```
        'use locale; $_ = lc($_) unless/^Make/'  *
```

Третья команда добавляет суффикс ***.bad*** к каждому файлу Fortran с суффиксом ".f" — давняя мечта многих программистов.

Четвертая команда переименовывает файлы в диалоге с пользователем. Имя каждого файла отправляется на стандартный вывод, а из стандартного ввода читается ответ. Если пользователь вводит строку, начинающуюся с **"y"** или "Y", то все экземпляры "foo" в имени файла заменяются на "bar".

Пятая команда с помощью find ищет в */tmp* файлы, имена которых заканчиваются тильдой. Файлы переименовываются так, чтобы они начинались с префикса .#. В сущности, мы переключаемся между двумя распространенными конвенциями выбора имен файлов, содержащих резервные копии.

воплощена вся мощь философии UNIX, основанной на утилитах и фильтрах. Конечно, можно написать специальную команду для преобразования символов в нижний регистр, однако ничуть не сложнее написать гибкую, универсальную утилиту с внутренним eval. Позволяя читать имена файлов из стандартного ввода, мы избавляемся от необходимости рекурсивного перебора каталога. Вместо этого мы используем функцию find, которая прекрасно справляется с этой задачей. Не стоит изобретать колесо, хотя модуль File::Find позволяет это сделать.

Смотри также

Описание функции в *perlfunc*(1); страницы руководства *mv*(1) и *rename*(2); документация по стандартному модулю File::Find.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

9.10. Деление имени файла на компоненты

Проблема

Имеется строка, содержащая полное имя файла. Из нее необходимо извлечь компоненты (имя, каталог, расширение (-я)).

Решение

Воспользуйтесь функциями стандартного модуля File::Basename.

```
use File::Basename;

$base = basename($path);
$dir  = dirname($path);
($base, $dir, $ext) = fileparse($path);
```

Комментарий

Функции деления имени файла присутствуют в стандартном модуле File::Basename. Функции dirname и basename возвращают соответственно каталог и имя файла:

```
$path = '/usr/lib/libc.a';
$file = basename($path);
$dir  = dirname($path);
        t
print "dir is $dir, file is $file\n";
# dir is /usr/lib,  file is libc.a
```

Функция fileparse может использоваться для извлечения расширений. Для этого передайте fileparse полное имя и регулярное выражение для поиска расширения. Шаблон необходим из-за того, что расширения не всегда отделяются точкой. Например, что считать расширением в ".tar.gz" — ".tar", ".gz" или ".tar.gz"? Передавая шаблон, вы определяете, какой из перечисленных вариантов будет возвращен:

```
$path = '/usr/lib/libc.a';
($name,$dir,$ext) = fileparse($path,'\..*');

print "dir is $dir, name is $name, extension is $ext\n";
# dir is /usr/lib/, name is libc, extension is .a
```

По умолчанию в работе этих функций используются разделитель, определяемый стандартными правилами вашей операционной системы. Для этого используется переменная $^0; содержащаяся в ней строка идентифицирует текущую систему. Ее значение определяется в момент построения и установки Perl. Значение по умолчанию можно установить с помощью функции fileparse_set_fstype. В результате изменится и поведение функций File::Basename при последующих вызовах:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
fileparse_set_fstype("MacOS");
$path = "Hard%20Drive:System%20Folder:README.txt";
($name,$dir,$ext) = fileparse($path,'\..*');

print "dir is $dir, name is $name, extension.is $ext\n";
* dir is Hard%20Drive:System%20Folder, name is README, extension is .txt
```

Расширение можно получить следующим образом:

```
sub extension {
    my $path = shift;
    my $ext = (fileparse($path,'\..*'))[2];
    $ext =^ s/^\.//;
          $ext;
}
```

Для файла ***source.c.bak*** вместо простого "bak" будет возвращено расширение "c. bak". Если **вы** хотите получить именно "bak", **в** качестве второго аргумента fileparse используйте **'\..*?'**.

Если передаваемое полное имя заканчивается разделителем каталогов (например, lib/), fileparse считает, что имя каталога равно "lib/", тогда как dirname считает его равным ".".

Смотри также

Описание переменной **$^0** *в perlvar(1);* документация по стандартному модулю File::Basename.

9.11. Программа: symirror

Программа из примера **9.6** рекурсивно воспроизводит каталог со всем содержимым и создает множество символических ссылок, указывающих на исходные файлы.

Пример 9.6. symirror

```
#!/usr/bin/perl -w
" symirror - дублирование каталога с помощью символических ссылок
use strict;
use File::Find;
use Cwd;

my ($srcdir, $dstdir);
my $cwd = getcwd();
die "usage: $0          mirrordir"

for (($srcdir, $dstdir) = @ARGV) {
    my $is_dir = -d;
    next if $is_dir;                        # Нормально
    if (defined ($is_dir)) {
        die "$0: $_ is not a directory\n";
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 9.6 (продолжение)

```
    } else {                                    # Создать каталог
        mkdir($dstdir, 07777) or die "can't mkdir $dstdir: $!";
    }
} continue {
    s#^(?!/)#$cwd/#;                # Исправить относительные пути
}

chdir $srcdir;
flnd(\&wanted, '.');

sub wanted {
    my($dev, $ino, $mode) = lstat($_);
    my $name = $File::Find::name;
    $mode &= 07777;                     # Сохранить права доступа
    $name =~ s!^\./!!;                # Правильное имя
    if (-d _) {                         # Затем создать каталог
        mkdir("$dstdir/$name", $mode)
            or die "can't mkdir $dstdir/$name: $!";
    } else {                            # Продублировать все остальное
        symlink("$srcdir/$name", "$dstdir/$name")
            or die "can't symlink $srcdir/$name to $dstdir/$name: $!'
    }
}
```

9.12. Программа: lst

Вам не приходилось отбирать из каталога самые большие или созданные последними файлы? В стандартной программе *ls* предусмотрены параметры для сортировки содержимого каталогов по времени (флаг -t) и для рекурсивного просмотра подкаталогов (флаг -R). Однако *ls* делает паузу для каждого каталога и выводит только его содержимое. Программа не просматривает все подкаталоги, чтобы потом отсортировать найденные файлы.

Следующая программа *lst* справляется с этой задачей. Ниже показан пример подробного вывода, полученного с использованием флага -l:

```
% lst -l /etc
12695 0600     1     root     wheel      512 Fri May 29 10:42:41 1998
      /etc/ssh_random_seed
12640 0644     1     root     wheel    10104 Mon Ma y 25  7:39:19 1998
      /etc/ld.so.cache
12626 0664     1     root     wheel    12288 Sun May 24 19:23:08 1998
      /etc/psdevtab
12304 0644     1     root      root      237 Sun May 24  13:59:33 1998
      /etc/exports
12309 0644     1     root      root     3386 Sun May 24 13:24:33 1998
      /etc/inetd.conf
12399 0644     1     root      root    30205 Sun Hay 24 10:08:37 1998
      /etc/sendmail.cf
18774 0644     1    gnat   perldoo     2199 Sun May 24   9:35:57 1998
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    /etc/X11/XMetroconfig
12636 0644      1      root    wheel      290 Sun May 24   9:05:40 1998
    /etc/mtab
12627 0640      1      root     root        0 Sun May 24   8:24:31 1998
    /etc/wtmplock
12310 0644      1      root  tchrist       65 Sun May 24   8:23:04 1998
    /etc/issue
```

Файл */etc/X11/XMetroconfig* оказался посреди содержимого */etc,* поскольку листинг относится не только к */etc*, но и ко всему, что находится внутри каталога.

К числу поддерживаемых параметров также относится сортировка по времени последнего чтения вместо записи (-u) и сортировка по размеру вместо времени (-s). Флаг -i приводит к получению списка имен из стандартного ввода вместо рекурсивного просмотра каталога функцией find. Если у вас уже есть готовый список имен, его можно передать *lst* для сортировки.

Исходный текст программы приведен в примере 9.7.

Пример 9.7. lst

```
#!/usr/bin/perl
# 1st - вывод отсортированного содержимого каталогов

use Getopt::Std;
use File::Find;
use File::stat;
use User::pwent;
use User::grent;

getopts('lusrcmi')          or die "DEATH;
Usage: $0 [-mucsril] [dirs ...]
 or    $0 -1 [-mucsrl] < filelist

Input format:
    -1  read pathnames from stdin
Output format:
    -1  long listing
Sort on:
    -m  use mtime (modify time) [DEFAULT]
    -u  use atime (access time)
    -c  use ctime (inode change time)
    -s  use size for  sorting
Ordering:

NB: You may only use select one  sorting option at a time.
DEATH

unless ($opt_i || @ARGV) { @ARGV = ('.') }

if ($Opt_C + $opt_u + $Opt_s + $opt_m > 1)
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 9.7 (продолжение)

```
    die "can only sort on one time or size";
}

$IDX = 'mtime';
$IDX = 'atime' if $opt_u;
$IDX = 'ctime' if $opt_c;
$IDX = 'size'  if $opt_s;

$TIME_IDX = $opt_s ? 'mtime' : $IDX;

"name = *File::Find::name;  # Принудительное импортирование переменной

# Флаг $opt_i заставляет wanted брать имена файлов
# из ARGV вместо получения от find.

if ($opt_i) {
     *name = *_;  # $name теперь является синонимом $_
     while (<>) { chomp; &wanted; }   # Все нормально, это не stdin
}  else {
    find(\&wanted, @ARGV);
}

# Отсортировать файлы по кэшированным значениям времени,
# начиная с самых новых.
@skeys = sort { $time{$b} <=> $time{$a} } keys %time;

# Изменить порядок, если в командной строке был указан флаг -r
@skeys          @skeys    $opt_r;

for (@skeys) {
    unless ($opt_l) {  8 Эмулировать ls -1, кроме прав доступа
        print "$_\n";
        next;
    }
    $now = localtime $stat{$_}->$TIME_IDX();
    printf "%6d %O4o %6d %8s %8s %8d %s %s\n",
  $stat{$_}->ino(),
  $stat{$_}->mode() & 07777,
  $stat{$_}->nlink(),
  user($stat{$_}->uid()),
  group($stat{$_}->gid()),
  $stat{$_}->size(),
  $now, $ ;
}

8 Получить от stat информацию о файле, сохраняя критерий
" сортировки (mtime, atime, ctime или size)
tf в хэше %time, индексируемом по имени файла.
8 Если нужен длинный список, весь объект stat приходится
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# сохранять в %stat. Да, это действительно хэш объектов.
sub wanted {
    my $sb = stat($_);  # XXX: stat или lstat?
           unless $sb;
    $time{$name} = $sb->$IDX();  # Косвенный вызов метода
    $stat{$name} = $sb if $opt 1;
}

# Кэширование преобразований идентификатора пользователя в имя
sub user {
    my $uid = shift;
    $user{$uid} = getpwuid($uid)->name || "#$uid"
        unless defined $user{$uid};
           $user{$uid};
}

# Кэширование преобразований номера группы в имя
sub group {
    my $gid = shift;
    $group{$gid} = getgrgid($gid)->name || "#$gid"
        unless defined $group{$gid};
           $group{$gid};
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

10 Подпрограммы

Огнем бессмертным наполняя смертных...
В. Оден, "Три песни ко Днюсвятой Сесилии"

Введение

Практика вставки/копирования кода довольно опасна, поэтому в больших программах многократно используемые фрагменты кода часто оформляются в виде подпрограмм. Для нас термины "подпрограмма" (subroutine) и "функция" (function) будут эквивалентными, поскольку в Perl они различаются ничуть не больше, чем в С. Даже объектно-ориентированные методы представляют собой обычные подпрограммы со специальнымсинтаксисом вызова, описанным в главе 13 "Классы, объекты и связи".

Подпрограмма объявляется с помощью ключевого слова sub. Пример определения простой подпрограммы выглядит так:

```
sub hello {
                                Глобальная переменная
    print "hi there\n!";
}
```

Типичный способ вызова этой подпрограммы выглядит следующим образом:

```
hello();     # Подпрограмма hello вызывается без аргументов/параметров
```

Перед выполнением программы Perl компилирует ее, поэтому место объявления подпрограммы не имеет значения. Определения не обязаны находиться в одном файле с основной программой. Они могут быть взяты из других файлов с помощью операторов do, (см. главу 12 "Пакеты, библиотеки и модули"), создаваться "на месте" с помощью ключевого слова eval или механизма AUTOLOAD или генерироваться посредством замыканий, используемых в шаблонах функций.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если вы знакомы с другими языками программирования, некоторые особенности функций Perl могут показаться странными. В большинстве рецептов этой главы показано, как применять эти особенности в свою пользу.

- Функции Perl не имеют формальных, именованных параметров, но это не всегда плохо (см. рецепты 10.1 и 10.7).
- Все переменные являются глобальными, если обратное не следует из объявления. Дополнительная информация приведена в рецептах 10.1 и 10.7.
- Передача или возвращение нескольких массивов или хэшей обычно приводит к потере ими "индивидуальности". О том, как избежать этого, рассказано в рецептах 10.5, 10.8, 10.9 и 10.11.
- Функция может узнать свой контекст вызова (списковый или скалярный), количество аргументов при вызове и даже имя функции, из которой она была вызвана. О том, как это сделать, рассказано в рецептах 10.4 и 10.6.
- Используемое в Perl значение `undef` может использоваться в качестве признака ошибки, поскольку ни одна допустимая строка или число никогда не принимает это значение. В рецепте 10.10 описаны некоторые неочевидные трудности, связанные с `undef`, которых следует избегать, а в рецепте 10.12 показано, как обрабатываются другие катастрофические случаи.
- В Perl функции обладают рядом интересных возможностей, редко встречающихся в других языках (например, анонимные функции, создание функций "на месте" и их косвенный вызов через указатель на функцию). Эти мистические темы рассматриваются в рецептах 10.14 и 10.16.

При вызове вида `$x = &func;` функция не получает аргументов, но зато может напрямую обращаться к массиву `@_` вызывающей стороны! Если убрать амперсанд и воспользоваться формой `func()` или `func`, создается новый, пустой экземпляр массива `@_`.

10.1. Доступ к аргументам подпрограммы

Проблема

В своей функции вы хотите использовать аргументы, переданные вызывающей стороной.

Решение

Все значения, переданные функции в качестве аргументов, хранятся в специальном массиве `@_`. Следовательно, первый аргумент хранится в элементе `$_[0]`, второй — в `$_[1]` и т. д. Общее число аргументов равно `scalar(@_)`.

Например:

```
sub hypotenuse {
          sqrt( ($_[0] .* 2) + ($_[1] *• 2) );
}

$diag = hypotenuse(3,4);  # $diag = 5
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В начале подпрограммы аргументы почти всегда копируются в именованные закрытые переменные для удобства и повышения надежности:

```
sub hypotenuse {
    my ($side1, $side2) = @_;
            sqrt( ($side1 ** 2) + ($side1 ** 2) );
}
```

Комментарий

Говорят, в программировании есть всего три удобных числа: ноль, единица и "сколько угодно". Механизм работы с подпрограммами Perl разрабатывался для упрощения написания функций со сколь угодно большим (или малым) числом параметров и возвращаемых значений. Все входные параметры хранятся в виде отдельных скалярных значений в специальном массиве @_, который автоматически становится локальным для каждой функции (см. рецепт 10.13). Для возв ращения значений из подпрограмм следует использовать команду аргументом. Если она отсутствует, возвращаемое значение представляет собой результат последнего вычисленного выражения.

Приведем несколько примеров вызова функции hypotenuse, определенной в решении:

```
print hypotenuse(3, 4), "\n";                # Выводит 5

@a = (3,·4):
print hypotenuse(@a), "\n";                  # Выводит 5
```

Если взглянуть на аргументы, использованные во втором вызове hypotenuse, может показаться, что мы передали лишь один аргумент — массив @a. Но это не так — элементы @a копируются в массив @_ по отдельности. Аналогично, при вызове функции с аргументами (@a, @b) мы передаем ей все аргументы из обоих массивов. При этом используется тот же принцип, что и при сглаживании списков:

```
@both = (@men, @women);
```

Скалярные величины в @_ представляют собой неявные синонимы для передаваемых значений, а не их копии. Таким образом, модификация элементов @_ в подпрограмме приведет к изменению значений на вызывающей стороне. Это тяжкое наследие пришло из тех времен, когда в Perl еще не было нормальных ссылок.

Итак, функцию можно записать так, чтобы она не изменяла свои аргументы — для этого следует скопировать их в закрытые переменные:

```
@nums = (1.4, 3.5, 6.7);
@ints = int_all(@nums);    ft    @nums не изменяется
sub int_all {
    my @retlist = @_;            # Сделать копию для
    for my $n (@retlist) { $n = int($n) }
            @retlist;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Впрочем, функция также может изменять значения переменных вызывающей стороны:

```
@nums = (1.4, 3.5, 6.7);
trunc_em(@nums);                " @nums = (1.3,6)
sub trunc_em {
    for (9_) { $_ = int($_) }  8 Округлить каждый аргумент
}
```

Таким функциям не следует передавать константы — н априм ер, `trunc_em(1.4, 3.5, 6.7)`. Если попытаться это сдел ать, будет возбуждено исключение Modification value attempted at... («Попытка модифицировать величину, доступную только для чтения»).

Встроенные функции `chop` и `chomp` работают именно так — они модифицируют переменные вызывающей стороны и возвращают удаленный символ(-ы). Многие привыкают к тому, что функции возвращают измененные значения, и частопишут в программах следующее:

```
$line = chomp(<>);                  # НЕВЕРНО
```

пока не поймут, что происходит в действительности. Учитывая широкие возможности для ошибок, перед модификацией `@_` в подпрограмме стоит дважды подумать.

Смотри также
perlsub(1).

10.2. Создание закрытых переменных в функциях

Проблема

В подпрограмме потребовалось создать временную переменную. Использование глобальных переменных нежелательно, поскольку другие подпрограммы могут получить к ним доступ.

Решение

Воспользуйтесь ключевым словом `my` для объявления переменной, ограниченной некоторой областью программы:

```
sub somefunc {
    my $variable;                  " Переменная $variable невидима
                                   № за пределами somefunc()
     my ($another, @an_array, %a_hash);     " Объявляем несколько
                                            tt переменных сразу
    * ...
›
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Оператор my ограничивает использование переменной и обращение к ней определенным участком программы. За пределами этого участка переменная недоступна. Такой участок называется *областью действия* (scope).

Переменные, объявленные с ключевым словом my, обладают *лексической областью действия* — это означает, что они существуют лишь в границах некоторого фрагмента исходного текста. Например, областью действия переменной $variable из решения является функция somefunc, в которой она была определена. Переменная создается при вызове somefunc и уничтожается при ее завершении. Переменная доступна внутри функции, но не за еепределами.

Лексическая область действия обычно представляет собой программный блок, заключенный в фигурные скобки, — например, определение тела подпрограммы somefunc или границы команд if, while, for, eval. Лексическая область действия также может представлять собой весь файл или строку, переданную eval. Поскольку лексическая область действия обычно является блоком, иногда мы говорим, что лексические переменные (переменные с лексической областью действия) видны только в своем блоке — имеется в виду, что они видны только в границах своей области действия. Простите нам эту неточность, иначе слова "область действия" и "подпрограмма" заняли бы половину этой книги.

Поскольку фрагменты программы, в которых видна переменная my, определяются во время компиляции и не изменяются позднее, лексическая область действия иногда не совсем точно называется "статической областью действия". Ее противоположностью является *динамическая* область действия, рассмотренная в рецепте 10.13.

Объявление my может сочетаться с присваиванием. При определении сразу нескольких переменных используются круглые скобки:

```
my ($name, $age) = @ARGV;
my $start        = fetch_time();
```

Эти лексические переменные ведут себя как обычные локальные переменные. Вложенный блок видит лексические переменные, объявленные в родительских по отношению к нему блоках, но не в других, не связанных с ними блоках:

```
my ($a, $b) = @pair;
my $c = fetch_time();

sub check_x {
    my $x = $_[0];
    my $y = "whatever";
    run_check();
    if ($condition) {
        print "got $x\n";
    }
}
```

В приведенном выше фрагменте блок if внутри функции может обращаться к закрытой переменной $x. Однако в функции run_check, вызванной из этой области, переменные $x и $y недоступны, потому что она предположительно определяется в другой области действия. Однако check_x может обращаться к $a, $b и $c из

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

внешней области, поскольку определяется в одной области действия с этими переменными.

Именованные подпрограммы не следует объявлять внутри объявлений других именованных подпрограмм. Такие подпрограммы, в отличие от полноценных замыканий, не обеспечивают правильной привязки лексических переменных. В рецепте **10.16** показано, как справиться с этим ограничением.

При выходе лексической переменной за пределы области действия занимаемая ей память освобождается, если на нее не существует ссылок, как для массива `@arguments` в следующем фрагменте:

```
sub save_array {
    my @arguments = @_;
    push(@Global_Array, \@arguments);
}
```

Система сборки мусора **Perl** знает о том, что память следует освобождать лишь для неиспользуемых объектов. Это и позволяет избежать утечки памяти при возвращении ссылки на закрытую переменную.

Смотри также

Раздел "Private Variables via my()» *perlsub*(1).

10.3. Создание устойчивых закрытых переменных

Проблема

Вы хотите, чтобы переменная сохраняла значение между вызовами подпрограммы, но не была доступна за ее пределами. Например, функция может запоминать, сколько раз она былавызвана.

Решение

"Заверните" функцию во внешний блок **и** объявите переменные `my` **в** области действия этого блока, а не **в** самой функции:

```
{
    my $variable;
    sub mysub {
        # ... обращение к $variable
    }
}
```

Если переменные требуют инициализации, снабдите блок ключевым словом `BEGIN`, чтобы значение переменных заведомо задавалось перед началом работы основной программы:

```
BEGIN {
    my $variable = 1;                    # Начальное значение
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    sub othersub {                          # ... обращение к $variable
    }
}
```

Комментарий

В отличие от локальных переменных в языках C и C++, **лексические** переменные Perl не всегда уничтожаются при выходе из области действия. Если нечто, продолжающее существовать, все еще помнит о лексической переменной, память не освобождается. В нашем примере mysub использует переменную $variable, поэтому Perl не освобождает память переменной при завершении блока, вмещающего определение mysub.

Счетчик вызовов реализуется следующим образом:

```
{
    my $counter;
    sub next_counter {
}
```

При каждом вызове next_counter функция увеличивает и возвращает переменную $pounter. При первом вызове переменная $counter имеет неопределенное значение, поэтому для оператора ++ она интерпретируется как 0. Переменная входит не в область действия next_counter, а в окружающий ее блок. Никакой внешний код не сможет изменить $counter без вызова next_counter.

Для расширения области действия обычно следует использовать ключевое слово BEGIN. В противном случае возможен вызов функции до инициализации переменной.

```
BEGIN {
    my $counter = 42;
    sub next_counter {
                                --$counter }
}
```

Таким образом, в Perl создается аналог статических переменных языка C. В действительности он даже лучше — переменная не ограничивается одной функцией, и обе функции могут совместно использовать свою закрытую переменную.

Смотри также

Раздел "Private Variables via my()» *perlsub*(1); раздел "Package Constructors and Destructors» *perlmod*(1); рецепт **11.4.**

10.4. Определение имени текущей функции

Проблема

Требуется определить имя функции, работающей в настоящий момент. Оно пригодится для сообщений об ошибках, которые не изменяются при копировании/вставке исходного текста подпрограммы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь функцией `caller`:

```
$this_function = (caller(0))[3];
```

Комментарий

Программа всегда может определить текущей номер строки с помощью специальной метапеременной `__LINE__`. Текущий файл определяется с помощью метапеременной `__FILE__`, а текущий пакет — `__PACKAGE__`. Однако не существует метапеременной для определения имени текущей подпрограммы, не говоря уже об имени той, из которой она была вызвана.

Встроенная функция ca.ller справляется со всеми затруднениями. В скалярном контексте она возвращает имя пакета вызывающей функции, а в списковом контексте возвращается список с разнообразными сведениями. Функции также можно передать число, определяющее уровень вложенности получаемой информации: 0 — ваша функция, 1 — функция, изкоторой она была вызвана, и т. д.

Полный синтаксис выглядит следующим образом ($i — количество уровней вложенности):

```
($package, $filename, $line, $subr, $has_args, $wantarray )= caller($i);
#   0         1         2      3       4          5
```

Возвращаемые значения имеют следующий смысл:

`$package`

Пакет, в котором был откомпилирован код:

`$filename`

Имя файла, в котором был откомпилирован код. Значение `-e` возвращается при запуске из командной строки, а значение `-` (дефис) — при чтении сценария из STDIN.

$line

Номер строки, из которой был вызван данный кадр стека:

`$subr`

Имя функции данного кадра, включающее ее пакет. Замыкания возвращают имена вида `main::__ANON__`, вызов по ним невозможен. Для eval возвращается `"(eval)"`.

`$has_args`

Признак наличия аргументов при вызове функции:

`$wantarray`

Значение, возвращаемое функцией `wantarray` для данного кадра стека. Равно либо `true`, либо `false`, либо `undef`. Сообщает, что функция была вызвана в списковом, скалярном или неопределенном контексте.

Вместо непосредственного вызова `caller`, продемонстрированного в решении, можно написать вспомогательные функции;

```
$me = whoami();
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$him = whowasi();

sub whoami  {  (caller(1))[3]  }
sub whowasi {  (caller(2))[3]  }
```

Аргументы 1 и 2 используются для функций первого и второго уровня вложенности, поскольку вызов whoami или whowasi будет иметь нулевой уровень.

Смотри также

Описание функций wantarray и caller в *perlfunc*(1); рецепт **10.6.**

10.5. Передача массивов и хэшей по ссылке

Проблема

Требуется передать функции несколько массивов или хэшей и сохранить их как отдельные сущности. Например, вы хотите выделить в подпрограмму алгоритм поиска элементов одного массива, отсутствующих в другой массиве, из рецепта 4.7. При вызове подпрограмма должна получать два массива, которые не должны смешиваться.

```
array_diff( \@array1, \@array2 );
```

Комментарий

Операции со ссылками рассматриваются в главе 11 "Ссылки и записи". Ниже показана подпрограмма, получающая ссылки на массивы, и вызов, в котором эти ссылки генерируются:

```
@a = (1, 2);
@b = (5, 8);
@c = add_vecpair( \@a, \@b );
print "@c\n";
6 10

sub add_vecpair {                  # Предполагается, что оба вектора
                                   # имеют одинаковую длину
    my ($x, $y) = @_;              # Скопировать ссылки на массивы


    for (my $i=0; $i < @$x;$i++) {
                    $x->[$i] + $y->[$i];
    }

            @result;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Функция обладает одним потенциальным недостатком: она не проверяет, что ей были переданы в точности два аргумента, являющиеся ссылками на массивы. Проверку можно организовать следующим образом:

```
unless (@_ == 2 &&          eq 'ARRAY' &&          eq 'ARRAY') {
    die "usage: add_vecpair ARRAYREF1 ARRAYREF2";
}
```

Если вы собираетесь ограничиться вызовом die в случае ошибки (см. рецепт 10.12), проверка обычно пропускается, поскольку при попытке разыменования недопустимой ссылки все равно возникает исключение.

Смотри также

Раздел "Pass by Reference» *perlsub*(1); раздел «Prototypes» *perlsub*(1); рецепт 10.11; глава 11.

10.6. Опред еление контекста вызова

Проблема

Требуется узнать, была ли ваша функция вызвана в скалярном или списковом контексте. Это позволяет решать разные задачи в разных контекстах, как это делается в большинстве встроенных функций Perl.

Решение

Воспользуйтесь функцией `wantarray()`, которая возвращает три разных значения в зависимости от контекста вызова текущей функции:

```
if (wantarray()) {
    # Списковый контекст
}
elsif (defined wantarray()) {
    # Скалярный контекст
}
else {
    # Неопределенный контекст
}
```

Комментарий

Многие встроенные функции, вызванные в скалярном контексте, работают совсем не так, как в списковом контексте. Пользовательская функция может узнать контекст своего вызова с помощью значения, возвращаемого встроенной функцией `wantarray`. Для спискового контекста `wantarray` возвращает `true`. Еслиовозвращается ложное, но определенное значение, функция используется в скалярном контексте. Если возвращается `undef`, отфункции вообще не требуется возвращаемого значения.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
if (wantarray()) {
    print "In list context\n";
          @many_things;
} elsif (defined wantarray()) {
    print "In scalar context\n";
          $one_thing;
} else {
    print "In void context\n";
             # Ничего
}

mysub();                    # Неопределенный контекст

$a = mysub();               # Скалярный контекст
if (mysub()) {  }           # Скалярный контекст

@a = mysub();               # Списковый контекст
print mysub();              # Списковый контекст
```

Смотри также

Описание функций в *perlfunc*(1).

10.7. Передача именованных параметров

Проблема

Требуется упростить вызов функции с несколькими параметрами, чтобы программист помнил смысл параметров, а не порядок их следования.

Решение

Укажите имена параметров при вызове:

```
thefunc(INCREMENT => "20s", START => "+5m", FINISH => "+30m");
thefunc(START => "+5m", FINISH => "+30m");
thefunc(FINISH => "+30m");
thefunc(START => "+5m", INCREMENT => "15s");
```

Затем в подпрограмме создайте хэш, содержащий значения по умолчанию и массив пар:

```
sub thefunc {
    my %args = (
        INCREMENT   => '10s',
        FINISH      => 0,
        START       => 0,
        @_,         # Список пар аргументов
    );
    if ($args{INCREMENT}  =^ /m$/ ) {       }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Функции, аргументы которых должны следовать в определенном порядке, удобны для небольших списков аргументов. Но с ростом количества аргументов становится труднее делать некоторые из них необязательными или присваивать им значения по умолчанию. Пропускать можно только аргументы, находящиеся в конце списка, и никогда — в начале.

Более гибкое решение — передача пар значений. Первый элемент пары определяет имя аргумента, а второй — значение. Программа автоматически документируется, поскольку смысл параметра можно понять, не читая полное определение функции. Более того, программистам, использующим такую функцию, не придется запоминать порядок аргументов, и они смогут пропускать любые аргументы.

Решение построено на объявлении в функции закрытого хэша, хранящего значения параметров по умолчанию. В конец хэша заносится массив текущих аргументов, @_ — значения по умолчанию заменяются фактическими значениями аргументов.

Смотри также

Глава 4 "Массивы".

10.8. Пропуск некоторых возвращаемых значений

Проблема

Имеется функция, которая возвращает много значений, однако вас интересуют лишь некоторые из них. Классический пример — функция stat; как правило, требуется лишь одно значение из длинного возвращаемого списка (например, режим доступа).

Решение

Присвойте результат вызова списку, некоторые позиции которого равны undef:

```
($a, undef, $c) = func();
```

Либо создайте срез списка возвращаемых значений и отберите лишь то, что вас интересует:

```
($a, $c) = (func())[0,2];
```

Комментарий

Применять фиктивные временные переменные слишком расточительно:

```
($dev,$ino,$DUMMY,$DUMMY,$uid) = stat($filename);
```

Чтобы ОТбрОСИТЬ ненужное ЗНачение, достаточно заменить фиктивные переменные на undef:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
($dev,$ino,undef,undef,$uid) = stat($filename);
```

Также можно создать срез и включить в него лишь интересующие вас значения:

```
($dev,$ino,$uid,$gid) = (stat($filename))[0,1,4,5];
```

Если вы хотите перевести результат вызова функции в списковый контекст и отбросить все возвращаемые значения (вызывая его ради побочных эффектов), начиная с версии 5.004, можно присвоить его пустому списку:

```
() = some_function();
```

Смотри также

Описание срезов в *perlsub*(1);рецепт 3.1.

10.9. Возврат нескольких массивов или хэшей

Проблема

Необходимо, чтобы функция возвратила несколько массивов или хэшей, однако возвращаемые значения сглаживаются в один длинный список скалярных величин.

Решение

Возвращайте ссылки на хэши или массивы:

```
                $hash_ref)   somefunc();

sub somefunc {
    my @array;
    my %hash;

    # ...

                \@array, \%hash
}
```

Комментарий

Как говорилось выше, все аргументы функции сливаются в общий список скалярных величин. То же самое происходит и с возвращаемыми значениями. Функция, возвращающая отдельные массивы или хэши, должна возвращать их по ссылке, и вызывающая сторона должна быть готова к получению ссылок. Например, возвращение трех отдельных хэшей может выглядеть так:

```
sub fn {

            (\%a, \%b, \%c); # или
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        \(%a, %b, %c); # то же самое
}
```

Вызывающая сторона должна помнить о том, что функция возвращает список ссылок на хэши. Она не может просто присвоить его списку из трех хэшей.

```
(%h0, %h1, %h2)  = fn();     # НЕВЕРНО!
@array_of_hashes = fn();     # например: $array_of_hashes[2]->{"keystring"}
($r0, $r1, $r2)  = fn();     # например: $r2->{"keystring"}
```

Смотри также

Общие сведения о ссылках в главе 11; рецепт 10.5.

10.10. Возвращение признака неудачного вызова

Проблема

Функция должна возвращать значение, свидетельствующее о неудачной попытке вызова.

Решение

Воспользуйтесь командой аргументов, которая в скалярном контексте возвращает `undef`, а в списковом — пустой список `()`.

Комментарий

аргументов означает следующее:

```
sub empty_retval {
        wantarray ? () : undef );
}
```

Ограничиться простым `undef` нельзя, поскольку в списковом контексте вы получите список из одного элемента: `undef`. Если функция вызывается в виде:

```
if (@a = yourfunc()) { ... >
```

то признак ошибки будет равен `true`, поскольку `@a` присваивается список `(undef)`, интерпретируемый в скалярном контексте. Результат будет равен 1 (количество элементов в `@a`), то есть истинному значению. Контекст вызова можно определить с помощью функции `wantarray`, однако аргументов обеспечивает более наглядное иизящное решение, которое работает в любых ситуациях:

```
unless ($a = sfunc()) { die "sfunc failed" }
unless (@a = afunc()) { die "afunc failed" }
unless (%a = hfunc()) { die "hfunc failed" }
```

Некоторые встроенные функции Perl иногда возвращают довольно странные значения. Например, `fcntl` и `ioctl` в некоторых ситуациях возвращают строку

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

"0 but true" (для удобства эта волшебная строка была изъята из бесчисленных предупреждений об ошибках преобразования флага **-w**). Появляется возможность использовать конструкции следующего вида:

```
ioctl(....) or die "can't ioctl: $!";
```

В этом случае программе не нужно отличать определенный ноль от неопределенного значения, как пришлось бы делать для функций glob. В числовой интерпретации "0 but true" является нулем. Необходимость в возвращении подобных значений возникает довольно редко. Более распространенный (и эффектный) способ сообщить о неудаче при вызове функции заключается в инициировании исключения (см. рецепт 10.12).

Смотри также

Описание функций wantarray и в *perlfunc*(1) рецепт 10.12.

10.11. Прото типы функций

Проблема

Вы хотите использовать прототипы функций, чтобы компилятор мог проверить типы аргументов.

Решение

В Perl существует нечто похожее на прототипы, но это сильно отличается от прототипов в традиционном понимании. Прототипы функций Perl больше напоминают принудительный контекст, используемый при написании функций, которые ведут себя аналогично некоторым встроенным функциям Perl (например, push и pop).

Комментарий

Фактическая проверка аргументов функции становится возможной лишь во время выполнения программы. Если объявить функцию до ее реализации, компилятор сможет использовать очень ограниченную форму прототипизации. Не путайте прототипы Perl с теми, что существуют в других языках. В Perl прототипы предназначены лишь для эмуляции поведения встроенных функций.

Прототип функции Perl представляет собой ноль и более пробелов, обратных косых черт или символов типа, заключенных в круглые скобки после определения или имени подпрограммы. Символ типа с префиксом \ означает, что аргумент в данной позиции передается по ссылке и должен начинаться с указанного символа типа.

Прототип принудительно задает контекст аргументов, используемых при вызове данной функции. Это происходит во время компиляции программы и в большинстве случаев вовсе не означает, что Perl проверяет количество или тип аргументов функции. Если Perl встретит вызов func(3, 5) для функции с прото-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

типом sub func($), он завершит компиляцию с ошибкой. Но если для того же прототипа встретится вызов func(@array), компилятор всего лишь преобразует @array в скалярный контекст; он не скажет: "Массив передавать нельзя — здесь должна быть скалярная величина".

Это настолько важно, что я повторю снова: не пользуйтесь прототипами Perl, если вы надеетесь, что компилятор будет проверять тип и количество аргументов.

Тогда зачем нужны прототипы? Существуют дваосновных применения, хотя во время экспериментов вы можете найти и другие. Во-первых, с помощью прототипа можно сообщить Perl количество аргументов вашей функции, чтобы опустить круглые скобки при ее вызове. Во-вторых, с помощью прототипов можно создавать подпрограммы с тем же синтаксисом вызова, что и у встроенных функций.

Пропуск скобок

Обычно функция получает список аргументов, и при вызове скобки ставить не обязательно:

```
@results = myfunc 3 , 5;
```

Без прототипа такая запись эквивалентна следующей:

```
@results = myfunc(3 , 5);
```

При отсутствии скобок Perl преобразует правую часть вызова подпрограммы в списковый контекст. Прототип позволяет изменить такое поведение:

```
sub myfunc($);
@results = myfunc 3 , 5;
```

Теперь эта запись эквивалентна следующей:

```
myfunc(3), 5 );
```

Кроме того, можно предоставить пустой прототип, показывающий, что функция вызывается без аргументов, как встроенная функция time. Именно так реализованы константы LOCK_SH, LOCK_EX и LOCK_UN в модуле Fcntl. Они представляют собой экспортируемые функции, определенные с пустым прототипом:

```
sub LOCK_SH () { 1 }
sub LOCK_EX () { 2 }
sub LOCK_UN () { 4 }
```

Имитация встроенных функций

Прототипы также часто применяются для имитации поведения таких встроенных функций, как push и shift, передающих аргументы без сглаживания. При вызове push(@array, 1, 2, 3) функция получает *ссылку* на @array вместо самого массива. Для этого в прототипе перед символом @ ставится обратная косая черта:

```
sub mypush (\@@) {
  my          = shift;
  my @remainder = @_;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

\@ в прототипе означает "потребовать, чтобы первый аргумент начинался с символа @, и передавать его по ссылке". Второй символ @ говорит о том, что остальные аргументы образуют список (возможно, пустой). Обратная косая черта, с которой начинается список аргументов, несколько ограничивает ваши возможности. Например, вам даже не удастся использовать условную конструкцию ?: для выбора передаваемого массива:

```
mypush( $x > 10 ? @a : @b , 3, 5 );        # НЕВЕРНО
```

Вместо этого приходится изощряться со ссылками:

```
mypush( @{ $x > 10 ? @a : @b }, 3, 5 );      # ВЕРНО
```

Приведенная ниже функция hpush работает аналогично push, но для хэшей. Функция дописывает в существующий хэш список пар "ключ/значение", переопределяя прежнее содержимое этих ключей.

```
sub hpush(\%@) {
    my $href = shift;
    while ( my ($k, $v) = splice(@_, 0, 2) ) {
        $href->{$k} = $v;
    }
}
hpush(%pieces, "queen" => 9, "rook" => 5);
```

Смотри также

Описание функции prototype в *perlfunc*(1); *perlsub*(1); рецепт **10.5.**

10.12. Обработка исключений

Проблема

Как организовать безопасный вызов функции, способной инициировать исключение? Как создать функцию, инициирующую исключение?

Решение

Иногда в программе возникает что-то настолько серьезное, что простого возвращения ошибки оказывается недостаточно, поскольку та может быть проигнорирована вызывающей стороной. Включите в функцию конструкцию die СТРОКА, чтобы инициировать исключение:

```
die "some message";            # Инициировать исключение
```

Чтобы перехватить исключение, вызывающая сторона вызывает функцию из eval, после чего узнает результат с помощью специальной переменной $@:

```
eval { func() };
if ($@) {
    warn "func raised an exception: $@";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Инициирование исключения — крайняя мера, и относиться к ней следует серьезно. В большинстве функций следует возвращать признак ошибки с помощью простого оператора Перехватывать исключения при каждом вызове функции скучно и некрасиво, и это может отпугнуть от применения исключений.

Но в некоторых ситуациях неудачный вызов функции должен приводить к аварийному завершению программы. Вместо невосстановимой функции exit следует вызвать die — по крайней мере, у программиста появится возможность вмешаться в происходящее. Если ни один обработчик исключения не был установлен с помощью eval, на этом месте программа аварийно завершается.

Чтобы обнаружить подобные нарушения, можно поместить вызов функции в блок eval. Если произойдет исключение, оно будет присвоено переменной $@; в противном случае переменная равна false.

```
eval { $val = func() >;
warn "func blew up: $@" if $@;
```

Блок eval перехватывает все исключения, а не только те, что интересуют вас. Непредусмотренные исключения обычно следует передать внешнему обработчику. Предположим, функция инициирует исключение, описываемое строкой "Full moon!". Можно спокойно перехватить это исключение и дать другим обработчикам просмотреть переменную $@. При вызове die без аргументов новая строка исключения конструируется на основании содержимого $@ и текущего контекста.

```
eval { $val = func() };
if ($@ && $@ !^ /Full moon!/) {
    die;      # Повторно инициировать неизвестные ошибки
}
```

Если функция является частью модуля, можно использовать модуль Carp и вызвать croak или confess вместо die. Единственное отличие die от croak заключается в том, что croak представляет ошибку с позиции вызывающей стороны, а не модуля. Функция confess по содержимому стека определяет, кто кого вызвал и с какими аргументами.

Другая интересная возможность заключается в том, чтобы функция могла узнать о полном игнорировании возвращаемого ею значения (то есть о том, что она вызывается в неопределенном контексте). В этом случае возвращение кода ошибки бесполезно, поэтому вместо него следует инициировать исключение.

Конечно, вызов функции в другом контексте еще не означает, что возвращаемое значение будет должным образом обработано. Но в неопределенном контексте оно заведомо не проверяется.

```
if (defined wantarray()) {

} else {
    die "pay attention to my error!";
}
```

Смотри также

Описание переменной $@ в *perlvar*(1); описание функций die и eval *в perlfunc(1);* рецепты 10.15, 12.2 и 16.21.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

10.13. Сохранение глобальных значений

Проблема

Требуется временно сохранить значение глобальной переменной.

Решение

Воспользуйтесь оператором local, чтобы сохранить старое значение и автоматически восстановить его при выходе из текущего блока:

```
$age = 18;            # Глобальная переменная
if (CONDITION) {
    local $age = 23;
    func();  #  Видит  временное  значение  23
} я Восстановить старое значение при выходе из блока
```

Комментарий

К сожалению, оператор Perl local не создает локальной переменной — это делается оператором my. local всего лишь сохраняет существующее значение на время выполнения **блока**, в котором он находится.

Однако в трех ситуациях вы *должны* использовать local вместо my.

1. Глобальной переменной (особенно $_) присваивается временное значение.
2. Создается локальный манипулятор файла или каталога или локальная функция.
3. Вы хотите временно изменить один элемент массива или хэша.

Применение local() для присваивания временных значений глобальным переменным

Первая ситуация чаще встречается для стандартных, нежели пользовательских переменных. Нередко эти переменные используются Perl в высокоуровневых операциях. В частности, любая функция, явно или косвенно использующая $_, должна иметь локальную копию **$_**. Об этом часто забывают. Одно из возможных решений приведено в рецепте 13.15.

В следующем примере используется несколько глобальных переменных. Переменная $/ косвенно влияет на поведение оператора чтения строк, используемого в операциях <FH>.

```
$para = get_paragraph(*FH);          # Передать glob файлового манипулятора
$para = get_paragraph(\*FH);         # Передать манипулятор по ссылке на glob
$para = get_paragraph(*IO{FH});      # Передать манипулятор по ссылке на IO
sub get_paragraph {
    my $fh = shift;
    local $/ = '';
    my $paragraph = <$fh>;
    chomp($paragraph);
        $paragraph;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Применение local() для создания локальных манипуляторов

Вторая ситуация возникает в случае, когда требуется локальный манипулятор файла или каталога, реже — локальная функция. Начиная с Perl версий 5.000, можно воспользоваться стандартными модулями Symbol, Filehandle или IO::Handle, но и привычная методика с тип-глобом по-прежнему работает. Например:

```
$contents = get_motd();
sub get_motd {
    local *MOTD;
    open(MOTD, "/etc/motd")       or die "can't open motd: $!";
    local $/ = undef;  # Читать весь файл
    local $_ = <MOTD>;
    close (MOTD);

}
```

Открытый файловый манипулятор возвращается следующим образом:

```
*MOTD;
```

Применение local() в массивах и хэшах

Третья ситуация на практике почти не встречается. Поскольку оператор local в действительности является оператором "сохранения значения", им можно воспользоваться для сохранения одного элемента массива или хэша, даже если сам массив или хэш является лексическим!

```
my @nums = (0 .. 5);
sub first{
    local $nums[3] = 3.14159;
    second();
}
sub second {
    print "@nums\n";
}
second();
0 1 2 3 4 5
first();
0 1 2 3.14159 4 5
```

Единственное стандартное применение — временные обработчики сигналов.

```
sub first <
    local $SIG{INT} = 'IGNORE';
    second();
}
```

Теперь во время работы second() сигналы прерывания будут игнорироваться. После выхода из first() автоматически восстанавливается предыдущее значение $SIG{INT}.

Хотя local часто встречается в старом коде, от него следует держаться подальше, если это только возможно. Поскольку local манипулирует значениями

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

глобальных, а не локальных переменных, директива use st rict ни к чему хорошему не приведет.

Оператор local создает *динамическую область действия.* Она отличается отдругой области действия, поддерживаемой Perl и значительно более понятной на интуитивном уровне. Речь идет об области действия my— *лексической области действия,* иногда называемой "статической".

В динамической области действия переменная доступна в том случае, если она находится в текущей области действия — или в области действия всехкадров (блоков) стека, определяемых во время выполнения. Все вызываемые функции обладают полным доступом к динамическим переменным, поскольку последние остаются глобальными, но получают временные значения. Лишь лексические переменные защищены от вмешательства извне. Если и это вас не убедит, возможно, вам будет интересно узнать, что лексические переменные примерно на 10 процентов быстрее динамических.

Старый фрагмент вида:

```
sub func {
    local($x, $y) =@_;
    #....
}
```

почти всегда удается заменить без нежелательных последствий следующим фрагментом:

```
sub func {
    my($x, $y) = @_;
    #
}
```

Единственный случай, когда подобная замена невозможна, — если работа программы основана на динамической области действия. Это происходит в ситуации, когда одна функция вызывает другую и работа второй зависит от доступа к временным версиям глобальных переменных $x и $y первой функции. Код, который работает с глобальными переменными и вместо нормальной передачи параметров издалека вытворяет нечто странное, в лучшем случае ненадежен. Хорошие программисты избегают подобных выкрутасов как чумы.

Если вам встретится старый код вида:

```
&func(*Global_Array);
sub func {
    local(*aliased_array) = shift;
    for (@aliased_array) { .... }
```

вероятно, его удастся преобразовать к следующей форме:

```
func(\@Global_Array);
sub func {
                    shift;
                       ....
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

До появления в Perl нормальной поддержки ссылок, использовалась старая стратегия передачи тип-глобов. Сейчас это уже дело прошлое.

Смотри также

Описание функций local и my в *perlfunc*(1); разделы "Private Variables via my()» и "Temporary Values via local()» *perlsub*(1); рецепты 10.2; 10.16.

10.14. Переопределение функции

Проблема

Требуется временно или постоянно переопределить функцию, однако функциям нельзя "присвоить" новый код.

Решение

Чтобы переопределить функцию, присвойте ссылку на новый код тип-глобу имени функции. Используйте local для временной замены.

```
undef &grow;                # Заглушить жалобы -w на переопределение
*grow = \&expand;
grow();                     # Вызвать expand()

{
    local *grow = \&shrink;     # Только в границах блока
        grow();                 # Вызывает shrink()
}
```

Комментарий

В отличие от переменных (но по аналогии с манипуляторами) функции нельзя напрямую присвоить нужное значение. Это всего лишь имя. Однако с ней можно выполнять многие операции, выполняемые с переменными, поскольку вы можете напрямую работать с таблицей символов с помощью тип-глобов вида *foo и добиваться многих интересных эффектов.

Если присвоить тип-глобу ссылку, то при следующем обращении к символу данного типа будет использовано новое значение. Именно это делает модуль Exporter при импортировании функции или переменной из одного пакета в другой. Поскольку операции выполняются непосредственно с таблицей символов пакета, они работают только для пакетных (глобальных) переменных, но не для лексических.

```
*one::var = \%two::Table;    # %one::var становится синонимом для %two::Table
*one::big = \&two::small;    # &one::big становится синонимом для &two::small
```

С тип-глобом можно использовать local, но не my. Из-за local синоним действует только в границах текущего блока.

```
local          \&barney;      # временно связать &fred`c &barney
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если **значение**, присваиваемое тип-глобу, представляет собой не ссылку, а другой тип-глоб, то замена распространяется на *все* типы с данным именем. Полное присваивание тип-глоба относится к скалярным величинам, массивам, хэшам, функциям, файловым манипуляторам, манипуляторам каталогов и форматам. Следовательно, присваивание *Top = *Bottom сделает переменную $Top текущего пакета синонимом для $Bottom, @Top — для @Bottom, %Top — для %Bottom и &Top — для &Bottom. Замена распространяется даже на соответствующие манипуляторы файлов и каталогов и форматы! Вероятно, это окажется лишним.

Присваивание тип-глобов в сочетании с **замыканиями** позволяет легко и удобно дублировать функции. Представьте, что вам понадобилась функция для генерации HTML-кода, работающего с цветами. Например:

```
$string = red("careful here");
print $string;
<FONT COLOR='red'>careful here</FONT>
```

Функция red выглядит так:

```
sub red { "<FONT COLOR='red'>@_</FONT>" }
```

Если вам потребуются другие цвета, пишется нечто подобное:

```
sub color_font {
    my $color = shift;
            "<FONT COLOR='$color'>@_</FONT>";
}
              color_font("red",
sub green { color_font("green", @_)  }
sub blue  { color_font("blue", @_)   }
sub purple { color_font("purple", @_) }
# И т. д.
```

Сходство функций наводит на мысль, что общую составляющую можно как-то выделить. Для этого следует воспользоваться косвенным присваиванием тип-глобы. Если вы используете рекомендуемую директиву use strict, сначала **отключите strict 'refs' для этого блока.**

```
@colors =        blue        yellow orange purple violet);
for my $name (@colors) {
    no strict 'refs';
    *$name = sub { "<FONT COLOR='$name'>@ </FONT>" };
}
```

Функции кажутся независимыми, однако фактически код был откомпилирован лишь один раз. Подобная методика экономит время компиляции и память. Для создания полноценного замыкания все переменные анонимной подпрограммы *должны* быть лексическими. Именно поэтому переменная цикла объявляется с ключевым словом my.

Перед вами одна из немногочисленных ситуаций, в которых создание прототипа для замыкания оправдано. Если вам захочется форсировать скалярный контекст для аргументов этих функций (вероятно, не лучшая идея), ее можно записать в следующем виде:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
*$name = sub ($) { "<FONT COLOR='$name'>$_[0]</FONT>" };
```

Однако прототип проверяется во время компиляции, поэтому приведенное выше присваивание произойдет слишком поздно и никакой пользы не принесет. Следовательно, весь цикл с присваиваниями следует включить в BEGIN-блок, чтобы форсировать его выполнение при компиляции.

Смотри также

Описание замыканий в раздел "Symbol tables" *perlmod*(1); рецепты 10.11; 11.4.

10.15. Перехват вызовов неопределенных функций с помощью AUTOLOAD

Проблема

Требуется перехватить вызовы неопределенных функций и достойно обработать их.

Решение

Объявите функцию с именем AUTOLOAD для пакета, вызовы неопределенных функций которого вы собираетесь перехватывать. Во время ее выполнения переменная $AUTOLOAD этого пакета содержит имя вызванной неопределенной функции.

Комментарий

В подобных ситуациях обычно применяются вспомогательные функции (proxy). При вызове неопределенной функции вместо автоматического инициирования исключения также можно перехватить вызов. Если пакет, к которому принадлежит вызываемая функция, содержит функцию с именем AUTOLOAD, то она будет вызвана вместо неопределенной функции, а специальной глобальной переменной пакета $AUTOLOAD будет присвоено полное имя функции. Затем функция AUTOLOAD сможет делать все, что должна была делать исходная функция.

```
sub AUTOLOAD {
    use vars qw($AUTOLOAD);
    my $color = $AUTOLOAD;
    $color =~ s/.*::/;
        "<FONT COLOR='$color'>@ </FONT>";
}
# Примечание: функция sub c определена
print chartreuse("stuff");
```

При вызове несуществующей функции main::chartreuse вместо инициирования исключения будет вызвана функция main::AUTOLOAD с аргументами, переданными Пакетная переменная $AUTOLOAD будет содержать строку main::

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Методика с присваиваниями тип-глобов из рецепта **10.14** быстрее и удобнее. Быстрее — поскольку вам не приходится запускать копию и заниматься подстановками. Удобнее — поскольку **вы** сможете делать следующее:

```
{
    local *yellow = \&violet;
    local (*red,
    print_stuff();
}
```

При работе `print_stuff` или любой вызванной ей функции все, что должно выводиться желтым цветом, выводится фиолетовым; красный цвет заменяется зеленым, и наоборот.

Однако подстановка функций не позволяет обрабатывать вызовы неопределенных функций. `AUTOLOAD` справляется с этой проблемой.

Смотри также

Раздел «Autoloading» *perlsub*(1); документация по стандартным модулям AutoLoader и AutoSplit; рецепты **10.12; 12.10; 13.11.**

10.16. Вложенные подпрограммы

Проблема

Требуется реализовать вложение подпрограмм, чтобы одна подпрограмма была видна и могла вызываться только из другой. Если попытаться применить очевидный вариант `sub F00 { sub BAR { } ... }`, Perl предупреждает о переменных, которые "не останутся общими".

Решение

Вместо того чтобы оформлять внутренние функции в виде обычных подпрограмм, реализуйте их в виде замыканий и затем временно присвойте их тип-глобу правого имени, чтобы создать локальную функцию.

Комментарий

Вероятно, **в** других языках программирования вам приходилось работать с вложенными функциями, обладающими собственными закрытыми переменными. **В** Perl для этого придется немного потрудиться. Интуитивная реализация приводит к выдаче предупреждения. Например, следующий фрагмент не работает:

```
sub outer {
    my $x = $_[0] + 35;
    sub inner {                    НЕВЕРНО
            inner();
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Обходное решение выглядит так:

```
sub outer {
    my $x = $_[0] + 35;
    local *inner = sub {
                inner();
}
```

Теперь благодаря временно присвоенному замыканию `inner()` может вызываться только из `outer()`. При вызове `inner()` получает нормальный доступ к лексической переменной **$x** из области действия `outer()`.

В сущности, мы создаем функцию, которая является локальной для другой функции — подобная возможность не поддерживается **в Perl** напрямую. Впрочем, ее реализация не всегда выглядит понятно.

Смотри также

Описание замыканий **в** раздел "Symbol tables" ***perlmod(1);*** рецепты **10.13-10.14.**

10.17. Сортировка почты

Программа **из** примера 10.1 сортирует почтовыйящик по темам. Для этого она читает сообщения по абзацам и ищет абзац, начинающийся с `"From:"`. Когда такой абзац будет найден, программа ищет **тему,** удаляет из нее все пометки `"Re:"`, преобразует в нижний регистр и сохраняет в массиве `@sub`. При этом сами сообщения сохраняются в массиве `@msgs`. Переменная `$msgno` следит за номером сообщения.

Пример 10.1. bysubl

```
#!/usr/bin/perl
# bysub1 - simple sort by subject
my(@msgs, @sub);
my $msgno = -1;
$/ = '';                    # Чтение по абзацам
while (<>) {
    if (/^From/m) {
        /^Subject:\s*(?:Re:\s*)*(.*)/mi;
        $sub[++$msgno] = lc($1) || '';
    }
    $msgs[$msgno] .= $_;
>
for my $i (sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 .. $#msgs)) {
    print $msgs[$i];
}
```

В этом варианте сортируются только индексы массивов. Ёсли темы совпадают, `cmp` возвращает 0, поэтому используется вторая часть `||`, в которой номера сообщений сравниваются в порядке их исходного следования.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если функции sort передается список (0, 1, 2, 3), после сортировки будет получена некоторая перестановка — например, (2, 1, 3, 0). Мы перебираем элементы списка в цикле for и выводим каждое сообщение.

В примере 10.2 показано, как бы написал эту программу программист с большим опытом работы на awk. Ключ -00 используется для чтения абзацев вместо строк.

Пример 10.2. bysub2

```
#!/usr/bin/perl -nOO
# bysub2 - сортировка по темам в стиле awk
BEGIN { $msgno = -1 }
$sub[++$msgno] = (/^Subject:\s*(?:Re:\s*)*(.*)/mi)[0] if /^From/m;
$msg[$msgno] .= $_;
END { print @msg[ sort { $sub[$a] cmp $sub[$b] || $a <=> $b } (0 ..$#msg) ] }
```

Параллельные массивы широко использовались лишь на ранней стадии существования Perl. Более элегантное решение состоит в том, чтобы сохранять сообщения в хэше. Анонимный хэш (см. главу 11) сортируется по каждому полю.

Программа из примера 10.3 построена на тех же принципах, что и примеры 10.1 и 10.2.

Пример 10.3. bysub3

```
#!/usr/bin/perl -00
# bysub3 - sort by subject
use strict;
my @msgs = ();
while (<>) {
    push @msgs, {
        SUBJECT => /^Subject:\s*(?:Re:\s*)*(.*)/mi,
        NUMBER  => scalar @msgs,   # Номер сообщения
        TEXT    => '',
    } if /^From/m;
    $msgs[-1]{TEXT} .= $_;
}

for my $msg (sort {
                        $a->{SUBJECT} cmp $b->{SUBJECT>
                                  ||
                        $a->{NUMBER} <=> $b->{NUMBER}
                    } @msgs
          )
{
    print $msg->{TEXT};
}
```

Работая с полноценными хэшами, нетрудно добавить дополнительные критерии сортировки. Почтовые ящики часто сортируются сначала по теме, а затем по дате сообщения. Основные трудности связаны с анализом и сравнением дат. Модуль Date::Manip помогает справиться с ними и возвращает строку, которую можно сравнивать с другими. Тем не менее программа *datesort* из примера 10.4, исполь-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

зующая Date::Manip, работает в 10 раз медленнее предыдущей. Анализ дат в непредсказуемых форматах занимает слишком много времени.

Пример 10.4. datesort

```
#!/usr/bin/perl -00
# datesort - сортировка почтового ящика по теме и дате
use strict;
use Date::Manip;
my @msgs = ();
while (o) {
    next unless /^From/m;
    my $date = '';
    if (/^Date:\s*(.*)/m) {
        ($date = $1) =~ s/\s+\(.*//;
        $date = ParseDate($date);
    }
    push @msgs, {
        SUBJECT => /^Subject:\s*(?:Re:\s*)*(.*)/mi,
        DATE    => $date,
        NUMBER  => scalar @msgs,
        TEXT    => '',
    };
} continue {
    $msgs[-1]{TEXT} ,= $ ;
}

for my $msg (sort {
                        $a->{SUBJECT} cmp $b->{SUBJECT}
                                        ||
                        $a->{DATE}    cmp $b->{DATE}
                                        ||
                        $a->{NUMBER}  <=> $b->{NUMBER}
                    } @msgs
          )
{
    print $msg->{TEXT};
}
```

Особого внимания в примере 10.4 заслуживает блок continue. Придостижении конца цикла (нормальном выполнении или переходе по next) этот блок выполняется целиком. Он соответствует третьему компоненту цикла for, но не ограничивается одним выражением. Это полноценный блок, который может состоять из нескольких команд.

Смотри также

Описание функции sort *в perlfunc(1);* описание переменной $/ *в perlvar(1)* и во введении главы 8 "Содержимое файлов"; рецепты 3.7, 4.15, 5.9 и 11.9.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Ссылки и записи 11

В эту маленькую паутинку
Я поймаю такую большую муху, как Кассио.

Шекспир,
«Отелло» акт II, сцена 1

Введение

В Perl существуют три основных типа данных: скалярные величины, массивы и хэши. Конечно, многие программы удается написать и без сложных структур данных, но обычно простых переменных и списков все же оказывается недостаточно.

Три встроенные структуры данных Perl в сочетании со ссылками позволяют строить сколь угодно сложные и функциональные структуры данных — те самые записи, которых так отчаянно не хватало в ранних версиях Perl. Правильно выбирая структуру данных и алгоритм, вы иногда выбираете между элегантной программой, которая быстро справляется со своей задачей, и убогой поделкой, работающей с черепашьей скоростью и нещадно пожирающей системные ресурсы.

Первая часть этой главы посвящена созданию и использованию простых ссылок. Во второй части рассказывается о применении ссылок для создания структур данных более высокого порядка.

Ссылки

Чтобы хорошо понять концепцию ссылок, сначала необходимо разобраться с тем, как в Perl хранятся значения переменных. С любой определенной переменной ассоциируется имя и адрес области памяти. Идея хранения адресов играет для ссылок особую роль, поскольку в ссылке хранятся данные о местонахождении другой величины. Скалярная величина, содержащая адрес области памяти, называется ссылкой. Значение, хранящееся в памяти по данному адресу, называется *субъектом* (рис. 11.1).

Субъект может относиться к одному из встроенных типов данных (скалярная величина, массив, хэш, ссылка, код или глоб) или представлять собой пользовательский тип, основанный на одном из встроенных типов.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

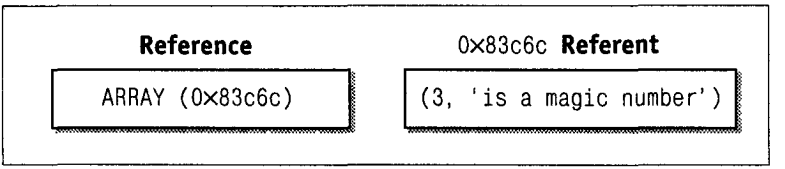


Рис. 11.1. Ссылка и субъект

Субъекты в Perl типизованы. Это означает, что ссылку на массив нельзя интерпретировать как ссылку на хэш. При подобных попытках инициируется исключение. В Perl не предусмотрен механизм преобразования типов, и это было сделано намеренно.

На первый взгляд кажется, что ссылка — обычный адрес с сильной типизацией. На самом деле это нечто большее. Perl берет на себя автоматическое выделение и освобождение памяти (сборку мусора) для ссылок так же, как и для всего остального. С каждым блоком памяти в Perl связан *счетчик ссылок*, который определяет количество ссылок на данный субъект. Память, используемая субъектом, возвращается в пул свободной памяти процесса лишь при обнулении счетчика ссылок. Тем самым гарантируется, что вы никогда не получите недопустимую ссылку — забудьте об аварийных завершениях и ошибках защиты, часто возникающих при неправильной работе с указателями в С.

Освобожденная память передается Perl для последующего использования, но лишь немногие операционные системы возвращают ее себе. Это связано с тем, что в большинстве схем распределения памяти используется стек, а при освобождении памяти в середине стека операционная система не сможет вернуть ее без перемещения всех остальных блоков. Перемещение нарушит целостность указателей и прикончит вашу программу.

Чтобы перейти от ссылки к субъекту, снабдите ссылку символом типа для тех данных, к которым вы обращаетесь. Например, если является ссылкой на скалярную величину, возможна следующая запись:

```
print              Выводится скалярная величина, на которую ссылается $sref
                   Присваивается субъекту
```

Для обращения к отдельному элементу массива или хэша, на который у вас имеется ссылка, используется ассоциативный оператор, оператор -> ("стрелка") — например, $rv->[37] или $rv->{"wilma"}. Помимо разыменования ссылок на массивы и хэши, стрелка также применяется при косвенном вызове функций через ссылки — например, $code_ref->("arg1","arg2") (см. рецепт 11.4). Если вы работаете с объектами, то с помощью стрелки можно вызывать их методы, $object->methodname("arg1","arg2"), как показано в главе 13 "Классы, объекты и связи".

Правила синтаксиса Perl делают разыменование сложных выражений нетривиальной задачей. Чередование правых и левых ассоциативных операторов не рекомендуется. Например, $$x[4] — то же самое, что и $x->[4]; иначе говоря, $x интерпретируется как ссылка на массив, после чего из массива извлекается четвертый элемент. То же самое записывается в виде ${$x}[4]. Если вы имели в виду "взять четвертый элемент @x и разыменовать его в скалярное выражение", воспользуйтесь ${$x[4]}. Старайтесь избегать смежных символов типов ($@%&) везде, кроме простых и однозначных ситуаций типа %hash = %$hashref.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Приведенный выше пример с можно переписать в виде:

```
print ${$sref};     # Выводится скалярная величина, на которую ссылается
${$sref} = 3;       # Присваивается субъекту
```

Некоторые программисты для уверенности используют только эту форму.

Функция ref получает ссылку и возвращает строку с описанием субъекта. Строка обычно принимает одно из значений SCALAR, ARRAY, HASH или CODE, хотя иногда встречаются и другие встроенные типы GLOB, REF, IO, Regexp и LVALUE. Если вызывается для аргумента, не являющегося ссылкой, функция возвращает false. При вызове ref для объекта (ссылки, для субъекта которой вызывалась функция bless) возвращается класс, к которому был приписан объект: CGI, IO::Socket или даже ACME::Widget.

Ссылки в Perl можно создавать для субъектов уже определенных или определяемых с помощью конструкций [ ], { } или sub { }. Использовать оператор \ очень просто: поставьте его перед субъектом, для которого создается ссылка. Например, ссылка на содержимое массива @array создается следующим образом:

```
$rv = \@array;
```

Создавать ссылки можно даже для констант; при попытке изменить значение субъекта происходит ошибка времени выполнения:

```
$pi = \3.14159;
$$pi = 4;   # Ошибка
```

Анонимные данные

Ссылки на существующие данные часто применяются для передачи аргументов функции, но в динамическом программировании они бывают неудобны. Иногда ситуация требует создания нового массива, хэша, скалярной величины или функции, но вам нехочется возиться с именами.

Анонимные массивы и хэши в Perl могут создаваться явно. При этом выделяется память для нового массива или хэша и возвращается ссылка на нее:

```
                                                Новый анонимный массив
$href = { "How" => "Now", "Brown" => "Cow" };   # Новый анонимный хэш
```

В Perl также существует возможность косвенного создания анонимных субъектов. Если попытаться присвоить значение через неопределенную ссылку, Perl автоматически создаст субъект, который вы пытаетесь использовать.

```
undef
@$aref = (1, 2, 3);
print
ARRAY(Ox80c04fO)
```

Обратите внимание: от undef мы переходим к ссылке на массив, не выполняя фактического присваивания. Perl автоматически создает субъект неопределенной ссылки. Благодаря этому свойству программа может начинаться так:

```
$a[4][23][53][21] = "fred";
print $a[4][23][53][21];
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
fred
print $a[4][23][53];
ARRAY(Ox81e2494)
print $a[4][23];
ARRAY(0x81e0748)
print $a[4];
ARRAY(0x822cd40)
```

В следующей таблице перечислены способы создания ссылок для именованных и анонимных скалярных величин, массивов, хэшей и функций. Анонимные **тип-глобы** выглядят слишком страшно и практически никогда не используются. Вместо них следует применять `IO::Handle->new()`.

Ссылка на	Именованный субъект	Анонимный субъект
Скалярная величина	`\$scalar`	`\do{my $anon}`
Массив	`\@array`	`< СПИСОК }`
Хэш	`\%hash`	`{ СПИСОК }`
Функция	`\&function`	`sub { КОД }`

Отличия именованных субъектов от анонимных поясняются на приведенных далее рисунках. На рис. 11.2 изображены именованные субъекты, а на рис. 11.3 — анонимные.

Иначе говоря, в результате присваивания `$a = \$b` переменные `$$a` и `$b` *занимают одну и ту же область памяти.* Если вы напишете `$$a = 3`, значение `$b` станет равно 3.

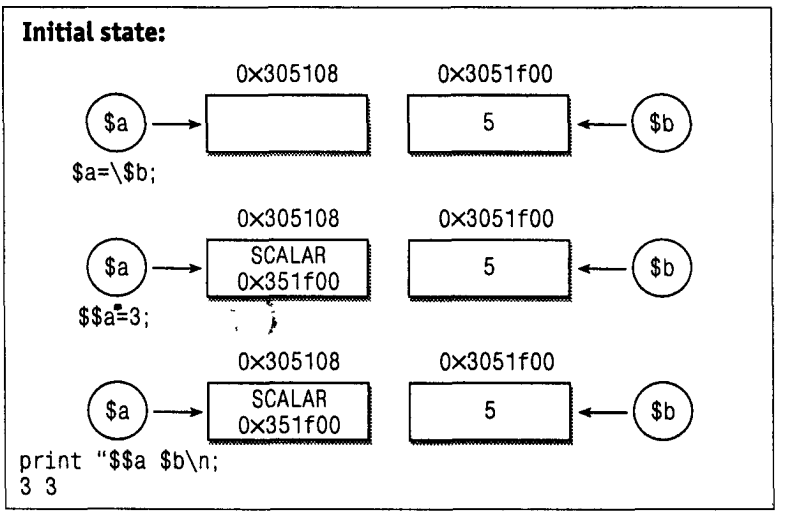


Рис. 11.2. **Именованные субъекты**

Все ссылки по определению оцениваются Ka K true, поэтому, если ваша функция возвращает ссылку, в случае ошибки можно вернуть undef и проверить возвращаемое значение следующим образом:

```
$op_cit = cite($ibid)          or die "couldn't make a
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

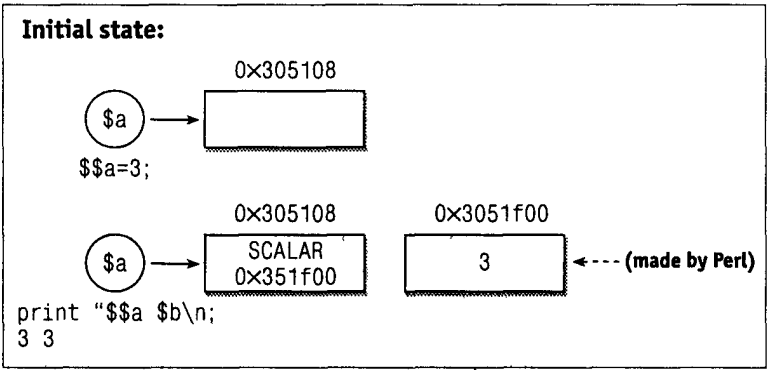


Рис. 11.3. Анонимные субъекты

Оператор undef может использоваться с любой переменной или функцией **Perl** для освобождения занимаемой ей памяти. Однако не следует полагать, что при вызове undef всегда освобождается память, вызываются деструкторы объектов и т. д. В действительности оператор всего лишь уменьшает счетчик ссылок на 1. Без аргумента undef дает неопределенное значение.

Записи

Ссылки традиционно применялись в Perl для обхода ограничения, согласно которому массивы и хэши могут содержать только скаляры. Ссылки являются скалярами, поэтому для создания массива массивов следует создать массив *ссылок* на массивы. Аналогично, хэши хэшей реализуются как хэши со ссылками на хэши; массивы хэшей — как массивы ссылок на хэши; хэши массивов — как хэши ссылок на массивы и т. д.

Имея в своем распоряжении эти сложные структуры, можно воспользоваться ими для реализации записей. Запись представляет собой отдельную логическую единицу, состоящую из различных атрибутов. Например, запись, описывающая человека, может содержать имя, адрес и дату рождения. В C подобные вещи называются структурами (structs), а в Pascal — записями (RECORDs). В Perl для них не существует специального термина, поскольку эта концепция может быть реализована разными способами.

Наиболее распространенный подход в Perl заключается в том, чтобы интерпретировать **хэш** как запись, где ключи хэша представляют собой имена полей записи, а ассоциированные величины — значения этих полей.

Например, запись "человек" может выглядеть так:

```
$Nat = { "Name"      =>  "Leonhard Euler",
                         "1729 Ramanujan Lane\nMathworld, PI 31416",
         "Birthday" => 0x5bb5580,
       };
```

Поскольку ссылка $NAT является скалярной величиной, ее можно сохранить в элементе хэша или массива с информацией о целой группе людей и далее использовать приемы сортировки, объединения хэшей, выбора случайных записей и т. д., рассмотренные в главах 4 и 5.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Атрибуты записи, в том числе и "человека" из нашего примера, всегда являются скалярами. Конечно, вместо строк можно использовать числа, но это банально. Настоящие возможности открываются в том случае, если атрибуты записи также представляют собой ссылки. Например, атрибут "Birthday" может храниться в виде анонимного массива,состоящего из трех элементов: день, месяц и год. Выражение `$person->{"BIrthday"}->[0]` выделяет из даты рождения поле "день". Дата также может быть представлена в виде хэша, для доступа к полям которого применяются выражения вида `$person->{"Birthday"}->{"day"}`. После включения ссылок в коллекцию приемов перед вами откроются многие нетривиальные и полезные стратегии программирования.

На этом этапе мы концептуально выходим за пределы простых записей и переходим к созданию сложных структур, которые представляют запутанные отношения между хранящимися в них данных. Хотя они *могут* использоваться для реализации традиционных структур данных (например, связанныхсписков), рецепты второй части этой главы не связаны ни с какими конкретными структурами. В них описываются обобщенные приемы загрузки, печати, копирования и сохранения обобщенных структур данных. Завершающая программа этой главы демонстрирует работу с бинарными деревьями.

Смотри также
perllol(1); *perldsc*(1).

11.1. Ссылки на массивы

Проблема

Требуется работать с массивом через ссылку.

Решение

Ссылка на массив создается следующим образом:

```
                        \@array;
$anon_array          = [1, 3, 5, 7, 9];
$anon_copy           = [ @array ];
@$implicit_creation = (2, 4, 6, 8, 10);
```

Чтобы разыменовать ссылку на массив, поставьте перед ней символ @:

```
push(@$anon_array, 11);
```

Или воспользуйтесь стрелкой с указанием индекса конкретного элемента в квадратных скобках:

Для получения индекса последнего элемента массива по ссылке или определения количества элементов в массиве применяется следующая запись:

```
$last_idx   = $#$aref;
$num_items = @$aref;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Дополнительные фигурные скобки повышают надежность и форсируют нужный контекст:

```
$last_idx  = $#{
$num_items = scalar @{
```

Комментарий

Рассмотрим примеры использования ссылок на массивы:

```
# Проверить, содержит ли $someref ссылку на массив
if (ref($someref) ne 'ARRAY') {
    die "Expected an array                    $someref\n";
}

print "@{$array_ref}\n";          # Вывести исходные данные

@order = sort @{                    Отсортировать их

push @{                  $item;     # Добавить в массив новый элемент
```

Если **вы** не можете выбрать между использованием ссылки на именованный массив и созданием нового **массива,** существует простое правило, которое **в** большинстве случаев оказывается верным. Получение ссылки на существующий массив используется либо для возврата ссылки за пределы области действия, либо при передаче массива функции по ссылке. Практически во всех остальных случаях используется `[@array]`, что **приводит к созданию** ссылки на новый массив с копиями старых значений.

Автоматический подсчет ссылок **в** сочетании с оператором \ обладает большими возможностями:

```
    my @array;
          \@array;
}
```

При каждом вызове функция выделяет для `@array` новый блок памяти. Если **бы мы** не вернули ссылку на `@array`, то занимаемая массивом память была **бы** возвращена при выходе из блока, то есть при завершении подпрограммы. Однако ссылка на `@array` продолжает существовать, поэтому Perl не освобождает память, и мы получаем ссылку на блок памяти, недоступный через таблицу символов. Такие блоки памяти называются ***анонимными,*** поскольку с ними не связано никакое имя.

К определенному элементу массива, на который указывает ссылка можно обратиться в форме делает то же самое и обладает большей наглядностью.

```
print $array_ref->[$N];          " Обращение к N-му элементу (лучший вариант)
print                            #          менее наглядно
print ${$array_ref}[$N];         # То же, но непонятно и уродливо
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Имея ссылку на массив, можно получить срез субъектного массива;

```
@$pie[3..5];                    # Срез массива, но читается плохо
@{$pie}[3..5];                  # Срез массива, читается лучше (?)
```

Срезы массивов, даже при обращении через ссылки, допускают присваивание. В следующей строке сначала происходит разыменование массива, после чего элементам среза присваиваются значения:

```
@{$pie}[3..5] = ("blackberry", "blueberry", "pumpkin");
```

Срез массива полностью идентичен списку отдельных элементов. Поскольку ссылку на список получить нельзя, вам не удастся получить ссылку на срез массива:

```
\@{$pie}[3..5];        НЕВЕРНО!
```

Для перебора в массиве применяется цикл `foreach` или `for`:

```
foreach $item ( @{$array_ref} ) {
    # Данные в $item
>

    ($idx              $#{              }; $idx++) {
    # Данные в $array_ref->[$idx]
}
```

Смотри также

perllol(1); рецепты 2.14; 4.5.

11.2. Создание хэшей массивов

Проблема

С каждым ключом хэша может быть ассоциирована лишь одна скалярная величина, однако вам хочется использовать один ключ для хранения и извлечения нескольких величин. Иначе говоря, вы хотите, чтобы ассоциированное значение представляло собой список.

Решение

Сохраните в элементе хэша ссылку на массив. Используйте `push` для присоединения новых элементов:

```
push(@{ $hash{"KEYNAME"} }, "new value");
```

Затем при выводе хэша разыменуйте значение как ссылку на массив:

```
        $string (keys %hash) {
    print "$string: @{$hash{$string}}\n";
```

Комментарий

В хэше могут храниться только скалярные величины. Впрочем, ссылки и являются скалярными величинами. Они помогают решить проблему сохранения не-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

скольких ассоциированных значений с одним ключом — в $hash{$key} помещается ссылка на массив, содержащий значения $key. Все стандартные операции с хэшами (вставка, удаление, перебор и проверка существования) могут комбинироваться с операциями массивов (push, splice

Присвоение ключу нескольких значений осуществляется следующим об-'разом:

```
$hash{"a key"} = [ 3, 4, 5 ];        # Анонимныймассив
```

Ключ с ассоциированным массивом используется так:

```
@values = @{ $hash{"a key"} };
```

Для присоединения новых значений к массиву, ассоциированному с конкретным ключом, используется функция push:

```
push @{ $hash{"a key"} }, $value;
```

Классическое применение этой структуры данных — инвертирование хэша, в котором одно значение ассоциируется с несколькими ключами. В хэше, полученном после инвертирования, один ключ ассоциирован с несколькими значениями. Эта проблема рассматривается в рецепте 5.8.

Учтите, что запись вида:

```
@residents = @{ $phone2name{$number} };
```

при действующей директиве use strict вызовет исключение, поскольку **вы** пытаетесь разыменовать неопределенную ссылку без автоматического создания. Приходится использовать другую формулировку:

```
@residents = exists( $phone2name{$number} )
             ? @{ $phone2name{$number} }
             : ();>
```

Смотри также

Раздел "Hashes ofArrays" *perldsc*(1); рецепт **5.8;** пример **"Хэш** с автоматическим дополнением" из рецепта **13.15.**

11.3. Получение ссылок на хэши

Проблема

Требуется работать с хэшем по ссылке. Например, ссылка может передаваться функции или входить во внешнюю структуру данных.

Решение

Получение ссылки на **хэш:**

```
        \%hash;
$anon_hash = { "key1" => "value1", "key2" => "value2 ..." };
$anon_hash_copy = { %hash };
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Разыменование ссылки на хэш:

```
%hash  = %$href,
$value = $href->{$key},
@slice = @$href{$key1, $key2, $key3},  # Обратите внимание  стрелки нет!
@keys  = keys %$hash,
```

Проверка того, является ли переменная ссылкой на хэш:

```
if (ref($someref) ne HASH ) {
    die Expected a has                $someref\n
}
```

Комментарий

Следующий пример выводит все ключи и значения двух заранее определенных хэшей:

```
foreach $href ( \%ENV, \%INC ) {    # ИЛИ  for $href( \(%ENV,%INC)) {
            $key ( keys %$href ) {
        print  $key =>                  ,
    }
}
```

Операции со срезами хэшей по ссылке выполняются так же, как со срезами массивов. Например:

```
@values              key1 ,  key2 ,  key3 },

for $val (@$hash_ref{ key1 ,  key2 ,  key3 }) {
    $val += 7,   # Прибавить 7 ккаждому значению в срезе хэша
}
```

Смотри также

Глава 5 «Хэши»; *perlref*(1); рецепт 11.9.

11.4. Получение ссылок на функции

Проблема

Требуется создать ссылку для вызова подпрограммы. Такая задача возникает при создании обработчиков сигналов, косвенно-вызываемых функций Tk и указателей на хэши функций.

Решение

Получение ссылки на функцию:

```
\&func,
```

Вызов функции по ссылке.

```
@returned = $cref->(@arguments),
            &$cref(@arguments),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Чтобы получить ссылку на функцию, достаточно снабдить ее имя префиксом \&. Кроме того, формулировка sub { } позволяет создавать анонимные функции. Ссылка на анонимную функцию может быть сохранена так же, как и любая другая.

В Perl **5.004** появилась постфиксная запись для разыменования ссылок на функции. Чтобы вызвать функцию по ссылке, раньше приходилось писать `&$funcname(@ARGS)`, где `$funcname` — имя функции. Возможность сохранить имя функции в переменной осталась и сейчас:

```
$funcname = "thefunc";
&$funcname();
```

однако подобное решение нежелательно по нескольким причинам. Во-первых, в нем используются символические, а не настоящие (жесткие) ссылки, поэтому при действующей директиве use strict 'refs' оно отпадает. Символические ссылки обычно не рекомендуются, поскольку они не могут обращаться к лексическим, а только к глобальным переменным, и для них не ведется подсчет ссылок.

Во-вторых, оно не содержит данных о пакете, поэтому выполнение фрагмента в другом пакете может привести к вызову неверной функции. Наконец, если функция была в какой-то момент переопределена (хотя это происходит нечасто), символическая ссылка будет обращаться к текущему определению функции, а жесткая ссылка сохранит старое определение.

Вместо того чтобы сохранять имя функции в переменной, создайте ссылку на нее с помощью оператора \. Именно так следует сохранять функцию в переменной или передавать ее другой функции. Ссылки на именованные функции можно комбинировать со ссылками на анонимные функции:

```
my %commands = (
    "happy" => \&joy,
    "sad"   => \&sullen,
    "done"  => sub { die "See ya!" },
    "mad"   => \&angry,
);

print "How are you? ";
chomp($string = <STDIN>);
if ($commands{$string}) {
    $commands{$string}->();
} else {
    print "No such command: $string\n";
}
```

Если вы создаете анонимную функцию, которая ссылается на лексическую (my) переменную из вмещающей области действия, схема подсчета ссылок гарантирует, что память лексической переменной не будет освобождена при наличии ссылок на нее:

```
sub counter_maker {
    my $start = 0;
                                        9 Замыкание
                $start++;               # Лексическая переменная
                                         й из вмещающей области действия
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    };
}

$counter = counter_maker();

for ($i = 0; $i < 5; $i ++) {
    print &$counter, "\n";
}
```

Даже несмотря на то что функция counter_maker завершилась, а переменная $start вышла из области действия, Perl не освобождает ее, поскольку анонимная подпрограмма (на которую ссылается $counter) все еще содержит ссылку на $start. Если повторно вызвать counter_maker, функция вернет ссылку на другую анонимную подпрограмму, использующую *другое* значение $start:

```
$counter1 = counter_maker();
$counter2 = counter_maker();

for ($i = 0; $i < 5; $i ++) {
    print &$counter1, "\n";
}

print &$counter1, " ", &$counter2, "\n";
0
1
2
3
4
5 0
```

Замыкания часто используются в косвенно-вызываемых функциях (callbacks). В графических интерфейсах и вообще в программировании, основанном на событиях, определенные фрагменты кода связываются с событиями нажатий клавиш, щелчков мышью, отображения окон и т. д. Этот код вызывается много позже, возможно, из совсем другой области действия. Переменные, используемые в замыкании, должны быть доступными к моменту вызова. Для нормальной работы они должны быть лексическими, а не глобальными.

Замыкания также используются в генераторах функций, то есть в функциях, которые создают и возвращают другие функции. Функция counter_maker является генератором. Приведем еще один простой пример:

```
sub timestamp {
    my $start_time = time();
                        time()   $start time };
}
$early = timestamp();
sleep 20;
$later = timestamp();
sleep 10;
printf "It's been %d seconds since early.\n", $early->();
printf "It's been %d seconds since later.\n", $later->();
It's been 30 seconds since early.
It's been 10 seconds since later.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Каждый вызов `timestamp` генерирует и возвращает новую функцию. Функция `timestamp` создает лексическую переменную `$start_time`, которая содержит текущее время (в секундах с начала эпохи). При каждом вызове замыкания оно возвращает количество прошедших секунд, которое определяется вычитанием начального времени из текущего.

Смотри также

Описание замыканий в *perlref*(1); рецепты 10.11; 11.4.

11.5. Получение ссылок на скаляры

Проблема

Требуется создать ссылку на скалярную величину и работать с ней.

Решение

Для создания ссылки на скалярную величину воспользуйтесь оператором \:

```
\$scalar;          Получение ссылки на именованный скаляр
```

Чтобы создать ссылку на анонимную скалярную величину (то есть скаляр, не являющийся переменной), присвойте нужное значение через разыменование неопределенной переменной:

```
undef
```

Ссылка на скалярную константу создается следующим образом:

```
=
```

Разыменование выполняется конструкцией ${...}:

```
print ${          };          # Разыменовать
          .= "string";  # Изменить значение субъекта
```

Комментарий

Если вам понадобилось создать много новых анонимных скаляров, воспользуйтесь функцией, возвращающей ссылку на лексическую переменную вне области действия, как объяснялось во введении:

```
sub new_anon_scalar {
    my $temp;
          \$temp;
}
```

Perl почти никогда не выполняет косвенного разыменования. Исключение составляют ссылки на файловые манипуляторы, программные ссылки на sort и ссылочный аргумент функции bless. Из-за этого для разыменования скалярной переменной следует снабдить ее префиксом $, чтобы получить все ее содержимое:

```
        new_anon_scalar();

print          $$sref\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
@array_of_srefs = ( new_anon_scalar(), new_anon_scalar() );
${ $array[0] } = 6.02e23;
${ $array[1] } = "avocado";
print "\@array contains: ", join(", ", map { $$_ } @array ), "\n";
```

Обратите внимание на фигурные скобки вокруг $array[0] и $array[1]. Если бы мы попытались ограничиться простым$$array[0], то впроцессе разыменования получили бы $array->[0]. Переменная $array интерпретировалась бы как ссылка на массив, поэтому в результате был бы возвращен элемент с нулевым индексом.

Приведем другие примеры, в которых фигурные скобки необязательны:

```
$var          = `uptime`;      # $var содержит текст
                                   "указывает на" $var
                /load/)  {}    # Косвенное обращение к $var
chomp                            Косвенное изменение $var
```

Как упоминалось во введении, для определения типа субъекта по ссылке применяется встроенная функция ref. При вызове ref для ссылки на скаляр возвращается строка "SCALAR":

```
# Проверить, содержит ли $someref ссылку на скаляр
if (ref($someref) ne 'SCALAR') {
    die "Expected a scalar                $someref\n";
}
```

Смотри также

11.6. Создание массивов ссылок на скаляры

Проблема

Требуется создать массив ссылок на скаляры. Такая задача часто возникает при передаче функциям переменных по ссылке, чтобы функция могла изменить их значения.

Решение

Чтобы создать массив, либо снабдите префиксом \ каждый скаляр в списке:

```
@array_of_scalar_refs = ( \$a, \$b );
```

либо просто поставьте \ перед всем списком, используя свойство дистрибутивности оператора \:

```
@array_of_scalar_refs = \( $a, $b );
```

Чтобы получить или задать значение элемента списка, воспользуйтесь конструкцией ${...}:

```
${ $array_of_scalar_refs[1] } = 12;        # $b = 12
```

Комментарий

В следующих примерах предполагается, что @array — простой массив, содержащий ссылки на скаляры (не путайте массив ссылок со ссылкой на массив). При косвенных обращениях к данным необходимы фигурные скобки.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
($a, $b, $c, $d) = (1 .. 4);          # Инициализировать
@array = (\$a, \$b, \$c, \$d);        # Ссылки на все скаляры
@array = \( $a, $b, $c, $d);          # То же самое!

${ $array[2] } += 9;                  # $c = 12

${ $array[ $#array ] } *= 5;          # $d = 20
${ $array[-1] }        *= 5;          # То же; $d = 100

$tmp   = $array[-1];                  # Использование временной переменной
$$tmp *= 5;                           # $d = 500
```

Две формы присваивания @array эквивалентны — оператор \ обладает свойством дистрибутивности. Следовательно, \ перед списком (но не массивом!) эквивалентно применению \ к каждому элементу списка. Следующий фрагмент изменяет значения переменных, ссылки на которые хранятся в массиве.

А вот как работать с массивом без явного индексирования.

```
use Math::Trig qw(pi);                # Загрузить константу pi
                (@array)                    товиться к изменению $a,$b,$c,$d
                                      Заменить обьемом сферы
}
```

В этом фрагменте используется формула вычисления объема сферы:

$V - 4/3\pi\rho^3$.

Переменная цикла перебирает все ссылки @array, а в заносятся сами числа, то есть исходные переменные $a, $b, $c и $d. Изменение цикле приводит к изменению этих переменных. Сначала мы возводим затем умножаем полученный результат на $4/3\pi$. При этом используется то обстоятельство, что присваивание в Perl возвращает левостороннее выражение. Это позволяет сцеплять операторы присваивания, как это делается с операторами **= и *=.

Вообще говоря, анонимные скаляры обычно бесполезны — ведь скалярная величина занимает столько же места, что и ссылка на нее. По этой причине не предусмотрены и специальные конструкции для их создания. Скалярные ссылки существуют только для поддержки синонимов, которые могут быть реализованы и другими способами.

Смотри также

Раздел "Assignment Operators» *perlop*(1).

11.7. Применение замыканий вместо объектов

Проблема

Вы хотите работать сзаписями, обладающими определенным состоянием, поведением и идентичностью, но вам не хочется изучать для этого объектно-ориентированное программирование.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Напишите функцию, которая возвращает (по ссылке) хэш ссылок на фрагменты кода. Все эти фрагменты представляют собой замыкания, созданные в общей области действия, поэтому при выполнении они будут совместно использовать одни и те же закрытые переменные.

Комментарий

Поскольку замыкание представляет собой совокупность кода и данных, одна из реализаций позволяет имитировать поведение объекта.

Следующий пример создает и возвращает хэш анонимных функций. Функция mkcounter получает начальное значение счетчика и возвращает ссылку, позволяющую косвенно оперировать им.

```
$c1 = mkcounter(20);
$c2 = mkcounter(77);

printf "next c1: %d\n", $c1->{NEXT}->();  # 21
printf "next c2: %d\n", $c2->{NEXT}->();  # 78
printf "next c1: %d\n", $c1->{NEXT}->();  # 22
printf "last c1: %d\n", $c1->{PREV}->();  # 21
printf "old  c2: %d\n", $c2->{RESET>->(); # 77
```

Каждая ссылка на хэш, $c1 и $c2, отдельно хранит информацию о своем состоянии. Реализация выглядит так:

```
sub mkcounter {
    my $count  = shift;
    my $start  = $count;
    my $bundle = {
                                  ++$count  },
        "PREV"                    --$count  },

        "SET"    => sub {  $count = shift   },
        "BUMP"   => sub {  $count += shift '},
        "RESET"  => sub {  $count = $start  },
    };
    $bundle->{"LAST"} = $bundle->{"PREV"};
            $bundle;
}
```

Поскольку лексические переменные, используемые замыканиями в ссылке на хэш $bundle, используются функцией, они не освобождаются. При следующем вызове mkcounter замыкания получают другой набор привязок переменных для того же кода. Никто не сможет обратиться к этим двум переменным за пределами замыканий, поэтому полная инкапсуляция гарантирована.

В результате присваивания, расположенного непосредственно перед return, значения "prev" и "last" будут ссылаться на одно и то же замыкание. Если вы разбираетесь в объектно-ориентированном программировании, можете считать их двумя разными сообщениями, реализованными с применением одного метода.

Возвращаемая нами совокупность не является полноценным объектом, поскольку не поддерживает наследования и полиморфизма (пока). Однако она несом-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ненно обладает собственным состоянием, поведением и идентификацией, а также обеспечивает инкапсуляцию.

Смотри также

Замыкания рассматриваются в *perlref*(1). Также см. главу 13, рецепты 10.11; 11.4.

11.8. Создание ссылок на методы

Проблема

Требуется сохранить ссылку на метод.

Решение

Создайте замыкание, обеспечивающее вызов нужного метода для объекта.

Комментарий

Ссылка на метод — это нечто большее, чем простой указатель на функцию. Вам также придется определить, для какого объекта вызывается метод, поскольку исходные данные для работы метода содержатся в объекте. Оптимальным решением будет использование замыкания. Если переменная $obj имеет лексическую область действия, воспользуйтесь следующим фрагментом:

```
$mref = sub { $obj->meth(@_) };
# Позднее...
$mref->("args", "go", "here");
```

Даже когда переменная $obj выходит из области действия, она остается в замыкании, хранящемся в $mref. Позднее при косвенном вызове метода будет использован правильный объект.

Учтите, что формулировка:

```
\$obj->meth;
```

работает не так, как можно предположить. Сначала она вызывает метод объекта, а затем дает ссылку либо на возвращаемое значение, либо на последнее из возвращаемых значений, если метод возвращает список.

Метод can из базового класса UNIVERSAL выглядит заманчиво, но вряд ли делает именно то, что вы хотите:

```
= $obj->can("meth");
```

Он дает ссылку на код соответствующего метода (если он будет найден), не несущую информации об объекте. В сущности, вы получаете обычный указатель на функцию. Информация об объекте теряется. Из-за этого и понадобилось замыкание, запоминающее как состояние объекта, так и вызываемый метод.

Смотри также

Описание методов во введении к главе 13; рецепты 11.7; 13.7.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

11.9. Конструирование записей

Проблема

Требуется создать тип данных для хранения атрибутов (запись).

Решение

Воспользуйтесь ссылкой на анонимный хэш.

Комментарий

Предположим, вам захотелось создать тип данных, содержащий различные атрибуты — аналог структур C или записей Pascal. Проще всего сделать это с помощью анонимного хэша. Следующий пример демонстрирует процесс инициализации и применения записи, содержащей информацию о работнике фирмы:

```
    NAME   =>  Jason  ,
    EMPNO  =>  132
    TITLE  =>  deputy peon  ,
    AGE    => 23,
    SALARY => 37_000,
    PALS   => [  Norbert  ,  Rhys  ,  Phineas  ],
},

printf 'I am %s, and my pals are %s \n ,

    join( ,  , @{$record->{PALS}}),
```

Впрочем, от отдельной записи толку мало — хотелось бы построить структуры данных более высокого уровня. Например, можно создать хэш %ByName, а затем инициализировать и использовать его следующим образом:

```
# Сохранить запись
$byname{ $record->{NAME}

# Позднее искать по имени
if ($rp = $byname{ Aron }) {        # false, если отсутствует missing
    printf  Aron is employee %d \n  ,  $rp->{EMPNO},


# Дать Джейсону нового друга
push @{$byname{ Jason }->{PALS}},
printf  Jason now has %d pals\n  ,  scalar @{$byname{ Jason }->{PALS}},
```

В результате %byname превращается в хэш хэшей, поскольку хранящиеся в нем значения представляют собой ссылки на хэши. Поиск работника по имени с применением такой структуры оказывается простой задачей. Если значение найдено в хэше, мы сохраняем ссылку на запись во временной переменной $rp, с помощью которой далее можно получить любое нужное поле.

ДЛЯ операций С %byname можно использовать стандартные средства работы с хэшами. Например, итератор each организует перебор элементов в произвольном порядке:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Перебор всех записей
while (($name,          = each %byname) {
    printf "%s is employee number %d\n", $name,
}
```

А как насчет поиска работников по номеру? Достаточно построить другую структуру данных — массив хэшей @employees. Если работники нумеруются непоследовательно (скажем, после 1 следует номер 159997), выбор массива окажется неудачным. Вместо этого следует воспользоваться **хэшем**, в котором номер работника ассоциируется с записью. Для последовательной нумерации подойдет и массив:

```
# Сохранить запись
$employees[

# Поиск по номеру
if ($rp = $employee[132]) {
    printf "employee number 132 is %s\n", $rp->{NAME};
}
```

При работе с подобными структурами данных обновление записи в одном месте обновляет ее везде. Например, следующая команда повышает жалование Джейсона на 3,5 %:

```
$byname{"Jason"}->{SALARY} *= 1.035;
```

Внесенные изменения отражаются во всех представлениях этих записей. Помните о том, что $byname{"Jason"} и $employees[132] ссылаются на одну и ту же запись, поскольку хранящиеся в них ссылки относятся к одному анонимному хэшу.

Как отобрать все записи, удовлетворяющие некоторому критерию? Для этого и была создана функция Например, в следующем фрагменте отбираются два подмножества записей — работников, чья должность содержит слово "peon", и тех, чей возраст равен 27 годам.

```
@peons =        $_->{TITLE} =~ /peon/i } @employees;
              $_->{AGE}   == 27 >      @employees;
```

Каждый элемент @peons и @tsevens представляет собой ссылку на запись, поэтому они, как и @employees, являются массивами хэшей.

Вывод записей в определенном порядке (например, по возрасту) выполняется так:

```
# Перебрать все записи
        $rp (sort { $a->{AGE} <=> $b->{AGE} } values %byname) {
    printf "%s is age %d.\n", $rp->{NAME}, $rp->{AGE};
    " или со срезом хэша через ссылку
    printf "%s is employee number %d.\n", @$rp{'NAME','EMPNO'};
}
```

Вместо того чтобы тратить время на сортировку по возрасту, можно просто создать для этих записей другое представление, @byage. Каждый элемент массива (например, $byage[27]) является массивом всех записей с данным возрастом. Фактически мы получаем массив массивов хэшей. Он строится так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Используем @byage, массив массивов записей
push @{ $byage[
```

Далее отбор осуществляется следующим образом:

```
for ($age = 0; $age <= $#byage; $age++), {
    next unless $byage[$age];
    print "Age $age: ";
    foreach $rp (@{$byage[$age]}) {
        print $rp->{NAME}, " ";
    }
    print "\n";
}
```

Аналогичное решение заключается в применении map, что позволяет избежать цикла

```
for ($age = 0; $age <= $#byage; $age++) {
    next unless $byage[$age];
    printf "Age %d: %s\n", $age,
        join(", ", map {$_->{NAME}} @{$byage[$age]});
}
```

Смотри также

Рецепты 4.13; 11.3.

11.10. Чтение и сохранение записей в текстовых файлах

Проблема

Требуется прочитать или сохранить хэш записи в текстовом файле.

Решение

Воспользуйтесь простым форматом, при котором каждое поле занимает отдельную строку вида:

```
ИмяПоля:Значение
```

и разделяйте записи пустыми строкайи.

Комментарий

Если у вас имеется массив записей, которые должны сохраняться в текстовом файле и читаться из него, воспользуйтесь простым форматом, основанным на заголовках почтовых сообщений. Из-за простоты формата ключи не йогут быть двоеточиями и переводами строк, а значения — переводами строк.

Следующий фрагмент записывает данные в файл:

```
                (@Array_of_Records)
for $key (sort keys %$record) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        print "$key: $record->{$key}\n";
    }
    print "\n";
}
```

Прочитать записи из файла тоже несложно:

```
$/ = "";                    # Режим чтения абзацев
while (o) {
    my @fields = { split /^([^:]+):\s*/m };
    shift @fields;          # Удалить начальное пустое поле
    push(@Array_of_Records, { @fields });
}
```

Функция `split` работает с `$_`, своим вторым аргументом по умолчанию, в котором находится прочитанный абзац. Шаблон ищет начало строки (не просто начало записи благодаря `/m`), за которым следует один или более'символов, не являющихся двоеточиями, затем двоеточие и необязательный пропуск. Если шаблон split содержит скобки, они возвращаются вместе со значениями. Возвращаемые значения заносятся в `@fileds` в порядке "ключ/значение"; пустое поле в начале убирается. Фигурные скобки в вызове `push` создают ссылку на новый анонимный хэш, куда копируется содержимое `@fields`. Поскольку в массиве сохранился порядок "ключ/значение", мы получаем правильно упорядоченное содержимое хэша.

Все происходящее сводится к операциям чтения и записи простого текстового файла, поэтому вы можете воспользоваться другими рецептами. Рецепт 7.11 поможет правильно организовать параллельный доступ. В рецепте 1.13 рассказано о сохранении в ключах и значениях двоеточий и переводов строк, а в рецепте 11.3 — о сохранении более сложных структур.

Если вы готовы пожертвовать элегантностью простого текстового файла в пользу быстрой базы данных с произвольным доступом, воспользуйтесь DBM-файлом (см. рецепт 11.14).

Смотри также

Описание функции split *в perlfunc(1);* рецепты 11.9, 11.13—11.14.

11.11. Вывод структур данных

Проблема

Требуется вывести содержимое структуры данных.

Решение

Если важна наглядность вывода, напишите нестандартную процедуру вывода.

В отладчике Perl воспользуйтесь командой x:

```
DB<1>          = [ { "foo" => "bar" }, 3, sub { print "hello, world\n' } ];
DB<2> x
  0  ARRAY(Ox1d033c)
    0  HASH(Ox7b390)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    'foo' = 'bar'>
  1  3
  2  CODE(0x21e3e4)
     - & in ???>
```

В программе воспользуйтесь функцией `Dumper` модуля Data::Dumper от CPAN:

```
use Data::Dumper;
print Dumper($reference);
```

Комментарий

Иногда для вывода структур данных в определенном формате пишутся специальные функции, но это часто оказывается перебором. В отладчике Perl существуют команды `x` и `X`, обеспечивающие симпатичный вывод. Команда `x` полезнее, поскольку она работает с глобальными и лексическими переменными, а `X` — только с глобальными. Передайте `x` ссылку на выводимую структуру данных.

```
D<1> x \@INC
  0  ARRAY(0x807d0a8)
     0  '/home/tchrist/perllib'
     1  '/usr/lib/perl5/i686-linux/5.00403'
     2  '/usr/lib/perl5'
     3  '/usr/lib/perl5/site_perl/i686-linux'
     4  '/usr/lib/perl5/site_perl'
     5  '.'
```

Эти команды используют библиотеку *dumpvar.pl*. Рассмотрим пример:

```
{ package main;          "dumpvar.pl" }
*dumpvar = \&main::dumpvar if _    _PACKAGE_    _ ne 'main';
dumpvar("main", "INC");              # Выводит и @INC, и %INC
```

Библиотека *dumpvar.pl* не является модулем, но мы хотим использовать ее как модуль и поэтому заставляем импортировать функцию `dumpvar`. Первые две строки форсируют импортирование функции `main::dumpvar` из пакета `main` в текущий пакет, предполагая, что эти функции отличаются. Выходные данные будут выглядеть так:

```
@INC = (
    0  '/home/tchrist/perllib/i686-linux'
    1  '/home/tchrist/perllib'
    2  '/usr/lib/perl5/i686-linux/5.00404'
    3  '/usr/lib/perl5'
    4  '/usr/lib/perl5/site_perl/i686-linux'
    5  '/usr/lib/perl5/site_perl'
    6  '.'
)
%INC = (
    'dumpvar.pl'  =  '/usr/lib/perl5/i686-linux/5.00404/dumpvar.pl'
    'strict.pm'  =  '/usr/lib/perl5/i686-linux/5.00404/strict.pm'
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Модуль Data::Dumper, доступный на CPAN, предоставляет более гибкое решение. Входящая в него функция Dumper получает список ссылок и возвращает строку с выводимой (и пригодной для eval) формой этих ссылок.

```
use Data::Dumper;
print Dumper(\@INC);
$VAR1 = [
        '/home/tchrist/perllib',
         '/usr/lib/perl5/i686-linux/5.00403',
        '/usr/lib/perl5',
         '/usr/lib/perl5/site_perl/i686-linux',
         '/usr/lib/perl5/site_perl',
```

Смотри также

Документация по модулю Data::Dumper с CPAN; раздел "The Perl Debugger" *perldebug*(1).

11.12. Копирование структуры данных

Проблема

Требуется скопировать сложную структуру данных.

Решение

Воспользуйтесь функцией dclone модуля Storable от CPAN:

```
use Storable;

$r2 = dclone($r1);
```

Комментарий

Существуют два типа копирования, которые иногда путают. *Поверхностное копирование* (surface copy) ограничивается копированием ссылок без создания копий данных, на которые они ссылаются:

```
@original = ( \@a, \@b, \@c );
@surface = @original;
```

Глубокое копирование (deep copy) создает абсолютно новую структуру без перекрывающихся ссылок. Следующий фрагмент копирует ссылки на один уровень вглубь:

```
@deep = map { [ @$_ ] } @original;
```

Если переменные @a, @b и @c сами содержат ссылки, вызов map не решит всех проблем. Написание специального кода для глубокого копирования структур — дело трудоемкое и быстро надоедающее.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Модуль Storable, доступный на CPAN, содержит функцию dclone, которая обеспечивает рекурсивное копирование своего аргумента:

```
use Storable qw(dclone);
$r2 = dclone($r1);
```

Функция работает только со ссылками или приписанными к конкретному пакету (blessed) объектами типа SCALAR, ARRAY и HASH; ссылки на CODE, GLOB и IO и другие экзотические типы не поддерживаются. Функция saf

такую возможность для одного адресного пространства посредством использования кэша ссылок, который при некоторых обстоятельствах вмешивается в процесс сборки мусора и работу деструкторов объектов.

Поскольку dclone принимает и возвращает ссылки, при копировании хэша ссылок в нее приходится включать дополнительные символы:

```
%newhash = %{ dclone(\%oldhash) };
```

Смотри также

Документация по модулям CPAN.

11.13. Сохранение структур данных на диске

Проблема

Требуется сохранить большую, сложную структуру данных на диске, чтобы ее не пришлось заново строить при каждом запуске программы.

Решение

Воспользуйтесь функциями store и retrieve модуля Storable с CPAN:

```
use Storable;
store(\%hash, "filename");

# later on...
$href = retrieve("filename");        # По ссылке
%hash = %{ retrieve("filename") };   # Прямо в хэш
```

Комментарий

Модуль Storable использует функции C и двоичный формат для обхода внутренних структур данных Perl и описания данных. По сравнению со строковой реализацией сохранения записей в Perl такой вариант работает эффективнее, однако он менее надежен.

Функции store и retrieve предполагают, что в передаваемых двоичных данных используется порядок байтов, стандартный для данного компьютера. Это означает, что созданные этими функциями файлы нельзя передавать между различными архитектурами. Функция nstore делает то же, что и сохраняет дан-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ные в каноническом (сетевом) порядке. Быстродействие при этом несколько снижается:

```
use Storable qw(nstore),
nstore(\%hash, filename ),
" Позднее
      = retrieve( filename ),
```

Независимо от того, какая функция сохраняла данные — store Или nstore, для их восстановления в памяти используется одна и та же функция retrieve. О переносимости должен заботиться создатель данных, а не их потребитель. Если создатель изменит свое решение, ему достаточно изменить программу всего в одном месте. Тем самым обеспечивается последовательный интерфейс со стороны потребителя, который ничего не знает об этих изменениях.

Функции store и блокируют файлы, с которыми они работают. Если вас беспокоят проблемы параллельного доступа, откройте файл самостоятельно, заблокируйте его (см. рецепт 7.11) и воспользуйтесь функцией store_fd или более медленной, но независимой от платформы версией,

Следующий фрагмент сохраняет хэш в файле с установкойблокировки. При открытии файла не используется флаг O_TRUNC, поскольку до стирания содержимого нам приходится ждать получения блокировки.

```
use Storable qw(nstore_fd)
use Fcntl qw( DEFAULT flock),
sysopen(DF, /tmp/datafile , O_RDWR|O_CREAT 0666)
  or die "can't open /tmp/datafile $! ,
flock(DF, LOCK_EX) or die "can't lock /tmp/datafile $! ,
nstore_fd(\%hash, "DF)
                        hash\n ,
truncate(DF,tell(DF)),
close(DF);
```

Другой фрагмент восстанавливает хэш из файла, также с применением блокировки:

```
use Storable;
use Fcntl qw( DEFAULT flock),
open(DF, < /tmp/datafile )      or die "can't open /tmp/datafile $! ,
flock(DF, LOCK_SH)              or die can t lock /tmp/datafile $! ,

close(DF);
```

Внимательное применение этой стратегии позволяет эффективно передавать большие объекты данных между процессами, поскольку файловый манипулятор канала или сокета представляет собой байтовый поток, похожий на обычный файл.

В отличие от связей с различными реализациями DBM, модуль Storable не ограничивается одними хэшами (или массивами, как DB_File). На диске могут сохраняться произвольные структуры данных. Вся структура должна читаться или записываться полностью.

Смотри также

Рецепт 11.14.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

11.14. Устойчивые структуры данных

Проблема

Существует сложная структура данных, которую требуется сделать устойчивой (persistent)[1].

Решение

Воспользуйтесь модулем MLDBM и либо DB_File (предпочтительно), либо GDBM_File:

```
use MLDBM qw(DB_File);
use Fcntl;

tie(%hash, 'MLDBM', 'testfile.db', O_CREAT|O_RDWR, 0666)
    or die "can't open tie to testfile.db: $!";

...  Операции с %hash

untie %hash;
```

Комментарий

Конечно, построение хэша из 100 000 элементов займет немало времени. Сохранение его на диске (вручную или с помощью Storable) также потребует немалых расходов памяти и вычислительных ресурсов.

Модули DBM решают эту проблему посредством связывания хэшей с файлами баз данных на диске. Вместо того чтобы читать всю структуру сразу, они извлекают данные лишь при необходимости. Для пользователя все выглядит так, словно состояние хэша сохраняется между вызовами программы.

К сожалению, значения устойчивого хэша должны представлять собой простые строки. Вам не удастся легко использовать базу данных для хранения хэша хэшей, хэша массивов и т. д. — только хэши строк.

Однако модуль MLDBM с CPAN позволяет сохранять ссылки в базе данных. Преобразование ссылок в строки для внешнего хранения осуществляется с помощью Data::Dumper:

```
use MLDBM qw(DB_File);
use Fcntl;
tie(%hash, 'MLDBM', 'testfile.db', O_CREAT|O_RDWR, 0666)
    or die "can't open tie to testfile.db: $!";
```

Теперь %hash может использоваться для выборки или сохранения сложных записей на диске. Единственный недостаток заключается в том, что к ссылкам нельзя обращаться напрямую. Приходится извлекать ссылку из базы, работать с ней, а затем снова сохранять в базе.

```
# Не будет работать!
$hash{"some key"}[4] = "fred";
```

[1] Термин "устойчивость" означает сохранение состояния между запусками программы. Также встречается термин "перманентность". — *Примеч. перев.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# ПРАВИЛЬНО
        $hash{"some key"};
           = "fred";
$hash{"some key"}
```

Смотри также

Документация по модулю MLDBM с CPAN; рецепты 14.1; 14.7; 14.11.

11.15. Программа: бинарные деревья

Встроенные типы данных Perl представляют собой мощные, динамические структуры. В большинстве программ этих стандартных возможностей оказывается вполне достаточно. Для выполнения простого поиска почти всегда следует использовать простые хэши. Как выразился Ларри, "Весь фокус в том, чтобы использовать сильные, а не слабые стороны Perl".

Однако хэши не обладают внутренним упорядочиванием элементов. Чтобы перебрать элементы хэша в определенном порядке, необходимо сначала извлечь ключи, а затем отсортировать их. При многократном выполнении это может отразиться на быстродействии программы, что, однако, вряд ли оправдывает затраты времени на разработку хитроумного алгоритма.

Древовидные структуры обеспечивают упорядоченный перебор. Как реализовать дерево на Perl? Для начала загляните в свой любимый учебник по структурам данных. Воспользуйтесь анонимным хэшем для представления каждого узла дерева и переведите алгоритмы, изложенные в книге, на Perl. Обычно это задача оказывается проще, чем кажется.

Программа в примере 11.1 демонстрирует простую реализацию бинарного дерева, построенную на базе анонимных хэшей. Каждый узел состоит из трех полей: левый потомок, правый потомок и значение. Важнейшее свойство упорядоченных бинарных деревьев заключается в том, что значение левого потомка всегда меньше, чем значение текущего узла, а значение правого потомка всегда больше.

Основная программа выполняет три операции. Сначала она создает дерево с 20 случайными узлами, затем выводит три варианта обхода узлов дерева и, наконец, запрашивает у пользователя ключ и сообщает, присутствует ли этот ключ в дереве.

Функция insert использует механизм неявной передачи скаляров по ссылке для инициализации пустого дерева при вставке пустого узла. Присваивание $_[0] созданного узла приводит к изменению значения на вызывающей стороне.

Хотя такая структура данных занимает гораздо больше памяти, чем простой хэш, и обычный перебор элементов в ней происходит медленнее, упорядоченные перемещения выполняются быстрее.

Исходный текст программы приведен в примере 11.1.

Пример 11.1.

```
#!/usr/bin/perl -w
#          пример работы с бинарным деревом
use strict;
my($root, $n);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Сгенерировать 20 случайных узлов
while ($n++ < 20) { insert($root, int(rand(1000)) }

# Вывести узлы дерева в трех разных порядках
print "Pre order:  ";                    print "\n";
print "In order:   ";  in_order($root);   print "\n";
print "Post order: ";  post_order($root); print "\n";

ft Запрашивать до получения EOF
for (print "Search? "; <>; print "Search? ") {
    chomp;
    my $found = search($root, $_);
    if ($found) { print "Found $_ at $found, $found->{VALUE}\n" }
    else        { print "No $_ in tree\n" }
```

```
# Функция вставляет передаваемое значение в правильную позицию
# передаваемого дерева. Если дерево не передается,
# для @_ используется механизм косвенной передачи по ссылке,
# что приводит к созданию дерева на вызывающей стороне.
sub insert {
    my($tree, $value) = @_;
    unless
                                                    новый узел
        $tree->{VALUE}  = $value;
        $tree->{LEFT}   = undef;
        $tree->{RIGHT}  = undef;
        $_[0]  =                        $_[0]  - ссылочный параметр!

    }
    if    ($tree->{VALUE} > $value) { insert($tree->{LEFT}, $value) >
    elsif ($tree->{VALUE} < $value) { insert($tree->{RIGHT}, value)  }
    else                            { warn "dup insert of $value\n"  }
                                    # XXX: узлы не должны повторяться
```

```
# Рекурсия по левому потомку,
# вывод текущего значения
# и рекурсия по правому потомку.
sub in_order {
    my($tree) = @_;
          unless
    in_order($tree->{LEFT});
    print $tree->{VALUE}, " ";
    in_order($tree->{RIGHT});
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 11.1 (продолжение)

```
# Вывод текущего значения,
# рекурсия по левому потомку
# и рекурсия по правому потомку.

                @_;
            unless
    print $tree->{VALUE}, " ";
    pre_order($tree->{LEFT});
    pre_order($tree->{RIGHT});
```

```
9 Функция определяет,  присутствует ли  передаваемое значение в дереве.
# Если значение присутствует, функция возвращает соответствующий узел.
# Поиск ускоряется за счет ограничения перебора нужной ветвью.
sub search {
    my($tree, $value) = @_;

    if ($tree->{VALUE} == $value) {

    }
    search($tree->{ ($value < $tree->{VALUE}) ? "LEFT" :  "RIGHT"}, $value)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пакеты, библиотеки и модули 12

Как и все обладатели библиотек, Аурелиан сознавал свою вину за то, что он недостаточно хорошо знал ее содержимое.

Хорхе Луис Борхес, "Теологи"

Введение

Представьте, что у вас есть две программы, каждая из которых хорошо работает сама по себе. Возникает идея — создать третью программу, объединяющую лучшие свойства первых двух. Вы копируете обе программы в новый файл и начинаете перемещать фрагменты. Выясняется, что в программах встречаются переменные и функции с одинаковыми именами, которые невозможно объединить. Например, каждая программа может содержать функцию `init` или глобальную переменную `$count`. При объединении эти компоненты вступают в конфликт.

Проблема решается с помощью *пакетов*. Пакеты используются в Perl для разделения глобального пространства имен. Они образуют основу как для традиционных модулей, так и для объектно-ориентированных классов. Подобно тому, как каталог содержит файлы, пакет содержит идентификаторы. Каждый глобальный идентификатор (переменная, функция, манипулятор файла или каталога, формат) состоит из двух частей: имени пакета и собственно идентификатора. Эти две части разделяются символами `::`. Например, переменная `$CGI::needs_binmode` представляет собой глобальную переменную с именем `$needs_binmode`, принадлежащую пакету CGI (до выхода версии 5.000 для этой цели использовался апостроф — например, `$CGI'needs_bin_mode`). Переменная `$Names::startup` — это переменная `$startup` пакета Names, а `$Dates::startup` — переменная `$startup` пакета Dates. Идентификатор `$startup` без имени пакета означает глобальную переменную `$startup` текущего пакета (при условии, что в данный момент не видна лексическая переменная `$startup`; о лексических переменных рассказано в главе 10 "Подпрограммы»). При указании неполного имени (то есть имени переменной без пакета) лексические переменные переопределяют глобальные. Лексическая переменная существует в области действия; глобальная — на уровне пакета. Если вам нужна глобальная переменная, укажите ее полное имя.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Ключевое слово package является объявлением, обрабатываемым на стадии компиляции. Оно устанавливает префикс пакета по умолчанию для неполных глобальных **идентификаторов**, по аналогии с тем, как chdir устанавливает префикс каталога по умолчанию для относительных путей. Влияние package распространяется до конца текущей области действия (блока в фигурных скобках, файла или eval) или до ближайшей команды package в той же области действия (см. следующий фрагмент). Все программы выполняются в пакете main, пока командой package в них не будет выбран другой пакет.

```
package Alpha;
$name = "first";

package Omega;
$name = "last";

package main;
print "Alpha is $Alpha::name, Omega is $Omega::name.\n";
Alpha is first, Omega is last.
```

В отличие от пользовательских идентификаторов, встроенные переменные со специальными именами (например, $_ и $.) и идентификаторы STDIN, STDOUT, STDERR, ARGV, ARGVOUT, ENV, INC и SIG без указания имени пакета считаются принадлежащими к пакету main. Благодаря этому STDIN, @ARGV, %ENV и $_ всегда означают одно и то же независимо от текущего пакета; например, @ARGV всегда относится к @main::ARGV, даже если вы измените пакет по умолчанию командой package. Уточненное имя @ElseWhere::ARGV относится к нестандартному массиву @ARGV и не обладает специальным значением. Не забудьте локализовать переменную $_, если вы используете ее в своем модуле.

Модули

Многократное использование кода в Perl осуществляется с помощью *модулей*. Модуль представляет собой файл, содержащий набор взаимосвязанных функций, которые используются другими программами и библиотечными модулями. У каждого модуля имеется *внешний* ***интерфейс*** — набор переменных и функций, предназначенных для использования за его пределами. Внутри модуля интерфейс определяется инициализацией некоторых пакетных переменных, с которыми работает стандартный модуль Exporter. За пределами модуля доступ к интерфейсу организуется посредством импортирования имен, что является побочным эффектом команды use. Внешний интерфейс модуля Perl объединяет все, что документировано для всеобщего применения. К недокументированному интерфейсу относится все, что не предназначено для широкой публики. Говоря о модулях в этой главе и о традиционных модулях вообще, мы имеем в виду **модули,** использующие Exporter.

Команды подключают модуль к вашей программе, хотя и обладают несколько разной семантикой. Команда require загружает модуль во время выполнения с проверкой, позволяющей избежать повторной загрузки модуля.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Команда use работает аналогично, но с двумя дополнительными свойствами: загрузкой модуля на стадии компиляции и автоматическим импортированием.

Модули, включаемые командой use, обрабатываются на стадии компиляции, а обработка `require` происходит во время выполнения. Это существенно, поскольку при отсутствии необходимого модуля программа даже не запустится — use не пройдет компиляцию сценария. Другое преимущество use перед заключается в том, что компилятор получает доступ к прототипам функций в подпрограммах модуля. Прототипы принимаются во внимание только компилятором, но не интерпретатором (впрочем, как говорилось выше, мы рекомендуем пользоваться прототипами только для замены встроенных команд, у которых они имеются).

Обработка команды `use` на стадии компиляции позволяет передавать указания компилятору. Директива (pragma) представляет собой специальный модуль, влияющий на процесс компиляции Perl-кода. Имена директив всегда записываются в нижнем регистре, поэтому при написании обычного модуля следует выбирать имена, начинающиеся с большой буквы. К числу директив, поддерживаемых Perl 5.004, принадлежат autouse, constant, diagnostics, integer, lib, locale, overload, `sigtrap`, strict, subs и vars. Каждой директиве соответствует отдельная страница руководства.

Другое отличие use и `require` заключается в том, что use выполняет неявное *импортирование* пакета включаемого модуля. Импортирование функции или переменной из одного пакета в другой создает некое подобие синонима — иначе говоря, появляются два имени, обозначающих одно и то же. Можно провести аналогию с созданием ссылки на файл, находящийся в другом каталоге, командой *ln /somedir/somefile*.После подключения уже не придется вводить полное имя для того, чтобы обратиться к файлу. Аналогично, импортированное имя не приходится уточнять именем пакета (или заранее объявлять с помощью use vars или use subs). Импортированные переменные можно использовать так, словно они являются частью вашего пакета. После импортирования `$English::OUTPUT_AUTOFLUSH` в текущий пакет на нее можно ссылаться в виде `$OUTPUT_AUTOFLUSH`.

Модули Perl должны иметь расширение *.pm*. Например, модуль FileHandle хранится в файле *FileHandle.pm*. Полный путь к файлу зависит от включаемых путей, хранящихся в глобальном массиве @INC. В рецепте 12.7 показано, как работать с этим массивом.

Если имя модуля содержит одну или несколько последовательностей `::`, они преобразуются в разделитель каталогов вашей системы. Следовательно, модуль File::Find в большинстве файловых систем будет храниться в файле File/Find.pm. Например:

```
        "FileHandle.pm";          #            время выполнения
        FileHandle;        tt        Предполагается ".pm";
                                   # то же, что и выше
use FileHandle;                   к Загрузка во время компиляции

        "Cards/Poker.pm";           Загрузка во время выполнения
        Cards: ;Poker;             # Предполагается ".pm";
                                   # то же, что и выше
use Cards::Poker;                 # Загрузка во время компиляции
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Правила импортирования/экспортирования

Процесс экспортирования демонстрируется ниже на примере гипотетического модуля Cards::Poker. Программа хранится в файле *Poker.pm* в каталоге Cards, то есть *Cards/Poker.pm* (о том, где должен находиться каталог Cards, рассказано в рецепте 12.7). Приведем содержимое этого файла с пронумерованными для удобства строками:

```
1   package Cards::Poker;
2   use Exporter;
3   @ISA = ('Exporter');
4   @EXPORT = qw(&shuffle @card_deck);
5   @card_deck = ();                    # Инициализировать глобальные
                                        # переменные пакета
6   sub shuffle { }                     # Определение
                                        # заполняется позднее
7   1;                                  # Не забудьте!
```

В строке 1 объявляется пакет, в который модуль поместит свои глобальные переменные и функции. Обычно модуль начинается с переключения на конкретный пакет, что позволяет ему хранить глобальные переменные и функции так, чтобы они не конфликтовали с переменными и функциями других программ. Имя пакета *должно* быть записано точно так же, как и при загрузке модуля соответствующей командой use.

Не пишите package Poker только потому, что модуль хранится в файле *Poker.pm*! Используйте package Cards::Poker, поскольку в пользовательской программе будет стоять команда use Cards::Poker. Эту распространенную ошибку трудно обнаружить. Если между командами package и use нет точного соответствия, проблемы возникнут лишь при попытке вызвать импортированную функцию или обратиться к импортированной переменной — те будут загадочным образом отсутствовать.

Строка 2 загружает модуль Exporter, управляющий внешним интерфейсом модуля (см. ниже). Строка 3 инициализирует специальный, существующий на уровне пакета массив @ISA строкой "Exporter". Когда в программе пользователя встречается команда use Cards::Poker, Perl неявно вызывает специальный метод, Cards::Poker->import(). В пакете нет метода import, но это нормально — такой метод есть в пакете Exporter, и вы *наследуете* его благодаря присваиванию @ISA (ISA = "is a", то есть «является»). Perl обращается к массиву @ISA пакета при обращении к неопределенному методу. Наследование рассматривается в главе 13 "Классы, объекты и связи". Пока не обращайте на него внимания, но не забывайте вставлять код строк 2 и 3 в каждый новый модуль.

Строка 4 заносит список ('&shuffle', '@card_deck') в специальный, существующий на уровне пакета массив @EXPORT. При импортировании модуля для переменных и функций, перечисленных в этом массиве, создаются синонимы в вызывающем пакете. Благодаря этому после импортирования вам не придется вызывать функцию в виде Poker::Deck::shuffle(23) — хватит простого shuffle(23). Этого не произойдет при загрузке Cards::Poker командой Cards::Poker; импортирование выполняется только для use.

Строки 5 и 6 готовят глобальные переменные и функции пакета к экспортированию (конечно, вы предоставите более конкретные инициализации и определе-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ния, чем в нашем примере). Добавьте другие переменные и функции, включая и те, которые не были включены в внешний интерфейс посредством @EXPORT. Об использовании модуля Exporter рассказано в рецепте 12.1.

Наконец, строка 7 определяет общее возвращаемое значение модуля. В нашем случае это просто 1. Если последнее вычисляемое выражение модуля не дает истинного значения, инициируется исключение. Обработка исключений рассматривается в рецепте 12.2. Подойдет любое истинное выражение, будь то 6.02e23 или "Because tchrist and gnat told us to put this here"; однако 1 — каноническая истинная величина, используемая почти во всех модулях.

Пакеты обеспечивают группировку и организацию глобальных идентификаторов. Они не имеют ничего общего с ограничением доступа. Код, откомпилированный в пакете Church, может свободно просматривать и изменять переменные пакета State. Пакетные переменные всегда являются глобальными И общедоступными. Но это вполне нормально, поскольку модуль представляет собой больше, чем простой пакет; он также является файлом, а файлы обладают собственной областью действия. Следовательно, если вам нужно ограничить доступ, используйте лексические переменные вместо глобальных. Эта тема рассматривается в рецепте 12.4.

Другие типы библиотечных файлов

Библиотека представляет собой набор неформально взаимосвязанных функций, используемых другими программами. Библиотеки не обладают жесткой семантикой модулей Perl, Их можно узнать по расширению файла *.pl* — например, *syslog.pl* и *chat2.pl.*

Библиотека Perl (а в сущности, любой файл, содержащий код Perl) может загружаться командой do 'file.pl' или il.pl'. Второй вариант лучше, поскольку в отличие от do require выполняет неявную проверку ошибок. Команда инициирует исключение, если файл не будет найден в пути @INC, не компилируется или не возвращает истинного значения при выполнении инициализирующего кода (последняя строка с 1, о которой говорилось выше). Другое преимущество require заключается в том, что команда следит за загруженными файлами с помощью глобального хэша %INC. Если %INC сообщает, что файл уже был загружен, он не загружается повторно.

Библиотеки хорошо работают в программах, однако в ситуациях, когда одна библиотека использует другую, могут возникнуть проблемы. Соответственно, простые библиотеки Perl в значительной степени устарели и были заменены более современными модулями. Однако некоторые программы продолжают пользоваться библиотеками, обычно загружая их командой require вместо do.

В Perl встречаются и другие расширения файлов. Расширение *.ph* используется для заголовочных файлов C, преобразованных в библиотеки Perl утилитой *h2ph* (см. рецепт 12.14). Расширение *.xs* соответствует исходному файлу C (возможно, созданному утилитой *h2xs*), скомпилированному утилитой *xsubpp* и компилятором C в машинный код. Процесс создания смешанных модулей рассматривается в рецепте 12.15.

До настоящего времени мы рассматривали лишь традиционные модули, которые экспортируют свой интерфейс, предоставляя вызывающей стороне прямой доступ к некоторым подпрограммам и переменным. К этой категории относится

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

большинство модулей. Но некоторые задачи — и некоторые программисты — связываются с хитроумными модулями, содержащими объекты. Объектно-ориентированный модуль редко использует механизм импортирования/экспортирования. Вместо этого он предоставляет объектно-ориентированный интерфейс с конструкторами, деструкторами, методами, наследованием и перегрузкой операторов. Данная тема рассматривается в главе 13.

Пользуйтесь готовыми решениями

Perl Archive Network) представляет собой гигантское хранилище практически всех ресурсов, относящихся к Perl, — исходные тексты, документацию, версии для альтернативных платформ и, что самое главное, модули. Перед тем как браться за новый модуль, загляните на CPAN и поищите там готовое решение. Даже если его не существует, может найтись что-нибудь похожее, полезное в вашей работе.

На CPAN можно обратиться по адресу *http://www.perl.com/CPAN/CPAN.ht*(или *ftp://www.perl.com/pub/perl/CPAN/CPAN.html).* В этом файле кратко описан каждый модуль, входящий в CPAN. Поскольку файл редактируется вручную, в нем могут отсутствовать описания последних модулей. Необходимую информацию можно получить по адресу *CPAN/RECENT* или *CPAN/RECENT.html.*

Каталог модулей находится по адресу *CPAN/modules*В Нем содержатся индексы всех зарегистрированных модулей, а также имеются три удобных подкаталога: *by_module* (сортировка по модулям), *by_author*(сортировка по авторам) и *by_category* (сортировка по категориям). В каждом подкаталоге перечислены одни и те же модули, но подкаталог *by_category*,вероятно, наиболее удобен. Находящиеся в нем подкаталоги соответствуют конкретным прикладным областям, среди которых — интерфейсы операционной системы, сетевые взаимодействия, модемы и межпроцессные коммуникации, интерфейсы баз данных, пользовательские интерфейсы, интерфейсы к другим языкам программирования, аутентификация, безопасность и шифрование, World Wide Web, HTML, HTTP, CGI и MIME, графика, операции с растровыми изображениями, построение графиков — и это лишь малая часть.

Смотри также

Разделы "Packages" и "Modules" в *perlmod*(1).

12.1. Определение интерфейса модуля

Проблема

Требуется определить внешний интерфейс модуля с помощью стандартного модуля Exporter.

Решение

Включите в файл модуля (например, *YourModule.pm*)приведенный ниже фрагмент, МНОГОТОЧИЯ заполняются в соответствии с инструкциями, приведенными в разделе «Комментарий».

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
package YourModule;
use strict;
use vars qw(@ISA @EXPORT @EXPORT_OK %EXPORT_TAGS $VERSION);

use Exporter;
$VERSION = 1.00;                " Или выше
@ISA = qw(Exporter);

@EXPORT      = qw(...);         # Автоматически экспортируемые имена
                                 # (набор :DEFAULT)
@EXPORT_OK   = qw(...);         # Имена, экспортируемые по запросу
%EXPORT_TAGS = (                # Определение имен для наборов
    TAG1 => [...],
    TAG2 => [...],




# Ваш программный код


1;                              # Так должна выглядеть последняя строка
```

Чтобы воспользоваться модулем YourModule в другом файле, выберите один из следующих вариантов:

```
use YourModule; •               # Импортировать в пакет имена по умолчанию
use YourModule qw(...);         # Импортировать в пакет перечисленные имена
use YourModule ();              я Не импортировать никаких имен
use YourModule qw(:TAG1);       # Импортировать набор имен
```

Комментарий

Внешний интерфейс модуля определяется с помощью стандартного модуля Exporter. Хотя в пакете можно определить собственный метод import, почти никто этого не делает.

Когда в программе встречается команда use YourModule, в действительности выполняется команда "YourModule. pm", за которой вызывается метод YourModule->import(). Это происходит во время компиляции. Метод import, унаследованный из пакета Exporter, ищет в вашем пакете глобальные переменные, управляющие его работой. Поскольку они должны быть пакетными, мы используем директиву use vars, чтобы избежать проблем с use str ict. Это следующие переменные.

$VERSION

При загрузке модуля можно указать минимальный допустимый номер версии. Если версия окажется ниже, use инициирует исключение.

```
use YourModule 1.86     # Если $VERSION < 1.86, происходит исключение
```

$EXPORT

Массив содержит список функций и переменных, экспортируемых в пространство имен вызывающей стороны, чтобы в дальнейшем к ним можно было

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

обращаться без уточнения имени пакета. Обычно используется список в форме qw():

```
@EXPORT = qw(&F1 &F2 @List);
@EXPORT = qw( F1 . F2 @List);        # То же
```

После выполнения простой команды use YourModule вы сможете вызывать функцию &F1 в виде F1() вместо YourModule::F1() и обращаться к массиву @List вместо @YourModule::List. Амперсанд (&) перед спецификацией экспортированной функции необязателен.

Чтобы загрузить модуль во время компиляции, но при этом запретить экспортирование каких-либо имен, воспользуйтесь специальной формой с пустым списком use Exporter().

@EXPORT_OK

Массив содержит имена, которые могут импортироваться по конкретному запросу. Если массив заполнен следующим образом:

```
@EXPORT_OK = qw(Op_Func %Table);
```

то пользователь сможет загрузить модуль командой:

```
use YourModule qw(Op_func %Table F1);
```

и импортировать только функцию Op_func, хэш %Table и функцию F1. Функция F1 присутствует в массиве @EXPORT. Обратите внимание: команда не выполняет автоматического импортирования F2 или @List, хотя эти имена присутствуют в @EXPORT. Чтобы получить все содержимое @EXPORT и плюс к тому все дополнительное содержимое @EXPORT_OK, воспользуйтесь специальным тегом :DEFAULT:

```
use YourModule qw(:DEFAULT %Table);
```

%EXPORT_TAGS

Хэщ используется большими модулями (типа CGI или POSIX) для высокоуровневой группировки взаимосвязанных импортируемых имен. Его значения представляют собой ссылки на массивы символических имен, каждое из которых должно присутствовать либо в @EXPORT, либо в @EXPORT_OK. Приведем пример инициализации:

```
%EXPORT_TAGS = (
        functions => [ qw(F1 F2 Op_func) ],
        Variables => [ qw(@List %Table)  ],
);
```

Импортируемое имя с начальным двоеточием означает импортирование группы имен. Например, команда:

```
use YourModule qw(:Functions %Table);
```

импортирует все имена из

```
@{ $YourModule::EXPORT_TAGS{Functions} },
```

то есть функции F1, F2 и Op_func, а затем — хэш %Table.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Хотя тег :DEFAULT не указывается в %EXPORT_TAGS, он обозначает все содержимое @EXPORT.

Все эти переменные не обязательно определять в каждом модуле. Ограничьтесь лишь теми, которые будут использоваться.

Смотри также

Документация по стандартному модулю Exporter; рецепты 12.7; 12.18.

12.2. Обработка ошибок

Проблема

Загружаемый модуль может отсутствовать в системе. Обычно это приводит к фатальной ошибке. Вы хотите обнаружить и перехватить эту ошибку.

Решение

Поместите require или use в eval, а eval — в блок BEGIN:

```
# Не импортировать
BEGIN {
    unless (eval "require $mod")
        warn "couldn't load $mod: $@",
```

Комментарий

Попытка загрузки отсутствующего или неполного модуля обычно должна приводить к аварийному завершению программы. Однако в некоторых ситуациях программа должна продолжить работу — например, попытаться загрузить другой модуль. Как и при других исключениях, для изолирования ошибок компиляции применяется конструкция eval.

Использовать eval { БЛОК } нежелательно, поскольку в этом случае будут перехватываться только исключения времени выполнения, а use относится к событиям времени компиляции. Вместо этого следует использовать конструкцию eval "СТРОКА", что позволит перехватывать и ошибки компиляции. Помните: вызов require простого слова[1] имеет несколько иной смысл, чем вызов

[1] "Простым словом" (bareword) называется слово, не имеющее специальной грамматической интерпретации и интерпретируемое как строка. — *Примеч. перев.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

для переменной. Команда добавляет расширение *.pm* и преобразует `::` в разделитель каталогов вашей операционной системы — в каноническом варианте / (как в URL), но в некоторых системах используются \, : и даже . .

Если вы хотите последовательно попытаться загрузить несколько модулей и остановиться на первом работающем, поступите так:

```
BEGIN {
    my($found, @DBs, $mod);
    $found = 0;
    @DBs = qw(Giant: :Eenie Giant: :Meanie Mouse: :Mynie Moe);
    for $mod (@DBs) {
            (eval          $mod")
             $mod->import();     fl     При необходимости
             $found = 1;
             last;
```

Смотри также

Рецепт 10.12; рецепт 12.3. Функции eval, die, use и require описаны *в perlfunc(1).*

12.3. Отложенное использование модуля

Проблема

Необходимо организовать загрузку модуля на определенной стадии работы программы или вообще отказаться от его загрузки при некоторых обстоятельствах.

Решение

Разбейте use на отдельные компоненты import, либо воспользуйтесь директивой use autouse.

Комментарий

Если программа проверяет свои аргументы и завершает работу с информационным сообщением или ошибкой, загружать неиспользуемые модули бессмысленно. Это лишь вызывает задержки и раздражает пользователей. Но как говорилось во введении, команды use обрабатываются во время компиляции, а не во время выполнения.

Наиболее эффективная стратегия состоит в проверке аргументов внутри блока BEGIN до загрузки модулей. Следующая программа перед загрузкой необходимых модулей проверяет, что она была вызвана ровно с двумя аргументами, каждый из которых является целым числом:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
BEGIN {
    unless (@ARGV == 2 && (2 == grep {/^\d+$/} @ARGV)) {
        die "usage: $0 num1 num2\n";


use Some::Module;
use More::Modules;                                          \
```

Похожая ситуация возникает в программах, которые при разных запусках могут использовать разные наборы модулей. Например, программа *factors* из главы 2 "Числа" загружает библиотеку вычислений с повышенной точностью лишь при вызове с флагом **-b**. Команда use в данном случае бессмысленна, поскольку она обрабатывается во время компиляции, задолго до проверки условия `if`. По этой причине мы используем команду

```
if ($opt_b) {
            Math::BigInt;
}
```

Math::BigInt является не традиционным, а объектно-ориентированным модулем, поэтому импортирование не требуется. Если у вас имеется список импортируемых объектов, укажите его в конструкции `qw()` так, как это было бы сделано для use. Например, вместо:

```
use Fcntl qw(O_EXCL O_CREAT O_RDWR);
```

можно использовать следующую запись:

```
        Fcntl;
Fcntl->import(qw(O_EXCL O_CREAT O_RDWR));
```

Откладывая импортирование до времени выполнения, мы сознательно идем на то, что оставшаяся часть программы не узнает об изменениях импортированной семантики, которые были бы видны компиляторупри использовании `use`. В частности, не будут своевременно видны прототипы функций и переопределения встроенных функций.

Возникает идея — инкапсулировать отложенную загрузку в подпрограмме. Следующее, простое на первый взгляд решение не работает:

```
sub load_module {
            $_[0];  "НЕВЕРНО
    import  $_[0];  "НЕВЕРНО
>
```

Понять причину неудачи непросто. Представьте себе вызов аргументом "`Math::BigFloat`". Если это простое слово, `::` преобразуется в разделитель каталогов операционной системы, а в конец добавляется расширение *.pm*. Но простая переменная интерпретируется как литерал — имя файла. Дело усугубляется тем, что Perl не имеет встроенной функции `import`. Существует лишь метод класса `import`, который мы пытаемся применить с сомнительным косвенным объектным синтаксисом. Как и в случае с косвенным применением файловых манипуляторов, косвенный объект можно использовать лишь для простой ска-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

лярной переменной, простого слова или блока, возвращающего объект. Выражения, а также отдельные элементы массивов или хэшей здесь недопустимы.

Усовершенствованный вариант выглядит так:

```
load_module('Fcntl', qw(O_EXCL O_CREAT O_RDWR));

sub load_module {
                    $_[0]";
    die if $@;
    $_[0]->import(@_[1 .. $"_]);
}
```

Но и он в общем случае не идеален. Функция должна импортировать имена не в свой пакет, а в пакет вызвавшей стороны. В принципе эта проблема решается, но процедура становится все сложнее и сложнее.

Удобное альтернативное решение — применение директивы autouse. Она появилась в Perl 5.004. Эта новая директива экономит время для редко загружаемых функций, откладывая их загрузку до момента фактического использования:

```
use autouse Fcntl => qw( O_EXCL() O_CREAT() O_RDWR() );
```

Круглые скобки после O_EXCL, O_CREAT и O_RDWR нужны для autouse, но не для use или import. Директива autouse принимает не только имена функций, но также позволяет передать прототип функции. В соответствии с прототипами константы Fcntl вызываются без аргументов, поэтому их можно использовать в программе как простые слова без возни с use strict.

Также помните, что проверка use strict осуществляется во время компиляции. Если модуль Fcntl подключается командой use, прототипы модуля Fcntl будут откомпилированы и мы сможем использовать константы без круглых скобок. Если использована команда require или вызов use заключен в eval, как это делалось выше, компилятор не сможет прочитать прототипы, поэтому константы Fcntl не будут использоваться без скобок.

За сведениями об особенностях директивы autouse обращайтесь к электронной документации.

Смотри также

Рецепт 12.2; документация по стандартному модулю Exporter (описание метода import); документация по стандартной директиве use autouse.

12.4. Ограничение доступа к переменным модуля

Проблема

Требуется сделать переменную или функцию закрытой (то есть разрешить ее использование только в границах пакета).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Общего решения не существует. Однако можно ограничить доступ на уровне файла, в котором находится модуль, — обычно этого достаточно.

Комментарий

Помните, что пакет всего лишь определяет способ группировки переменных и функции и потому не поддерживает ограничения доступа. Все содержимое пакета по определению является глобальным и доступным отовсюду. Пакеты лишь группируют, ничего не скрывая.

Ограничение доступа возможно только с применением лексических переменных. Предположим, модуль реализован в виде файла *Module.pm*, а все его глобальные имена принадлежат пакету Module. Поскольку файл по определению образует самостоятельную область действия, а лексические переменные ограничиваются ею, создание лексической переменной с файловой областью действия фактически эквивалентно переменной, ограниченной данным модулем.

Однако переключение пакетов внутри области действия может привести к тому, что лексические переменные этой области остаются видны в любом месте области. Дело в том, что команда package всего лишь устанавливает новый префикс для глобальных идентификаторов.

```
package Alpha;
my $aa = 10;


package Beta;
my $bb = 20;      .     .
   $x = "blue";

package main;
print "$aa, $bb, $x, $Alpha::x, $Beta::x\n";
                 blue
```

На это ли вы рассчитывали? Две лексические переменные, $aa и $bb, ос таются в области действия, поскольку они не вышли за границы текущего блока, файла или eval. Считайте, что глобальные и лексические переменные существуют в разных изменениях, Никак не связанных друг с другом. Пакетные команды не имеют ничего общего с лексическими переменными. После установки текущего префикса первая глобальная переменная $x в действительности представляет собой $Alpha::x, а вторая — $Beta::x, поскольку промежуточная команда package изменила префикс по умолчанию. Доступ к пакетным идентификаторам при указании полного имени может осуществляться откуда угодно, как это делается в команде print.

Итак, пакеты не позволяют ограничивать доступ — зато на это способны модули, поскольку они находятся в файлах, а файл всегда обладает собственной областью действия. Приведенный ниже простой модуль находится в файле *Flipper.pm* и экспортирует две функции, flip_words и flip_boundary. Первая функция переставляет слова строки в обратном порядке, а вторая изменяет определение границы слова.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Flipper.pm
package Flipper;
usestrict;

        Exporter;
use vars qw(@ISA @EXPORT $VERSION);
@ISA     = qw(Exporter);
@EXPORT  = qw(flip_words flip_boundary);
$VERSION = 1.0;

my $Separatrix = ' ';  # По умолчанию пробел; предшествует функциям

sub flip_boundary {
                    = $Separatrix;
    if (@_) { $Separatrix = $_[0] }

}
sub flip_words {                                                    ' : '
    my $line  = $_[0];
    my @words = split($Separatrix, $line);
            join($Separatrix,          @words);
}
1;
```

Модуль задает значения трех пакетных переменных, необходимых для работы Exporter, а также инициализирует лексическую переменную `$Separatrix` уровня файла. Как говорилось выше, эта переменная ограничивается границами файла, а не пакета. Весь код той же области действия, расположенный после ее объявления, прекрасно видит `$Separatrix`. Хотя глобальные переменные не экспортировались, к ним можно обращаться по полному имени — например, `$Flipper::VERSION`.

Лексические переменные, существующие в некоторой области действия, нельзя прочитать или изменить вне этой области, которая в данном случае соответствует всему файлу после объявления переменной. На лексические переменные нельзя ссылаться по полному имени или экспортировать их; экспортирование возможно лишь для глобальных переменных. Если кому-либо за пределами модуля потребуется просмотреть или изменить лексические переменные файла, они должны обратиться с запросом к модулю. Именно здесь в игру вступает функция `flip_boundary`, обеспечивающая косвенный доступ к закрытым компонентам модуля.

Работа приведенного выше модуля ничуть не изменилась бы, будь `$Separatrix` пакетной глобальной переменной, а не файловой лексической. Теоретически к ней можно было бы обратиться снаружи так, что модулю об этом ничего не было известно. С другой стороны, не стоит увлекаться чрезмерными ограничениями и щедро уснащать модули лексическими переменными с файловой областью действия. У вас уже имеется пространство имен (в нашем примере — Flipper), в котором можно сохранить все необходимые идентификаторы. Собственно, для этого оно и предназначено. Хороший стиль программирования на Perl почти всегда избегает полностью уточненных идентификаторов.

Если уж речь зашла о стиле, регистр символов в идентификаторах модуля Flipper выбирался не случайно. В соответствии с руководством по стилю программирования на Perl, символами верхнего регистра записываются идентификато-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ры, имеющие специальное значение для Perl. Имена функций и локальных переменных записываются в нижнем регистре. Устойчивые переменные модуля (файловые лексические или пакетные глобальные) начинаются с символа верхнего регистра. Если идентификатор состоит из нескольких слов, то для удобства чтения эти слова разделяются символами подчеркивания. Пожалуйста, не разделяйте слова символами верхнего регистра безподчеркиваний — в конце концов, вряд ли вам захотелось бы читать эту книгу без пробелов.

Смотри также

perlstyle(1); рецепты 10.2–10.3. Лексические переменные с файловой областью действия рассматриваются в *perlmod*(1).

12.5. Определение пакета вызывающей стороны

Проблема

Требуется узнать текущий или вызывающий пакет.

Решение

Текущий пакет определяется так:

```
$this_pack = __PACKAGE__;
```

Пакет вызывающей стороны определяется так:

```
$that_pack = caller();
```

Комментарий

Метапеременная `__PACKAGE__` возвращает пакет, в котором был откомпилирован текущий код. Значение не интерполируется в строках, заключенных в кавычки:

```
print "I am in package __PACKAGE__\n";      # НЕВЕРНО!
I am in package __PACKAGE__
```

Необходимость узнать пакет вызывающей стороны чаще возникает в старом коде, которому в качестве входных данных была передана строка для `eval`, файловый манипулятор, формат или имя манипулятора каталога. Рассмотрим гипотетическую функцию `runit`:

```
package Alpha;
runit('$line = <TEMP>');

package Beta;
sub runit {
    my $codestr = shift;
    eval $codestr;
    die if $@;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Такой подход работает лишь в том случае,если переменная $line является глобальной. Для лексических переменных он не годится. Обходное решение — сделать так, чтобы функция runit принимала ссылку на функцию:

```
package Beta;
sub runit {                    .
    ray $codestr = shift;
    my $hispack = caller;
    eval "package $hispack; $codestr";
    die if $@;
>
```

Новое решение не только работает с лексическими переменными, но и обладает дополнительным преимуществом — синтаксис кода проверяется во время компиляции, а это существенный плюс.

При передаче файлового манипулятора стоит воспользоваться более переносимым решением — функцией Symbol::qualify. Она получает имя и пакет, для которого оно уточняется. Если имя нуждается в уточнении, оно исправляется, а в противном случае остается без изменений. Однако это решение заметно уступает по эффективности прототипу *.

Следующий пример читает и возвращает п строк из файлового манипулятора. Перед тем как работать с манипулятором, функция qualify уточняет его.

```
open (FH, "< /etc/termcap")
    or die "can't open /etc/termcap: $!";
                                'FH');

use Symbol ();
use Carp;
                {
    my ($count, $handle) = @_;
    my(@retlist, $line);

    croak "count must be > 0" unless $count > 0;
    $handle = Symbol::qualify($handle, (caller())[0]);
    croak "need open filehandle" unless defined fileno($handle);

    push(@retlist, $line) while defined($line = <$handle>) && $count--;
          @retlist;
}
```

Если при вызове функции файловый манипулятор всегда передается в виде тип-глоба *FH, ссылки на глоб *FH или с помощью объектов FileHandle или IO::Handle, уточнение не потребуется. Оно необходимо лишь на случай передачи минимального "FH".

Смотри также

Документация по стандартному модулю Symbol; рецепт 12.12, Специальные метапеременные FILE , LINE и PACKAGE описаны в *perldata*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

12.6. Автоматизированное выполнение завершающего кода

Проблема

Требуется создать для модуля начальный и завершающий код, вызываемый автоматически без вмешательства пользователя.

Решение

Начальный код реализуется просто — разместите нужные команды вне определений подпрограмм в файле модуля. Завершающий код помещается в блок END модуля.

Комментарий

В некоторых языках программист должен вызвать инициализирующий код модуля, прежде чем вызывать какие-либо его функции. Аналогично, при завершении программы от программиста может потребоваться вызов завершающего кода, выполняющего деинициализацию модуля.

В Perl дело обстоит иначе. Инициализирующий код модуля образуют команды, не входящие ни в одну подпрограмму модуля. Этот код выполняется непосредственно при загрузке модуля. Пользователю никогда не приходится следить за вызовом начального кода, поскольку это происходит автоматически.

Для чего нужен автоматический вызов завершающего кода? Все зависит от модуля. Допустим, вам захотелось записать информацию о завершении в системный журнал, приказать серверу базы данных актуализировать все незаконченные операции, обновить состояние экрана или вернуть терминал в исходное состояние.

Предположим, модуль должен регистрировать начало и завершение своей работы в журнале. Вставьте следующий фрагмент в блок END, чтобы он выполнялся при завершении программы:

```
$Logfile = "/tmp/mylog" unless defined $Logfile;
open(LF, ">>$Logfile")
    or die "can't append to $Logfile: $!";  '
select(((select(LF), $|=1))[0]);  " Отменить буферизацию LF
logmsg("startup");

sub logmsg {
    my $now = scalar gmtime;
    print LF "$0 $$ $now: @_\n"
        or die "write to $Logfile failed: $!";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Первая часть кода, не входящая в объявления функций, выполняется во время загрузки модуля. Для этого от пользователя модуля не потребуется никаких специальных действий. Впрочем, для кого-нибудь это может оказаться неприятным сюрпризом, поскольку при недоступности журнала die вызовет сбой при выполнении use или

Блоки END не отличаются от других функций завершения — trap 0 вкомандном интерпретаторе, atexit в языке C или глобальные деструкторы в объектно-ориентированных языках. Порядок выполнения END противоположен порядку загрузки модулей; иначе говоря, первым выполняется блок END последнего загруженного модуля. Завершающий код вызывается независимо от причины завершения — нормального достижения конца основной программы, непосредственного вызова функции exit или необработанного исключения (например, die или ошибки деления на ноль).

Однако с неперехваченными сигналами дело обстоит иначе. При завершении по сигналу блоки завершения не вызываются. Проблема решается следующей директивой:

```
use sigtrap qw(die normal-signals error-signals)
```

END также не вызывается в случае, если процесс вызывает функцию exec, поскольку процесс остается тем же самым, изменяется лишь программа. Все стандартные атрибуты (идентификатор процесса и его родителя, идентификаторы пользователя и группы, маска доступа, текущий каталог, переменные окружения, ограничения ресурсов и накопленная статистика), открытые файловые дескрипторы (однако см. описание переменной $"F в perlvar(1)) сохраняются. Другой подход привел бы к лишним вызовам блоков завершения в программах с ручной обработкой fork и exec. Это было бы нежелательно.

Смотри также

Стандартная директива use sigtrap описана в *perlmod*(1), а переменная $"F — в *perldata*(1). Функции fork и exec рассматриваются в *perlmod*(1).

12.7. Ведение собственного каталога модулей

Проблема

Вы не хотите включать собственные модули в стандартную библиотеку расширений системного уровня.

Решение

Возможно несколько вариантов: воспользоваться параметром командной строки Perl -I; присвоить значение переменной окружения PERL5LIB; применить директиву use lib (возможно, в сочетании с модулем FindBin).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Массив @INC содержит список каталогов, которые просматриваются при каждой компиляции кода из другого файла, библиотеки или модуля командой do, или use. Содержимое массива легко вывести из командной строки:

```
% perl -e 'for (@INC) { printf "%d %s\n", $i++, $_ }'
0   /usr/local/perl/lib/i686-linux/5.004
1  /usr/local/perl/lib
2    /usr/local/perl/lib/site_perl/i686-linux
3   /usr/local/perl/lib/site_perl
4 .
```

Первые два элемента (0 и 1) массива @INC содержат обычные платформенно-зависимый и платформенно-независимый каталоги, с которыми работают все стандартные библиотеки, модули и директивы. Этих каталогов два, поскольку некоторые модули содержат данные или форматирование, имеющие смысл лишь для конкретной архитектуры. Например, модуль Config содержит информацию, относящуюся лишь к некоторым архитектурам, поэтому он находится в 0 элементе массива. Здесь же хранятся модули, содержащие откомпилированные компоненты на С (например, *Socket.so*). Однако большинство модулей находится в элементе 1 (независимый от платформы каталог).

Следующая пара, элементы 2 и 3, по своим функциям аналогична элементам О и 1, но относится к конкретной системе. Допустим, у вас имеется модуль, который не поставлялся с Perl, — например, модуль, загруженный с CPAN или написанный вами. Когда вы (или, что более вероятно, ваш системный администратор) устанавливаете этот модуль, его компоненты попадают в один из этих каталогов. Эти каталоги следует использовать для любых модулей, удобный доступ к которым должен быть в вашей системе.

Последний стандартный элемент, "." (текущий рабочий каталог), используется только в процессе разработки и тестирования программ. Если модули находятся в каталоге, куда вы перешли последней командой chdir, все хорошо. Если в любом другом месте — ничего не получится.

Иногда ни один из каталогов, указанных в @INC, не подходит. Допустим, у вас имеются личные модули или ваша рабочая группа использует свой набор модулей, относящихся только к данному проекту. В этом случае необходимо дополнить поиск по стандартному содержимому @INC.

В первом варианте решения используется флаг командной строки -Iсписок_каталогов. После флага указывается список из одного или нескольких каталогов, разделенных двоеточиями[1]. Список вставляется в начало массива @INC. Этот вариант удобен для простых командных строк и потому может использоваться на уровне отдельных команд (например, при вызове простой однострочной программы из сценария командного интерпретатора).

Подобную методику не следует использовать в строках #!. Во-первых, редактировать каждую программу в системе скучно. Во-вторых, в некоторых старых

[1] Или запятыми в MacOS.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

операционных системах имеются ошибки, связанные с ограничением длины этой строки (обычно 32 символа, включая #!). В этом случае очень длинный путь (например, #/opt/languages/free/extrabits/perl) приведет к появлению таинственной ошибки "Command not found". Perl пытается заново просканировать строку, но этот механизм недостаточно надежен и полагаться на него не стоит.

Нередко самое удачное решение заключается в использовании переменной окружения PERL5LIB, значение которой обычно задается в стартовом сценарии интерпретатора. Если системный администратор задаст переменную в стартовом файле системного уровня, результаты будут доступны для всех пользователей. Предположим, ваши модули хранятся в каталоге *~/perllib*. Включите одну из следующих строк в стартовый файл командного интерпретатора (в зависимости от того, каким интерпретатором вы пользуетесь):

```
# Синтаксис для sh, bash, ksh и zsh
$ export PERL5LIB=$HOME/perllib

# Синтаксис для csh или tcsh
% setenv PERL5LIB ~/perllib
```

Возможно, самое удобное решение с точки зрения пользователя — включение директивы use lib в начало сценария. При этом пользователям программы вообще не придется выполнять специальных действий для ее запуска, Допустим, у нас имеется гипотетический проект Spe программы которого используют собственный набор библиотек. Такие программы могут начинаться с команды:

```
use lib "/projects/spectre/lib";
```

Что делать, если точный путь к библиотеке неизвестен? Ведь проект может устанавливаться в произвольный каталог. Конечно, можно написать детально проработанную процедуру установки с динамическим обновлением сценария, но даже в этом случае путь будет жестко фиксироваться на стадии установки. Если позднее файлы переместятся в другой каталог, библиотеки не будут найдены.

Модуль FindBin легко решает эту проблему. Он пытается вычислить полный путь к каталогу выполняемого сценария и присваивает его важной пакетной переменной $Bin. Обычно он применяется для поиска модулей в одном каталоге с программой или в каталоге lib того же уровня.

Рассмотрим пример для первого случая. Допустим, у вас имеется программа */wherever/spectre/myprog*, которая ищет свои модули в каталоге */wherever/spectrem*, однако вы не хотите жестко фиксировать этот путь:

```
use FindBin;
use lib $FindBin::Bin;
```

Второй случай — если ваша программа находится в каталоге *bin/myprog*, но ее модули должны находиться в каталоге

```
use FindBin qw($Bin);
use lib "$Bin/../lib";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Документация по стандартной директиве use lib и стандартному модулю FindBin. Переменная окружения PERL5LIB описана в *perl*(1). Переменные окружения рассматриваются в руководстве по синтаксису командного интерпретатора.

12.8. Подготовка модуля к распространению

Проблема

Вы хотите подготовить модуль в стандартном формате распространения, чтобы им можно было легко поделиться с другом. Или, что еще лучше, вы собираетесь загрузить модуль на CPAN и сделать его общедоступным.

Решение

Начните со стандартной утилиты Perl *h2xs*. Предположим, вы хотите создать модуль Planets или Astronomy::Orbits. Введите следующие команды:

```
% h2xs -XA -n Planets
% h2xs -XA -n Astronomy::Orbits
```

Эти команды создают подкаталоги *./Planets/* и *./Astronomy/Orbits/* соответственно. В каталогах находятся все компоненты, необходимые для начала работы. Флаг -n задает имя создаваемого модуля, -X запрещает создание компонентов XS (внешних подпрограмм), а -A означает, что модуль не будет использовать AutoLoader.

Комментарий

Написать модуль несложно, если знать, как это делается. Написание "правильного" модуля похоже на заполнение юридического контракта — перед вами множество мест для инициалов, подписей и дат, и все нужно заполнить правильно. Если вы что-нибудь пропустите, контракт не имеет законной силы. Вместо того чтобы нанимать специалиста, можно воспользоваться утилитой *h2xs*. Она создает "скелет" файла модуля с заполненными данными об авторских правах, а также другие файлы, необходимые для правильной установки и документирования модуля, для включения его в CPAN или распространения среди друзей.

Название утилиты *h2xs* может сбить с толку, поскольку XS представляет собой интерфейс внешних подпрограмм Perl для компоновки с C или C++. Однако утилита *h2xs* также в высшей степени удобна для подготовки распространяемых модулей, даже если они и не используют интерфейс XS.

Давайте рассмотрим один из модулей, созданных утилитой *h2xs*. Поскольку модуль будет называться Astronomy::Orbits, вместо команды use Orbits пользователь должен вводить use Astronomy::Orbits. Следовательно, нам потребуется

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

дополнительный подкаталог Astronomy, в котором будет размещаться каталог Orbits. Приведем первую и, вероятно, самую важную строку *Orbits.pm*:

```
package Astronomy::Orbits;
```

Команда определяет йакет (префикс по умолчанию) для всех глобальных идентификаторов (переменных, функций, файловых манипуляторов и т. д.) данного файла. Следовательно, переменная `@ISA` в действительности является глобальной переменной `@Astronomy::Orbits::ISA`.

Как было сказано во введении, использовать команду package Orbits только потому, что она находится в файле *Orbits.pm*, будет ошибкой. Команда package в модуле должна точно совпадать с формулировкой use или означает присутствие префикса каталога, а также совпадение регистра символов. Более того, необходим промежуточный каталог Astronomy. Утилита *h2xs* позаботится обо всем, включая правило установки в Make-файле. Если вы готовите модуль вручную, помните об этом (см. рецепт 12.1).

Если вы собираетесь использовать автоматическую загрузку (см. рецепт 12.10), уберите флаг -A из вызова *h2xs*. В результате будет создан фрагмент вида:

```
        Exporter;
        AutoLoader;
@ISA = qw(Exporter AutoLoader);
```

Если ваш модуль использует и Perl и C (см. рецепт 12.14), уберите флаг -X из вызова *h2xs*. Сгенерированный фрагмент выглядит так:

```
        Exporter;
        DynaLoader;
@ISA = qw(Exporter DynaLoader);
```

Далее перечисляются переменные модуля Exporter (см. рецепт 12.1). Если вы пишете объектно-ориентированный модуль (см. главу 13), вероятно, вам вообще не придется использовать Exporter.

Подготовка завершена. Переходите к написанию кода своего модуля. Когда модуль будет готов к распространению, преобразуйте модуль в tar-архив для удобства распространения. Для этого используется команда *make dist* в командном интерпретаторе (имя программы make может зависеть от системы).

```
%make dist
```

Команда создает файл с именем вида Astronomy-Orbits-1.03.tar.Z.

Чтобы зарегистрироваться в качестве разработчика CPAN, обратитесь по адресу *http://www.perl.com/CPAN/modules/04pause.html*

Смотри также

h2xs(1); документация по стандартным модулям Exporter, AutoLoader, AutoSplit и ExtUtils::MakeMaker. По адресу *http://www.perl.com/CPAN* можно найти ближайший зеркальный узел и рекомендации, касающиеся предоставления модулей.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

12.9. Ускорение загрузки модуля с помощью SelfLoader

Проблема

Вам хочется быстро загрузить очень большой модуль.

Решение

Воспользуйтесь модулем SelfLoader:

```
        Exporter;
        SelfLoader;
@ISA = qw(Exporter SelfLoader);
#
# Прочие инициализации и объявления
#
__DATA__
sub abc { .... }
sub def { .... }
```

Комментарий

При загрузке модуля командой необходимо прочитать содержимое всего файла модуля и откомпилировать его (во внутренние деревья лексического анализа, не в байт-код или машинный код). Для очень больших модулей эта раздражающая задержка совершенно не нужна, если вам нужны всего несколько функций из конкретного файла.

Модуль SelfLoader решает эту проблему, откладывая компиляцию каждой подпрограммы до ее фактического вызова. Использовать SelfLoader несложно: достаточно расположить подпрограммы вашего модуля под маркером DATA, чтобы они были проигнорированы компилятором, обратиться к SelfLoader с помощью и включить SelfLoader в массив @ISA модуля. Вот и все, что от вас требуется. При загрузке модуля SelfLoader создает заглушки для функций, перечисленных в секции DATA. При первом вызове функции заглушка компилирует настоящую функцию и вызывает ее.

В модулях, использующих SelfLoader (или AutoLoader — см. рецепт 12.10), действует одно важное ограничение. Функции, загружаемые SelfLoader или AutoLoader, не имеют доступа к лексическим переменным файла, в чьем блоке DATA они находятся, поскольку они компилируются функцией eval в импортированном блоке AUTOLOAD. Следовательно, динамически сгенерированные функции компилируются в области действия AUTOLOAD модуля SelfLoader или AutoLoader.

Как скажется применение SelfLoader на быстродействии программы — положительно или отрицательно? Ответ на этот вопрос зависит от количества функций в модуле, от их размера и от того, вызываются ли они на протяжении всего жизненного цикла программы или нет.

Модуль SelfLoader не следует применять на стадии разработки и тестирования модулей. Достаточно закомментировать строку __DATA__, и функции станут видны во время компиляции.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Документация по стандартному модулю SelfLoader; рецепт **12.10.**

12.10. Ускорение загрузки модуля с помощью AutoLoader

Проблема

Вы хотите воспользоваться модулем AutoLoader.

Решение

Простейшее решение — воспользоваться утилитой ***h2xs*** для создания каталога и всех необходимых файлов. Предположим, у вас имеется каталог ***~/perllib,*** содержащий ваши личные библиотечные модули.

```
% h2xs -Xn Sample
% cd Sample
% perl Makefile.PL LIB=~/perllib
% (edit Sample.pm)
% make install
```

Комментарий

Модуль AutoLoader, как и SelfLoader, предназначен для ускорения работы программы. Он также генерирует функции-заглушки, которые заменяются настоящими функциями при первом вызове. Но вместо того чтобы искать все функции в одном файле под маркером `DATA__`, AutoLoader ищет определение каждой функции **в** отдельном файле. Например, если модуль Sample.pm содержит две функции, foo и `bar`, то AutoLoader будет искать их **в** файлах *Sample/auto/foo.al* и *Sample/auto/bar.al* соответственно. Модули, загружающие функции с помощью AutoLoader, работают быстрее тех, что используют SelfLoader, но за это приходится расплачиваться созданием дополнительных файлов, местом на диске и повышенной сложностью.

Процесс подготовки выглядит сложно. Вероятно, сделать это вручную действительно непросто. **К** счастью, **h2xs** оказывает громадную помощь. Помимо создания каталога с шаблонами Sample.pm **и** других необходимых файлов, утилита также генерирует Make-файл, который использует модуль AutoSplit для разделения функций модуля по маленьким файлам, по одной функции на файл. Правило `make install` устанавливает их так, чтобы они находились автоматически. Все, что от вас нужно, — разместить функции модуля после строки `__END__` (вместо строки `__DATA__` в SelfLoader), которая, как вы убедитесь, генерируется автоматически.

Как и в случае с SelfLoader, разработку и тестирование модуля лучше осуществлять без AutoLoader. Достаточно закомментировать строку `__END__`, пока модуль не придет к окончательному виду.

При работе с AutoLoader действуют те же ограничения видимости файловых лексических переменных, что и для SelfLoader, поэтому использование файловых

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

лексических переменных для хранения закрытой информации состояния не подойдет. Если вопрос хранения состояния становится настолько важным и труднореализуемым, подумайте о том, чтобы написать объектный модуль вместо традиционного.

Смотри также

Документация по стандартному модулю SelfLoader; *h2xs(1);* рецепт **12.9.**

12.11. Переопределение встроенных функций

Проблема

Вы хотите заменить стандартную функцию собственной версией.

Решение

Импортируйте нужную функцию из другого модуля **в** свое пространство имен.

Комментарий

Многие (хотя и не все) встроенные функции Perl могут переопределяться. **К** этому шагу следует относиться серьезно, но **в** принципе это возможно. Например, необходимость **в** переопределении может возникнуть при работе на платформе, которая не поддерживает эмулируемой функции. Также переопределение используется для создания интерфейсных оболочек для встроенных функций.

Не все зарезервированные слова одинаковы. Те, что возвращают отрицательное число в функции C `keyword()` файла *token.c* исходной поставки Perl, могут переопределяться. **В** версии **5.004** не допускалось переопределение следующих ключевых слов: `chop`, `defined`, `delete`, `do`, `dump`, `each`, `else`, `elsif`, `eval`, `exists`, `for`, `foreach`, `format`, `glob`, `if`, `keys`, `last`, `local`, `m`, `map`, `my`, `next`, `no`, `package`, `pop`, `pos`, `print`, `printf`, `prototype`, `push`, `q`, `qq`, `qw`, `qx`, `return`, `s`, `scalar`, `shift`, `sort`, `splice`, `split`, `study`, `sub`, `tie`, `tied`, `tr`, `undef`, `unless`, `unshift`, `untie`, `until`, `use`, `while` и `y`.

Стандартный модуль Perl Cwd переопределяет функцию `chdir`. Также переопределение встречается во многих модулях с функциями, возвращающими списки: File::stat, Net::hostent, Net::netent, Net::protoent, Net::servent, Time::gmtime, Time::localtime, Time::tm, User::grent и User::pwent. Эти модули содержат переопределения встроенных функций (например, `stat` или `getpwnam`), которые возвращают объект с возможностью доступа по имени — например, `getpwnam("daemon")->dir`. Для этого они переопределяют исходные, списковые версии этих функций.

Переопределение осуществляется импортированием функции из другого пакета. Импортирование действует только в импортирующем пакете, а не во всех возможных пакетах. Простого переобъявления недостаточно, функцию необходимо импортировать. Это защищает от случайного переопределения встроенных функций.

Предположим, вы решили заменить встроенную функцию `time`, которая возвращает целое количество секунд, другой, возвращающей вещественное число. Для

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

этого можно создать модуль FineTime с необязательным экспортированием функции time:

```
package FineTime;
usestrict;
            rter;
use vars qw(@ISA @EXPORT_OK);
@ISA = qw(Exporter);
@EXPORT_OK = qw(time);

sub time() {       }
```

Затем пользователь, желающий использовать усовершенствованную версию time, пишет что-то вроде:

```
use FineTime qw(time);
$start = time();
1 while print time() - $start, "\n";
```

Предполагается, что в вашей системе есть функция, соответствующая приведенной выше спецификации. Некоторые решения, которые могут работать в вашей системе, рассматриваются в рецепте 12.14.

Переопределение методов и операторов рассматривается в главе 13.

Смотри также

Раздел "Overriding Built-in functions*perlfSMb(l).

12.12. Вывод сообщений об ошибках и предупреждений по аналогии со встроенными функциями

Проблема

Ваш модуль генерирует ошибки и предупреждения, однако при использовании warn или die пользователь видит имя вашего файла и номер строки. Вы хотите, чтобы функции модуля вели себя по аналогии со встроенными функциями и сообщали об ошибках с точки зрения пользовательского, а не вашего кода.

Решение

Соответствующие функции присутствуют в стандартном модуле Carp. Вместо warn используйте функцию carp, а вместо die — функцию croak (для коротких сообщений) или confess (длядлинных сообщений).

Комментарий

Некоторые функции модуля, как и встроенные функции, могут генерировать предупреждения или ошибки. Предположим, вы вызвали функцию sqrt с отри-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

цательным аргументом (и не воспользовались модулем Math::Complex) — возникает исключение с выводом сообщения вида "Can't take sqrt of -3 at /tmp/negroot line 17", где */tmp/negroot* — имя вашей программы. Но если вы напишете собственную функцию с использованием die:

```
sub even_only {
    my $n = shift;
    die "$n is not even" if $n & 1;  # Один из способов проверки
    #....
}
```

то в сообщении вместо пользовательского файла, из которого вызывалась ваша функция, будет указан файл, в котором была откомпилирована функция even_only. На помощь приходит модуль Carp. Вместо die мы используем функцию croak:

```
use Carp;
sub even_only {
    my $n = shift;
    croak "$n is not even" if $n % 2;  Другой способ
    #
}
```

Если вы хотите просто вывести сообщение с номером строки пользовательской программы, где произошла ошибка, вызовите carp вместо warn (в отличие от warn и die, завершающий перевод строки в сообщениях carp и croak не имеет особой интерпретации). Например:

```
use Carp;
sub even_only {
    my$n=shift;
    if ($n & 1) {          # Проверка нечетности
        carp "$n is not even, continuing";
        ++$n;
    >
    #
}
```

Многие встроенные функции выводят предупреждения лишь при использовании ключа командной строки -w. Переменная $^W сообщает о его состоянии. Например, предупреждения можно выдавать лишь при наличии запроса от пользователя:

```
carp "$n is not even, continuing" if $^W;
```

Наконец, в модуле Carp существует третья функция — confess. Она работает аналогично croak за исключением того, что при аварийном завершении выводится полная информация о состоянии стека, вызовах функций и Значениях аргументов.

Смотри также

Описание функций warn и die в *perlmod(1)*; описание метапеременных WARN и __DIE_ в разделе "Global Special Arrays" *perlvar*(1) и в рецепте 16.15; документация по стандартному модулю Carp; рецепт 19.2.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

12.13. Косвенные ссылки на пакеты

Проблема

Требуется сослаться на переменную или функцию в пакете, имена которых неизвестны до момента выполнения программы, однако синтаксис `$packname::$varname` недопустим.

Решение

Воспользуйтесь символическими ссылками:

```
{
    no strict 'refs';
    $val = ${ $packname . "::" . $varname };
    @vals = @{ $packname . "::" . $aryname };
    &{ $packname . "::" . $funcname }("args");
    ($packname . "::" . $funcname) -> ("args");
}
```

Комментарий

Объявление пакета имеет смысл во время компиляции. Если имя пакета или переменной неизвестно до времени выполнения, придется прибегнуть к символическим ссылкам и организовать прямые обращения к таблице символов пакета. и постройте строку с полным именем интересующей вас переменной или функции. Затем разыменуйте полученную строку так, словно она является нормальной ссылкой Perl.

До выхода Perl версии 5 программистам в подобных случаях приходилось использовать eval:

```
eval "package $packname; \$'$val = \$$varname"; " Задать $main'val
die if $@;
```

Как видите, такой подход затрудняет построение строки. Кроме того, такой код работает относительно медленно. Впрочем, вам никогда не придется делать это лишь для того, чтобы косвенно обращаться к переменным по именам. Символические ссылки обеспечивают необходимый компромисс.

Функция eval также используется для определения функций во время выполнения программы. Предположим, вы хотите иметь возможность вычислять двоичные и десятичные логарифмы:

```
printf "log2 of 100 is %.2f\n", log2(100);
printf "log10 of 100 is %.2f\n", log10(100);
```

В Perl существует функция log для вычисления натуральных логарифмов. Давайте посмотрим, как использовать eval для построения функций во время выполнения программы. Мы создадим функции с именами от log2 до log999:

```
$packname = 'main';
for ($i = 2; $i < 1000; $i++) {
    $logN = log($i);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    eval "sub ${packname}::log$i { log(shift) / $logN }";
    die if $@;
}
```

По крайней мере в данном случае это не нужно. Следующий фрагмент делает то же самое, но вместо того, чтобы компилировать новую функцию 998 раз, мы откомпилируем ее всего единожды в виде замыкания. Затем мы воспользуемся символическим разыменованием в таблице символов и присвоим одну и ту же ссылку на функцию по многим именам:

```
$packname = 'main';
for ($i = 2; $i < 1000; $i++) {
    my $logN = log($i);
    no strict 'refs';
    *{"${packname}::log$i"} = sub { log(shift) / $logN };
}
```

Присваивая ссылку тип-глобу, вы всего лишь создаете синоним для некоторого имени. На этом принципе построена работа Exporter. Первая строка следующего фрагмента вручную экспортирует имя функции Colors::blue в текущий пакет. Вторая строка назначает функцию main::blue синонимом функции Colors::azure.

```
*blue      = \&Colors::blue;
*main::blue = \&Colors::azure;
```

Принимая во внимание гибкость присваиваний тип-глобов и символических ссылок, полноценные конструкции eval "СТРОКА" почти всегда оказываются излишеством, последней надеждой отчаявшегося программиста. Ничего худшего себе и представить нельзя — разве что если бы они были недоступны.

Смотри также

Раздел "Symbolic References» *perlsub*(1); рецепт 11.4

12.14. Применение h2ph для преобразования заголовочных файлов C

Проблема

Полученный от кого-то код выдает устрашающее сообщение об ошибке:

```
Can't locate sys/syscall.ph in @INC (did you run h2ph?)
(@INC contains: /usr/lib/perl5/i686-linux/5.00404 /usr/lib/perl5
/usr/lib/perl5/site_perl/i686-linux /usr/lib/perl5/site_perl .)
at some_program line 7.
```

Вы хотите понять, что это значит и как справиться с ошибкой.

Решение

Попросите системного администратора выполнить следующую команду с правами привилегированного пользователя:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
% cd /usr/include; h2ph sys/syscall.h
```

Однако многие заголовочные файлы включают другие заголовочные файлы; иными словами, придется преобразовать их все:

```
% cd /usr/include; h2ph *.h */*.h
```

Если вы получите сообщение о слишком большом количестве файлов или если некоторые файлы в подкаталогах не будут найдены, попробуйте другую команду:

```
% cd /usr/include; find . -name '*.h' -print | xargs h2ph
```

Комментарий

Файлы с расширением *.ph* создаются утилитой *h2ph,* которая преобразует директивы препроцессора С из `#include`-файлов в Perl. Это делается для того, чтобы программа на Perl могла работать с теми же константами, что и программа на С. Утилита *h2xs* обычно оказывается более удачным решением, поскольку вместо кода Perl, имитирующего код С, она предоставляет откомпилированный код С. Однако работа с *h2xs* требует намного большего опыта программирования (по крайней мере, в том, что касается С), чем *h2ph.*

Если процесс преобразования *h2ph* работает, все прекрасно. Если нет — что ж, вам не повезло. Усложнение системных архитектур и заголовочных файлов приводит к более частым отказам *h2ph.* Если повезет, необходимые константы уже будут присутствовать в модулях Fcntl, Socket или POSIX. В частности, модуль POSIX реализует константы из *sys/file.h, sys/errno.h* и *sys/wait.h.* Кроме того, он обеспечивает выполнение нестандартных операций с терминалом (см. рецепт 15.8).

Так что же можно сделать с файлом *.ph*? Рассмотрим несколько примеров. В первом примере непереносимая функция syscall используется для вызова системной функции `gettimeofday`. Перед вами реализация модуля FineTime, описанного в рецепте 12.11.

```
# Файл FineTime.pm
package main;
        'sys/syscall.ph';
die "No SYS_gettimeofday in sys/syscall.ph"
    unless defined &SYS_gettimeofday;

package FineTime;
    use strict;
            rter;
use vars qw(@ISA @EXPORT_OK);
@ISA = qw(Exporter);
@EXPORT_OK = qw(time);

sub time() {
            pack("LL",                  buffer to two longs
    syscall(&main::SYS_gettimeofday, $tv, undef) >= 0
        or die "gettimeofday: $!";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    my($seconds, $microseconds) = unpack("LL", $tv);
          $seconds + ($microseconds / 1_000_000);
}

1;
```

Если вам приходится вызывать старых файлов *.pl* или *.ph,* сделайте это из главного пакета (package main в приведенном выше коде). Эти старые библиотеки всегда помещают свои символические имена в текущий пакет, а main служит "местом встречи". Чтобы использовать имя, уточните его, как мы поступили с main::SYS_gettimeofday.

Файл ***sys/ioctl.ph,*** если вам удастся построить его **в** своей системе, открывает доступ **к** функциям ввода/вывода вашей системы через функции ioctl. **К** их числу принадлежит функция TIOCSTI **из** примера **12.1.** Сокращение TIOCSTI означает "управление терминальным вводом/выводом, имитация терминального ввода" (terminal **I/O** control, simulate terminal input). **В** системах, где эта функция реализована, она вставляет один символ **в** поток устройства, чтобы при следующем чтении из устройства со стороны любого процесса **был** получен вставленный символ.

Пример 12.1. jam

```
#!/usr/bin/perl -w
# jam - вставка символов в STDIN
        'sys/ioctl.ph';
die "no TIOCSTI" unless defined &TIOCSTI;
sub jam {
    local $SIG{TTOU} = "IGNORE"; # "Остановка для вывода на терминал"
    local *TTY;  # Создать локальный манипулятор
    open(TTY, "+</dev/tty")                or die "no tty: $!";
    for (split(//, $_[0])) <
        ioctl(TTY, &TIOCSTI, $_)           or die "bad TIOCSTI:  $!";
    }
    close(TTY);
}
jam("@ARGV\n");
```

Поскольку преобразование ***sys/ioctl.h*** может вызвать некоторые сложности, вероятно, для получения кода TIOCSTI вам придется запустить следующую программу на C:

```
% cat > tio.c "EOF && cc tio.c && a.out
#include  <sys/ioctl.h>
main() { printf("%#08x\n", TIOCSTI); }
EOF
0x005412
```

Функция ioctl также часто применяется для определения размеров текущего окна в строках/столбцах и даже в пикселях. Исходный текст программы приведен в примере 12.2.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 12.2. winsz

```
#!/usr/bin/perl
# winsz - определение размеров окна в символах и пикселях
        'sys/ioctl.ph';
die "no TIOCGWINSZ " unless defined &TIOCGWINSZ;
open(TTY, "+</dev/tty")                    or die "No tty: $!";
unless (ioctl(TTY, &TIOCGWINSZ, $winsize='')) {
    die sprintf "$0. ioctl TIOCGWINSZ (%08x: $!)\n", &TIOCGWINSZ;
}
($row, $col, $xpixel, $ypixel) = unpack('S4', $winsize);
print "(row,col) = ($row,$col)";
print "  (xpixel,ypixel) = ($xpixel,$ypixel)" if $xpixel || $ypixel;
print "\n";
```

Как видите, для экспериментов с файлами .ph, распаковкойдвоичныхданных и вызовами syscall и ioctl необходимо хорошо знать прикладной интерфейс C, обычно скрываемый Perl. Единственное, что требует такого же уровня знаний C — это интерфейс XS. Одни считают, что программисты должны бороться с искушением и за версту обходить подобные непереносимые решения. По мнению других, жесткиетребования, поставленные перед рядовым программистом, оправдывают самые отчаянные меры.

К счастью, все большее распространение получают менее хрупкие механизмы. Для большинства этих функций появились модули CPAN. Теоретически они работают надежнее, чем обращения к файлам *.ph*.

Смотри также

Описание функций syscall и ioctl в *perlmod*(1);инструкции по работе с *h2ph* в файле *INSTALL* исходной поставки Perl; *h2ph(1);* рецепт 12.15.

12.15. Применение h2xs для создания модулей с кодом C

Проблема

Вам хотелось бы работать с функциями C из Perl.

Решение

Воспользуйтесь утилитой h2xs для построения необходимых файлов шаблонов, заполните их соответствующим образом и введите:

```
% perl Makefile PL
% make
```

Комментарий

При написании модуля Perl необязательно ограничиваться одним Perl. Как и для любого другого модуля, выберите имя и вызовите для него утилиту *h2xs*. Мы со-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

здадим функцию FineTime::time с той же семантикой, что и в предыдущем рецепте, но на этот раз реализуем ее на C.

Сначала выполните следующую команду:

```
% h2xs -cn FineTime
```

Если бы у нас был файл *.h* с объявлениями прототипов функций, его можно было бы включить, но поскольку мы пишем модуль с нуля, используется флаг -c — тем самым мы отказываемся от построения кода, преобразующего директивы #define. Флаг **-n** требует создать для модуля каталог *FineTime/*, в котором будут находиться следующие файлы:

Файл	Список файлов в поставке
Changes	Протокол изменений
Makefile.PL	Мета-make-файл
FineTime.pm	Компоненты Perl
FineTime.xs	Будущие компоненты C
test.pl	Тестовая программа

Перед тем как вводить команду make, необходимо сгенерировать make-файл для текущей системной конфигурации с помощью шаблона Makefile.PL. Вот как это делается:

```
% perl Makefile.PL
```

Если код XS вызывает библиотечный код, отсутствующий в нормальном наборе библиотек Perl, сначала добавьте в *Makefile.pl* новую строку. Например, если мы хотим подключить библиотеку *librpm.a* из каталога */usr/redhat/lib*, то нам надо изменить строку Makefile.PL:

```
'LIBS'      => [''],    # e.g., '-lm'
```

и привести ее к виду:

```
'LIBS'      => ['-L/usr/redhat/lib -lrpm'],
```

Наконец, отредактируйте файлы *FineTime.pm* и *FineTime.xs*. В первом случае большая часть работы уже сделана за нас. Нам остается создать список экспортируемых функций. На этот раз мы помещаем его в @EXPORT_OK, чтобы нужные функции запрашивались пользователем по имени. Файл *FineTime.pm* выглядит так:

```
package FineTime;
use strict;
use vars qw($VERSION @ISA @EXPORT_OK);
        Exporter;
        DynaLoader;
@ISA = qw(Exporter DynaLoader);
@EXPORT_OK = qw(time);
$VERSION = '0.01';
bootstrap FineTime $VERSION;
1;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Make автоматически преобразует файл *FineTime.xs* в *FineTime.c* и общую библиотеку, которая на большинстве платформ будет называться *FineTime.so*. Преобразование выполняется утилитой *xsubpp*, описанной в ее собственной странице руководства и *perlxstut*(1). *Xsubpp* автоматически вызывается в процессе построения.

Кроме хороших познаний в C, вы также должны разбираться в интерфейсе C-Perl, который называется XS (eXternal Subroutine). Подробности и нюансы XS выходят за рамки этой книги. Автоматически сгенерированный файл *FineTimejcs* содержит заголовочные файлы, специфические для Perl, а также объявление MODULE. Мы добавили несколько дополнительных файлов и переписали код новой функции time. На C пока не похоже, но после завершения работы *xsubpp* все придет в норму.

Использованный нами файл *FineTimejcs* выглядит так:

```
#include <unistd.h>
#include <sys/time.h>
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

MODULE = FineTime          PACKAGE = FineTime

double
time()
    CODE:
        struct timeval tv;
        gettimeofday(&tv,0);
        RETVAL = tv.tv sec + ((double) tv.tv_usec) / 1000000;
    OUTPUT:
        RETVAL
```

Определение функции с именем, присутствующем в стандартной библиотеке C, не вызовет проблем при компиляции — это не настоящее имя, а лишь псевдоним, используемый Perl. Компоновщик C увидит функцию с именем XS_FineTime_time, поэтому конфликта не будет.

При выполнении команды make install происходит следующее (с небольшими исправлениями):

```
% make install
mkdir ./blib/lib/auto/FineTime
cp FineTime.pm ./blib/lib/FineTime.pm
/usr/local/bin/perl -I/usr/lib/perl5/i686-linux/5.00403  -I/usr/lib/perl5
/usr/lib/perl5/ExtUtils/xsubpp  -typemap
  /usr/lib/perl5/ExtUtils/typemap  FineTime.xs
FineTime.tc && mv FineTime.tc FineTime.ccc -c -Dbool=char -DHAS_BOOL
    -O2-DVERSION=\"0.01\" -DXS_VERSION=\"0.01\" -fpic
    -I/usr/lib/perl5/i686-linux/5.00403/CORE
FineTime.cRunning Mkbootstrap for FineTime ()
chmod 644 FineTime.bs
LD_RUN_PATH="" cc -o blib/arch/auto/FineTime/FineTime.so
            -L/usr/local/lib FineTime.o
chmod 755 blib/arch/auto/FineTime/FineTime.so
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
cp FineTime.bs ./blib/arch/auto/FineTime/FineTime.bs
chmod 644 blib/arch/auto/FineTime/FineTime.bs
Installing /home/tchrist/perllib/i686-linux/./auto/FineTime/FineTime.so
Installing /home/tchrist/perllib/i686-linux/./auto/FineTime/FineTime.bs
Installing /home/tchrist/perllib/./FineTime.pm
Writing /home/tchrist/perllib/i686-linux/auto/FineTime/.packlist •
Appending installation info to /home/tchrist/perllib/i686-linux/perllocal.pod
```

Когда все будет готово, в интерпретаторе вводится следующая команда:

```
% perl -I ^/perllib -MFineTime=time -le '1 while print time()' | head
888177070.090978
888177070.09132
888177070.091389
888177070.091453
888177070.091515
888177070.091577
888177070.091639
888177070.0917
888177070.091763
888177070.091864
```

Смотри также

Документация по стандартному модулю ExtUtils::MakeMaker; *h2ph*(1) и *xsubpp*(1). Вызовы функций C из Perl описаны в perlxstut(l) и perlxs(l), а вызовы функций Perl из C — в perlembed(l). Внутренний API Perl рассматривается в *perlcall*(1) и *perlguts*(1). По адресу *http://www.perl.com/CPAN/authors/Dean_Roehrich/* находится подробное руководство по XS с рекомендациями по организации интерфейса с C++.

12.16. Документирование модуля в формате pod

Проблема

Вы хотите документировать свой модуль, но не знаете, какой формат следует использовать.

Решение

Включите документацию в файл модуля в формате pod.

Комментарий

Сокращение pod означает "plain old documentation", то есть "простая документация". Документация в формате pod включается в программу с применением очень простого формата разметки. Как известно, программисты сначала пишут программу, а документацию... не пишут вообще. Формат pod был разработан для максимальной простоты документирования, чтобы с этой задачей справился даже лентяй. Иногда это даже помогает.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если во время анализа исходного текста Perl обнаруживает строку, начинающуюся со знака = (там, где ожидается новая команда), он игнорирует весь текст до строки, начинающейся с `=cut`, после чего продолжает анализировать код. Это позволяет смешивать в программах или файлах модулей Perl код и документацию. Поскольку формат pod является сугубо текстовым, никакого особого форматирования не требуется. Трансляторы стараются проявить интеллект и преобразуют вывод так, чтобы программисту не приходилось особым образом форматировать имена переменных, вызовы функций и т. д.

Вместе с Perl поставляется несколько программ-трансляторов, которые фильтруют документацию в формате pod и преобразуют ее в другой формат вывода. Утилита *pod2man* преобразует pod в формат *troff*, используемый в программе man или в системах верстки и печати. Утилита *pod2html* создает Web-страницы, работающие в системах, не принадлежащих к семейству UNIX. Утилита *pod2text* преобразует pod в простой ASCII-текст. Другие трансляторы *fyod2ipf,pod2fm,pod2text, pod2latex* и *pod2ps)* могут входить в поставку Perl или распространяются через CPAN.

Многие книги пишутся в коммерческих текстовых редакторах с ограниченными сценарными возможностями... но только не эта! Она была написана в формате pod в простых текстовых редакторах (Том использовал *vi*, а Нат — *emacs*). На стадии технической правки книга была преобразована в формат *troff* специальным транслятором *pod2ora*, написанным Ларри. Окончательный вариант книги был получен преобразованием pod-файлов в формат FrameMaker.

Хотя в *perlpod*(1) приведено общее описание pod, вероятно, этот формат удобнее изучать на примере готовых модулей. Если вы начали создавать собственные модули с помощью утилиты *h2xs,* то у вас уже имеются образцы. Утилита Makefile знает, как преобразовать их в формат man и установить страницы руководства так, чтобы их могли прочитать другие. Кроме того, программа *perldoc* может транслировать документацию pod с помощью *pod2text*.

Абзацы с отступами остаются без изменений. Другие абзацы переформатируются для размещения на странице. В pod используются лишь два вида служебной разметки: абзацы, начинающиеся со знака = и одного или нескольких слов, и внутренние последовательности в виде буквы, за которой следует текст в угловых скобках. Теги абзацев определяют заголовки, перечисляемые элементы списков и служебные символы, предназначенные для конкретного транслятора. Последовательности в угловых скобках в основном используются для изменения начертания (например, выбора полужирного, курсивного или моноширинного шрифта). Приведем пример директивы `=head2` в сочетании с изменениями шрифта:

```
=head2  Discussion

If we had a dot-h file with function prototype declarations, we
could include that, but since we're writing this one from scratch,
we'll use the -c flag to omit building code to translate any
#define symbols. The -n flag says to          a module
named FineTime/, which will have the following files.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Последовательность =for определяет код для выходных файлов конкретного формата. Например, в этой книге, главным образом написанной в формате pod, присутствуют вызовы стандартных средств *troff* eqn, tbl и pic. Ниже показан пример внутреннего вызова eqn, который обрабатывается лишь трансляторами, производящими данные в формате *troff:*

```
=for troff
.EQ
log sub n (x) = { {log sub e  (x)} over {log sub e  (n)} }
.EN
```

Формат pod также позволяет создавать многострочные комментарии. В языке С комментарий /*....*/ может включать несколько строк текста — вам не придется ставить отдельный маркер в каждой строке. Поскольку Perl игнорирует директивы pod, этим можно воспользоваться для блочного комментирования. Весь фокус заключается в том, чтобы найти директиву, игнорируемую трансляторами pod. Например, можно воспользоваться тегом for later или for nobody:

```
=for later
next if 1 .. ?^$?;
s/^(.)/>$1/;
s/(.{73})          */$1<SNIP>/;

=cut back to perl
```

или парой =begin и =end:

```
=begin comment

if (!open(FILE, $file)) {
    unless ($opt_q) {
        warn "$me: $file: $!\n";
        $Errors++;
    }
    next FILE;
}

$total = 0;
$matches = 0;

=end comment
```

Смотри также

Раздел "POD: Enbedded Documentation" *в perlsyn(1);perlpod(1),pod2man(1), pod2html*(1) и *pod2text*(1).

12.7. Построение и установка модуля CPAN

Проблема

Требуется установить файл модуля, загруженный с CPAN или взятый с компакт-диска.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Введите в интерпретаторе следующие команды (на примере установки модуля Some::Module версии 4.54):

```
% gunzip Some-Module-4.54.tar.gz
% tar xf Some-Module-4.54
% cd Some-Module-4.54
% perl Makefile.PL
% make
% make test
% make install
```

Комментарий

Модули Perl, как и большинство программ в Сети, распространяются в архивах *tar*, сжатых программой GNU *zip*[1]. Если tar выдает предупреждение об ошибках контрольных сумм каталогов checksum errors"), значит, вы испортили двоичный файл, приняв его в текстовом формате.

Вероятно, для установки модуля в системные каталоги необходимо стать привилегированным пользователем с соответствующими правами доступа. Стандартные модули обычно устанавливаются в каталог */usr/lib/perl5*, прочие — в каталог */usr/lib/perl5/site_perl.*

Рассмотрим процесс установки модуля MD5:

```
% gunzip MD5-1.7.tar.gz
% tar xf MD5-1.7.tar
% cd MD5-1.7
% perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for MD5
X make
mkdir ./blib
mkdir ./blib/lib
cp MD5.pm ./blib/lib/MD5.pm
AutoSplitting MD5 (./blib/lib/auto/MD5)
/usr/bin/perl -I/usr/local/lib/perl5/i386 ...

op MD5.bs ./blib/arch/auto/MD5/MD5.bs
chmod 644 ./blib/arch/auto/MD5/MD5.bsmkdir ./blib/man3
Manifying ./blib/man3/MD5.3
% make test
PERL_DL_NONLAZY=1 /usr/bin/perl -I./blib/arch -I./blib/lib
                                -I/usr/looal/lib/perl5 test.pl
```

[1] Этот формат отличается от формата zip, используемого на большинстве компьютеров с Windows, однако новые версии Winzip читают его. До выхода Perl 5.005 для построения модулей CPAN использовалась стандартная версия Perl для Win32, а не ActiveState. Также существуют бесплатные версии tar и GNU tar для систем Microsoft.

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг


```
1..14
ok 1
ok 2

ok 13
ok 14
% sudo make install
Password:
Installing /usr/local/lib/perl5/site_perl/i386-freebsd/./auto/MD5/
    MDS.so
Installing /usr/local/lib/perl5/site_perl/i386-freebsd/./auto/MD5/
    MD5.bs
Installing /usr/local/lib/perl5/site_perl/./auto/MD5/autosplit.ix
Installing /usr/local/lib/perl5/site_perl/./MD5.pm
Installing /usr/local/lib/perl5/man/man3/./MD5.3
Writing /usr/local/lib/perl5/site_perl/i386-freebsd/auto/MD5/.packlist
Appending installation info to /usr/local/lib/perl5/i386-freebsd/
5.00404/perllocal.pod
```

Если ваш системный администратор где-то пропадает или у него нет времени на установку, не огорчайтесь. Используя Perl для построения **make-файла** по шаблону ***Makefile.PL***, можно выбрать альтернативный каталог для установки.

```
# Если вы хотите установить модули в свой каталог
% perl Makefile.PL LIB=~/lib

# Если у вас имеется полная поставка
% perl Makefile.PL PREFIX=~/perl5-private
```

Смотри также

Документация по стандартному модулю **ExtUtils::MakeMaker**. Файл *INSTALL* в исходной поставке Perl содержит сведения о построении двоичного файла *perl* со статической компоновкой.

12.18. Пример: шаблон модуля

Ниже приведен "скелет" модуля. Если вы собираетесь написать собственный модуль, попробуйте скопировать и отредактировать его.

```
package Some::Module;  # Должен находиться в Some/Module.pm

use strict;

        Exporter;
use vars        qw($VERSION @ISA @EXPORT @EXPORT_OK %EXPORT_TAGS);

# Установка версии для последующей проверки
$VERSION      = 0.01;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
@ISA         = qw(Exporter);
@EXPORT      = qw(&func1  &func2 &func4);
%EXPORT_TAGS = ( );     # например: TAG => [ qw!name1 name2!  ],

fl Здесь находятся экспортируемые глобальные переменные,
# а также функции с необязательным экспортированием
@EXPORT_OK   = qw($Var1 %Hashit &func3);

use vars qw($Var1  %Hashit);
# Здесь находятся неэкспортируемые глобальные имена пакета
use vars      qw(@more $stuff);

# Инициализировать глобальные переменные пакета,
# начиная с экспортируемых
$Var1   = '';
%Hashit  = ();

# Затем все остальные (к которым можно обращаться
# в виде $Some::Module::stuff)
$stuff  = '';
@more   = ();

# Все лексические переменные с файловой областью действия
# должны быть созданы раньше функций, которые их используют.

# Лексические переменные, доступ к которым
# ограничивается данным файлом.
my $priv_var    = '';
my %secret_hash = ();

tt Закрытая функция, оформленная в виде замыкания
tt и вызываемая через &$priv_func.
my $priv_func = sub {
    # Содержимое функции.
>;

# Все ваши функции, экспортируемые и нет;
# не забудьте вставить что-нибудь в заглушки {}
sub   func1   { .... }    tt без прототипа
sub  func2()  { .... }    # прототип - void
sub  func3($$)  { — }     # прототип - 2 скаляра

# Функция не экспортируется автоматически, но может вызываться!
sub  func4(\%)  { .... }    # прототип - 1 ссылка на хэш

END { }    ff    Завершающий код модуля (глобальный деструктор)

1;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

12.19. Программа: поиск версий и описаний установленных модулей

Perl распространяется вместе с множеством модулей. Еще больше модулей можно найти в CPAN. Следующая программа выводит имена, версии и описания всех модулей, установленных в вашей системе. Она использует стандартные модули (например, File::Find) и реализует некоторые приемы, описанные в этой главе.

Программа запускается следующей командой:

```
% pmdesc
```

Она выводит список модулей с описаниями:

```
FileHandle (2.00) - supply object methods for filehandles
IO::File (1.06021) - supply object methods for filehandles
IO::Select (1.10) - 00 interface to the select system call
IO::Socket (1.1603) - Object interface to socket communications
```

С флагом -v программа pmdesc выводит имена каталогов, в которых находятся файлы:

```
% pmdesc -v

<<<Modules from /usr/lib/perl5/i686-linux/5.00404>>>

FileHandle (2.00) - supply object methods for filehandles
```

Флаг -w предупреждает о том, что модуль не включает документации в формате pod, а флаг -s сортирует список модулей в каждом каталоге.

Исходный текст программы приведен в примере 12.3.

Пример 12.3. pmdesc

```
#!/usr/bin/perl -w
# pmdesc - вывод описаний файлов pm
# tchrist@perl.com

use strict;
use File::Find      qw(find);
use Getopt::Std     qw(getopts);
use Carp;

use vars (
    q!$opt_v!,          # Вывод отладочной информации
    q!$opt_w!,          # Предупреждения об отсутствующих
                        # описаниях модулей
    q!$opt_a!,          # Вывод относительных путей
    q!$opt_s!,          # Сортировка данных по каждому каталогу
);
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 12.3 (продолжение)

```
$| = 1.

getopts( wvas )             or die bad usage ,

@ARGV = @INC unless @ARGV,

# Глобальные переменные  Я бы предпочел обойтись без этого
use vars (
    q'$Start_Dir',     # Каталог верхнего уровня, для которого
                       # вызывалась функция find
    q'%Future',        # Другие каталоги верхнего уровня,
                       # для которых find вызывается позднее
),

my $Module,

# Установить фильтр для сортировки списка модулей,
# если был указан соответствующий флаг
if ($opt_s) {
    if (open(ME, -| )) {
        $/ =   ,
        while (<ME>) {
            chomp,
            print join( \n , sort split /\n/),  \n ,
        }
        exit,
    }
}

MAIN  {
    my %visited,
    my ($dev,$ino),

    @Future{@ARGV} = (1) x @ARGV,

    foreach $Start_Dir (@ARGV) {
        delete $Future{$Start_Dir},

        print  \n<<Modules from $Start_Dir>>\n\n'
            if $opt_v,

        next unless ($dev,$ino) = stat($Start_Dir),
        next if $visited{$dev,$ino}++,
        next unless $opt_a || $Start_Dir =^ m'^/',

        find(\&wanted, $Start_Dir),
    }
    exit
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

8 Вычислить имя модуля по файлу и каталогу

```
subfflodname{
    local $_ = $File::Find::name;

    if (index($_, $Start_Dir . '/') == 0) {
        substr($ , 0, 1+length($Start_Dir)) = '';
    >

    s { /              }    {::}gx;
    s { \.p(m|od)$     }    {}x;


}

# Решить, нужен ли нам данный модуль
sub wanted {
    if ( $Future{$File::Find::name} ) {
        warn "\t(Skipping $File::Find::name, qui venit in futuro.)\n"
            if 0 and $opt_v;
        $File::Find::prune = 1;
        return;
    }
            unless /\.pm$/ && -f;
    $Module = &modname;
    # skip obnoxious modules
    if ($Module =~ /^CPAN(\Zi::)/) {
        warn("$Module -- skipping because it misbehaves\n");

    }

    my    $file = $_;

    unless (open(POD, "< $file")) {
        warn "\tcannot open $file: $!";
            # if $opt_w;

    }

    $: = " -:";

    local $/ = '';
    local $_;
    while (<POD>) {
        if (/=head\d\s+NAME/) {
            chomp($_ = <POD>);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 12.3 (продолжение)

```
            if (defined ($v = getversion($Module))) {
                print  $Module ($v)  ,
            } else {
                print  $Module  ,
            }
            print  - $_\n ,

        }
    }

    warn  \t(MISSING DESC FOR $File  Find  name)\n
        if $opt_w,

          0,
}

# Загрузить модуль и вывести его номер версии,
# перенаправляя ошибки в /dev/null
sub getversion {
    my $mod = shift,

    my $vers =  $^X -m$mod -e  print \$${mod>  VERSION  2>/dev/null ,
    $vers =^ s/^\s*( *?)\s*$/$1/, # Удалить лишние пропуски
            ($vers || undef),
}

format  =
 ^<<<<<<<<<<<<<<<<<~~~^<<<<<<<<<<<<<<<<
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Классы, объекты и связи 13

По всему миру я призываю массы на борьбу с классами.

Уильям Гладстон, речь в Ливерпуле, 28 июня 1886 г.

Введение

Наряду со ссылками и модулями в Perl версии 5.000 появились объекты. Как обычно, Perl не заставляет всех использовать единственно правильный стиль, а поддерживает несколько разных стилей. Благодаря этому люди решают свои задачи так, как им нравится.

При написании программ необязательно пользоваться объектами, в отличие от языка Java, где программы представляют собой экземпляры объектов. Однако при желании можно написать Perl-программу, в которой используется практически весь арсенал приемов объектно-ориентированного программирования. В Perl поддерживаются классы и объекты, одиночное и множественное наследование, методы экземпляров и методы классов, переопределение методов, конструкторы и деструкторы, перегрузка операторов, методы-посредники с автозагрузкой, делегирование, иерархия объектов и два уровня сборки мусора.

Вы можете выбрать ровно столько объектно-ориентированных принципов, сколько захочется. Связи (ties) являются единственной частью Perl, где объектно-ориентированный подход обязателен. Но даже здесь об этом должен знать лишь программист, занимающийся реализацией модуля; случайный пользователь остается в блаженном неведении относительно внутренних механизмов. Связи, рассматриваемые в рецепте 13.14, позволяют организовать прозрачный перехват обращений к переменной. Напри мер, с помощью связей можно создать хэш с возможностью поиска по ключу или по значению.

Под капотом

Если спросить десятерых программистов, что такое "объектная ориентация", вы получите десять разных ответов. Люди рассуждают об "абстракции" и "инкапсуляции", пытаются выделить основные черты объектно-ориентированных языков

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

программирования и придумать для них умные термины, чтобы потом писать статьи и книги. Не все объектно-ориентированные языки обладают одинаковыми возможностями, но все они считаются объектно-ориентированными. Конечно, в результате появляются все новые статьи и книги.

Мы будем использовать терминологию из документации Perl и страницы руководства *perlobj*(1). *Объект* представляет собой переменную, принадлежащую к некоторому *классу*. *Методами* называются функции, ассоциируемые с классом или объектом. В Perl класс представляет собой пакет — а обычно и модуль. Объект является ссылкой на что-то, что было приведено (blessed) к классу. Приведение ассоциирует субъект с классом. Для этого используется функция bless, вызываемая с одним или двумя аргументами. Первым аргументом является ссылка на приводимый объект, а необязательным вторым аргументом — пакет, к которому осуществляется приведение.

```
$object = {};                        # Ссылка на хэш
bless($object, "Data::Encoder");     # Привести $object к классу
                                      # Data::Encoder
bless($object);                      # Привести $object к текущему пакету
```

Имя класса соответствует имени пакета (Data::Encoder в приведенном выше примере). Поскольку классы являются модулями (обычно), код класса Data::Encoder находится в файле *Data/Encoder.pm*. Структура каталогов, как и для традиционных модулей, существует исключительно для удобства; она никак не связана с наследованием, ограничением доступа к переменным или чем-нибудь еще. Однако в отличие от традиционных модулей, объектные модули очень редко используют Exporter. Вся работа должна вестись только через вызовы методов, но не через импортированные функции или переменные.

После приведения объекта вызов функции ref для ссылки на него возвращает имя класса вместо фундаментального типа субъекта:

```
$obj = [3,5];
print          " ", $obj->[1], "\n";
bless($obj, "Human::Cannibal");
print ref($obj), " ", $obj->[1], "\n";

ARRAY 5
Human::Cannibal 5
```

Как видите, приведенную ссылку все еще можно разыменовать. Чаще всего объекты реализуются с помощью приведенных ссылок на хэши. Вы можете использовать любые типы ссылок, но ссылки на хэш обеспечивают максимальную гибкость. Они позволяют создавать в объекте поля данных с произвольными именами:

```
$obj->{Stomach} = "Empty";   я Прямое обращение к данным объекта
$obj->{NAME}    = "Thag";
 # Символы верхнего регистра в имени поля
 # помогают выделить его (необязательно)
```

Хотя Perl позволяет любому коду за пределами класса напрямую обращаться к данным объекта, это считается нежелательным. Согласно общепринятому мне-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

нию, работа с данными должна вестись только с использованием методов, предназначенных для этой цели. У разработчика класса появляется возможность изменить его реализацию без модификации всего кода приложений, использующих данный класс.

Методы

Для вызова методов используется оператор, оператор `->`. В следующем примере мы вызываем метод `encode()` объекта `$object` с аргументом `"data"` и сохраняем возвращаемое значение в переменной `$encoded`:

```
$encoded = $object->encode("data");
```

Перед нами *метод объекта,* поскольку мы вызываем метод конкретного объекта. Также существуют *методы классов,* то есть методы, вызываемые по имени класса:

```
$encoded = Data::Encoder->encode("data");
```

При вызове метода вызывается функция соответствующего класса с неявной передачей в качестве аргумента либо ссылки (для метода объекта), либо строки (для метода класса). В рецепте 13.17 показано, как вызывать методы с именами, определяемыми во время выполнения.

В большинстве классов существуют специальные методы, возвращающие новые объекты — *конструкторы.* В отличие от некоторых объектно-ориентированных языков, конструкторы Perl не имеют специальных имен. В сущности, конструктор можно назвать, как вам захочется. Программисты C++ обожают присваивать своим конструкторам в Perl имя `new`. Мы рекомендуем выбирать имя конструктора так, чтобы оно имело смысл в контексте решаемой задачи. Например, конструкторы расширения Tk в Perl называются по именам создаваемых ими элементов (widgets). Менее распространенный подход заключается в экспортировании функции, имя которой совпадает с именем класса; см. пример в разделе "Пример. Перегруженный класс StrNum" в рецепте 13.14.

Типичный конструктор выглядит следующим образом:

```
sub new {
    my $class = shift;
    my $self  = {};          #  Выделить новый хэш для объекта
    bless($self, $class);
            $self;
}
```

Вызов конструктора выглядит так:

```
$object = Class->new();
```

Если дело обходится без наследования или иных выкрутасов, это фактически эквивалентно

```
$object = Class::new("Class");
```

Первым аргументом функции `new()` является имя класса, к которому приводится новая ссылка. Конструктор должен передать эту строку `bless()` в качестве второго аргумента.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В рецепте 13.1 также рассматриваются функции, возвращающие приведенные ссылки. Конструкторы не обязаны быть методами класса. Также встречаются методы объектов, возвращающие новые объекты (см. рецепт 13.6).

Деструктором называется функция, которая выполняется при уничтожении субъекта, соответствующего данному объекту, в процессе сборки мусора. В отличие от конструкторов имена деструкторов жестко фиксируются. Методу-деструктору должно быть присвоено имя DESTROY. Этот метод, если он существует, вызывается для всех объектов непосредственно перед освобождением памяти. Наличие деструктора (см. рецепт 13.2) необязательно.

Некоторые языки на уровне синтаксиса позволяют компилятору ограничить доступ к методам класса. В Perl такой возможности нет — программа может вызывать любые методы объекта. Автор класса должен четко документировать *открытые* методы (те, которые можно использовать). Пользователям класса следует избегать недокументированных (тоесть неявно *закрытых)* методов.

Perl не различает методы, вызываемые для класса (методы классов), и методы, вызываемые для объекта (методы экземпляров). Если вы хотите, чтобы некоторый метод вызывался только как метод класса, поступите следующим образом:

```
sub class_only_method {
    my $class = shift;
    die "class method                    $class;
    # Дополнительный код
}
```

Чтобы метод вызывался только как метод экзе мпляра, воспользуйтесь следующим кодом:

```
sub instance_only_method {
    ray $self = shift;
    die "instance method called on class" unless     $self;
    # Дополнительный код
}
```

Если в вашей программе вызывается неопределенный метод объекта, Perl не будет жаловаться на стадии компиляции; вместо этого произойдет исключение во время выполнения. Аналогично, компилятор не перехватывает ситуации, при которой методу, который должен вызываться для простых чисел, передается комплексный аргумент. Метод представляет собой обычный вызов функции, пакет которой определяется во время выполнения. Методы, как и все косвенные функции, не имеют проверки прототипа — проверка выполняется на стадии компиляции. Даже если бы вызовы методов учитывали наличие прототипа, в Perl компилятор не сможет автоматически установить точный тип или интервал аргумента функции. Прототипы Perl предназначены для *форсирования* контекста аргумента функции, а не для проверки интервала. Странности прототипов Perl описаны в рецепте ,10.11.

Чтобы предотвратить инициирование исключений для неопределенных методов, можно использовать механизм AUTOLOAD для перехвата вызовов несуществующих методов. Данная возможность рассматривается в рецепте 13.11.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Наследование

Отношения наследования определяют иерархию классов. При вызове метода, не определенного в классе, поиск метода с указанным именем осуществляется в иерархии. Используется первый найденный метод. Наследование позволяет строить классы "на фундаменте" других классов, чтобы код не приходилось переписывать заново. Классы являются одной из форм многократного использования кода и потому способствуют развитию Лени — главной добродетели программиста.

В некоторых языках существует специальный синтаксис наследования. В Perl каждый класс (пакет) может занести список своих суперклассов, то есть родителей в иерархии, в глобальную (нелексическую!) пакетную переменную @ISA. Этот список просматривается во время выполнения программы, при вызове метода, не определенного в классе объекта. Если первый пакет, указанный в @ISA, не содержит искомого метода, но имеет собственный массив @ISA, то Perl перед продолжением поиска рекурсивно просматривает @ISA *этого* пакета.

Если поиск унаследованного метода заканчивается неудачей, проверка выполняется заново, но на этот раз ищется метод с именем AUTOLOAD. Поиск метода $ob->meth(), где объект $ob принадлежит классу P, происходит в следующей последовательности:

- P::meth
- Любой метод S::meth() в пакетах S из @P::ISA, рекурсивно.
- UNIVERSAL::meth
- Подпрограмма P::AUTOLOAD.
- Любой метод S::AUTOLOAD() в пакетах S из @P::ISA, рекурсивно.
- Подпрограмма UNIVERSAL::AUTOLOAD.

В большинстве классов массив @ISA состоит из одного элемента — такая ситуация называется *одиночным наследованием*. Если массив @ISA содержит несколько элементов, говорят, что класс реализует *множественное наследование*. Вокруг достоинств и недостатков множественного наследования идут постоянные споры, но Perl поддерживает эту возможность.

В рецепте 13.9 рассматриваются основы наследования и базовые принципы построения классов, обеспечивающие удобство субклассирования. В рецепте 13.10 мы покажем, как субкласс переопределяет методы своих суперкласов.

Perl не поддерживает наследования данных. Класс может напрямую обращаться к данным другого класса, но делать этого не следует. Это не соответствует принципам инкапсуляции и нарушает абстракцию. Если вы последуете рекомендациям из рецептов 13.10 и 13.12, это ограничение не вызовет особых проблем.

Косвенный вызов методов

Косвенный вызов методов:

```
$lector = new Human::Cannibal;
feed $lector "Zak";
move $lector "New York";
```

представляет собой альтернативный вариант синтаксиса для:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$lector = Human::Cannibal->new();
$object->feed("Zak");
$object->move("New York");
```

Косвенный вызов методов привлекателен для англоязычных программистов и хорошо знаком программирующим на C++ (где подобным образом используется new). Не поддавайтесь соблазну. Косвенный вызов обладает двумя существенными недостатками. Во-первых, он должен подчиняться тем же ненадежным правилам, что и позиция файлового манипулятора в print и printf:

```
printf STDERR "stuff here\n";
```

Эта позиция, если она заполняется, должна содержать простое слово, блок или имя скалярной переменной; скалярные выражения недопустимы. Это приводит к невероятно запутанным проблемам, как в двух следующих строках:

```
move $obj->{FIELD};          g Вероятно, ошибка
move $ary[$i];               # Вероятно, ошибка
```

Как ни странно, эти команды интерпретируются следующим образом:

```
$obj->move->{FIELD};         " Сюрприз!
$ary->move->[$i];            # Сюрприз!
```

вместо ожидаемого:

```
$obj->{FIELD}->move();       # Ничего подобного
$ary[$i]->move;              # Ничего подобного
```

Вторая проблема заключается в том, что во время компиляции Perl приходится гадать, что такое name и move — функции или методы. Обычно Perl угадывает правильно, но в случае ошибки функция будет откомпилирована как метод, и наоборот. Это может привести к появлению невероятно хитрых ошибок, которые очень трудно обнаружить. Формулировке -> эти раздражающие неоднозначности не присущи, поэтому мы рекомендуем пользоваться только ею.

Некоторые замечания по объектной терминологии

В объектно-ориентированном мире одни и те же концепции часто описываются разными словами. Если вы программировали на другом объектно-ориентированном языке, возможно, вам захочется узнать, как знакомые термины и концепции представлены в Perl.

Например, объекты часто называются *экземплярами* (instances) классов, а методы этих объектов — *методами экземпляров*. Поля данных, относящиеся к каждому объекту, часто называются *данными* экземпляров или *атрибутами объектов*, а поля данных, общие для всех членов класса, — *данньши класса, атрибутами класса* или *статическими переменными класса*.

Кроме того, термины *базовый класс* и *суперкласс* описывают одно и то же понятие (родитель или другой предок в иерархии наследования), тогда как термины *производный класс* и *субкласс* описывают противоположное отношение (непосредственный или отдаленный потомок в иерархии наследования).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Программисты на C++ привыкли использовать *статические методы, виртуальные методы* и *методы экземпляров,* но Perl поддерживает только *методы классов* и *методы объектов.* В действительности в Perl существует только общее понятие "метод". Принадлежность метода к классу или объекту определяется исключительно контекстом использования. Метод класса (со строковым аргументом) можно вызвать для объекта (с аргументом-ссылкой), но вряд ли это приведет к разумному результату.

Программисты C++ привыкли к глобальным (то есть существующим на уровне класса) конструкторам и деструкторам. В Perl они идентичны соответственно инициализирующему коду модуля и блоку `END{}`.

С позиций C++ все методы Perl являются виртуальными. По этой причине их аргументы никогда не проверяются на соответствие прототипам функции, как это можно сделать для встроенных и пользовательских функций. Прототипы проверяются компилятором во время компиляции. Функция, вызванная методом, определяется лишь во время выполнения.

Философское отступление

В своих объектно-ориентированных аспектах Perl предоставляет полную свободу выбора: возможность делать одни и те же вещи несколькими способами (приведение позволяет создать объект из данных любого типа), возможности модификации классов, написанных другими (добавление функций в их пакеты), а также полная возможность превратить отладку программы в сущий ад — если вам этого сильно захочется.

В менее гибких языках программирования обычно устанавливаются более жесткие ограничения. Многие языки с фанатичным упорством отстаивают закрытость данных, проверку типов на стадии компиляции, сложные сигнатуры функций и другие возможности. Все эти возможности отсутствуют в объектах Perl, поскольку они вообще не поддерживаются Perl. Помните об этом, если объектно-ориентированные аспекты Perl покажутся вам странными. Все странности происходят лишь от того, что вы привыкли к философии других языков. Объектно-ориентированная сторона Perl абсолютно разумна — если мыслить категориями Perl. Для любой задачи, которую нельзя решить на Perl по аналогии с Java или C++, найдется прекрасно работающее решение в идеологии Perl. Программист-параноик даже сможет обеспечить полную закрытость: в *perltoot*(1) рассказано о том, как с помощью приведения замыканий получить объекты, по степени закрытости не уступающие объектам C++ (и даже превосходящие их).

Объекты Perl не плохи; просто они другие.

Смотри также

В литературе по объектно-ориентированному программированию Perl упоминается очень редко. Изучение объектно-ориентированных аспектов языка лучше всего начать с документации Perl — особенно с учебника по объектам *perltoot*(1). За справочной информацией обращайтесь к *perlobj*(1). Вероятно, этот документ понадобится вам при чтении руководства *perlbot*(1), полного объектно-ориентированных фокусов.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

13.1. Конструирование объекта

Проблема

Необходимо предоставить пользователю возможность создания новых объектов.

Решение

Создайте конструктор. В Perl метод-конструктор не только инициализирует объект, но и предварительно выделяет память для него — как правило, с использованием анонимного хэша. Конструкторы C++, напротив, вызываются после выделения памяти. В объектно-ориентированном мире конструкторы C++ было бы правильнее назвать ***инициализаторами***.

Канонический конструктор объекта в Perl выглядит так:

```
sub new {
    my $class = shift;
    my $self  = {},
    bless($self, $class);
          $self;
}
```

Данный фрагмент эквивалентен следующей строке:

```
sub new { bless( { >,  shift ) }
```

Комментарий

Любой метод, который выделяет память для объекта и инициализирует его, фактически является конструктором. Главное, о чем следует помнить, — ссылка становится объектом лишь после того, как для нее будет вызвана функция bless. Простейший, хотя и не особенно полезный конструктор выглядит так:

```
sub new { bless({ " }
```

Давайте включим в него инициализацию объекта:

```
sub new {
    my $self = { },  #  Выделить анонимный хэш
    bless($self);
    # Инициализировать два атрибута/поля/переменных экземпляра
    $self->{START} = time();
    $self->{AGE}   = 0;
          $self;
}
```

Такой конструктор не очень полезен, поскольку в нем используется одноаргументная форма bless, которая всегда приводит объект ***в текущий пакет***. Это означает, что полезное наследование от него становится невозможным; сконструированные объекты всегда будут приводиться к классу, в котором была откомпилирована функция new. При наследовании этот класс не обязательно совпадет с тем, для которого вызывался данный метод.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Проблема решается просто: достаточно организовать в конструкторе обработку первого аргумента. Для метода класса он представляет собой имя пакета. Передайте имя класса функции bless в качестве второго аргумента:

```
sub new {
    my $classname  = shift;      tt      Какой класс мы конструируем?
    my $self       = {>;              # Выделить память
                   $classname);         Привести к нужному типу
    $self->{START}  = time();         # Инициализировать поля данных
    $self->{AGE}    = 0;
                                          вернуть
}
```

Теперь конструктор будет правильно наследоваться производными классами.

Выделение памяти и приведение можно отделить от инициализации данных экземпляра. В простых классах это не нужно, однако такое разделение упрощает наследование; см. рецепт 13.10.

```
sub new {
    my $classname  = shift;     ft      Какой класс мы конструируем?
    my $self    `   = {};             й  Выделить память                    *
    bless($self,  $classname);        й  Привести к нужному типу
    $self->_init(@_);                 й  Вызвать _init
                                      й с остальными аргументами
            $self;
>

й "Закрытый" метод для инициализации полей. Он всегда присваивает START
# текущее время, а AGE r 0.  При вызове с аргументами _init
fl интерпретирует их как пары ключ/значение и инициализирует ими объект.
sub _init {
    my $self = shift;
    $self->{START} = time();
    $self->{AGE}   = 0;
    if (@_) {
        my %extra = @_;
        @$self{keys %extra} = values %extra;
    }
}
```

Смотри также

perltoot(1) и *perlobj*(1); рецепты 13.6; 13.9—13.10.

13.2. Уничтожение объекта

Проблема

Некоторый фрагмент кода должен выполняться в случае, если надобность в объекте отпадает. Например, объект может использоваться в интерфейсе с внешним миром или содержать циклические структуры данных — в этих случаях он

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

должен "убрать за собой". При уничтожении объекта может происходить удаление временных файлов, разрыв циклических связей, корректное отсоединение от сокета или уничтожение порожденных процессов.

Решение

Создайте метод с именем DESTROY. Он будет вызываться в том случае, когда на объект не остается ни одной ссылки или при завершении программы (в зависимости от того, что произойдет раньше). Освобождать память не нужно; лишь выполните все завершающие действия, которые имеют смысл для данного класса.

```
sub DESTROY {
    my $self = shift;
    printf("$self dying at %s\n", scalar localtime),
}
```

Комментарий

У каждой истории есть начало и конец. История объекта начинается с выполнения конструктора, который явно вызывается при создании объекта. Жизненный цикл объекта завершается в *деструкторе* — методе, который неявно вызовется при уходе объекта из жизни. Весь завершающий код, относящийся к объекту, помещается в деструктор, который должен называться DESTROY.

Почему деструктору нельзя присвоить произвольное имя, как это делается для конструктора? Потому что конструктор явно вызывается по имени, а деструктор — нет. Уничтожение объекта выполняется автоматически через систему сборки мусора Perl, реализация которой в настоящее время основана на системе подсчета ссылок. Чтобы знать, какой метод должен вызываться при уничтожении объекта, Perl требует присвоить деструктору имя DESTROY. Если несколько объектов одновременно выходят из области действия, Perl не гарантирует вызова их деструкторов в определенном порядке.

Почему имя DESTROY пишется в верхнем регистре? В Perl это обозначение говорит о том, что данная функция вызывается автоматически. К числу других автоматически вызываемых функций принадлежат BEGIN, END, AUTOLOAD и все методы связанных объектов (см. рецепт 13.15) — например, STORE и FETCH.

Пользователь не должен беспокоиться о том, когда будет вызван конструктор. Просто это произойдет в нужный момент. В языках, не поддерживающих сборку мусора, программисту приходится явно вызывать деструктор для очистки памяти и сброса состояния — и надеяться на то, что он не ошибся в выборе момента. Беднягу можно только пожалеть.

Благодаря автоматизированному управлению памятью в Perl деструкторы объектов используются редко. Но даже в случаях, когда они нужны, явный вызов деструктора — вещь не только излишняя, но и попросту опасная. Деструктор будет вызван системой времени исполнения в тот момент, когда объект перестанет использоваться. В большинстве классов деструкторы не нужны, поскольку Perl сам решает основные проблемы — такие, как освобождение памяти.

Система сборки мусора не поможет лишь в одной ситуации — при наличии циклических ссылок в структуре данных:

```
$self->{WHATEVER} = $self;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В этом случае циклическую ссылку приходится удалять вручную, чтобы при работе программы не возникали утечки памяти. Такой вариант чреват ошибками, но это лучшее, что мы можем сделать. Впрочем, в рецепте 13.13 представлено элегантное решение этой проблемы. Однако вы можете быть уверены, что при завершении программы будут вызваны деструкторы всех ее объектов. При завершении работы интерпретатора выполняется тотальная сборка мусора. Даже недоступные или циклические объекты не переживут последней чистки. Следовательно, можно быть уверенным в том, что объект *когда-нибудь* будет уничтожен должны образом, даже если выход из программы никогда не происходит. Если Perl работает внутри другого приложения, вторая форма сборки мусора встречается чаще (при каждом завершении интерпретатора).

Метод DESTROY *не вызывается* при завершении программы, вызванной функцией exec.

Смотри также

perltoot(1) и *perlobj*(1); рецепты 13.10; 13.13.

13.3. Работа с данными экземпляра

Проблема

Для работы с каждым атрибутом данных объекта (иногда называемым переменной экземпляра или свойством) необходим специальный метод доступа. Как написать функцию для работы с данными экземпляра?

Решение

Напишите пару методов для чтения и присваивания соответствующего ключа в хэше объекта:

```
sub get_name {
    my $self = shift;
            $self->{NAME};
}

sub set_name {
    my $self      = shift;
    $self->{NAME} = shift;
}
```

Или воспользуйтесь одним методом, который решает ту или иную задачу в зависимости от того, был ли передан аргумент при вызове:

```
sub name {
    my $self = shift;
    if (@_) { $self->{NAME} = shift >
            $self->{NAME};
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Иногда при установке нового значения полезно вернуть старое:

```
sub age {
    my $self = shift;
                $self->{AGE};
    if (@_) { $self-><AGE} = shift }

}
# Пример одновременного чтения и записи атрибута
$obj->age( 1 + $obj->age );
```

Комментарий

Работа методов зависит от того, как вы организуете открытый интерфейс к объекту. Нормальный класс не любит, чтобы окружающие копались у него во внутренностях. Для каждого атрибута данных должен существовать метод, обеспечивающий его чтение или обновление. Если пользователь пишет фрагмент вида:

```
$him = Person->new();
$him->{NAME} = "Sylvester";
$him->{AGE}  = 23;
```

он нарушает интерфейс объекта и напрашивается на неприятности.

Для номинально закрытых атрибутов вы просто не создаете методы, позволяющие обращаться к ним.

Интерфейс на базе функций позволяет изменить внутреннее представление, не рискуя нарушить работу программ. Он позволяет выполнять любые проверки диапазона, а также выполнять необходимое форматирование или преобразование данных.

Продемонстрируем сказанное на примере улучшенной версии метода name:

```
use Carp;
sub name {
    my $self = shift;
            $self->{NAME} unless @_;
    local $_ = shift;
    croak "too many arguments" if @_;
    if ($^W) {
        /[^\s\w'-]/          && carp "funny characters in name";
        /\d/                 && carp "numbers in name";
        /\S+(\s+\S+)+/       ||                  tiword name";
        /\S/                 || carp "name is blank";
    }
    s/(\w+)/\u\L$1/g;        # Начинать с символа верхнего регистра
    $self->{NAME} = $_;
}
```

Если пользователи (или даже другие классы посредством наследования) обращаются к полю "NAME" напрямую, вы уже не сможете добавить подобный код. Настаивая на косвенном обращении ко всем атрибутам данных через функции, вы оставляете за собой свободу выбора.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Программисты, которым приходилось работать с объектами C++, привыкли к тому, что к атрибутам объекта можно обращаться из методов в виде простых переменных. Модуль Alias с CPAN обеспечивает эту и многие другие возможности — например, создание открытых методов, которые могут вызываться объектом, но недоступны для кода за его пределами.

Рассмотрим пример создания класса Person с применением модуля Alias. Обновление "магических" переменных экземпляра автоматически обновляет поля данных в хэше. Удобно, правда?

```
package Person;

# То же, что и раньше...
sub new {
     my $that  = shift; •
     my $class = ref($that) ||  $that;
     my $self = {
           NAME  => undef,
           AGE   => undef,
           PEERS  =>  [],

    bless($self, $class);
           $self;
}

use Alias qw(attr);
use vars  qw($NAME $A6E  $PEERS);

sub name {
    my $self =  attr shift;
    if (@_) { $NAME = shift; }
              $NAME;
}

sub age {
    my $self = attr  shift;
    if (@_) { $AGE = shift; }
              $A6E;
}

sub peers {
    my $self =  attr shift;
    if (@_) { @PEERS = @_; }
              @PEERS;
}

sub exclaim {
    my $self = attr  shift;
           sprintf "Hi, I'm %s, age %d, working with %s",
            $NAME, $AGE, join(", ", @PEERS);
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub happy_birthday {
    my $self = attr  shift;
            ++$AGE;
}
```

Директива use vars понадобилась из-за того, что Alias играет с пакетными глобальными переменными, имена которых совпадают с именами полей. Чтобы использовать глобальные переменные при действующей директиве use strict, необходимо заранее объявить их. Эти переменные локализуются в блоке, содержащем вызов attr(), словно они объявлены с ключевым словом local. Таким образом, они остаются глобальными пакетными переменными с временными значениями.

Смотри также

perltoot(1), *perlobj*(1) *ttperlbot(1);* документация по модулю Alias с CPAN; рецепты 13.11-13.12.

13.4. Управление данными класса

Проблема

Вам нужен метод, который вызывается для класса в целом, а не для отдельного объекта. Например, он может обрабатывать глобальный атрибут данных, общий для всех экземпляров класса.

Решение

Первым аргументом метода класса является не ссылка, как в методах объектов, а строка, содержащая имя класса. Методы классов работают с данными пакета, а не данными объекта, как показывает приведенный ниже метод population:

```
package Person;

$Body_Count = 0;

sub population {         $Body_Count  }

sub new {                                    8 Конструктор
    $Body_Count++;
            bless({}, shift);
}

sub DESTROY {  --$BodyCount  }                Я Деструктор

# Позднее пользователь может написать:
package main;

for (1.,10) {  push @people,  Person->new }
printf            %d people alive.\n",  Person->population();

                      alive.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Обычно каждый объект обладает определенным состоянием, полная информация о котором хранится в самом объекте. Значение атрибута данных одного объекта никак не связано со значением этого атрибута в другом экземпляре того же класса. Например, присваивание атрибуту gender объекта her никак не влияет на атрибут gender объекта him, поскольку это разные объекты с разным состоянием:

```
$him = Person->new();
$him->gender("male");

$her = Person->new();
$her->gender("female");
```

Представьте атрибут, общий для всего класса — изменение атрибута для одного экземпляра приводит к его изменению для остальных экземпляров. Подобно тому, как имена глобальных переменных часто записываются с большой буквы, некоторые программисты предпочитают записывать имя символами верхнего регистра, если метод работает с данными класса, а не с данными экземпляра. Рассмотрим пример использования метода класса с именем Max_Bounds:

```
FixedArray->Max_Bounds(100);      # Устанавливается для всего класса
$alpha = FixedArray->new();
printf "Bound on alpha is %d\n", $alpha->Max_Bounds();
100
$beta = FixedArray->new();
$beta->Max_Bounds(50);            # Также устанавливается для всего класса
printf "Bound on alpha is %d\n", $alpha->Max_Bounds();
50
```

Реализация выглядит просто:

```
package FixedArray;
$Bounds = 7;   # default
sub new { bless( {}, shift ) }
sub Max_Bounds {
    my $proto  = shift;
    $Bounds    = shift if @_;          # Разрешить обновления
        $Bounds;
}
```

Чтобы фактически сделать атрибут доступным только для чтения, просто удалите команды обновления:

```
sub Max_Bounds { $Bounds }
```

Настоящий параноик сделает $Bounds лексической переменной, которая ограничена областью действия файла, содержащего класс. В этом случае никто не сможет обратиться к данным класса через $FixedArray::Bounds. Работать с данными придется через интерфейсные методы.

Следующий совет поможет вам строить расширяемые классы: храните данные объекта в пространстве имен объекта (в хэше), а данные класса — в пространстве имен класса (пакетные переменные или лексические переменные с файловой об-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ластью действия). Только методы класса могут напрямую обращаться к атрибутам класса. Методы объектов работают только с данными объектов. Если методу объекта потребуется обратиться к данным класса, его конструктор должен сохранить ссылку на эти данные в объекте. Пример:

```
sub new {
    my $class = shift;
    my $self = bless({}, $class);
                                \$Bounds;
            $self;
}
```

Смотри также

perltoot(1), *perlobj*(1) и *perlbot*(1); рецепт 13.3; пример использования метода places в разделе "Пример. Перегруженный класс FixNum» в рецепте 13.14.

13.5. Использование класса как структуры

Проблема

Вы привыкли работать со структурированными типами данных — более сложными, чем массивы и хэши Perl (например, структуры C и записи Pascal). Вы слышали о том, что классы Perl не уступают им по возможностям, но не хотите изучать объектно-ориентированное программирование.

Решение

Воспользуйтесь стандартным модулем Class::Struct для объявления C-подобных структур:

```
use Class::Struct;            # Загрузить модуль построения структур

struct Person => {            # Создать определение класса "Person"
    name  => '$',             #    Имя - скаляр
    age   => '$',             #    Возраст - тоже скаляр
    peers => '@',             #    Но сведения о друзьях - массив (ссылка)
};

my $p = Person->new();        # Выделить память для пустой структуры Person

$p->name("Jason Smythe");                  # Задать имя
$p->age(13);                               # Задать возраст
$p->peers( ["Wilbur", "Ralph",             Задать друзей

# Или так:
@{$p->peers> = ("Wilbur", "Ralph", "Fred");

# Выбрать различные значения, включая нулевого друга
printf "At age %d, %s's first friend is %s.\n",
    $p->age, $p->name, $p->peers(0);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Функция `Class::Struct::struct` автоматически создает классы, дублирующие структуры. Она создает класс с именем, передаваемым в первом аргументе, и генерирует для него конструктор new и методы доступа к полям.

В определении структуры ключи соответствуют именам полей, а значения — типам данных. Существуют три основных значения типа: '$' для скаляров, '@' для массивов и '%' для хэшей. Каждый метод доступа может вызываться без аргументов (выборка текущего значения) или с аргументами (присваивание значения). Для полей с типом "массив" или "хэш" вызов метода без аргументов возвращает ссылку на весь массив или хэш, вызов с одним аргументом получает значение по указанному индексу[1], а вызов с двумя аргументами задает значение для указанного индекса.

Однако тип может быть именем другой структуры (или любого класса), имеющей конструктор new.

```
use Class::Struct;

struct Person => {name => '$',       age  => '$'};
struct Family => {head => 'Person',            '$', members => '@'};

$folks  = Family->new();
$dad    = $folks->head;
$dad->name("John");
$dad->age(34);

printf("%s's age is %d\n", $folks->head->name, $folks->head->age);
```

Чтобы организовать дополнительную проверку параметров, напишите собственные версии методов доступа, переопределяющие версии по умолчанию. Предположим, вы хотите убедиться, что значение возраста состоит из одних цифр и не превышает нормальной продолжительности человеческой жизни. Функция может выглядеть так:

```
sub Person::age {
    use Carp;
    my ($self, $age) = @_;
    if   (@_  > 2) {  confess "too many  arguments" }
    elsif (@_                  $struct-><'age'}      }
    elsif (@_ == 2) {
        carp "age '$age' isn't numeric"   if $age !^ /^\d+/;
        carp "age '$age' is
        $self->{'age'} = $age;
    }
}
```

Если предупреждения должны выводиться лишь при наличии флага -w в командной строке, проверьте переменную $^W:

[1] Если только оно не является ссылкой; в этом случае используется субъектная структура данных с проверкой типа.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
if ($^W) {
    carp "age  '$age' isn't numeric"   if $age !" /^\d+/;
    carp "age '$age' is                  $age > 150;
}
```

Если при наличии флага **-w** выводится предупреждение, а без него функция должна инициировать исключение, воспользуйтесь следующим фрагментом. Пусть стрелка вас не смущает; это косвенный вызов функции, а **не** вызов метода.

```
my $gripe = $^W ? \&carp : \&croak;
$gripe->("age '$age' isn't numeric")    if $age Г /^\d+/;
$gripe->("age '$age' is                          150;
```

Как и большинство классов, наш класс реализован **в** виде хэша, Это упрощает отладку и сопровождение кода. Представьте себе возможность вывода структуры в отладчике. Однако модуль **Class::Struct** также поддерживает реализацию на базе массива, для этого достаточно перечислить поля в квадратных скобках вместо фигурных:

```
struct Family => [head => 'Person',                    members => '@'];
```

Существуют эмпирические данные, свидетельствующие о том, что выбор массива вместо хэша снижает расходы памяти от 10 до 50 % и примерно на 33 % ускоряет доступ. За это приходится расплачиваться менее содержательной отладочной информацией и трудностями при написании переопределяющих функций (таких, как приведенная выше функция **Person::age)**. Обычно представление объекта в виде массива усложняет наследование. В данном случае это не так, поскольку С-подобные структуры обеспечивают намного более понятную реализацию агрегирования.

Директива use **fields** в Perl версии 5.005 повышает скорость за счет дополнительных затрат памяти и обеспечивает проверку имен полей на стадии компиляции.

Если все поля принадлежат к одному типу, то запись вида:

```
' struc't Card => {
    name    => '$',
    color   => '$',
    cost    => '$',
    type    => '$',
               '$',
    text    => '$',
};
```

упрощается с помощью функции **map**:

```
struct Card => map { $_ => '$' > qw(name color cost type        text);
```

А если вы программируете на С **и** предпочитаете указывать тип поля перед его именем, а не наоборот, просто измените **их** порядок:

```
struct hostent => { reverse qw{
    $ name
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    @aliases
    $ addrtype
    $ length
    @ addr_list
}};
```

Вы даже можете создавать синонимы в стиле `#define` (впрочем, такая возможность выглядит сомнительно), позволяющие обращаться к одному полю по нескольким именам. В C можно написать:

```
#define h_type h_addrtype
"define h_addr h_addr_list[0]
```

В Perl можно попробовать следующий вариант:

```
" Сделать (hostent object)->type()
# эквивалентным (hostent object)->addrtype()
*hostent::type = \&hostent::addrtype;

# Сделать (hostenv object)->addr()
И эквивалентным (hostenv object)->addr_llst(0)
sub hostent::addr { shift->addr_list(0,@_) }
```

Как видите, вы можете добавлять методы в класс (или функции в пакет) простым объявлением функции в нужном пространстве имен. Для этого необязательно находиться в файле с определением класса, создавать субкласс или делать что-то хитроумное и запутанное. Однако вариант с субклассированием все же смотрится намного лучше:

```
package Extra::hostent;
use Net::hostent;
@ISA = qw(hostent);
sub addr { shift->addr_list(0,@_) }
1;
```

Это решение взято из стандартного класса **Net::hostent.** Обратитесь к исходным текстам этого модуля, это весьма вдохновляющее чтение. Впрочем, авторы не несут ответственности за возможные последствия вашего вдохновения.

> **Смотри также**

perltoot(1), *perlobj*(1) и *perlbot*(1); документация по стандартному модулю **Class::Struct**; исходный текст стандартного модуля Net::hostent; документация по модулю Alias с CPAN; рецепт 13.3.

13.6. Клонирование объектов

Проблема

Вы хотите написать **конструктор,** который может вызываться для существующего объекта.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Начните свой конструктор примерно так:

```
my $proto  = shift;
my $class  =              || $proto;
                             $proto;
```

Переменная $class содержит класс, к которому выполняется приведение, а переменная либо равна false, либо ссылается на клонируемый объект.

Комментарий

Иногда требуется создать объект, тип которого совпадает с типом другого, существующего объекта. Вариант:

```
$ob1  =  SomeClass->new();
# Далее
$ob2  =      $ob1)->new();
```

выглядит не очень понятно. Вместо этого хотелось **бы** иметь конструктор, который может вызываться для класса или существующего объекта. **В** качестве метода класса он возвращает новый объект, инициализированный по умолчанию. **В** качестве метода экземпляра он возвращает новый объект, инициализированный данными объекта, для которого он **был** вызван:

```
$ob1 = Widget->new();
$ob2 = $ob1->new();
```

Следующая версия new учитывает эти соображения:

```
sub new {
    my $proto  = shift;
    my $class  =              || $proto;


    my $self;
    # Проверить, переопределяется ли new из @ISA
    if (@ISA && $proto->SUPER::can('new') {
        $self = $proto->SUPER::new(@_);
    } else {
        $self = {};
        bless ($self,  $proto);
    }
    bless($self, $class);

    $self->{PARENT}   =
    $self->{START}    = time();    # Инициализировать поля данных
    $self->{AGE}      = 0;
            $self;
}
```

Инициализация не сводится к простому копированию данных из объекта-прототипа. Если вы пишете класс связанного списка или бинарного дерева, при вы-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

зове в качестве метода экземпляра ваш конструктор может вернуть новый объект, включенный в дерево или список.

Смотри также
perlobj(1);рецепты **13.1; 13.9; 13,13.**

13.7. Косвенный вызов методов

Проблема

Требуется вызвать метод по имени, которое станет известно лишь во время выполнения программы.

Решение

Сохраните имя метода в строковом виде в скалярной переменной и укажите имя переменной там, где обычно указывается имя метода — справа от оператора **->:**

```
$methname = "flicker";
$obj->$methname(10);          # Вызывает $ob->flicker(10);

# Три метода объекта вызываются по именам
        $m                  stop) ) {
    $ob]->$m();
}
```

Комментарий

Имя метода не всегда известно на стадии компиляции. Как известно, получить адрес метода нельзя, но можно сохранить его имя. Если имя хранится в скалярной переменной `$meth`, то для объекта `$crystal` этотметод вызывается так: `$crystal->$meth()`.

```
@methods = qw(name rank serno);
%his_info = map { $_ => $ob->$_() } @methods;

# Эквивалентно:

%his_info = (
    'name'  => $ob->name(),
    'rank'  => $ob->rank(),
    'serno' => $ob->serno(),
);
```

Если вам никак не обойтись без получения адреса метода, попробуйте переосмыслить свой алгоритм. Например, вместо неправильной записи `\$ob->method()`, при которой \ применяется к возвращаемому значению или значениям метода, поступите следующим образом:

```
$ob->method(@_)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Когда придет время косвенного вызова этого метода, напишите:

```
"fred");
```

и это даст правильный вызов метода:

```
$obj->method(10, "fred");
```

Такое решение работает даже в том случае, если $ob находится вне области действия и потому является предпочтительным.

Ссылку на код, возвращаемую методом `can()` класса UNIVERSAL, вероятно, не следует использовать для косвенного вызова методов. Нельзя быть уверенным в том, что она будет соответствовать правильному методу для объекта произвольного класса.

Например, следующий фрагмент крайне сомнителен:

```
$obj->can('method_name')->($obj_target, @arguments)
     $obj_target->isa(     $obj
```

Ссылка, возвращаемая can, может и не соответствовать правильному методу для $obj2. Вероятно, разумнее ограничиться проверкой метода `can()` в логическом условии.

Смотри также
perlobj(1); рецепт **11.8.**

13.8. Определение принадлежности субкласса

Проблема

Требуется узнать, является ли объект экземпляром некоторого класса **или** одного **из** его субклассов. Например, надо выяснить, можно ли вызвать для объекта некоторый метод.

Решение

Воспользуйтесь методами специального класса UNIVERSAL:

```
$obj->isa("HTTP::Message");                  # Как метод объекта
HTTP::Response->isa("HTTP::Message");        # Как метод класса

if ($obj->can("method_name")) { .... }       # Проверка метода
```

Комментарий

Для нас было бы очень удобно, чтобы все объекты в конечном счете происходили от общего базового класса. Тогда их можно было бы наделить общими методами, не дополняя по отдельности каждый массив `@ISA`. В действительности такая возможность существует. Хотя вы этого не видите, но Perl считает,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

что в конце @ISA находится один дополнительный элемент — пакет с именем UNIVERSAL.

В версии 5.003 класс UNIVERSAL не содержал ни одного стандартного метода, но вы могли занести в него все, что считали нужным. Однако в версии 5.004 UNIVERSAL уже содержит несколько методов. Они встроены непосредственно в двоичный файл Perl и потому на их загрузку не расходуется дополнительное время. К числу стандартных методов относятся isa, can и VERSION. Метод isa сообщает, "является ли" (is a) объект или класс чем-то другим, избавляя вас от необходимости самостоятельно просматривать иерархию:

```
$has_io = $fd->isa("IO::Handle");
$itza_handle = IO::Socket->isa("IO::Handle");
```

Также существует мнение, что обычно лучше попробовать вызвать метод. Считается, что явные проверки типов вроде показанной выше слишком ограничивают свободу действий.

Метод can вызывается для объекта или класса и сообщает, соответствует ли его строковый аргумент допустимому имени метода для данного класса. Он возвращает ссылку на функцию данного метода:

```
$his_print_method = $obj->can('as_string');
```

Наконец, метод VERSION проверяет, содержит ли класс (или класс объекта) пакетную глобальную переменную $VERSION с достаточно высоким значением:

```
Some_Module->VERSION(3.0);
$his_vers = $obj->VERSION();
```

Тем не менее нам обычно не приходится вызывать VERSION самим. Вспомните: имена функций, записанные в верхнем регистре, означают, что функция вызывается Perl автоматически. В нашем случае это происходит, когда в программе встречается строка вида:

```
use Some_Module 3.0;
```

Если вам захочется включить проверку версии в класс Person, описанный выше, добавьте в файл Person.pm следующий фрагмент:

```
use vars qw($VERSION);
$VERSION = '1.01';
```

Затем в пользовательской программе ставится команда use Person 1.01; — это позволяет проверить версию и убедиться в том, что она равна указанной или превышает ее. Помните, что версия не обязана точно совпадать с указанной, а должна быть *не меньше* ее. Впрочем, в настоящее время параллельная установка нескольких версий одного модуля не поддерживается.

Смотри также

Документация по стандартному модулю UNIVERSAL. Ключевое слово use описано в perlfunc(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

13.9. Создание класса с поддержкой наследования

Проблема

Вы не уверены в том, правильно ли вы спроектировали свой класс и может ли он использоваться в наследовании.

Решение

Воспользуйтесь "проверкой пустого субкласса".

Комментарий

Допустим, вы реализовали класс Person *с* конструктором new и методами age и name. Тривиальная реализация выглядит так:

```
package Person;
sub new {
    my $class = shift;
    my $self = { };
                $self, $class;
}
sub name {
    my $self = shift;
    $self->{NAME} = shift if @_;
          $self->{NAME};
}
sub age {
    my $self = shift;
    $self->{AGE} = shift if @_;

}
```

Пример использования класса может выглядеть так:

```
use Person;
my $dude = Person->new();
$dude->name("Jason");
$dude->age(23);
printf "%s is age %d.\n", $dude->name, $dude->age;
```

Теперь рассмотрим другой класс с именем Employee:

```
package Employee;
use Person;
@ISA = ("Person");
1;
```

Ничего особенно интересного. Класс всего лишь загружает класс Person и заявляет, что все необходимые методы Employee наследует от Person. Поскольку Employee не имеет собственных методов, он получит от Person *все* методы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Мы хотим, чтобы поведение класса Person полностью воспроизводилось в Employee.

Создание подобных пустых классов называется "проверкой пустого субкласса"; иначе говоря, мы создаем производный класс, который не делает ничего, кроме наследования от базового. Если базовый класс спроектирован нормально, то производный класс в точности воспроизведет его поведение. Это означает, что при простой замене имени класса все остальное будет работать:

```
use Employee;
my $empl = Employee->new(),
$empl->name("Jason");
$empl->age(23);
printf "%s is age %d \n", $empl->name, $empl->age;
```

Под "нормальным проектированием" имеется в виду использование только двухаргументной формы bless, отказ от прямого доступа к данным класса и отсутствие экспортирования. В определенной выше функции Person::new() мы проявили необходимую осторожность: в конструкторе используются некоторые пакетные данные, но ссылка на них хранится в самом объекте. Другие методы обращаются к пакетным данным через эту ссылку, поэтому проблем быть не должно.

Но почему мы сказали *"функции Person::new()»* — разве это не метод? Дело в том, что метод представляет собой функцию, первый аргумент которой определяет имя класса (пакет) или объект (приведенную ссылку). Person::new — это функция, которая в конечном счете вызывается методами Person->new и Employee->new. Хотя вызов метода очень похож на вызов функции, они все же отличаются. Если вы начнете путать функции с методами, то очень скоро у вас не останется ничего, кроме неработающих программ. Во-первых, функции отличаются от методов фактическими конвенциями вызова — метод вызывается с дополнительным аргументом. Во-вторых, вызовы функций не поддерживают наследования, а методы — поддерживают.

Если вы привыкнете к вызовам вида:

Вызов метода	Вызов функции
Person->new()	Person::new("Person")
Employee->new()	Person::new("Employee")

```
$him = Person::new();                    # НЕВЕРНО
```

в программе возникнет нетривиальная проблема, поскольку функция не получит ожидаемого аргумента "Person" и несможет привести его к переданному классу. Еще хуже, если вам захочется вызвать функцию Employee::new(). Такой функции не существует! Это всего лишь вызов унаследованного метода.

Мораль: не вызывайте функции там, где нужно вызывать методы.

Смотри также

perltoot(1), *perlobj*(1) и *perlbot*(1); рецепты **13.1; 13.10.**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

13.10. Вызов переопределенных методов

Проблема

Конструктор переопределяет конструктор суперкласса. Вы хотите вызвать конструктор суперкласса из своего конструктора.

Решение

Используйте специальный класс, SUPER:

```
sub meth {
    my $self = shift;
    $self->SUPER::meth();
}
```

Комментарий

В таких языках, как C++, где конструкторы не выделяют память, а ограничиваются инициализацией объекта, конструкторы базовых классов вызываются автоматически. В таких языках, как Java и Perl, приходится вызывать их самостоятельно.

Для вызова методов конкретного класса используется формулировка $self->SUPER::meth(). Она представляет собой расширение обычной записи с началом поиска в определенном базовом классе и допустима только в переопределенных методах. Сравните несколько вариантов:

```
$self->meth();              # Вызвать первый найденный meth
$self->Where::meth();       # Начать поиск с пакета "Where"
$self->SUPER::meth();   tf  Вызвать переопределенную версию
```

Вероятно, простым пользователям класса следует ограничиться первым вариантом. Второй вариант возможен, но не рекомендуется. Последний вариант может вызываться только в переопределенном методе.

Переопределяющий конструктор должен вызвать конструктор своего класса SUPER, в котором выполняется выделение памяти и приведение объекта, и ограничиться инициализацией полей данных. В данном случае код выделения памяти желательно отделять от кода инициализации объекта. Пусть имя начинается с символа подчеркивания — условного обозначения номинально закрытого метода, аналога таблички "Руками не трогать".

```
sub new {
    my $classname = shift;          # Какой класс мы конструируем?
    my $self      = $classname->SUPER::new(@_);
    $self->_init(@_);
          $self;                    # Вернуть
}

sub _init {
    my $self = shift;
    $self->{START}   = time();   # Инициализировать поля данных
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $self->{AGE}     = 0;
    $self-><EXTRA}   = { @_ };   # Прочее
>
```

И SUPER::new и _init вызываются со всеми остальными аргументами, что позволяет передавать другие инициализаторы полей:

```
$obj   Widget->new(
```

∴ Стоит ли сохранять пользовательские параметры **в** отдельном **хэше** — решайте сами.

Обратите внимание: SUPER работает только для первого переопределенного метода. Если в массиве @ISA перечислено несколько классов, будет обработан только первый. Ручной перебор @ISA возможен, но, вероятно, не оправдывает затраченных усилий.

```
my $self = bless {},  $class;
for my $class (@ISA) {
    my $meth = $class . "::_init";
    $self->$meth(@_) if $class->can("_init");
}
```

В этом ненадежном фрагменте предполагается, что все суперклассы инициализируют свои объекты не в конструкторе, а в _init. Кроме того, предполагается, что объект реализуется через ссылку на хэш.

Смотри также

Класс SUPER рассматривается *perltoot(1)* и *perlobj*(1).

13.11. Генерация методов доступа с помощью AUTOLOAD

Проблема

Для работы с полями данных объекта нужны методы доступа, а вам не хочется писать повторяющийся код.

Решение

Воспользуйтесь механизмом AUTOLOAD для автоматического построения методов доступа — это позволит обойтись без самостоятельного написания методов при добавлении новых полей данных.

Комментарий

Механизм AUTOLOAD перехватывает вызовы неопределенных методов. Чтобы ограничиться обращениями к полям данных, мы сохраним список допустимых полей в хэше, Метод AUTOLOAD будет проверять, присутствует ли в хэше запрашиваемое поле.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
package Person;
use strict,
use Carp;
use vars qw($AUTOLOAD %ok_field);

# Проверка четырех атрибутов
for my $attr ( qw(name age peers ) { $ok_field{$attr}++; }

sub AUTOLOAD {
    my $self = shift;
    my $attr = $AUTOLOAD;
    $attr =~ s/.*::/;
            unless $attr =^ /[^A-Z]/;  # Пропустить DESTROY и другие
                                       # методы, имена которых
                                       # записаны в верхнем регистре
    croak "invalid attribute method: ->$attr()" unless $ok_field{$attr};
    $self->{uc $attr} = shift if @_;
            $self->{uc $attr};
}
sub new {
    my $proto  = shift;
    my $class  =            || $proto;
               =            && $proto;
    my $self = {};
    bless($self, $class);

            $self;
}
1;
```

Класс содержит конструктор new и четыре метода атрибутов: name, age, peers Модуль используется следующим образом:

```
use Person;
my ($dad, $kid);
$dad = Person->new;
$dad->name("Jason");
$dad->age(23);
$kid = $dad->new;
$kid->name("Rachel");
$kid->age(2);
printf "Kid's          %s\n", $kid->parent->name;
Jason
```

В иерархиях наследования это решение вызывает некоторые затруднения. Предположим, вам понадобился класс Employee, который содержит все атрибуты данных класса Person и еще два атрибута (например, salary и boss). Класс Employee не может определять методы своих атрибутов с помощью унаследованного варианта Person::AUTOLOAD — следовательно, каждому классу нужна собственная функция AUTOLOAD. Она проверяет атрибуты данного класса, но вместо вызова croak приотсутствии атрибута вызывает переопределенную версию суперкласса.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

С учетом этого AUTOLOAD может выглядеть так:

```
sub AUTOLOAD {
    my $self = shift;
    my $attr = $AUTOLOAD;
    $attr =~ s/.*::/;
            $attr eq 'DESTROY';

    if ($ok_field{$attr}) {
        $self->{uc $attr} = shift if @_;
            $self->{uc $attr};
    } else {
        my $superior = "SUPER::$attr";
        $self->$superior(@_);
    }
}
```

Если атрибут отсутствует в списке, мы передаем его суперклассу, надеясь, что он справится с его обработкой. Однако такой вариант AUTOLOAD наследовать нельзя; каждый класс должен иметь собственную версию, поскольку работа с данными осуществляется напрямую, а не через объект.

Еще худшая ситуация возникает, если класс A наследует от классов B и C, каждый из которых определяет собственную версию AUTOLOAD — в этом случае при вызове неопределенного метода A будет вызвана функция AUTOLOAO лишь одного из двух родительских классов.

С этими ограничениями можно было **бы** справиться, но всевозможные заплатки, исправления и обходные пути вскоре начинают громоздиться друг на друге. Для сложных ситуаций существуют более удачные решения.

Смотри также

Рецепты **10.15; 13.12.** Пример использования AUTOLOAD приведен в *perltoot*(1).

13.12. Решение проблемы наследования данных

Проблема

Вы хотите унаследовать от существующего класса и дополнить его несколькими новыми методами, но не знаете, какие поля данных используются родительским классом. Как безопасно дополнить **хэш** объекта новым пространством имен и не повредить данные предков?

Решение

Снабдите каждое имя поля **префиксом,** состоящим из имени класса и разделителя, — например, одного или двух подчеркиваний.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

В недрах стандартной объектно-ориентированной стратегии Perl спрятана одна неприятная проблема: знание точного представления класса нарушает иллюзию абстракции. Субкласс должен находиться в чрезвычайно близких отношениях со своими базовыми классами.

Давайте сделаем вид, что все мы входим в одну счастливую объектно-ориентированную семью и объекты всегда реализуются с помощью хэшей — мы попросту игнорируем классы, в чьих представлениях используются массивы, и наследуем лишь от классов на основе модели хэша (как показано в *perlbot*(1), эта проблема решается с помощью агрегирования и делегирования). Но даже с таким предположением наследующий класс не может с абсолютной уверенностью работать с ключами хэша. Даже если мы согласимся ограничиваться методами доступа для работы с атрибутами, значения которых задавались не нами, как узнать, что устанавливаемый нами ключ не используется родительским классом? Представьте себе, что в вашем классе используется поле `count`, но поле с таким же именем встречается в одном из пра-пра-правнуков. Имя `_count` (подчеркивание обозначает номинальную закрытость) не поможет, поскольку потомки могут сделать то же самое.

Одно из возможных решений — использовать для атрибутов префиксы, совпадающие с именем пакета. Следовательно, если вы хотите создать поле age в классе Employee, для обеспечения безопасности можно воспользоваться `Employee_age`. Метод доступа может выглядеть так:

```
sub Employee::age {
    my $self = shift;
    $self->{Employee_age} = shift if @_;
          $self->{Employee_age};
}
```

Модуль Class::Spirit, описанный в рецепте **13.5**, предоставляет еще более радикальное решение. Представьте себе один файл:

```
package Person;
use Class::Attributes;    * Объясняется ниже
        qw(name
```

и другой файл:

```
package Employee;
@ISA = qw(Person);
use Class::Attributes;
mkattr qw(salary age boss);
```

Вы обратили внимание на общий атрибут age? Если эти атрибуты должны быть логически раздельными, то мы не сможем использовать `$self->{age}` даже для текущего объекта внутри модуля! Проблема решается следующей реализацией функции `Class::Attributes::mkattr`:

```
package Class::Attributes;
use strict;
use Carp;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Exporter ();
use vars qw(@ISA @EXPORT);
@ISA = qw(Exporter);
@EXPORT = qw(mkattr);
sub mkattr {
    my $hispack = caller();
    for my $attr (@_) {
        my($field, $method);
        $method = "${hispack}::$attr";
        ($field = $method) =" s/:/_/g;
        no strict 'refs';
        *$method = sub {
            my $self = shift;
            confess "too many arguments" if @_ > 1;
            $self->{$field} = shift if @_;
                $self->{$field};
        };
    }
}
1;
```

В этом случае $self->{Person_age} и $self->{Employee_age} остаются раздельными. Единственная странность заключается в том, что $obj->age даст **лишь** первый из двух атрибутов. **В** принципе атрибуты можно было бы различать с помощью формулировок $obj ->Person::age и $obj ->Employee::age, но грамотно написанный код Perl не должен ссылаться на конкретный пакет с помощью ::, за исключением крайних случаев. Если это оказывается неизбежным, вероятно, ваша библиотека спроектирована не лучшим образом.

Если вам не нравится подобная запись, то внутри класса Person достаточно использовать age($self), и вывсегда получите age класса Person, тогда как в классе Employee age($self) дает версию age класса Employee. Это объясняется тем, что мы вызываем функцию, а не метода.

Смотри также

Документация по директивам use fieldsи use base для Perlверсии **5.005;** рецепт **10.14.**

13.13. Использование циклических структур данных

Проблема

Имеется структура данных, построенная на циклических ссылках. Система сборки мусора Perl, использующая подсчет ссылок, не заметит, когда данная структура перестает использоваться. Вы хотите предотвратить утечки памяти в программе.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Создайте не-циклический объект-контейнер, содержащий указатель на структуру данных с циклическими ссылками. Определите для объекта-контейнера метод DESTROY, который вручную уничтожает циклические ссылки.

Комментарий

Многие интересные структуры данных содержат ссылки на самих себя. Например, это может происходить в простейшем коде:

```
$node->{NEXT} = $node;
```

Как только в вашей программе встречается такая команда, возникает цикличность, которая скрывает структуру данных от системы сборки мусора Perl с подсчетом ссылок. В итоге деструкторы будут вызваны при выходе из программы, но иногда ждать долго не хочется.

Связанный список также обладает циклической структурой: каждый узел содержит указатель на следующий узел, указатель на предыдущий узел и значение текущего узла. Если реализовать его на Perl с применением ссылок, появится циклический набор ссылок, которые также не будут автоматически уничтожаться с исчезновением внешних ссылок на узлы.

Проблема не решается и созданием узлов, представляющих собой экземпляры специального класса **Ring.** На самом деле мы хотим, чтобы данная структура уничтожалась Perl по общим правилам — а это произойдет в том случае, если объект реализуется в виде структуры, содержащей ссылку на цикл. В следующем примере ссылка хранится в поле "DUMMY":

```
package Ring;

# Вернуть пустую циклическую структуру
sub new {
    my $class = shift;
    my $node  = { };
    $node->{NEXT} = $node->{PREV} = $node;
    my $self  = {  DUMMY => $node, COUNT => 0 >;
    bless $self, $class;
    return $self;
}
```

Цикличностью обладают узлы кольца, но не сам возвращаемый объект-кольцо. Следовательно, следующий фрагмент не вызовет утечек памяти:

```
use Ring;

$COUNT = 1000;
for (1 .. 20) {
    my $r = Ring->new();
    for ($i = O; $i < $COUNT; $i++) { $r->insert($i) >
}
```

Даже если мы создадим двадцать колец по тысяче узлов, то перед созданием нового кольца старое будет уничтожено. Пользователю класса не придется бес-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

покоиться об освобождении памяти в большей степени, чем дл я простых строк. Иначе говоря, все происходит автоматически, как и должно происходить.

Однако при реализации класса необходимо написать деструктор, который вручную уничтожает узлы:

```
# При уничтожении Ring уничтожить содержащуюся в нем кольцевую структуру
sub DESTROY {
    my $ring = shift;
    my $node;
    for ( $node  =  $ring->{DUMMY}->{NEXT};
          $node !=  $ring->{DUMMY};
          $node  =  $node->$NEXT}  )
    {
            $ring->delete_node($node);
    }
    $node->{PREV> = $node->{NEXT} = undef;
}

8 Удалить узел из циклической структуры
sub delete_node {
    my ($ring, $node) = @_;
    $node->{PREV}->{NEXT} = $node->{NEXT};
    $node->{NEXT}->{PREV} = $node->{PREV};
    --$ring->{COUNT};
}
```

Ниже приведено еще несколько методов, которые следовало бы включить в класс. Обратите внимание на то, что вся реальная работа выполняется с помощью циклических ссылок, скрытых внутри объекта:

```
# $node = $ring->search( $value ) : найти $value в структуре $ring
sub search {
    my ($ring, $value) = @_;
    my $node = $ring->{DUMMY}->{NEXT};
    while ($node != $ring->{DUMMY} && $node->{VALUE} != $value) {
          $node = $node->{NEXT};
    }
           $node;
}

# $ring->insert( $value ) : вставить $value в структуру $ring
sub insert_value {
    my ($ring, $value) = @_;
    my $node = { VALUE => $value };
    $node->{NEXT> = $ring->{DUMMY}->{NEXT};
    $ring->{DUMMY}->{NEXT}->{PREV} = $node;
    $ring->{DUMMY}->{NEXT} = $node;
    $node->{PREV} = $ring->{DUMMY};
    ++$ring->{COUNT};
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# $ring->delete_value( $value ) : удалить узел по значению
sub delete_value {
    my ($ring, $value) = @_;
    my $node = $ring->search($value);
              $node == $ring->{DUMMY};
    $ring->delete_node($node);
}

1;
```

Смотри также

Раздел "Garbage Collection» *perlobj*(1).

13.14. Перегрузка операторов

Проблема

Вы хотите использовать знакомые операторы (например, == или +) с объектами написанного вами класса или определить интерполированное значение для вывода объектов.

Решение

Воспользуйтесь директивой use overload. Нижеприведены два самых распространенных и часто перегружаемых оператора:

```
use overload ('<=>' => \&threeway_compare);
sub threeway_compare {
    my ($s1, $s2) = @_;
    uc($s1->{NAME}) cmp uc($s2->{NAME});
}

use overload (       => \&stringify );
sub stringify {
    my $self = shift;
            sprintf "%s (%05d)",                          "
             ucfirst(lc($self->{NAME})),
             $self->{IDNUM};
}
```

Комментарий

При работе со встроенными типами используются некоторые операторы (например, оператор + выполняет сложение, а . — конкатенацию строк). Директива use overload позволяет перегрузить эти операторы так, чтобы для ваших собственных объектов они делали что-то особенное.

Директиве передается список пар "оператор/функция":

```
package TimeNumber;
use overload '+' => \&my_plus,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            '-'  => \&my_minus,
            '*'  => \&my_star,
            '/'  => \&my_slash;
```

Теперь эти операторы можно использовать с объектами класса TimeNumber, и при этом будут вызываться указанные функции. Функции могут делать все, что вам захочется.

Приведем простой пример перегрузки + для работы с объектом, содержащим количество часов, мйнут и секунд. Предполагается, что оба операнда принадлежат к классу, имеющему метод new, которыйможет вызываться в качестве метода объекта, и что структура состоит из перечисленных ниже имен:

```
sub my_plus {
    my($left, $right) = @_;
    my $answer = $left->new();
    $answer->{SECONOS} = $left->{SECONDS} + $right->{SECONDS};
    $answer->{MINUTES} = $left->{MINUTES} + $right->{MINUTES};
    $answer->{HOURS>   = $left->{HOURS}   + $right->{HOURS};

    if  ($answer->{SECONDS} >= 60) {
        $answer->{SECONDS} %= 60;
        $answer->{MINUTES} ++;
    }

    if  ($answer->{MINUTES} >= 60) {
        $answer->{MINUTES} %= 60;
        $answer->{HOURS}   ++;
    }

          $answer;

}
```

Числовые операторы рекомендуется перегружать лишь в том случае, если объекты соответствуют какой-то числовой конструкции — например, комплексным числам или числам с повышенной точностью, векторам или матрицам. В противном случае программа становится слишком сложной, а пользователи делают неверные предположения относительно работы операторов. Представьте себе класс, который моделирует страну. Если вы создадите оператор для сложения двух стран, то почему нельзя заняться вычитанием? Как видите, перегрузка операторов для нечисловых математических объектов быстро приводит к абсурду.

Объекты (а в сущности, и любые ссылки) можно сравнивать с помощью == и eq, но в этом случае вы узнаете лишь о совпадении их адресов (при этом == работает примерно в 10 раз быстрее, чем eq). Поскольку объект является всего лишь высокоуровневым представлением обычного машинного адреса, во многих ситуациях требуется определить собственный критерий того, что следует понимать под равенством двух объектов.

Даже для нечисловых классов особенно часто перегружаются два оператора: сравнения и строковой интерполяции. Допускается перегрузка как оператора <=>, так и cmp, хотя преобладает второй вариант. После того как для объекта будет

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

определен оператор <=>, вы также сможете использовать операторы ==, !=, <, <=, > и >= для сравнения объектов. Если отношения порядка нежелательны, ограничьтесь перегрузкой ==. Аналогично, перегруженная версия cmp используется в lt, gt и других строковых сравнениях лишь при отсутствии их явной перегрузки.

Оператор строковой интерполяции обозначается странным именем "" (две кавычки). Он вызывается каждый раз, когда происходит строковое преобразование — например, внутри кавычек или апострофов или при вызове функции print.

Прочитайте документацию по директиве overload, прилагаемую к Perl. Перегрузка операторов Perl откроет перед вами некоторые нетривиальные возможности — например, методы строковых и числовых преобразований, автоматическая генерация отсутствующих методов и изменение порядка операндов при необходимости (например, в выражении 5 + $a, где $a является объектом).

Пример. Перегруженный класс StrNum

Ниже приведен класс StrNum, в котором числовые операторы используются для работы со строками. Да, мы действительно собираемся сделать то, против чего настраивали вас, то есть применить числовые операторы к нечисловым объектам, однако программисты по опыту работы в других языках всегда ожидают, что + и == будут работать со строками. Это всего лишь несложный пример, демонстрирующий перегрузку операторов. Подобное решение почти наверняка не будет использоваться в коммерческой версии программы из-за проблем, связанных с быстродействием. Кроме того, перед вами один из редких случаев использования конструктора, имя которого совпадает с именем класса, — наверняка это порадует программистов ср знанием C++ и Python.

```
#'/usr/bin/perl
# show_strnum - пример перегрузки операторов
use StrNum;

$x = StrNum("Red"); $y = StrNum("Black");
$z = $x + $y; $r = $z * 3;
print "values are $x, $y, $z, and $r\n";
print "$x is ", $x < $y^ "LT" : "GE", " $y\n";
```

```
values are Red, Black, RedBlack, and 0
Red is GE Black
```

Исходный текст класса приведен в примере 13.1.

Пример 13.1. StrNum

```
package StrNum;

use Exporter ();
@ISA = 'Exporter';
@EXPORT = qw(StrNum); # Необычно

use overload (
        '<r>'   => \&spaceship,
        'cmp'   => \&spaceship,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        '""'     => \&stringify,
        'bool'   => \&boolify,
        '0+'     => \&numify,
        '+'      => \&concat,

);

# Конструктор
sub StrNum($) {
    my ($value) = @_;
            bless \$value;
}

sub stringify { ${ $_[0] } }
sub numify    { ${ $_[0] > }
sub boolify   { ${ $_[0] } }

# Наличие <=> дает нам <, == и т. д.
sub spaceship {
    my ($s1, $s2, $inverted) = @_;
            $inverted ? $$s2 cmp $$s1 : $$s1 cmp $$s2;
}

# Использует stringify
sub concat {
    my ($s1, $s2, $inverted) = @_;
            StrNum $inverted ? ($s2 . $s1) : ($s1 . $s2);
}

# Использует stringify

    my ($s1, $s2, $inverted) = @_;
            StrNum $inverted ? ($s2 x $s1) : ($s1 x $s2);
}

1;
```

Пример. Перегруженный класс FixNum

В этом классе перегрузка оператора позволяет управлять количеством десятичных позиций при выводе. При этом во всех операциях используется полная точность. Метод `places()` вызывается для класса или конкретного объекта и задает количество выводимых позиций справа от десятичной точки.

```
#!/usr/bin/perl
# demo_fixnum - show operator overloading
use FixNum;

FixNum->places(5);

$x = FixNum->new(40);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$y = FixNum->new(12);

print "sum of $x and $y is",  $x + $y,"\n";
print "product of $x and $y is ", $x * $y, "\n";

$z = $x / $y;
printf "$z has %d places\n", $z->places;
$z->places(2) unless $z->places;
print "div of $x by $y is $z\n";
print "sq          that is ", $z * $z, "\n";
```

```
sum of STRFixNum: 40 and STRFixNum: 12 is STRFixNum: 52
product of STRFixNum: 40 and STRFixNum: 12 is STRFixNum: 480
STRFixNum: 3 has 0 places
div of STRFixNum: 40 by STRFixNum: 12 is STRFixNum: 3.33
square of that is STRFixNum: 11.11
```

Исходный текст класса приведен в примере 13.2. Из математических операций в нем перегружаются только операторы сложения, умножения и деления. Также перегружен оператор <=>, обеспечивающий выполнение всех сравнений, оператор строковой интерполяции и оператор числового преобразования. Оператор строковой интерполяции выглядит необычно, но это было сделано для удобства отладки.

Пример. 13.2 FixNum

```
package FixNum;

usestrict;

my $PLACES = 0;

sub new {
    my $proto   = shift;
    my $class   =              || $proto;
                                  $proto;

    my $v = shift;
    my $self = {
        VALUE  => $v,
        PLACES => undef,
    };
    if            defined
        $self->{PLACES} =
    } elsif ($v =~ /(\.\d*)/) {
        $self->{PLACES} = length($1) - 1;
    }  else {
        $self->{PLACES} = 0;
    }
            bless $self, $class;
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub places {
    my $proto = shift;
                                $proto;
                              || $proto;

    if (@_) {
        my $places = shift;
        ($self ? $self->{PLACES} : $PLACES) = $places;
    }
            $self ? $self->{PLACES} : $PLACES;
}

sub _max { $_[0] > $_[1] ^ $_[0] : $_[1] }

use overload  '+'   => \&add,
              *     => \&multiply,
             '/'    => \&divide,
             '<=>'  => \&spaceship,
                    => \&as_string,
             '0+'   => \&as_number;

sub add {
    my ($this, $that, $flipped) = @_;
                $this->new( $this->{VALUE}   $that->{VALUE}
                    _max($this->{PLACES}, $that->{PLACES}

}

sub multiply {
    my ($this, $that, $flipped) = @_;
                $this->new( $this->{VALUE} * $that->{VALUE} );
                    _max($this->{PLACES}, $that->{PlACES} ));

}

sub divide {
    my ($this, $that, $flipped) = @_;
                $this->new( $this->{VALUE} / $that->{VALUE} );
                    _max($this->{PLACES}, $that->{PLACES}

}

sub as_string {
    my $self = shift;
            sprintf("STR%s: %.*f",
        defined($self->{PLACES}) ^ $self->{PLACES} : $PLACES,
            $self->{VALUE});
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 13.2 (продолжение)

```
sub as_number {
    my $self = shift;
          $self->{VALUE};
}

sub spaceship {
    my ($this, $that, $flipped) = @_;
    $this->{VALUE} <=> $that->{VALUE};
}

1;
```

Смотри также

Документация по стандартной директиве use overload, а также модулям Math::BigInt и Math::Complex.

13.15. Создание "магических" переменных функцией tie

Проблема

Требуется организовать специальную обработку переменной или манипулятора.

Решение

Воспользуйтесь функций tie, чтобы создать объектные связи для обычной переменной.

Комментарий

Каждый, кому приходилось работать с DBM-файлами в Perl, уже использовал связанные объекты. Возможно, самый идеальный вариант работы с объектами — тот, при котором пользователь их вообще не замечает. Функция tie связывает переменную или манипулятор с классом, после чего все обращения к связанной переменной или манипулятору перехватываются специальными методами.

Наиболее важными являются следующие методы tie: FETCH (перехват чтения), STORE (перехват записи) и конструктор, которым является один из методов TIESCALAR, TIEARRAY, TIEHASH или TIEHANDLE.

Пользовательский код	Выполняемый код
tie $s, "SomeClass"	SomeClass->TIESCALAR()
$p = $s	$p = $obj->FETCH()
$s = 10	$obj->STORE(10)

Откуда берется объект $obj? Вызов tie приводит к вызову конструктора TIESCALAR соответствующего класса. Perl прячет возвращенный объект и тайком использует его при последующих обращениях.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Ниже приведен простой пример класса, реализующего кольцевую структуру данных. При каждом чтении переменной выводится следующее значение из кольца, а при записи в кольцо заносится новое значение.

```
#!/usr/bin/perl
# demo_valuering - демонстрация связывания
use ValueRing;
tie $color, 'ValueRing', blue);
print "$color $color $color $color $color $color\n";
red blue red blue red blue

$color = 'green';
print "$color $color $color $color $color $color\n";
        blue          blue
```

Простая реализация класса ValueRing приведена в примере **13.3.**

Пример 13.3. ValueRing

```
package ValueRing;

# Конструктор для связывания скаляров
sub TIESCALAR {
    my ($class, @values) = @_;
    bless \@values, $class;
          \@values;
}

# Перехватывает чтение
sub FETCH {
    my $self = shift;
    push(@$self, shift(@$self));
          $self->[-1];
}

# Перехватывает запись
sub STORE {
    my ($self, $value) = @_;
    unshift @$self, $value;

}

1;
```

Вероятно, такой пример кажется надуманным, но он показывает, как легко создать связь произвольной сложности. Для пользователя $color остается старой доброй переменной, а не объектом. Все волшебство спрятано под связью. При связывании скалярной переменной совсем не обязательно использовать скалярную ссылку; мы использовали ссылку на массив, но **вы** можете выбрать любой другой вариант. Обычно при связывании **любых** переменных используется ссылка на **хэш,** поскольку она обеспечивает наиболее гибкое представление объекта.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Для массивов и хэшей возможны и более сложные операции. Связывание манипуляторов появилось лишь в версии 5.004, а до появления версии 5.005 возможности применения связанных массивов были несколько ограничены, но связывание хэшей всегда поддерживалось на высоком уровне. Поскольку полноценная поддержка связанных хэшей требует реализации множества методов объекта, многие пользователи предпочитали наследовать от стандартного модуля Tie::Hash, в котором существуют соответствующие методы по умолчанию.

Ниже приведены некоторые интересные примеры связывания.

Пример связывания. Запрет $_

Этот любопытный связываемый класс подавляет использование неявной переменной $_. Вместо того чтобы подключать его командой use, что приведет к косвенному вызову метода import() класса, воспользуйтесь командой no для вызова редко используемого метода unimport(). Пользователь включает в программу следующую команду:

После этого любые попытки использования нелокализованной глобальной переменной $_ приводят к инициированию исключения.

Рассмотрим применение модуля на небольшом тестовом примере:

```
#!/usr/bin/perl
Я nounder_demo - запрет использования $_ в программе

@tests = (
    "Assignment"  => sub { $_ = "Bad" },
    "Reading"     => sub { print },
    "Matching"    => sub { $x = /badness/ },
    "Chop"        => sub { chop },
    "Filetest"    => sub { -x },
    "Nesting"     => sub { for (1..3) { print } },
);

while ( ($name, $code) = splice(@tests, 0, 2) ) {
    print "Testing $name: ";
    eval { &$code };
    print $@ ? "detected" : "missed!";
    print "\n";
}
```

Результат выглядит так:

```
Testing Assignment: detected
Testing Reading: detected
Testing Hatching: detected
Testing Chop: detected
Testing Filetest: detected
Testing Nesting: 123missed!
```

В последнем случае обращение к переменной не было перехвачено, поскольку она была локализована в цикле for.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Исходный текст примере 13.4. Обратите внимание, каким маленьким он получился. Функция tie вызывается модулем в инициализирующем коде.

Пример 13.4.

```
package
use Carp;
sub TIESCALAR {
    my $class = shift;
    my $dummy;
        bless \$dummy => $class;
}
sub FETCH { croak "Read access to \$_ forbidden" }
sub STORE { croak "Write access to \$_ forbidden" >
sub unimport { tie($_, __PACKAGE__) >
sub import { untie $_ >
tie($_, __PACKAGE__) unless tied $_;
1;
```

Чередование вызовов use и no для этого класса в программе не принесет никакой пользы, поскольку они обрабатываются во время компиляции, а не во время выполнения. Чтобы снова воспользоваться переменной $_, локализуйте ее.

Пример связывания. Хэш с автоматическим дополнением

Следующий класс создает хэш, который автоматически накапливает повторяющиеся ключи в массиве вместо их замены.

```
#!/usr/bin/perl
# appendhash_demo - хэш с автоматическим дополнением
use Tie::AppendHash;
tie%tab, 'Tie::AppendHash';

$tab{beer} = "guinness";
$tab{food} = "potatoes";
$tab{food} = "peas";

while (my($k, $v) = each %tab) {
    print "$k => [@$v]\n";
}
```

Результат выглядит так:

```
food => [potatoes peas]
beer => [guinness]
```

Простоты ради мы воспользовались шаблоном модуля для связывания хэша, входящим в стандартную поставку (см. пример 13.5). Для этого мы загружаем модуль Tie::Hash и затем наследуем от класса Tie::StdHash (да, это действительно разные имена - файл *Tie/Hash.pm* содержит классы Tie::Hash и Tie::StdHash, несколько отличающиеся друг от друга).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 13.5. Tie::AppendHash

```
package Tie::AppendHash;
use strict;
use Tie::Hash;
use Carp;
use vars qw(@ISA);
@ISA = qw(Tie::StdHash);
sub STORE {
    my ($self, $key, $value) = @_;
    push @{$self->{key}}, $value;
}
1;
```

Пример связывания. Хэш без учета регистра символов

Ниже приведен другой, более хитроумный пример связываемого хэша. На этот раз хэш автоматически преобразует ключи к нижнему регистру.

```
#!/usr/bin/perl
ft folded_demo - хэш с автоматическим преобразованием регистра
use Tie::Folded;
tie %tab, 'Tie::Folded';

$tab<VILLAIN} = "big ";
$tab{herOine} =      riding hood";
$tab{villain}¬= "bad wolf";

while ( my($k, $v) = each %tab ) {
    print "$k is $v\n";
}
```

Результат демонстрационной программы выглядит так:

```
heroine                hood
villain is big bad wolf
```

Поскольку на этот раз перехватывается большее количество обращений, класс из примера 13.6 получился более сложным, чем в примере 13.5.

Пример 13.6. Tie::Folded

```
package Tie::Folded;
use strict;
use Tie::Hash;
use vars qw(@ISA);
@ISA = qw(Tie::StdHash);
sub STORE {
    my ($self, $key, $value) = @_;
            $self->{lc $key} = $value;
    }
sub FETCH {
    my ($self, $key) = @_;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            $self->{lc $key};
}
sub EXISTS {
    my ($self, $key) = @_;
            exists $self->{lc $key};
}
sub DEFINED <
    my ($self, $key) = @_;
            defined $self->{lc $key};
}
1;
```

Пример. Хэш с возможностью поиска по ключу и по значению

Следующий хэш позволяет искать элементы как по ключу, так и по значению. Для этого метод STORE заносит в хэш не только значениепо ключу, но и обратную пару — ключ по значению.

Если сохраняемое значение представляет собой ссылку, возникают затруднения, поскольку обычно ссылка не может использоваться в качестве ключа хэша. Проблема решается классом Tie::RefHash, входящим в стандартную поставку. Мы унаследуем от него.

```
#!/usr/bin/perl -w
# revhash_demo - хэш с возможностью поиска по ключу "или* по значению
use strict;
use Tie::RevHash;
my %tab;
tie %tab, 'Tie::RevHash';
%tab = qw<
    Red         Rojo
    Blue        Azul
                Verde
};
$tab{EVIL} = [ "No way!", "Way!!" ];

while ( my($k, $v) = each %tab ) {
    print          "[@$k]"       "    ",
                ^ "[@$v]"       "\n";
}
```

о выдает следующий результат:

```
[No way! Way!!] = EVIL>
EVIL => [No way! Way!!]
Blue => Azul
        Verde
Rojo => Red
Red => Rojo
Azul => Blue
Verde => Green
```

Исходный текст модуля приведен в примере 13.7. Оцените размеры!

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 13.7. Tie::RevHash

```
package Tie::RevHash;
use Tie::RefHash;
use vars qw(@ISA);
@ISA = qw(Tie::RefHash);
sub STORE {
    my ($self, $key, $value) = @_;
    $self->SUPER::STORE($key, $value);
    $self->SUPER::STORE($value, $key);
}

sub DELETE {
    my ($self, $key) = @_;
    my $value = $self->SUPER::FETCH($key);
    $self->SUPER::DELETE($key);
    $self->SUPER::DELETE($value);
}

1;
```

Пример связывания. Манипулятор с подсчетом обращений

Пример связывания для файлового манипулятора выглядит так:

```
use Counter;
tie *CH, 'Counter';
while (<CH>) {

    print "Got $\n";
}
```

При запуске эта программа выводит Got 1, Got 2 и так далее — пока вы не прервете ее, не перезагрузите компьютер или не наступит конец света (все зависит от того, что случится раньше). Простейшая реализация приведена в примере 13.8.

Пример 13.8. Counter

```
package Counter;
sub TIEHANDLE <
    my $class = shift;
    my $start = shift;
            bless \$start => $class;
}
sub READLINE {
    my $self = shift;
            ++$$self;
}
1;
```

Пример связывания. Дублирование вывода по нескольким манипуляторам

Напоследок мы рассмотрим пример связанного манипулятора, который обладает tee-подобными возможностями — он объединяет STDOUT и STDERR:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Tie::Tee;
tie "TEE, 'Tie::Tee', *STDOUT, *STDERR;
print TEE "This line goes both places.\n"
```

Или более подробно:

```
#!/usr/bin/perl
# demo_tietee
use Tie::Tee;
use Symbol;

@handles = (*STDOUT);
for $i'( 1 .. 10 ) {
    push(@handles, $handle = gensym());
    open($handle, ">/tmp/teetest.$i");
}

tie *TEE, 'Tie::Tee', @handles;
print TEE "This lines goes many places.\n";
```

Содержимое файла *Tie/Tee.pm* показано в примере 13.9.

Пример 13.9. Tie::Tee

```
package Tie::Tee;

sub TIEHANDLE {
    my $class   = shift;
    my $handles = [@_];

    bless $handles, $class;
            $handles;
}

sub PRINT {
                shift;
    my $handle;
    my $success = 0;

    foreach $handle (@$href) {
        $success += print $handle @_;
    }

            $success == @$href;
}

1;
```

Смотри также

Функция tie описана в *perlfunc*(1) и *perltie*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14 Базы данных

Все, чего я прошу, — это информация.
Чарльз Диккенс, "Дэвид Копперфильд"

Введение

Базы данных встречаются везде, где происходит обработка данных. На простейшем уровне базой данных можно считать любой файл, а на самом сложном — дорогую и сложную реляционную базу данных, обрабатывающую тысячи транзакций в секунду. Между этими полюсами расположены бесчисленные механизмы ускоренного доступа к более или менее структурированным данным. Perl поддерживает работу с базами данных на любом из этих уровней.

На заре компьютерной эпохи люди заметили, что базы данных на основе плоских файлов плохо подходят для работы с большими объемами информации. Плоские файлы улучшались посредством введения записей фиксированной длины или индексирования, однако обновление требовало все больших затрат, и некогда простые приложения увязали в болоте ввода/вывода.

Умные программисты почесали в затылках и разработали более удачное решение. Поскольку хеш, находящийся в памяти, обеспечивает более удобный доступ к данным по сравнению с массивом, хеш на диске также упростит работу с данными по сравнению с «массивообразным» текстовым файлом. За ускорение доступа приходится расплачиваться объемом, но дисковое пространство в наши дни стоит дешево (во всяком случае, так принято считать).

Библиотека DBM предоставляет в распоряжение программистов простую и удобную базу данных. С хешами, ассоциированными cDBM-файлами, можно выполнять те же операции, что и с хешами в памяти. В сущности, именно так построена вся работа с базами данных DBM в Perl. Вы вызываете `dbmopen` с именем хеша и именем файла, содержащего базу данных. Затем при любом обращении к хешу Perl выполняет чтение или запись в базе данных DBM на диске.

Рецепт 14.1 демонстрирует процесс создания базы данных DBM, а также содержит рекомендации относительно ее эффективного использования. Хотя с файлами DBM допускаются все операции, разрешенные для Простых хешей, возникают

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

проблемы быстродействия, неактуальные для хешей в памяти. Рецепты 14.2 и 14.4 разъясняют суть этих проблем и показывают, как справиться с ними. С файлами DBM также можно выполнять операции, недоступные для обычных хешей. Два примера таких операций рассматриваются в рецептах 14.6 и 14.7.

Разные реализации DBM обладают разными возможностями. Старая функция dbmopen позволяла использовать лишь ту библиотеку DBM, с которой был построен Perl. Если вы хотели использовать dbmopen для чтения базы данных одного типа и записи в другой тип — с читайте, что вам не повезло. Положение было исправлено в Perl версии 5, где появилась возможность связать хеш с произвольным классом объекта — см. главу 13 «Классы, объекты исвязи".

В следующей таблице перечислены некоторые доступные библиотеки DBM.

Особенности	NDBM	SDBM	GDBM	DB
Программное обеспечение для связи поставляется с Perl	Да	Да	Да	Да
Исходные тексты поставляются с Perl	Нет	Да	Нет	Нет
Возможность распространения исходных текстов	Нет	Да	GPL[1]	Да
Доступность через FTP	Нет	Да	Да	Да
Легкость построения	–	Да	Да	Нормально
Частое применение в UNIX	Да[3]	Нет	Нет[4]	Нет[4]
Нормальное построение в UNIX	–	Да	Да	Да[5]
Нормальное построение в Windows	–	Да	Да	Да[6]
Размеркода	[7]	Малый	Большой	Большой
Использование диска	[9]	Малое	Большое	Нормальное
Скорость	[9]	Низкая	Нормальная	Высокая
Ограничение размера блока	4Кб	1Кб[10]	Нет	Нет
Произвольный порядок байтов	Нет	Нет	Нет	Да
Порядок сортировки, определяемый пользователем	Нет	Нет	Нет	Да
Поиск по неполному ключу	Нет	Нет	Нет	Да

Применение кода с общей лицензией GPL в программах должно удовлетворять некоторым условиям. За дополнительной информацией обращайтесь на *www.gnu.org*.

См. библиотечный метод DB_File. Требует символических ссылок.

На некоторых компьютерах может входить в библиотеку совместимости с BSD.

Кроме бесплатных версий UNIX - Linux, OpenBSD и NetBSD.

При наличии ANSI-компилятора C.

До выхода единой версии 5.005 существовало несколько разных версий Perl для Windows-систем, включая стандартный порт, построенный по обычной поставке Perl, и ряд специализированных портов. DB, как и большинство модулей CPAN, строится только в стандартной версии.

Зависит от поставщика.

Уменьшается при компиляции для одного метода доступа.

Зависит от поставщика.

По умолчанию, но может переопределяться (с потерей совместимости для старых файлов).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

NDBM присутствует в большинстве систем семейства BSD. GDBM представляет собой GNU-реализацию DBM. SDBM входит в поставку Xll и в стандартную поставку Perl. DB означает библиотеку Berkeley DB. Хотя остальные библиотеки фактически реализуют заново исходную библиотеку DB, код Berkeley DB позволяет работать с тремя разными типами баз данных и старается устранить многие недостатки, присущие другим реализациям (затраты дискового пространства, скорость и размер).

Строка "Размер кода" относится к размеру откомпилированной библиотеки, а строка "Использование диска" — к размеру создаваемых ей файлов баз данных. Размер блока определяет максимальный размер ключа или значения в базе. Строка "Произвольный порядок байтов" говорит о том, использует ли система баз данных аппаратный порядок следования байтов или создает переносимые файлы. Сортировка в пользовательском порядке позволяет сообщить библиотеке, в каком порядке должны возвращаться списки ключей, а поиск по неполному ключу позволяет выполнять приблизительный поиск в базе.

Большинство программистов Perl предпочитает берклиевские реализации. На многих системах эта библиотека уже установлена, и Perl может ей пользоваться. Другим мы рекомендуем найти эту библиотеку в CPAN и установить ее. Это заметно упростит вашу жизнь.

DBM-файлы содержат пары "ключ/значение". В терминологии реляционных баз данных вы получаете базу данных, которая содержит всего одну таблицу с двумя полями. В Рецепте 14.8 показано, как использовать модуль MLDBM с CPAN для хранения сложных структур данных в DBM-файлах.

При всех своих достоинствах модуль MLDBM не может преодолеть главное ограничение: критерием для извлечения записи является содержимое лишь одного столбца, ключа хеша. Если вам понадобится сложный запрос, могут возникнуть непреодолимые трудности. В таких случаях подумайте о специализированной системе управления базами данных (СУБД). Проект DBI содержит модули для работы с Oracle, Sybase, mSQL, MySQL, другими системами.

По адресам *http://www.hermetica.com/technologia/perl/DBI/index.html* и *http://www.perl/com/CPAN/modules/by-category/07_Database_Interfaces/* в настоящее время имеются следующие модули:

AcsiiDB	DBI Db	MLDBM	OLE	Pg	Sybase
CDB File	DBZ File	Fame	Msql '	ObjStore	
DBD	DB File	Ingperl	MySQL	Oraperl	Sprite
XBase					

14.1. Создание и использование DBM-файла

Проблема

Вы хотите создать, заполнить, просмотреть или удалить значения из базы данных DBM.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь функцией dbmopen или tie, чтобы открыть базу и сделать ее доступной через хэш. Затем работайте с хэшем, как обычно. После завершения работы вызовите dbmclose или untie.

dbmopen

```
use DB_File;                        # необязательно; переопределяет
                                    # стандартный вариант
dbmopen %HASH, FILENAME, 0666       # открыть базу данных через %HASH
    or die "Can't open FILENAME: $!\n";

$V = $HASH{KEY};                    # Получить данные из базы
$HASH{KEY} = VALUE;                 # Занести данные в базу
if (exists $HASH{KEY}) f            # Проверить наличие данных в базе
    # ...
}
delete $HASH{KEY};                  # Удалить данные из базы
dbmclose %HASH;                     # Закрыть базу данных
```

tie

```
use DB_File;                        в Загрузить модуль баз данных

tie %HASH, "DB_File", FILENAME      `# Открыть базу данных
    or die "Can't open FILENAME: $!\n";   в через %HASH

$V = $HASH{KEY};                    # Получить данные из базы
$HASH{KEY} = VALUE;                 " Занести данные в базу
if (exists $HASH{KEY}) {            в Проверить наличие данных в базе
    # ...
}
delete $HASH{KEY};                  # Удалить данные из базы
untie %hash;                        # Закрыть базу данных
```

Комментарий

Работа с базой данных через хэш отличается широкими возможностями и простотой. В вашем распоряжении оказывается хэш, состояние которого сохраняется и после завершения программы. Кроме того, он работает намного быстрее, чем хэш, полностью загружаемый при каждом запуске; даже если хэш состоит из миллиона элементов, ваша программа запустится практически мгновенно.

Программа из примера 14.1 работает с базой данных так, словно она является обычным хэшем. Для нее даже можно вызывать keys или each. Кроме того, для связанных DBM-хэшей реализованы функции exists и defined. В отличие от обычного хэша, для DBM-хеша эти функции идентичны.

Пример 14.1. userstats

```
#!/usr/bin/perl -w
# userstats - вывод статистики о зарегистрированных пользователях.
" При вызове с аргументом выводит данные по конкретным пользователям.
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 14.1 (продолжение)

```
use DB_File;

$db = '/tmp/userstats.db';      # База для хранения данных между запусками

tie(%db, 'DB_File', $db)       or die "Can't open DB_File $db : $!\n";

if (@ARGV) {
    if ("@ARGV" eq "ALL") {
        @ARGV = sort keys %db;
    }
            $user (@ARGV) {
            print "$user\t$db{$user}\n";
    }
> else {
    @who = 'who';                           # Запустить who(1)
    if ($?) <
        die "Couldn't run who: $?\n";       # Аварийное завершение
    >
    # Извлечь имя пользователя (первое в строке) и обновить
            $line (@who) {
        $line =- /^(\S+)/;
        die "Bad line from who: $line\n" unless $1;
        $db{$1}++;
    }
}

untie %db;
```

Мы воспользовались командой *who* для получения списка зарегистрированных пользователей. Обычно результат выглядит следующим образом:

```
gnat      ttyp1    May 29 15:39    (coprolith.frii.com)
```

Если вызвать программу *userstats* без аргументов, она проверяет зарегистрированных пользователей и соответствующим образом обновляет базу данных.

Передаваемые аргументы интерпретируются как имена пользователей, о которых следует вывести информацию. Специальный аргумент `"ALL"` заносит в `@ARGV` отсортированный список ключей DBM. Для больших хэшей с множеством ключей это обойдется слишком дорого — лучше связать хэш с B-деревом (см. рецепт 14.6).

Смотри также

Документация по стандартным модулям GDBM_File, NDBM_File, SDBM_File, DB_File; *perltie*(1); рецепт 13.15. Влияние `umask` на процесс создания файлов рассматривается в рецепте 7.1.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14.2. Очистка DBM-файла

Проблема

Требуется стереть все содержимое DBM-файла.

Решение

Откройте базу данных и присвойте ей (). При этом можно использовать функцию dbmopen:

```
dbmopen(%HASH, $FILENAME, 0666)     or die "Can't open FILENAME: $!\n";
%HASH = ();
dbmclose %HASH;
```

или tie:

```
use DB_File;

tie(%HASH, "DB_File", $FILENAME)   or die "Can't open FILENAME: $!\n";
%HASH = ();
untie %hash;
```

Существует и другое решение — удалить файл и открыть его заново в режиме создания:

```
unlink $FILENAME
    or die "Couldn't unlink $FILENAME to empty the database: $!\n";
dbmopen(%HASH, $FILENAME, 0666)
    or die "Couldn't        $FILENAME database: $!\n";
```

Комментарий

Возможно, удаление файла с последующим созданием выполняется быстрее, чем очистка, но при этом возникает опасность подмены, которая может нарушить работу неосторожной программы или сделать ее уязвимой для нападения. В промежуток между удалением файла и его повторным созданием нападающий может создать ссылку, указывающую на жизненно важный файл */etc/precious*, с тем же именем, что и у вашего файла. При открытии файла библиотекой DBM содержимое */etc/precious* будет уничтожено.

При удалении базы данных DB_ File с повторным созданием теряются значения всех настраиваемых параметров — размер страницы, фактор заполнения и т. д. Это еще один веский довод в пользу присваивания связанному хэшу пустого списка.

Смотри также

Документация по стандартному модулю DB_File; рецепт 14.1. Функция unlink описана в *perlfunc(1)*.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14.3. Преобразование DBM-файлов

Проблема

У вас имеется файл в одном формате DBM, однако другая программа желает получить данные в другом формате DBM.

Решение

Прочитайте ключи и значения из исходного DBM-файла и запишите их в другой файл в другом формате DBM, как показано в примере 14.2.

Пример 14.2. db2gbdm

```
#!/usr/bin/perl -w
# db2gdbm: преобразование DB в GDBM

use strict;

use DB_File;
use GDBM_File;

unless (@ARGV == 2) {
    die "usage: db2gdbm infile outfile\n";
}

my ($infile, $outfile) = @ARGV;
my (%db_in, %db_out);

# Открыть файлы
tie(%db_in, 'DB_File', $infile)
    or die "Can't tie $infile: $!";
tie(%db_out, 'GDBM_File', $outfile, GDBM_WRCREAT, 0666)
    or die "Can't tie $outfile: $!";

# Скопировать данные (не пользуйтесь %db_out = %db_in,
# потому что для больших баз это работает медленно)
while (my($k, $v) = each %db_in) {
    $db_out{$k} = $v;
}

# Функции untie вызываются автоматически при завершении программы
untie %db_in;
untie %db_out;
```

Командная строка выглядит так:

```
% db2gdbm /tmp/users.db /tmp/users.gdbm
```

Комментарий

Если в одной программе используются различные типы DBM-файлов, вам придется использовать интерфейс `tie`, а не `dbmopen`. Дело в том, что интерфейс

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

dbmopen позволяет работать лишь с одним форматом баз данных и поэтому считается устаревшим.

Копирование хэшей простым присваиванием (%new = %old) работает и для DBM-файлов, однако сначала все данные загружаются в память в виде списка. Для малых хэшей это несущественно, но для больших DBM-файлов затраты могут стать непозволительно большими. Для хэшей баз данных лучше использовать перебор с помощью функции each.

Смотри также

Документация по стандартным модулям GDBM_File, NDBM_File, SDBM_File, DB_File; рецепт 14.1.

14.4. Объединение DBM-файлов

Проблема

Требуется объединить два DBM-файла в один с сохранением исходных пар "ключ/значение".

Решение

Либо объедините базы данных, интерпретируя их хэши как списки:

```
%OUTPUT = (%INPUT1, %INPUT2);
```

либо (более разумный вариант) организуйте перебор пар "ключ/значение":

```
%OUTPUT = ();
                \%INPUT1, \%INPUT2 ) {
    while (my($key, $value) = each(%$href)) (
        if (exists $OUTPUT{$key}) {
            # Выбрать используемое значение
            # и при необходимости присвоить $OUTPUT{$key}
        } else {
            $OUTPUT{$key} = $value;
        }
    }
}
```

Комментарий

Прямолинейный подход из рецепта 5.10 обладает тем же недостатком. Объединение хэшей посредством списковой интерпретации требует, чтобы хэши были предварительно загружены в память, что может привести к созданию огромных временных списков. Если вы работаете с большими хэшами и/или не располагаете достаточной виртуальной памятью, организуйте перебор ключей в цикле each — это позволит сэкономить память.

Между этими двумя способами объединения есть еще одно отличие— в том, как они поступают с ключами, присутствующими в обоих базах. Присваивание

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

пустого списка просто заменяет первое значение вторым. Итеративный перебор позволяет принять решение, как поступить с дубликатом. Возможные варианты — выдача предупреждения или ошибки, сохранение первого экземпляра, замена первого экземпляра вторым, конкатенация обоих экземпляров. Используя модуль MLDBM, можно даже сохранить оба экземпляра в виде ссылки на массив из двух элементов.

Смотри также

Рецепты 5.10; 14.8.

14.5. Блокировка DBM-файлов

Проблема

Необходимо обеспечить одновременный доступ к DBM-файлу со стороны нескольких параллельно работающих программ.

Решение

Воспользуйтесь реализацией механизма блокировки DBM, если он имеется, и заблокируйте файл функцией flock либо обратитесь к нестандартной схеме блокировки из рецепта 7.21.

Комментарий

SDBM и GDBM не обладают возможностью блокировки базы данных. Вам придется изобретать нестандартную схему блокировки с применением дополнительного файла.

В GDBM используется концепция доступа для чтения или записи: файл GDBM в любой момент времени может быть открыт либо многими читающими процессами, либо одним записывающим. Тип доступа (чтение или запись) выбирается при открытии файла. Иногда это раздражает.

Версия 1 Berkeley DB предоставляет доступ к файловому дескриптору открытой базы данных, позволяя заблокировать его с помощью flock. Блокировка относится к базе в целом, а не к отдельным записям. Версия 2 реализует собственную полноценную систему транзакций с блокировкой.

В примере 14.3 приведен пример блокировки базы данных с применением Berkeley DB. Попробуйте многократно запустить программу в фоновом режиме, чтобы убедиться в правильном порядке предоставления блокировок.

Пример 14.3. dblockdemo

```
#!/usr/bin/perl
# dblockdemo - демонстрация блокировки базы данных dbm
use DB_File;
use strict;

sub LOCK_SH { 1 }                # На случай, если у вас нет
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub LOCK_EX { 2 }                  # стандартного модуля Fcntl.
sub LOCK_NB { 4 }                  # Конечно, такое встречается редко,
sub LOCK_UN { 8 }                  # но в жизни всякое бывает.

my($oldval, $fd, $db, %db, $value, $key);

$key    = shift || 'default';
$value  = shift || 'magic';
$value .= " $$";

$db = tie(%db, 'DB_File', '/tmp/foo.db', O_CREAT|O_RDWR, 0666)
    or die "dbcreat /tmp/foo.db $!";
$fd = $db->fd;                     # Необходимо для блокировки
print "$$: db fd is $fd\n";
open(DB_FH, "+<&=$fd")
    or die "dup $!";

unless (flock (DB_FH, LOCK_SH | LOCK_NB)) {
    print "$$: CONTENTION;          during write update!
                Waiting for read lock ($!) ....";
    unless (flock (DB_FH, LOCK_SH)) { die "flock: $!" }
}
print "$$: Read lock granted\n";

$oldval = $db{$key};
print "$$: Old value was $oldval\n";
flock(OB_FH, LOCK_UN);

unless (flock (DB_FH, LOCK_EX | LOCK_NB)) {
    print "$$: CONTENTION; must have exclusive lock!
                Waiting for write lock ($!) ....";
    unless (flock (DB_FH, LOCK_EX)) { die "flock: $!" }
}

print "$$: Write lock granted\n";
$db{$key} = $value;
$db->sync;  # to flush
sleep 10;

flock(DB_FH, LOCK_UN);
undef $db;
untie %db;
close(DB_FH);
print "$$: Updated db to $key=$value\n";
```

Смотри также

Документация по стандартному модулю DB_File; рецепты 7.11; 16.12.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14.6. Сортировка больших DBM-файлов

Проблема

Необходимо обработать большой объем данных, которые должны передаваться в DBM-файл в определенном порядке.

Решение

Воспользуйтесь возможностью связывания B-деревьев модуля DB_File и предоставьте функцию сравнения:

```
use DB_File;

# Указать функцию Perl, которая должна сравнивать ключи
# с использованием экспортированной ссылки на хэш $DB_BTREE

    my ($key1, $key2) = @_ ;
    "\L$key1" cmp "\L$key2" ;
};

tie(%hash, "DB_File", $filename, O_RDWR|O_CREAT, 0666, $DB_BTREE)
    or die "can't tie $filename: $!";
```

Комментарий

Основной недостаток хэшей (как в памяти, так и в DBM-файлах) заключается в том, что они не обеспечивают нормального упорядочения элементов. Модуль Tie::IxHash с CPAN позволяет создать хэш в памяти с сохранением порядка вставки, но это не поможет при работе с базами данных DBM или произвольными критериями сортировки.

Модуль DB_File содержит изящное решение этой проблемы за счет использования B-деревьев. Одно из преимуществ B-дерева перед обычным DBM-хэшем — его упорядоченность. Когда пользователь определяет функцию сравнения, любые вызовы keys, values и each автоматически упорядочиваются. Так, программа из примера 144 создает хэш, ключи которого всегда сортируются без учета регистра символов.

Пример 14.4. sortdemo

```
#!/usr/bin/perl
tt sortdemo - автоматическая сортировка dbm
usestrict;
use DB_File;

$DB_BTREE->{'compare'} = sub {
    my ($key1, $key2) = @_ ;
    "\L$key1" cmp "\L$key2" ;
};

my %hash;
my $filename = '/tmp/sorthash.db';
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
tie(%hash, "DB_File", $filename, O_RDWR|O_CREAT, 0666, $DB_BTREE)
    or die "can't tie $filename: $!";

my $i = 0;
for my $word (qw(Can't you go camp down by Gibraltar)) {
    $hash{$word} = ++$i;


while (my($word, $number) = each %hash) {
    printf "%-12s %d\n", $word, $number;
}
```

По умолчанию записи баз данных B-деревьев DB_File сортируются по алфавиту. Однако в данном случае мы написали функцию сравнения без учета регистра, поэтому применение each для выборки всех ключей даст следующий результат:

```
by          6
camp        4
Can't       1
down        5
Gibraltar   7
go          3
you         2
```

Эта возможность сортировки хэша настолько удобна, что ей стоит пользоваться даже без базы данных на диске. Если передать tie вместо имени файла undef, DB_File создаст файл в каталоге */tmp*, а затем немедленно уничтожит его, создавая анонимную базу данных:

```
tie(%hash, "DB_File", undef, O_RDWR|O_CREAT, 0666, $DB_BTREE)
        or die "can't tie: $!";
```

Обеспечивая возможность сравнения, для своей базы данных в виде B-дерева, необходимо помнить о двух обстоятельствах. Во-первых, при создании базы необходимо передавать новую функцию сравнения. Во-вторых, вы не сможете изменить порядок записей после создания базы; одна и та же функция сравнения должна использоваться при каждом обращении к базе.

Базы данных BTREE также допускают использование повторяющихся или неполных ключей. За примерами обращайтесь к документации.

Смотри также

Рецепт 5.6.

14.7. Интерпретация текстового файла в виде строковой базы данных

Проблема

Требуется организовать работу с текстовым файлом как с массивом строк с привилегиями чтения/записи. Например, это может понадобиться для того, чтобы вы могли легко обновить N-ю строку файла.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Модуль DB_File позволяет связать текстовый файл с массивом.

```
use DB_File;

tie(@array, "DB_File", "/tmp/textfile", O_RDWR|O_CREAT, 0666, $DB_RECNO)
    or die "Cannot open file 'text': $!\en" ;

$array[4] = "a new line";
untie @array;
```

Комментарий

Обновление текстового файла на месте может оказаться на удивление нетривиальной задачей (см. главу 7 "Доступ к файлам»). Привязка RECNO позволяет удобно работать с файлом как с простым массивом строк — как правило, все полагают, что именно этот вариант является наиболее естественным.

Однако этот способ работы с файлами отличается некоторыми странностями. Прежде всего, нулевой элемент связанного массива соответствует первой строке файла. Еще важнее то, что связанные массивы не обладают такими богатыми возможностями, как связанные хэши. Положение будет исправлено в будущих версиях Perl — в сущности, "заплаты" существуют уже сейчас.

Как видно из приведенного выше примера, интерфейс связанного массива ограничен. Чтобы расширить его возможности, методы DB_File имитируют стандартные операции с массивами, в настоящее время не реализованные в интерфейс связанных массивов Perl. Сохраните значение, возвращаемое функцией tie, или получите его позднее для связанного хэша функцией tied. Для этого объекта можно вызывать следующие методы:

```
$X->push(СПИСОК)
```

Заносит элементы списка в конец массива.

```
$value = $X->pop
```

Удаляет и возвращает последний элемент массива.

```
$X->shift
```

Удаляет и возвращает первый элемент массива.

```
$X->unshift(СПИСОК)
```

Заносит элементы списка в начало массива.

```
$X->length
```

Возвращает количество элементов в массиве.

Пример 14.5показывает, как все эти методы используются на практике. Кроме того, он работает с интерфейсом API так, как рассказано в документации модуля DB_File (большая часть рецепта позаимствована из документации DB_file с согласия Пола Маркесса, автора Perl-порта Berkeley DB; материал использован с его разрешения).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 14.5.

```
#!/usr/bin/perl  -w
#               применение низкоуровневого API для привязок
use strict;
use vars qw(@lines $dbobj $file $i);
use DB_File;

$file = "/tmp/textfile";
unlink $file;               # На всякий случай

$dbobj = tie(@lines, "DB_File", $file, O_RDWR|O_CREAT, 0666, $DB_RECNO)
    or die "Cannot open file $file: $!\n";

# Сначала создать текстовый файл.
$lines[0] = "zero";
$lines[1] = "one";
$lines[2] = "two";
$lines[3] =
$lines[4] = "four";

# Последовательно вывести записи.
#
# Метод length необходим из-за того, что при использовании
# связанного массива в скалярном контексте
# не возвращается количество элементов в массиве.

print "\nORIGINAL\n";
foreach $i (0 .. $dbobj->length - 1) {
    print "$i: $lines[$i]\n"; '


# Методы push и pop
$a = $dbobj->pop;
$dbobj->push("last");
print "\nThe last            [$a]\n";

# Методы shift и unshift
$a = $dbobj->shift;
$dbobj->unshift("first");
print "The first            [$a]\n";

# Использовать API для добавления новой записи после записи 2.
$i = 2;
$dbobj->put($i, "Newbie", R_IAFTER);

# и еще одной новой записи после записи 1.
$i = 1;
$dbobj->put($i, "New One", R_IBEFORE);

# Удалить запись 3
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 14.5 (продолжение)

```
$dbobj->del(3);

# Вывести записи в обратном порядке
print "\nREVERSE\n";
for ($i = $dbobj->length - 1; $i >= 0; -- $i) {
    print "$i: $lines[$i]\n";
}

# То же самое, но на этот раз с использованием функций API
print "\nREVERSE again\n";
my ($s, $k, $v) = (0, 0, 0);
for ($s = $dbobj->seq($k, $v, R_LAST);
     $s == 0;
     $s = $dbobj->seq($k, $v, R_PREV))
{
    print "$k: $v\n"
}

undef $dbob];
untie @lines;
```

Результат выглядит так:

```
ORIGINAL
0: zero
1: one
2: two

4: four

    last              [four]
The first              [zero]

REVERSE
5: last

3: Newbie
2: one
1: New One
0: first

REVERSE again
5: last

3: Newbie
2: one
1: New One
0: first
```

Обратите внимание: для перебора массива `@lines` вместо

```
$item (@lines)
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

следует использовать либо

```
foreach $i (0 .. $dbobj->length - 1) { }
```

либо

```
for ($done_yet = $dbobj->get($k, $v, R_FIRST);
      not $done_yet;
      $done_yet = $dbobj->get($k, $v, R_NEXT) )
{
      # Обработать ключ или значение
}
```

Кроме того, при вызове метода put мы указываем индекс записи с помощью переменной $i вместо того, чтобы передать константу. Дело в том, что put возвращает в этом параметре номер записи вставленной строки, изменяя его значение.

Смотри также

Документация по стандартному модулю DB_File.

14.8. Хранение сложных структур данных в DBM-файлах

Проблема

В DBM-файле требуется хранить не скаляры, а что-то иное. Например, вы используете в программе хэш хэшей и хотите сохранить его в DBM-файле, чтобы с ним могли работать другие или чтобы его состояние сохранялось между запусками программы.

Решение

Воспользуйтесь модулем MLDBM от CPAN — он позволяет хранить в хэше более сложные структуры, нежели строки или числа.

```
use MLDBM 'DB_File';
tie(%HASH, 'MLDBM', [... прочие аргументы DBM]) or die $!;
```

Комментарий

MLDBM использует модуль Data::Dumper (см. рецепт 11.14) для преобразования структур данных в строки и обратно, что позволяет хранить их в DBM-файлах. Модуль не сохраняет ссылки; вместо них сохраняются данные, на которые эти ссылки указывают:

```
# %hash - связанный хэш
$hash{"Tom Christiansen"} = [ "book author", 'tchrist@perl.com' ];
$hash{"Tom Boutell"}                 author", 'boutell@boutell.com' ];

# Сравниваемые имена
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$name1 = "Tom Christiansen";
$name2 = "Tom Boutell";

$tom1 = $hash{$name1};        # Получить локальный указатель
$tom2 = $hash{$name2};        # И еще один

print "Two Toming: $tom1 $tom2\n";

ARRAY(0x73048)ARRAY(0x73e4c)
```

Каждый раз, когда **MLDBM** извлекает структуру данных из файла **DBM,** строится новая копия данных. Чтобы сравнить данные, полученные **из** базы данных MLDBM, необходимо сравнить значения полей этой структуры:

```
if ($tom1->[0] eq $tom2->[0] &&
    $tom1->[1] eq $tom2->[1]) <
    print         having runtime fun with one Tom made two.\n";
> else {
    print "No two Toms     ever alike.\n";
}
```

Этот вариант эффективнее следующего:

```
if ($hash{$name1}->[0] eq $hash{$name2}->[0] &&    # НЕЭФФЕКТИВНО
    $hash{$name1}->[1] eq $hash{$name2}->[1]) {
    print         having runtime fun with one Tom made two.\n";
> else {
    print "No two Toms are everalike.\n";
}
```

Каждый раз, когда в программе встречается конструкция `$hash{...}`, происходит обращение к DBM-файлу. Приведенный выше неэффективный код обращается к базе данных четыре раза, тогда как код с временными переменными `$tom1` и `$tom2` обходится всего двумя обращениями.

Текущие ограничения механизма **tie** не позволяют сохранять или модифицировать компоненты **MLDBM** напрямую:

```
$hash{"Tom Boutell"}->[0] = "Poet Programmer";     # НЕВЕРНО
```

Любые операции чтения, модификации и присваивания для частей структуры, хранящейся **в** файле, должны осуществляться через временную переменную:

```
$entry = $hash{"Tom Boutell"};                     # ВЕРНО
$entry->[0] = "Poet Programmer";
$hash{"Tom Boutell"} = $entry;
```

Если MLDBM использует базу данных с ограниченным размером значений (например, SDBM), вы довольно быстро столкнетесь с этими ограничениями. Чтобы выйти из положения, используйте GDBM_File или DB_File, в которых размер ключей или значений неограничивается. Предпочтение отдается библиотеке DB_File, поскольку она использует нейтральный порядок байтов, что позволяет использовать базу данных в архитектурах как с начальным старшим, так и с начальным младшим байтом.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Документация по модулям Data::Dumper, MLDBM и Storable от CPAN; рецепты 11.13; 14.9.

14.9. Устойчивые данные

Проблема

Вы хотите, чтобы значения переменных сохранялись между вызовами программы.

Решение

Воспользуйтесь модулем MLDBM для сохранения значений между вызовами программы:

```
use MLDBM 'DB_File';

my ($VARIABLE1,$VARIABLE2);
                        '/projects/foo/data';
BEGIN <
    my %data;
    tie(%data, 'MLDBM',
        or die "Can't tie to                    $!";
    $VARIABLE1 = $data{VARIABLE1};
    $VARIABLE2 = $data{VARIABLE2};
    # ...
    untie %data;
}
END {
    my %data;
    tie (%data, 'MLDBM',
        or die "Can't tie to                    $!";
    $data{VARIABLE1} = $VARIABLE1;
    $data{VARIABLE2} = $VARIABLE2;
    # ...
    untie %data;
}
```

Комментарий

Существенное ограничение MLDBM заключается в том, что структуру нельзя дополнить или изменить по ссылке без присваивания временной переменной. Мы сделаем это в простой программе из примера 14.6, присваивая
ред вызовом push. Следующая конструкция просто невозможна:

```
push(@{$db{$user}}, $duration);
```

Прежде всего, этому воспротивится MLDBM. Кроме того, $db{$user} может отсутствовать в базе (ссылка на массив не создается автоматически, как это делалось бы в том случае, если бы хэш %db не был связан с DBM-файлом). Именно по-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

этому мы проверяем exists $db{$user} перед тем, как присваивать ходное значение. Мы создаем пустой массив в случае, если он не существовал ранее.

Пример 14.6. mldbm-demo

```
#!/usr/bin/perl -w
# mldbm_demo - применение MLDBM с DB_File

use MLDBM "DB_File";

$db = "/tmp/mldbm-array";

tie %db, 'MLDBM', $db
  or die "Can't open $db : $!";

while(<DATA>) {
    chomp;
    ($user, $duration) = split(/\s+/, $_);
                exists $db{$user} ? $db{$user} : [];
    push(@$array_ref, $duration);
    $db{$user} =
>

foreach $user (sort keys %db) {
    print "$user: ";
    $total = 0;
            $duration (@{ $db{$user} }) {
        print "$duration ";
        $total += $duration;
    }
        print "($total)\n";
    }

__ENO__
gnat        15.3
tchrist     2.5
jules       22.1
tchrist     15.9
gnat        8.7
```

Новые версии MLDBM позволяют выбрать не только модуль для работы с базами данных (мы рекомендуем DB_File), но и **модуль сериализации** (рекомендуем Storable). В более ранних версиях **сериализация** ограничивалась модулем **Data::Dumper**, который работает медленнее Storable. Для использования DB_File со Storable применяется следующая команда:

```
use MLDBM qw(DB_File Storable);
```

Смотри также

Документация по модулям **Data::Dumper**, MLDBM и Storable с CPAN; **рецепты** 11.13; 14.8.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

14.10. Выполнение команд SQL с помощью DBI и DBD

Проблема

Вы хотите направить запрос SQL в систему управления базами данных (например, Oracle, Sybase, mSQL или MySQL) и обработать полученные результаты.

Решение

Воспользуйтесь модулями DBI (DataBase Interface) и DBD (DataBase Driver) от CPAN:

```
use DBI;

$dbh = DBI->connect('OBI:driver', 'username', 'auth',
            { PrintError => 1, RaiseError => 1})
    or die "connecting: $DBI::errstr";
$dbh->do(SQL)
    or die "doing: ", $dbh->errstr;
$sth = DBI->prepare(SQL)
                        ", $dbh->errstr;
$sth->execute
    or die "executing: ", $sth->errstr;
while (@row = $sth->fetchrow_array) {
    " ...
}
$sth->finish;
$dbh->disconnect;
```

Комментарий

DBI является посредником между программой и всеми драйверами, предназначенными для работы с конкретными СУБД. Для большинства операций нужен манипулятор базы данных (в приведенном выше примере — `$dbh`). Он ассоциируется с конкретной базой данных и драйвером при вызове `DBI->connect`.

Первый аргумент `DBI->connect` представляет собой строку, состоящую из трех полей, разделенных двоеточиями. Он определяет *источник данных* — СУБД, к которой вы подключаетесь. Первое поле всегда содержит символы DBI, а второе — имя драйвера, который вы собираетесь использовать (Oracle, mysql и т. д.). Оставшаяся часть строки передается модулем DBI запрошенному модулю драйвера (например, DBD::mysql) и идентифицирует базу данных.

Второй и третий аргументы выполняют аутентификацию пользователя.

Четвертым, необязательным аргументом является ссылка на хэш с определением атрибутов подключения. Если атрибут `PrintError` равен `true`, при каждом неудачном вызове метода DBI будет выдавать предупреждение. Присваивание `RaiseError` имеет аналогичный смысл, за исключением того, что вместо `warn` будет использоваться `die`.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Простые команды SQL (не возвращающие записи данных) могут выполняться методом do манипулятора базы данных. При этом возвращается логическая истина или ложь. Для команд SQL, возвращающих записи данных (например, SELECT), необходимо сначала вызвать метод манипулятора базы данных, чтобы создать манипулятор команды. Далее запрос выполняется методом exec ute, вызванным для манипулятора команды, а записи извлекаются методами выборки fetchrow_array или fetchrow_hashref (возвращает ссылку на хэш, в котором имя поля ассоциируется со значением).

После завершения работы с базой не забудьте отключиться от нее методом disconnect. Если манипулятор базы данных выходит из области действия без предварительного вызова disconnect, модуль **DBI** выдает предупреждение. Эта мера предосторожности предназначена для тех СУБД, которые должны возвращать память системе и требуют корректного отключения от сервера. Перед отключением манипулятора базы данных манипуляторы команд должны получить неопределенное **значение**, выйти **из** области действия или для них должен быть вызван метод finish. Если этого не сделать, **вы** получите предупреждение следующего вида:

```
disconnect(DBI::db=HASH(Ox9df84)) invalidates 1 active cursor(s)
  at -e line 1.
```

Модуль DBI содержит FAQ *tyerldoc DBI::FAQ*) и стандартную документацию *tyerldoc DBI*). Также существует документация для драйверов конкретных СУБД (например, *perldoc DBD::mysql*). Прикладной интерфейс DBI не ограничивается простейшим подмножеством, рассмотренным нами; он предоставляет разнообразные возможности выборки результата и взаимодействия со специфическими средствами конкретных СУБД (например, сохраняемыми процедурами). За информацией обращайтесь к документации по модулю драйвера.

Программа из Примера 14.7 создает и заполняет таблицу пользователей в MySQL, после чего выполняет в ней **поиск**. Она использует атрибут RaiseError и потому обходится без проверки возвращаемого значения для каждого метода.

Пример 14.7. **dbusers**

```
#!/usr/bin/perl -w
# dbusers - работа с таблицей пользователей в MySQL
use DBI;
use User::pwent;

$dbh = DBI->connect('DBI:mysql:dbname:mysqlserver.domain.com:3306',
                    'user', 'password',
                    { RaiseError => 1 })
  or die "connecting : $DBI::errstr\n";

$dbh->do("CREATE TABLE users (uid INT, login CHAR(8))");

$sql_fmt = "INSERT INTO users VALUES( %d, %s )";
while ($user = getpwent) {
    $sql = sprintf($sql_fmt, $user->uid, $dbh->quote($user->name));
    $dbh->do($sql);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

$sth = $dbh->prepare("SELECT * FROM users WHERE uid < 50'),
$sth->execute,

while ((@row) = $sth->fetchrow) {
    print join(", ", map {defined $_ ^ $_ . "(null)"} @row), "\n";
}
$sth->finish;

$dbh->do('DROP TABLE users'),

$dbh->disconnect,
```

Смотри также

Документация по DBI и модулям DBD с CPAN, *http://www.hermetica.com/technologia/perl/DBI/index.html* и *http://www.perl.com/CPAN/modules/by-category/07_Database_Interfaces/*.

14.11. Программа: ggh — поиск в глобальном журнале Netscape

Следующая программа выводит содержимое файла Netscape *history.db*. При вызове ей может передаваться полный **URL** или (один) шаблон. Если программа вызывается без аргументов, она выводит все содержимое журнала. Если не задан параметр -database, используется файл *~/.netscape/history.db*.

В каждой выводимой строке указывается URL и время работы. Время преобразуется в формат localtime параметром -localtime (по умолчанию) или в представление gmtime параметром -gmtime или остается в первоначальном формате (параметр -epoch), что может пригодиться для сортировки по дате.

Шаблон задается единственным аргументом, не содержащим : //.

Чтобы вывести данные по одному или нескольким URL, передайте их в качестве аргументов:

```
% ggh http://www.perl.com/index.html
```

Вывод сведений о адресах, которые **вы** помните лишь приблизительно (шаблоном считается единственный аргумент, не содержащий ://):

```
% ggh perl
```

Вывод всех адресатов электронной почты:

```
% ggh mailto.
```

Для вывода всех посещенных сайтов со списками FAQ используется шаблон Perl с внутренним модификатором /i:

```
'(?i)\bfaq\b'
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если вы не хотите, чтобы внутренняя дата была преобразована в формат localtime, используйте параметр -epoch:

```
% ggh -epoch http://www.perl.com/perl/
```

Если **вы** предпочитаете формат gmtime, используйте параметр -gmtime:

```
% ggh -gmtime http://www.perl.com/perl/
```

Чтобы просмотреть весь файл, не задавайте значения аргументов (вероятно, данные следует перенаправить в утилиту постраничного вывода):

```
% ggh | less
```

Чтобы отсортировать выходные данные по дате, укажите флаг -epoch:

```
% ggh -epoch | sort -rn | less
```

Для сортировки по времени в формате местного часового пояса используется более сложная командная строка:

```
% ggh -epoch | sort -rn |perl -pés/\d+/localtime $&/e' lless
```

Сопроводительная документация Netscape утверждает, что **в** журнале используется формат **NDBM.** Это не соответствует действительности: на самом деле использован формат Berkeley **DB,** поэтому вместо NDBM_File (входит **в** стандартную поставку всех систем, на которых работает Perl) **в** программе загружается DB_File. Исходный текст программы приведен в примере 14.8.

Пример 14.8. ggh

```
tf!/usr/bin/perl-w.
# ggh - поиск данных в журнале netscape
$USAGE = <<EO_COMPLAINT;
usage: $0 [-database dbfilename] [-help]
          [-epochtime | -localtime | -gmtime]
                                        ...
EO_COMPLAINT

use Getopt::Long;

($opt_database, $opt_epochtime, $opt_localtime,
 $opt_gmtime,                   $opt_help,
 $pattern,                                  )      = (0) x 7;

usage() unless GetOptions qw{ database=s

                              epochtime localtime gmtime
                              help
                            };

if ($opt_help) { print $USAGE; exit; }

usage("only one of localtime, gmtime, and epochtime allowed")
    if $opt_localtime + $opt_gmtime + $opt_epochtime > 1;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $pattern =
} elsif (@ARGV && $ARGV[0] !^ m(://)) {
    $pattern = shift;
>

usage("can't mix URLs and explicit patterns")
    if $pattern && @ARGV;

if ($pattern && !eval { '' =^ /$pattern/; 1 } ) {
    $@ =' s/ at \w+ line \d+\.//;
    die "$0: bad pattern $@";
>

        DB_File; DB_File->import();    Отложить загрузку до выполнения
$| = 1;                                #·Для перенаправления данных

$dotdir  = $ENV{HOME}    || $ENV{LOGNAME};
$HISTORY = $opt_database || "$dotdir/.netscape/history.db";

die "no netscape history dbase in $HISTORY: $!" unless -e $HISTORY;
die "can't dbmopen $HISTORY: $!" unless dbmopen %hist_db, $HISTORY, 0666;

tf Следующая строка - хак, поскольку программисты C,
# которые работали над этим, путали strlen и strlen+1.
И Так мне сказал jwz :-)
$add_nulls   = (ord(substr(each %hist_db, -1)) == 0);

" XXX: Сейчас следовало бы сбросить скалярные ключи, но
# не хочется тратить время на полный перебор,
# необходимый для связанных хэшей.
#   Лучше закрыть и открыть заново?

                "";
$byte_order  = "V";   fl   На PC не  понимают "N" (сетевой порядок)

if (@ARGV) {
            $href (@ARGV)
                                ($add_nulls && "\0");
        unless ($binary_time =
            warn "$0: No history entry for HREF $href\n";
            next;
        }
        $epoch_secs = unpack($byte_order, $binary_time);
        $stardate    = $opt_epochtime ? $epoch_secs
                                      : $opt_gmtime ? gmtime    $epoch_secs
                                                    : localtime $epoch_secs;
        print "$stardate $href\n";
    }
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 14.8 (продолжение)

```
} else {
    while (          $binary_time)        %hist_db
        chop $href if $add_nulls,
        #                    binary times are missing
        $binary_time = pack($byte_order, 0) unless $binary_time,
        $epoch_secs = unpack($byte_order, $binary_time),
        $stardate = $opt_epochtime  ?  $epoch_secs
                                  : $opt_gmtime ? gmtime    $epoch_secs
                                                localtime $epoch_secs,
        print "$stardate $href\n" un                    ! ~ /$pattern/o,
    }
}

sub usage {
    print STDERR "@_\n" if @_;
    die $USAGE,
}
```

Смотри также

Рецепт 6.17.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пользовательские интерфейсы 15

Потом разбились Окна — и тогда
Пропало все перед глазами...
Э. Дикинсон,
"Я слышала жужжанье
Мухи — когда я умерла"

Введение

Все, чем мы пользуемся — видеомагнитофоны, компьютеры, телефоны и даже книги, — имеет свой пользовательский интерфейс. Интерфейс есть и у наших программ. Какие аргументы должны передаваться вкомандной строке? Можно ли перетаскивать мышью файлы? Должны ли мы нажимать Enter после каждого ответа или программа читает входные данные по одному символу?

В этой главе мы не будем обсуждать *проектирование* пользовательского интерфейса — на эту тему и так написано множество книг. Вместо этого мы сосредоточим внимание на *реализации* интерфейсов — передаче аргументов в командной строке, посимвольному чтению с клавиатуры, записи в любое место экрана и программированию графического интерфейса.

Простейшим пользовательским интерфейсом обычно считается так называемый *консольный интерфейс*. Программы с консольным интерфейсом читают целые строки и выводят данные также в виде целых строк. Примером консольного интерфейса являются фильтры (например, *grep*) и утилиты (например, *mail*). В этой главе консольные интерфейсы почти не рассматриваются, поскольку им уделено достаточно внимания в остальных частях книги.

Более сложный вариант — так называемый *полноэкранный интерфейс*. Им обладают такие программы, как *elm* или *lynx*. Они читают по одному символу и могут выводить данные в любой позиции экрана. Этот тип интерфейса рассматривается в рецептах 15.4, 15.6, 15.9—15.11.

Последнюю категорию интерфейсов составляют графические пользовательские интерфейсы (GUI, Graphic User Interface). Программы с графическим интерфейсом работают не только с отдельными символами, но и с отдельными пикселями. В графических интерфейсах часто используется метафора окна — программа создает окна, отображаемые на пользовательском устройстве вывода.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Окна заполняются элементами (widgets) — например, полосами прокрутки или кнопками. Netscape Navigator, как и ваш менеджер окон, обладает полноценным графическим интерфейсом. Perl позволяет работать со многими инструментальными пакетами GUI, однако мы ограничимся пакетом Tk, поскольку он является самым распространенным и переносимым. См. рецепты 15.14, 15.15 и 15.19.

Не путайте пользовательский интерфейс программы со средой, в которой она работает. Среда определяет тип запускаемых программ. Скажем, при регистрации на терминале с полноэкранным вводом/выводом вы сможете работать с консольными приложениями, но не с графическими программами. Давайте кратко рассмотрим различные среды.

Некоторые из них позволяют работать лишь с программами, обладающими чисто консольным интерфейсом. Упрощенный интерфейс позволяет объединять их в качестве многократно используемых компонентов больших сценариев; такое объединение открывает чрезвычайно широкие возможности. Консольные программы прекрасно подходят для автоматизации **работы,** поскольку они не зависят от клавиатуры или экрана. Они используют лишь STDIN и **STDOUT,** да и то не всегда. Обычно эти программы обладают наилучшей переносимостью, поскольку они ограничиваются базовым вводом/выводом, поддерживаемым практически в любой системе.

Типичный рабочий сеанс, в котором участвует терминал с экраном и клавиатурой, позволяет работать как с консольными, так и полноэкранными интерфейсами. Программа с полноэкранным интерфейсом взаимодействует с драйвером терминала и хорошо знает, как вывести данные в любую позицию экрана. Для автоматизации работы таких программ создается **псевдотерминал,** с которым взаимодействует программа (см. рецепт 15.13).

Наконец, некоторые оконные системы позволяют выполнять как консольные и полноэкранные, так и графические программы. Например, можно запустить (консольная программа) из *vi* (полноэкранная программа) в окне *xterm* (графическая программа, работающая в оконной среде). Графические программы автоматизируются труднее всего, если только они не обладают альтернативным интерфейсом на основе вызова удаленных процедур (RPC).

Существуют специальные инструментальные пакеты для программирования в полноэкранных и графических средах. Такие пакеты (*curses* для полноэкранных программ; Tk — для графических) улучшают переносимость, поскольку программа не зависит от особенностей конкретной системы. Например, программа, написанная с применением **curses,** работает практически на **любом** терминале, При этом пользователю не приходится думать о том, какие служебные команды используются при вводе/выводе. **Tk-программа** будет работать и в UNIX и в Windows — при условии, что в ней не используются специфические функции операционной системы.

Существуют и другие варианты взаимодействия с пользователем, в первую очередь — через Web. Программирование для Web подробно рассматривается в главах 19 и 20, поэтому в этой главе мы не будем задерживаться на этой теме.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

i5.1. Лексический анализ аргументов

Проблема

Вы хотите, чтобы пользователь могу повлиять на поведение вашей программы, передавая аргументы в командной строке. Например, параметр -v часто управляет степенью детализации вывода.

Решение

Передача односимвольных параметров командной строки обеспечивается стандартным модулем Getopt::Std:

```
use Getopt::Std;

# -v ARG, -D ARG, -o ARG, присваивает $opt_v, $opt_D, $opt_o
getopt("vDo");
# -v ARG, -D ARG, -o ARG, присваивает $args{v}, $args{D}, $args{o}
getopt("vDo", \%args);

getopts("vDo:");          # -v, -D, -oARG, присваивает
                           # $opt_v, $opt_D, $opt_o
getopts("vDo:", \%args); и -v, -D, -o ARG, присваивает
                           " sets $args{v}, $args{D}, $args{o}
```

Или воспользуйтесь модулем Getopt::Long, чтобы работать с именованными аргументами:

```
use Getopt::Long;

GetOptions( "verbose"   => \$verbose,     # --verbose
            "Debug"     => \$debug,       # --Debug
            "output=s" => \$output );     # --output=string
```

Комментарий

Многие классические программы (такие, как *ls* и *rm*) получают односимвольные параметры (также называемые флагами или ключами командной строки) — н например, -1 или -r. В командных строках *ls -l* и *rm -r* аргумент является логической величиной: он либо присутствует, либо нет. Иначе дело обстоит в командной строке *gcc -o compiledfilesource.c*, где *compiledfile*— значение, ассоциированное с параметром **-o**. Логические параметры можно объединять в любом порядке; например, строка:

```
% rm -r -f /tmp/testdir
```

эквивалентна следующей:

```
% rm -rf /tmp/testdir
```

Модуль Getopt::Std, входящий в стандартную поставку Perl, анализирует эти традиционные типы параметров. Его функция getopt получает одну строку, где каждый символ соответствует некоторому параметру, анализирует аргументы командной строки в массиве @ARGV и для каждого параметра присваивает значение глобальной переменной. Например, значение параметра -D будет храниться в пе-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ременной $opt_0. Параметры, анализируемые с помощью getopt, не являются логическими (то есть имеют конкретное значение).

Модуль Getopt::Std также содержит функцию getopts, которая позволяет указать, является ли параметр логическим или принимает значение. Параметры со значениями (такие, как параметр -o программы *gcc*) обозначаются двоеточием, как это сделано в следующем фрагменте:

```
use Getopt::Std;
getopts("o:");
if ($opt_o) {
    print "Writing output to $opt_o";
```

Обе функции, getopt и getopts, могут получать второй аргумент — ссылку на хэш. При наличии второго аргумента значения вместо переменных $opt_X сохраняются в $hash{X}:

```
use Getopt::Std;

%option = ();
getopts("Do:", \%option);

if ($option{D}) {
    print "Debugging mode enabled.\n";
}

# Если параметр -o не задан, направить результаты в "-".
й Открытие "-" для записи означает STDOUT
$option{o} = "-" unless defined $option{o};

print "Writing output to file $option{o}\n" unless $option{o> eq "-";
open(STOOUT, "> $option{o}")
     or die "Can't open $option{o} for output: $!\n";
```

Некоторые параметры программы могут задаваться целыми словами вместо отдельных символов. Обычно они имеют специальный префикс — двойной дефис:

```
% gnutar --extract --file latest.tar
```

Значение параметра -file также может быть задано с помощью знака равенства:

```
% gnutar --extract --file=latest,tar
```

Функция GetOptions модуля Getopt::Long анализирует эту категорию параметров. Она получает хэш, ключи которого определяют параметры, а значения представляют собой ссылки на скалярные переменные:

```
use Getopt::Long;

GetOptions( "extract" => \$extract,
            "file=s" => \$file );

if ($extract) {
    print "I'm extracting.\n";
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
die "I wish I had a file" unless defined $file;
print "Working on the file $file\n";
```

Если ключ хэша содержит имя параметра, этот параметр является логическим. Соответствующей переменной присваивается false, если параметр не задан, или 1 в противном случае. Getopt::Long не ограничивается логическими параметрами и значениями Getopt::Std. Возможны следующие описания параметров:

Описание	Значение	Комментарий
option	Нет	Задается в виде --option или не задается вообще
option!	Нет	Может задаваться в виде --option или --nooption
option=s	Да	Обязательный строковый параметр: --option=somestring
option:s	Да	Необязательный строковый параметр: --option или --option=somestring
option=i	Да	Обязательный целый параметр: —option=35
option:i	Да	Необязательный целый параметр: --option или --option=35
option=f	Да	Обязательный вещественный параметр: --option=3.141
option:f	Да	Необязательный вещественный параметр: --option или -- option=3.141

Смотри также

Документация по стандартным модулям getopt::Long и Getopt::Std; примеры ручного анализа аргументов встречаются в рецептах **1.5, 1.17,** 6.22, **7.7, 8.19** и **15.12.**

15.2. Проверка интерактивного режима

Проблема

Требуется узнать, была ли ваша программа запущена в интерактивном режиме или нет. Например, запуск пользователем из командного интерпретатора является интерактивным, а запуск из *cron* - нет.

Решение

Воспользуйтесь оператором -t для проверки STDIN и STDOUT:

```
sub I_am_interactive {
            STDIN && -t STDOUT;
}
```

В POSIX-совместимых системах проверяются группы процессов:

```
use POSIX qw/getpgrp tcgetpgrp/;

sub I_am_interactive {
    local *TTY;  # local file handle
    open(TTY, "/dev/tty") or die "can't open /dev/tty: $!";
    my $tpgrp = tcgetpgrp(fileno(TTY));
    my $pgrp  = getpgrp();
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    close TTY,
            ($tpgrp == $pgrp);
}
```

Комментарий

Оператор -t сообщает, соответствует ли файловый манипулятор или файл терминальному устройству (tty); такие устройства являются признаком интерактивного использования. Проверка сообщит лишь о том, была ли ваша программа перенаправлена. Если программа запущена из командного интерпретатора, при перенаправлении STDIN и STDOUT первая версия `I_am_interactive` возвращает false. При запуске из *cron* `I_am_interactive` также возвращает false.

Второй вариант проверки сообщает, находится ли терминал в монопольном распоряжении программы. Программа, чей ввод и вывод был перенаправлен, все равно при желании может управлять своим терминалом, поэтому POSIX-версия I_am_interactive возвращает `true`. Программа, запущенная из `cron`, не имеет собственного терминала, поэтому I_am_interactive возвратит false.

Какой бы вариант I_am_interactive вы ни выбрали, он используется следующим образом:

```
while (1) {
    if (I_am_interactive()) {
        print "Prompt. ",
    }
    $line = <STDIN>;
    last unless defined $line,
    # Обработать $line
}
```

Или более наглядно:

```
sub prompt < print "Prompt: ' if I_am_interactive() }
for (prompt(), $line = <STDIN>; prompt()) {
    # Обработать $line
}
```

Смотри также

Документация по стандартному модулю POSIX. Оператор проверки файлов -t описан в *perlop*(1).

15.3. Очистка экрана

Проблема

Требуется очистить экран.

Решение

Воспользуйтесь модулем Term::Cap для посылки нужной последовательности символов. Скорость вывода терминала можно определить с помощью модуля

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

POSIX::Termios (или можно предположить **9600** бит/с). Ошибки, возникающие при работе с POSIX::Termios, перехватываются с помощью eval:

```
use Term Cap,

$OSPEED = 9600,
eval {

    my $termios = POSIX Termios->new(),
    $termios->getattr,
    $OSPEED = $termios->getospeed,
},

$terminal = Term Cap->Tgetent({OSPEED=>$OSPEED}),
$terminal->Tputs( cl , 1, STDOUT),
```

Или выполните команду clear:

```
system( clear ),
```

!

Комментарий

Если вам приходится часто очищать экран, кэшируйте возвращаемое значение Term::Cap или команды clear:

```
$clear = $terminal->Tputs( cl ),
$clear = clear ,
```

Это позволит очистить экран сто раз подряд безстократного выполнения clear:

```
print $clear,
```

Смотри также

Man-страницы *clear*(1) и *termcap*(1)(если они есть); документация по стандартному модулю Term::Cap; документация по модулю Term::Lib с CPAN.

15.4. Определение размера терминала или окна

Проблема

Требуется определить размер терминала или окна. Например, **вы** хотите отформатировать текст так, чтобы он не выходил за правую границу экрана.

Решение

Воспользуйтесь функцией `ioctl` (см. рецепт 12.14) или модулем Term::ReadKey с CPAN:

```
use Term ReadKey,

($wchar, $hchar, $wpixels, $hpixels) = GetTerminalSize(),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Функция GetTerminalSize возвращает четыре элемента: ширину и высоту в символах, а также ширину и высоту в пикселях. Если операция не поддерживается для устройства вывода (например, если вывод был направлен в файл), возвращается пустой список.

Следующий фрагмент строит графическое представление @values приусловии, что среди элементов нет ни одного отрицательного:

```
use Term.·ReadKey;

($width) = GetTerminalSize(),
die "You must have at least 10 characters' unless $width >= 10,

$max = 0;
foreach (@values) <
    $max = $_ if $max < $_,
}

$ratio = ($width-10)/$max,          # Символов на единицу
        (@values)
    printf("%8.1f %s\n", $_, "*" x ($ratio*$_)),
}
```

Смотри также

Документация по модулю Term::ReadKey с CPAN; рецепт **12.14.**

15.5. Изменение цвета текста

Проблема

Вы хотите выводить на экране символы разных цветов. Например, цвет может использоваться для выделения текущего режима или сообщения **об** ошибке.

Решение

Воспользуйтесь модулем Term::ANSIColor с CPAN для передачи терминалу последовательностей изменения цвета **ANSI:**

```
use Term··ANSIColor,

print color("red"), "Danger, Will Robinson!\n", color("reset"),
print "This is just normal text \n";
print colored("<BLINK>Do you hurt yet?</BLINK>", 'blink");
```

Или воспользуйтесь вспомогательными функциями модуля Term::ANSIColor:

```
use Term· ANSIColor qw(·constants),

print RED, "Danger, Will Robinson!\n", RESET;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Модуль Term::ANSIColor готовит служебные последовательности, которые опознаются некоторыми (хотя далеко не всеми) терминалами. Например, в *color-xterm* этот рецепт работает. В обычной программе *xterm* или на терминале *vt100* он работать не будет.

Существуют два варианта использования модуля: либо с экспортированными функциями color($АТРИБУТ) и $АТРИБУТ), либо с вспомогательными функциями (такими, как BOLD, BLUE и RESET).

Атрибут может представлять собой комбинацию цветов и модификаторов. Цвет символов принимает следующие значения: black, blue, magenta (черный, красный, зеленый, желтый, синий, малиновый) Цвет фона принимает значения on_black, on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan и on_white (черный, красный, зеленый, желтый, синий, малиновый, голубой и белый). Допускаются следующие модификаторы: clear, underline, blink, concealed (очистка, сброс, **жирный**, подчеркивание, подчеркивание, мерцание, инверсия и скрытый). Clear и являются синонимами (как и underline с сбросе восстанавливаются цвета, действовавшие при запуске программы, а при выводе скрытого текста цвет символов совпадает с цветом фона.

Атрибуты могут объединяться:

```
print color( on_black ), venom lack\n ,
print color( on_yellow ), kill that fellow\n ,

print color( on_cyan blink ) garish!\n ,
print
```

Этот фрагмент можно было записать в виде:

```
print venom lack\n on_black ),
print kill fellow\n on_yellow )

print garish!\n , , on_cyan , blink ),
```

или:

```
use Term ANSIColor qw( constants),

print BLACK ON_WHITE, black on white\n
print WHITE, ON_BLACK, white on black\n ,
print GREEN, ON_CYAN, BLINK, garish!\n ,
print RESET,
```

где BLACK — функция, экспортированная из Term::ANSIColor.

Не забывайте вызвать print RESET или color( ) в конце программы, если вызов распространяется на весь текст. Если этого не сделать, ваш терминал будет раскрашен весьма экзотическим образом. Сброс даже можно включить в блок END:

```
END { print color("reset") )
```

чтобы при завершении программы цвета были гарантированно сброшены.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Атрибуты, распространяющиеся на несколько строк текста, могут привести в замешательство некоторые программы или устройства. Если у вас возникнут затруднения, либо вручную установите атрибуты в начале каждой строки, либо используйте предварительно присвоив переменной $Term::ANSIColor::EACHLINE разделитель строк:

```
$Term::ANSIColor::EACHLINE = $/;
print                    ON_WHITE, BOLD, BLINK);
This way
each line
has its own
attribute set.
EOF
```

Смотри также

Документация по модулю Term::AnsiColor с CPAN.

15.6. Чтение с клавиатуры

Проблема

Требуется прочитать с клавиатуры один символ. Например, на экран выведено меню с клавишами ускоренного вызова, и вы не хотите, чтобы пользователь нажимал клавишу Enter при выборе команды.

Решение

Воспользуйтесь модулем Term::ReadKey с CPAN, чтобы перевести терминал в режим прочитать символы из STDIN и затем вернуть терминал в обычный режим:

```
use Term::ReadKey;

ReadMode
$key = ReadKey(0);
ReadMode 'normal';
```

Комментарий

Модуль Term::ReadKey может переводить терминал в разные режимы. лишь один из них. В этом режиме каждый символ становится доступным для программы сразу же после ввода (см. пример 15.1). Кроме того, в нем происходит эхо-вывод символов; пример режима без эхо-вывода рассматривается в рецепте 15.10.

Пример 15.1. sascii

```
#!/usr/bin/perl -w
# sascii - Вывод ASCII-кодов для нажимаемых клавиш

use Term::ReadKey;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
ReadMode('cbreak');
print                    their ASCII values   Use Ctrl-C to quit \n',

while (1) {
    $char = ReadKey(0),
    last unless defined $char;
    printf(" Decimal. %d\tHex' %x\n", ord($char), ord($char));
}

ReadMode('normal');
```

Режим ешает драйверу терминала интерпретировать символы конца файла и управления. Если вы хотите, чтобы ваша программа могла прочитать комбинации Ctrl+C (обычно посылает процессу SIGINT) или Ctrl+D (признак конца файла в UNIX), используйте режим raw.

Вызов ReadKey с нулевым аргументом означает, что мы хотим выполнить нормальное чтение функцией getc. При отсутствии входных данных программа ожидает их появления. Кроме того, можно передать аргумент -1 (неблокирующее чтение) или положительное число, которое определяет тайм-аут (продолжительность ожидания в целых секундах; дробные значения секунд не допускаются). Неблокирующее чтение и чтение с тайм-аутом возвращает либо undef приотсутствии входных данных, либо строку нулевой длины при достижении конца файла.

Последние версии Term::ReadKey также включают ограниченную поддержку систем, не входящих в семейство UNIX.

Смотри также

Документация по модулю Term::ReadKey с CPAN; рецепты 15.8—15.9. Функции getc и sysread описаны в *perlfunc*(1).

15.7. Предупреждающие сигналы

Проблема

Требуется выдать предупреждающий сигнал на терминале пользователя.

Решение

Воспользуйтесь символом "\a" для выдачи звукового сигнала:

```
print '\aWake up!\n',
```

другой вариант — воспользуйтесь средством терминала "vb" для выдачи визуального сигнала:

```
use Term  Cap;

$OSPEED = 9600;
eval {
            POSIX,
    my $termios = POSIX  Termios->new();
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $termios->getattr,
    $OSPEED = $termios->getospeed,
},

$terminal = Term Cap->Tgetent({OSPEED=>$OSPEED}),
$vb =    ,
eval <
    $terminal->Trequire( vb ),
    $vb = $terminal->Tputs( vb ,  1),
},

print $vb,                                  я Визуальный сигнал
```

Комментарий

Служебныйсимвол \a — то же самое, что и "\cG ,'\007' и "\x07'. Все эти обозначения относятся к символу ASCII BEL, который выдает на терминал противный звонок. Вам не приходилось бывать в переполненном терминальном классе в конце семестра, когда десятки новичков одновременно пытаются перевести *vi* в режим ввода? От этой какофонии можно сойти с ума. Чтобы не злить окружающих, можно использовать визуальные сигналы. Идея проста: терминал должен показывать, а не звучать (по крайней мере, не в многолюдных помещениях). Некоторые терминалы вместо звукового сигнала позволяют на короткое время поменять цвет символов с цветом фона, чтобы мерцание привлекло внимание пользователя.

Визуальные сигналы поддерживаются не всеми терминалами, поэтому мы включили их вызов в eval. Если визуальный сигнал не поддерживается, `Trequire` инициирует `die`, при этом переменная $vb останется равной "'. В противном случае переменной $vb присваивается служебная последовательность для выдачи сигнала.

Более разумный подход к выдаче сигналов реализован в графических терминальных системах (таких, как *xterm*). Многие из них позволяют включить визуальные сигналы на уровне внешнего приложения, чтобы программа, тупо выводящая `chr(7)`, была менее шумной.

Смотри также

Раздел "Quote и Quote-like Operators" в *perlop*(1);документация по стандартному модулю Term::Cap.

15.8. Использование termios

Проблема

Вы хотите напрямую работать с характеристиками своего терминала.

Решение

Воспользуйтесь интерфейсом **POSIX** `termios`.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Представьте себе богатые возможности команды *stty* — можно задать все, от служебных символов до управляющих комбинаций и перевода строки. Стандартный модуль POSIX обеспечивает прямой доступ к низкоуровневому терминальному интерфейсу и позволяет реализовать *stty*-подобные возможности в вашей программе.

Программа из примера 152 показывает, какие управляющие символы используются вашим терминалом для стирания в предыдущей и текущей позиции курсора (вероятно, это клавиши "забой" и Ctrl+U). Затем она присваивает им исторические значения, # и @, и предлагает ввести какой-нибудь текст. В конце своей работы программа восстанавливает исходные значения управляющих символов.

Пример 15.2. demo

```
#!/usr/bin/perl -w
# Демонстрация работы с интерфейсом POSIX termios

use POSIX qw( termios_h),

$term = POSIX  Termios->new,
$term->getattr(fileno(STDIN)),

$erase = $term->getcc(VERASE),
$kill = $term->getcc(VKILL),
printf  Erase is character %d, %s\n ,  $erase, uncontrol(chr($erase)),
printf  Kill is character %d, %s\n ,  $kill  uncontrol(chr($kill))

$term->setcc(VERASE, ord( # )),
$term->setcc(VKILL, ord( @ ))
$term->setattr(1, TCSANOW),

print( erase is #, kill is @, type something  '),
$line = <STDIN>,
print  You typed  $line ,

$term->setcc(VERASE, $erase),
$term->setcc(VKILL, $kill),
$term->setattr(1, TCSANOW),

sub uncontrol {
    local $_ = shift,
    s/([\200-\377])/sprintf( M-%c ,ord($1) & 0177)/eg,
    s/([\0-\37\177])/sprintf( ^%c ,ord($1) ^ 0100)/eg,

}
```

Следующий модуль, HotKey, реализует функцию readkey на Perl. Он не обладает никакими преимуществами по сравнению с Term::ReadKey, а всего лишь показывает интерфейс termios в действии:

```
# HotKey pm
package HotKey,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
@ISA = qw(Exporter)
@EXPORT                cooked

use strict,
use POSIX qw( termios_h),
my ($term, $oterm, $echo  $noecho, $fd_stdin),

$fd_stdin = fileno(STDIN),
$term     = POSIX  Termios->new(),
$term->getattr($fd_stdin),
$oterm      = $term->getlflag(),

$echo      = ECHO | ECHOK | ICANON,
$noecho    = $oterm & ~$echo,

           {
     $term->setlflag($noecho),  8 Эхо-вывод не нужен
     $term->setcc(VTIME, 1),
     $term->setattr($fd_stdin, TCSANOW),
>

sub cooked {
     $term->setlflag($oterm),
     $term->setcc(VTIME, 0),
     $term->setattr($fd_stdin, TCSANOW),
}


     my $key =

                    $key,
     cooked()
            $key
}

END { cooked() >

1,
```

Смотри также

Документация по стандартному модулю POSIX; рецепты 15.6; 15.9.

15.9. Проверка наличия входных данных

Проблема

Требуется узнать, имеются ли необработанные входные **данные,** не выполняя их фактического чтения.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь модулем Term::ReadKey от CPAN и попытайтесь прочитать символ в неблокирующем режиме, для этого используется аргумент -1:

```
use Term  ReadKey,

ReadMode (

1f (defined ($char = ReadKey(-1)) ) {
    й Имеется необработанный ввод $char
} else {
    " Необработанного ввода нет
}

ReadMode ( normal ),                " Восстановить нормальные
                                   # параметры терминала
```

Комментарий

Аргумент -1' функции ReadKey означает неблокирующее чтение символа. Если символа нет, ReadKey возвращает undef.

Смотри также

Документация по модулю Term::ReadKey с CPAN; рецепт 15.6.

15.10. Ввод пароля

Проблема

Требуется прочитать данные с клавиатуры без эхо-вывода не экране. Например, вы хотите прочитать пароль так, как это делает *passwd*, то есть без отображения пароля пользователя.

Решение

Воспользуйтесь модулем Term::ReadKey с CPAN, установите режим ввода noecho, после чего воспользуйтесь функцией ReadLine:

```
use Term  ReadKey,

ReadMode  noecho ,
$password = ReadLine 0,
```

Комментарий

Пример 15.3 показывает, как организовать проверку пароля пользователя. Если в вашей системе используются скрытые пароли, getpwuid вернет зашифрованный пароль лишь привилегированному пользователю. Всем остальным в соответствующем поле базы данных возвращается лишь *, что совершенно бесполезно при проверке пароля.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 15.3. checkuser

```
#!/usr/bin/perl -w
# checkuser - чтение и проверка пароля пользователя
use Term::ReadKey;

print "Enter your password: ";
ReadMode 'noecho';
$password = ReadLine 0;
chomp $password;
ReadMode 'normal';

print "\n",

($username, $encrypted) = ( getpwuid $< )[0,1];

if (crypt($password, $encrypted) ne $encrypted) {
    die "You are not $username\n";
} else <
    print "Welcome, $username\n";
}
```

Смотри **также**

Документация по модулю Term::ReadKey с CPAN; man-страницы *crypt*(3) и *passwd*(5) вашей системы (если есть). Функции crypt и getpwuid описаны в *perlfunc*(1).

15.11. Редактирование входных данных

Проблема

Вы хотите, чтобы пользователь мог отредактировать строку перед тем, как отсылать ее вам для чтения.

Решение

Воспользуйтесь стандартной библиотекой Term::ReadLine в сочетании с модулем Term::ReadLine::Gnu с CPAN:

```
use Term::ReadLine;

$term = Term::ReadLine->new("APP DESCRIPTION");
$OUT = $term->OUT || *STDOUT;

$term->addhistory($fake_line);
$line = $term->readline(PROMPT);

print $OUT "Any program output\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Программа из примера 15.4 работает как простейший командный интерпретатор Она читает строку и передает ее для выполнения. Метод readline читает строку с терминала с поддержкой редактирования и вызова истории команд. Вводимая пользователем строка автоматически включается в историю команд.

Пример 15.4. vbsh

```
#!/usr/bin/perl -w
# vbsh -  очень плохой  командный интерпретатор
use strict,

use Term  ReadLine,
use POSIX qw( sys_wait_h),

my $term = Term  ReadLine->new( Simple Shell ),
my $OUT = $term->OUT() || *STDOUT
my $cmd,

while (defined ($cmd = $term->readline( $  ) )) {
    my @output =  $cmd ,
    my $exit_value  = $? " 8,
    my $signal_num  = $? & 127,
    my $dumped_core = $? & 128,
    printf $OUT  Program terminated with status %d from signal %d%s\n ,
           $exit_value, $signal_num,
           $dumped_core ^          dumped)
    print @output,
    $term->addhistory($seed_line),
}
```

Чтобы занести в историю команд свою строку, воспользуйтесь методом addhistory:

```
$term->addhistory($seed_line),
```

В историю нельзя заносить больше одной строки за раз. Удаление строк из истории команд выполняется методом remove_history, которому передается индекс в списке истории: 0 соответствует первому (самому старому) элементу, 1 — второму и т. д. до самых последних строк.

```
$term->remove_history($line_number)
```

Для получения списка истории команд используется метод GetHistory:

```
@history = $term->GetHistory,
```

Смотри также

Документация по стандартным модулям Term::ReadLine и Term::ReadLine::Gnu с CPAN.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

15.12. Управление экраном

Проблема

Вы хотите выделять символы повышенной интенсивностью, перехватывать нажатия специальных клавиш или выводить полноэкранные меню, но не желаете беспокоиться о том, на каком устройстве вывода работает пользователь.

Решение

Воспользуйтесь модулем Curses с CPAN, который использует библиотеку *curses*(3) вашей системы.

Комментарий

Библиотека *curses* обеспечивает простое, эффективное и аппаратно-независимое выполнение полноэкранных операций. С его помощью можно писать высокоуровневый код вывода данных на логическом экране по символам или по строкам. Чтобы результаты вывода появились на экране, вызовите функцию `refresh`. Вывод, сгенерированный библиотекой, описывает только изменения виртуального экрана с момента последнего вызова особенно существенно для медленных подключений.

Работа с модулем Curses демонстрируется программой из примера 15.5. Вызовите ее с аргументами, описывающими командную строку запускаемой программы:

```
      ps aux
% rep netstat
      -2.5 lpq
```

Сценарий в цикле вызывает команду и выводит ее данные на экран, обновляя лишь ту часть, которая изменилась с момента предыдущего запуска. Такой вариант наиболее эффективен при малых изменениях между запусками. В правом нижнем углу экрана выводится текущая дата в инвертированном изображении.

По умолчанию ожидает 10 секунд перед повторным запуском команды. Чтобы изменить период задержки, передайте нужное количество секунд (допускается дробное число) в качестве аргумента, как это было сделано выше при вызове *lpq*. Кроме того, нажатие любой клавиши во время ожидания приводит к немедленному выполнению команды.

Пример 15.5. rep

```
#!/usr/bin/perl -w
# rep - циклическое выполнение команды
use strict;
use Curses;

my $timeout = 10;
if (@ARGV && $ARGV[0] =~ /^-(\d+\.?\d*)$/) {
    $timeout = $1;
    shift;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

die usage $0 [ -timeout ] cmd args\n unless @ARGV,

initscr(),    tt    Инициализировать экран
noecho(),

nodelay(1),        # Чтобы функция getch() выполнялась без блокировки

$SIG<INT} = sub { done( Ouch! ) },
sub done { endwin(), print @_\n , exit, }

while (1) {
    while ((my $key = getch()) ne ERR) {    # Возможен ввод
        done( See ya ) if $key eq q         # нескольких символов
    }
    my @data =  (@ARGV) 2>&1 ,              # Вывод+ошибки
    for (my $i = 0, $1 < $LINES, $i++) {
        addstr($i, 0, $data[$i] ||     x$COLS),
    }

    standout(),
    addstr($LINES-1, $COLS - 24, scalar localtime),
    standend(),

    move(0,0),
    refresh(),                              # Обновить экран

    my ($in, $out) = ( , ),
    vec($in,fileno(STDIN),1) = 1,           # Искать символ в stdin
    select($out = $in,undef,undef,$timeout),# Ожидание
}
```

С помощью Curses можно узнать, когда пользователь нажал кла вишу со стрел-)й или служебную клавишу (например, Home или Insert). Обычно это вызывает **труднения,** поскольку эти клавиши **кодируются** несколькими байтами. С Curses :е просто:

```
keypad(1),                  # Включить режим ваода
$key = getch(),             # с цифровой клавиатуры
if ($key eq k      ||       # Режим vi
    $key eq \cP    ||       # Режим emacs
    $key eq KEY_UP)         # Стрелка
{
    # Обработать клавишу
}
```

Другие функции Curses позволяют читать текст в определенной позиции экрана, управлять выделением символов и даже работать в нескольких окнах.

Модуль perlmenu, также хранящийся на CPAN, построен на базе низкоуровневого модуля **Curses.** Он обеспечивает высокоуровневые операции с меню и экранными формами. Приведем пример экранной формы из поставки perlmenu:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                    Template Entry Demonstration

        Data Example                                  Record #
Name    [                                                      ]
Addr    [                                                      ]
City    [                    ]        State  [__]      Zip  [\\\\\]

Phone   (\\\) \\\-\\\\                          Password  [         ]

Enter all information available
Edit fields with left/right arrow heys or  delete
Switch fields with  Tab  or up/down arrow keys
Indicate completion by          Return
                 with Control-L
Abort this demo      with  Control-X
```

Пользователь вводит текст в соответствующих полях. Обычный текст обозначается символами подчеркивания, числовые данные — символами \, а неотображаемые данные — символами ^.Такие обозначения напоминают форматы Perl, за исключением того, что формы предназначены для вывода, а не для ввода данных.

Смотри **также**

- Man-страница *curses*(3) вашей системы (если есть); документация по модулям Curses и perlmenu с CPAN; раздел "Formats" *perlform*(1);рецепт 3.10.

15.13. Управление другой программой с помощью Expect

Проблема

Вы хотите автоматизировать процесс взаимодействия с полноэкранной программой, которая работает с терминалом, не ограничиваясь STDIN и STDOUT.

Решение

Воспользуйтесь модулем Expect с CPAN:

```
use Expect,

$command = Expect->spawn( program to run )
    or die  Couldn t start program  $'\n ,

# Запретить вывод программы в STDOUT
$command->log_stdout(0),

# 10 секунд подождать появления  Password
unless ($command->expect(10, 'Password )) {
    # Тайм-аут
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
>

# 20 секунд подождать вывода текста, совпадающего с /[lL]ogin ?/
unless ($command->expect(20, -re => [lL]ogin ? )) {
    И Таймаут
>

# Бесконечно долго ждать появления invalid
unless ($command->expect(undef, invalid )) {
    # Произошла ошибка, вероятно, работа программы нарушена
}

# Послать программе Hello, world и перевод строки
print $command Hello, world\n ,

# Если программа завершается сама, предоставить ей такую возможность
$command->soft_close(),

# Если программа должна быть закрыта извне, завершить ее
$command->hard_close(),
```

Комментарий

Для работы модуля Expect необходимы два других модуля с CPAN: IO::Pty и IO:Stty. Expect создает псевдотерминал для взаимодействия с программами, которые непременно должны общаться с драйвером терминального устройства. Такая возможность часто используется для изменения пароля в программе *passwd*. К числу других программ, для которых также необходим настоящий терминал, принадлежат *telnet* (модуль Net::Telnet из рецепта 18.6 более функционален и обладает улучшенной переносимостью) и *ftp*.

Запустите нужную программу с помощью Expec t->spawn, передайте ей имя программы и аргументы — либо в виде одной строки, либо в виде списка. Expect запускает программу и возвращает либо представляющий ее объект, либо undef, если запустить программу не удалось.

Для ожидания вывода программой конкретной строки применяется метод expect. Его первый аргумент равен либо числу секунд, в течение которых ожидается вывод строки, либо undef для бесконечного ожидания. Ожидаемая строка является вторым аргументом expect. Чтобы определить ее с помощью регулярного выражения, передайте в качестве второго аргумента , третьего — строку с шаблоном. Затем можно передать другие строки или шаблоны

```
$which = $command->expect(30, invalid , succes , error , boom ),
if ($which) {
    # Найдена одна из указанных строк
}
```

В скалярном контексте expect возвращает номер аргумента, для которого произошло совпадение. В предыдущем примере expect вернет 1 при выдаче программой строки invalid , 2 - привыводе 'succes и т. д. Если ни одна строка или шаблон не совпали, expect возвращает false.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В списковом контексте expect возвращает список из пяти элементов. Первый элемент определяет номер совпавшей строки или шаблона (идентично возвращаемому значению в скалярном контексте). Второй элемент — строка с описанием причины возврата из expect. Приотсутствии ошибок второй аргумент равен undef. Возможные варианты ошибок: "1:TIMEOUT", "2:EOF", "3. spawn id(..) died" и "4:. ." (смысл этих сообщений описан в *Expect*(3)).Третий элемент в возвращаемом списке expect равен совпавшей строке. Четвертый элемент — текст до совпадения, а пятый — текст после совпадения.

Передача данных программе, находящейся под управлением Expect, сводится к простейшему вызову print. Единственная трудность состоит в том, что терминалы, устройства и сокеты отличаются по тем последовательностям, которые они передают и принимают в качестве разделителя строк, — мы покинули убежище стандартной библиотеки ввода/вывода C, поэтому автоматическое преобразование в "\n" не происходит. Рекомендуем начать с "\r"; если не получится, попробуйте "\n" и "\r\n".

После завершения работы с запущенной программой у вас есть три возможности. Во-первых, можно продолжить работу с главной программой; вероятно, запущенная программа будет принудительно завершена по завершении главной программы. Однако в этом случае плодятся лишние процессы. Во-вторых, если запущенная программа должна нормально завершиться после вывода всех данных или по некоторому внешнему условию (как, например, *telnet* при выходе из удаленного командного интерпретатора), вызовите метод soft_close. Если запущенная программа будет работать бесконечно (например, *tail -f*),вызовите метод hard_close; онуничтожает запущенный процесс.

Смотри также

Документация по модулям Expect, IO:Pty и IO:Stty от CPAN.

15.14. Создание меню с помощью Tk

Проблема

Требуется создать окно, в верхней части которого находится меню.

Решение

Воспользуйтесь элементами Tk Menubutton и Frame:

```
use Tk;

$main = MainWindow->new();

# Создать для меню горизонтальную область
# в верхней части окна
$menubar = $main->Frame(-relief          => 'raised',
                        -borderwidth     => 2)
                 ->pack (-anchor         => "nw ,
                        -fill            => "x");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Создать кнопку с надписью "File" для вызова меню.
$file_menu = $menubar->Menubutton(-text      => "File",
                                  -underline => 1)
                    ->pack      (-side      => "left" );
# Создать команды меню "File"
$file_menu->command(-label   => "Print",
                    -command => \&Print);
```

То же самое можно сделать намного проще, если воспользоваться сокращенной записью -menuitems:

```
$file_menu = $menubar->Menubutton(-text     => "File",
                                  -underline=> 1,
                                  -menuitems=> [
              [ Button => "Print",-command  =>\&Print  ],
               [ Button => "Save",-command  => \&Save   ] ])
                              ->pack(-side    => "left"),
```

Комментарий

Меню приложения можно рассматривать как совокупность четырех компонентов: области **(Frame)**, кнопок меню (Menubutton), меню **(Menus)** и команд меню (Menu Entries). Область представляет собой горизонтальную полосу в верхней части **окна**, в котором находится меню. Внутри области находится набор кнопок меню, открывающих различные меню: File, Edit, Format, Buffers и т. д. Когда пользователь щелкает на кнопке меню, на экране появляется соответствующее меню — вертикальный список команд.

В меню могут включаться *разделители* — горизонтальные линии, отделяющие один набор команд от другого.

С *командами* (например, Print в меню File) ассоциируются фрагменты кода. При выборе команды меню вызывается соответствующая функция. Обычно это делается так:

```
$file_menu->command(-label   => "Quit Immediately",
                    -command => sub { exit } );
```

С разделителями действия не связываются:

```
$file_menu->separator();
```

Команда-флажок может находиться в установленном (on) или сброшенном (off) состоянии, и с ней ассоциируется некоторая переменная. Если переменная находится в установленном состоянии, рядом с текстом команды-флажка стоит специальная пометка (маркер). Если переменная сброшена, маркер отсутствует. При выборе команды-флажка переменная переходит в противоположное состояние.

```
$options_menu->checkbutton(-label     =>                    File",
                           -variable  =>\$debug,
                           -onvalue   => 1,
                           -offvalue  => 0),
```

Группа *команд-переключателей* ассоциируется с одной переменной. В любой момент времени установленной может быть лишь одна команда-переключатель,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ассоциированная с переменной. При выборе команды-переключателя переменной присваивается ассоциированное значение:

```
$debug_menu->radiobutton(-label    =>  Level 1 ,
                         -variable => \$log_level,
                         -value    => 1)

$debug_menu->radiobutton(-label    =>  Level 2 ,
                         -variable => \$log_level,
                         -value    => 2),

$debug_menu->radiobutton(-label    =>  Level 3 ,
                         -variable => \$log_level,
                         -value    => 3),
```

Вложенные меню создаются с помощью *каскадных команд*. Например, в Netscape Navigator кнопка меню File содержит каскадную команду New, которая открывает подменю с несколькими вариантами. Создать каскадную команду сложнее, чем любую другую: вы должны создать каскадную команду, получить ассоциированное с ней новое меню и создать команды в этом меню.

```
# Шаг 1  создать каскадную команду меню
$format_menu->cascade          (-label    =>  Font ),

# Шаг 2  получить только что созданное меню
$font_menu = $format_menu->cget( -menu ),

# Шаг 3  заполнить это меню
$font_menu->radiobutton        (-label    =>  Courier
                                -variable => \$font_name,
                                -value    =>  courier ),
$font_menu->radiobutton        (-label    =>  Times Roman ,
                                -variable => \$font_name,
                                -value    =>  times ),
```

Отсоединяемый разделитель позволяет перемещать меню, в котором он находится. По умолчанию все кнопки меню и каскадные команды открывают меню, в верхней части которого находится отсоединяемый разделитель. Чтобы создать меню без него, воспользуйтесь параметром `-tearoff`:

```
$format_menu = $menubar->Menubutton(-text      =>  Format ,
                                    -underline => 1
                                    -tearoff   => 0)
                        ->pack,

$font_menu  = $format_menu->cascade(-label     =>  Font ,
                                    -tearoff   => 0),
```

Параметр `-menuitems` метода `Menubutton` представляет собой сокращенную форму для создания команд меню. В нем передается ссылка на массив с описаниями команд `Menubutton`. В свою очередь, каждая команда описывается анонимным мас-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

сивом. Первые два элемента массива команды определяют тип кнопки ("command", "radiobutton", "checkbutton", "cascade" или "tearoff") и название меню.

```
my $f = $menubar->Menubutton(-text => "File", -underline => 0,
                             -menuitems =>
    t
      [Button => 'Copy',         -command => \&edit_copy ],
      [Button => 'Cut',          -command => \&edit_cut ],
      [Button => 'Paste',        -command => \&edit_paste  ],
      [Button => 'Delete',       -command => \&edit_delete ],
      [Separator => ''],
      [Cascade => 'Object    ', -tearoff => 0,
                                -menuitems => [
         [ Button => "Circle",  -command => \&edit_circle ],
         [ Button => "Square" ,  -command => \&edit_square ],
         [ Button => "Point",   -command => \&edit_point  ] ] ],
    ])->grid(-row => 0, -column => 0, -sticky => 'w'),
```

Смотри также

Документация по модулю Tk с CPAN.

15.15. Создание диалоговых окон с помощью Tk

Проблема

Требуется создать диалоговое окно, то есть новое окно верхнего уровня с кнопками для его закрытия. Диалоговое окно также может содержать другие элементы - например, надписи и текстовые поля для ввода информации. Например, в диалоговом окне можно ввести регистрационные данные и закрыть его после передачи сведений или в том случае, если пользователь не захочет регистрироваться.

Решение

В простых случаях воспользуйтесь элементом Tk::DialogBox:

```
use Tk  DialogBox,

$dialog = $main->DialogBox( -title   => "Register This Program",
                            -buttons => [  Register', "Cancel" ] );

# Добавьте элементы в диалоговое окно методом $dialog->Add()                '

# Позднее, когда понадобится отобразить диалоговое окно
$button = $dialog->Show();
if ($button eq 'Register') {
    B
} elsif ($button eq Cancel' ) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    #
} else {
    # Такого быть не должно
}
```

Комментарий

Диалоговое окно состоит из набора кнопок (в нижней части) и произвольных элементов (в верхней части). Вызов `Show` выводит диалоговое окно на экран и возвращает кнопку, выбранную пользователем.

Пример 15.6 содержит полную программу, демонстрирующую принципы работы с диалоговыми окнами.

Пример 15.6. tksample3

```
ft'/usr/bin/perl -w
# tksample3 - работа с диалоговыми окнами

use Tk,
use Tk DialogBox,

$main = MainWindow->new(),

$dialog = $main->DialogBox( -title => Register ,
                            -buttons => [ Register , Cancel ],
                           ),

# В верхней части окна пользователь вводит имя, при этом
# надпись (Label) действует как подсказка

$dialog->add( Label , -text => Name )->pack(),
$entry = $dialog->add( Entry , -width => 35)->pack(),

# Диалоговое окно вызывается кнопкой
$main->Button( -text => Click Registration Form ,
               -command => \&register)  ->pack(-side => left ),
$main->Button( -text => Quit ,
               -command => sub { exit } ) ->pack(-side => left ),

MainLoop,


#
#
# Вызывает диалоговое окно регистрации
"

    my $button,
    my $done = 0,

    do {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        # Отобразит диалоговое окно
        $button = $dialog->Show,

        # Действовать в зависимости от того, какая кнопка была нажата.
        if ($button eq 'Register') {
                my $name = $entry->get,

            if (defined($name) && length($name)) {
                print 'Welcome to the fold, $name\n";
                $done = 1;
            } else {
                print "You didn t give me your name!\n',
            }
        } else {
            print "Sorry you decided not to register.\n";
            $done = 1,
        }
    } until $done,
}
```

В верхней части диалогового окна расположены два элемента: надпись и текстовое поле. Для ввода дополнительной информации понадобятся другие надписи и текстовые поля.

Диалоговые окна часто применяются для вывода предупреждений или сообщений об ошибках. Пример 15.7 показывает, как вывести в диалоговом окне результаты вызова функции `warn`.

Пример 15.7. tksample4

```
tf!/usr/bin/perl        -w
# tksample4 - диалоговые окна для предупреждений

use Tk;
use Tk  DialogBox,

my $main,

# Создать обработчик предупреждений, который отображает
# предупреждение в диалоговом окне Tk

BEGIN {
    $SIG{__WARN__} = sub {
        if (defined $main) {
            my $dialog = $main->DialogBox( -title   =>  'Warning',
                                           -buttons => [  "Acknowledge'  ]);
            $dialog->add("Label",  -text  =>  $_[0])->pack;
            $dialog->Show;
        } else {
            print STDOUT join( '\n', @_), 'n';
        }
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 15.7 (продолжение)

```
    },
}

# Команды вашей программы

$main = MainWindow->new(),

$main->Button( -text    => 'Make A Warning",
               -command => \&make_warning) ->pack(-side => "left'),
$main->Button( -text    => "Quit',
               -command => sub { exit > )  ->pack(-side => "left');

MainLoop,

# Фиктивная подпрограмма для выдачи предупреждения

sub make_warning {
    my $a,
    my $b = 2 * $a,
}
```

Смотри также

Страница руководства Tk::DialogBox в документации по модулю Tk с CPAN; страница руководства *menu*(n) (если она есть).

15.16. Обр аботка событий масштабирования в Tk

Проблема

Вы написали программу на базе Tk, но при изменении размеров окна пользователем нарушается порядок элементов.

Решение

Перехватывая событие запретить пользователю изменять размеры окна:

```
use Tk,

$main = MainWindow->new(),

$main->bind('<Configure>' => sub {
    $xe = $main->XEvent;
    $main->maxsize($xe->w, $xe->h),
    $main->minsize($xe->w, $xe->h),
}),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Кроме того, можно определить особенности масштабирования элементов при изменении размеров контейнера с помощью метода pack:

```
$widget->pack( -fill => both , -expand => 1
$widget->pack( -fill => x ,    -expand => 1
```

Комментарий

По умолчанию упакованные элементы изменяют размеры вместе с контейнером — они не масштабируют себя или свое содержимое в соответствии с новым размером. В результате между элементами возникают пустые места, а ихсодержимое обрезается или искажается.

Первое решение — вообще запретить изменение размеров. Мы перехватываем событие <Configure>, которое возникает при изменении размера или положения элемента, и регистрируем косвенно-вызываемую функцию (callback) для восстановления прежнего размера окна. Именно так обеспечивается фиксированный размер окон с сообщениями об ошибках.

Иногда запрещать изменение размеров окна нежелательно; в этом случае необходимо определить, как каждый элемент должен реагировать на изменения. Для этого используются аргументы метода pack: -fillуправляет той областью, внутри которой должен находиться элемент, а -expand говорит о том, должен ли элемент изменять свой размер для заполнения доступного места. Параметр -expand принимает логические значения, true или false. Строковый параметр -fill обозначает оси, по которым может изменяться размер элемента: x, y, both или none.

Для правильной работы необходимы оба параметра: -expand без -fill не узнает, в какой области должен увеличиваться элемент. -fill без -expand захватит область нужного размера, но сохранит прежние размеры.

Разные части вашего приложения ведут себя по-разному. Например, главная область Web-броузера при изменении размера окна, вероятно, должна изменить свои размеры в обоих направлениях. Метод pack для такого элемента выглядит так:

```
$mainarea->pack( -fill => both , -expand => 1),
```

Однако меню, расположенное над главной областью, может расшириться по горизонтали, но не по вертикали:

```
$menubar->pack( -fill => x , -expand => 1 ),
```

С изменением размеров связана и другая задача — закрепление элементов в определенной точке контейнера. Например, полоса меню закрепляется в левом верхнем углу контейнера следующим образом:

```
$menubar->pack (-fill   => x ,
                -expand => 1,
                -anchor => nw  ),
```

Теперь при изменении размеров окна меню останется на своем месте и не будет выровнено по центру пустой области.

Смотри также

Страницы руководства *pack*(n), *XEvent*(3) и *XConfigureEvent*(3)(если есть).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

15.17. Удаление окна сеанса DOS в Perl/Tk для Windows

Проблема

Вы написали программу для Windows-версии Perl и Tk, однако при каждом запуске программы открывается окно DOS-сеанса.

Решение

Запускайте программу из другого сценария Perl. В примере 15.8 содержится пример загрузчика, который запускает программу *realprogram* без окна DOS.

Пример 15.8. loader

```
#!/usr/bin/perl -w
# loader - запуск сценариев Perl без раздражающего окна DOS
use strict;
use Win32;
use Win32::Process;

ft Создать объект процесса.

Win32::Process::Create($Win32::Process::Create::ProcessObj,
     'C:/perl5/bin/perl.exe',            # Местонахождение Perl
     'perl realprogram',                 ft
    0,                                   й Не наследовать
    DETACHED_PROCESS,                    #
     ".") or                              " Текущий каталог
die print_error();

sub print_error() {
            Win32::FormatMessage( Win32::GetLastError() );
}
```

Комментарий

Программа проще, чем кажется. Окно DOS появляется потому, что интерпретатор Perl был откомпилирован как консольное приложение. Для чтения из STDIN и записи в STDOUT ему нужно окно DOS. Это нормально в приложениях, работающих в режиме командной строки, но если все общение с пользователем организовано с помощью Tk, то окно DOS не понадобится.

Загрузчик использует модуль Win32::Process для запуска программы в качестве нового процесса. Этот процесс отделяется от текущего, поэтому при завершении загрузчика окно DOS пропадет вместе с ним. Ваша программа будет прекрасно работать, не отягощенная пережитками прошлого.

Если произойдет какая-нибудь беда и программа не запустится, загрузчик умрет с выдачей сообщения об ошибке Windows.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Документация по модулю Win32::Process, входящая в поставки Perl для систем Microsoft Windows.

15.18. Программа: tcapdemo

Описание

Следующая программа очищает экран и рисует на нем до тех пор, пока не будет прервана. Она показывает, как использовать модуль Term::Cap для очистки экрана, перемещения курсора и записи в любую позицию экрана. В ней также используется рецепт 16.6.

Пример 15.9. tcapdemo

```
#!/usr/bin/perl -w
# tcapdemo - прямое позиционирование курсора

use POSIX;
use Term::Cap;

init();                    # Инициализация Term::Cap.
zip();                     # Рисование линий на экране.
finish();                    # Последующая очистка.
exit();

# Две вспомогательные функции.                    очевиден,
# clear_end очищает до конца экрана.
                    $tcap->Tputs('cl', 1, *STDOUT) }
sub clear_end    { $tcap->Tputs('cd', 1, *STDOUT) }

# Переместить курсор в конкретную позицию.
sub gotoxy {
    my($x, $y) = @_;
    $tcap->Tgoto('cm', $x, $y, *STDOUT);
}

# Определить скорость терминала через модуль POSIX и использовать
# для инициализации Term::Cap.
sub init {
    $l = 1;
    $delay = (shift() || 0) * 0.005;
    my $termios = POSIX::Termios->new();
    $termios->getattr;
    my $ospeed = $termios->getospeed;
    $tcap = Term::Cap->Tgetent ({ TERM => undef, OSPEED => $ospeed });

}
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 15.9 (продолжение)

```
# Рисовать линии на экране, пока пользователь
# не нажмет Ctrl-C
sub zip {

    ($maxrow, $maxcol) = ($tcap->{_li} - 1, $tcap->{_co} - 1),

    @chars = qw(* - / | \ _ ),
    sub circle { push(@chars, shift @chars), }

    $interrupted = 0,
    $SIG{INT} = sub { ++$interrupted },

    $col = $row = 0,
    ($row_sign, $col_sign) = (1,1),

    do {
        gotoxy($col, $row)
        print $chars[0],
        select(undef, undef, undef, $delay), ˚

        $row += $row_sign,
        $col += $col_sign,

        if    ($row == $maxrow) { $row_sign = -1, circle, }
        elsif ($row == 0 )      { $row_sign = +1, circle, }

        if    ($col == $maxcol) { $col_sign = -1, circle, }
        elsif ($col == 0 )      < $col_sign = +1, circle, }

    } until $interrupted,

}

# Очистить экран
sub finish {
    gotoxy(0, $maxrow),
    clear_end(),
}
```

Смотри также

Документация по стандартному модулю Term::Cap; *termcap*(5) (если есть).

15.19. Программа: tkshufflepod

Эта короткая программа с помощью Tk выводит список всех заголовков `=head1` в файле и позволяет изменить порядок их следования перетаскиванием. Клавиша s сохраняет изменения, а q завершает программу. Двойной щелчок на элемен-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

те списка выводит его содержимое в элементе Pod. Текст раздела записывается во временный файл, находящийся в каталоге */tmp*; файл удаляется при уничтожении элемента Pod.

При запуске программе передается имя просматриваемого pod-файла:

```
% tkshufflepod chap15 pod
```

Мы часто использовали эту программу при работе над книгой.

Исходный текст программы приведен в примере 15.10.

Пример 15.10. tkshufflepod

```
#!/usr/bin/perl -w
# tkshufflepod - изменение порядка разделов =head1 в pod-файле

use Tk;
use strict;



my $podfile;        я Имя открываемого файла
my $m;              # Главное окно
my $1;              # Элемент Listbox
my ($up, $down);    # Перемещаемые позиции
my @sections;       # Список разделов pod
my $all_pod;        # Текст pod-файла (используется при чтении)

# Прочитать pod-файл в память и разбить его на разделы

$podfile = shift || - ;

undef $/;
open(F, < $podfile )
  or die Can t open $podfile $!\n ;
$all_pod = <F>
close(F);
@sections = split(/(?==head1)/, $all_pod);

# Превратить @sections в массив анонимных массивов Первый элемент
# каждого массива содержит исходный текст сообщения, а второй -
# текст, следующий за =head1 (заголовок раздела)

        (@sections)
    /( *)/,
    $_ = [ $_, $1 ],
}

# Запустить Tk и вывести список разделов

$m = MainWindow->new();
$1 = $m->Listbox('-width' => 60)->pack('-expand' => 1, '-fill' => both');
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 15.10 (продолжение)

```
            $section (@sections)
        $l->insert("end", $section->[1]),
    }

    # Разрешить перетаскивание для элемента Listbox.
    $l->bind( '<Any-Button>'     => \&down );
    $l->bind(  <Any-ButtonRelease>  => \&up );

    # Разрешить просмотр при двойном щелчке
    $l->bind( '<Double-Button>'     => \&view );

    # 'q' завершает программу, а 's' сохраняет изменения.
    $m->bind( '<q>'    => sub { exit } );
    $m->bind( '<s>'     => \&save );

    MainLoop;

    fl down(widget): вызывается, когда пользователь щелкает в Listbox

    sub down {
        my $self = shift,
        $down = $self->curselection,,
    }

    tf up(widget): вызывается, когда пользователь отпускает
    # кнопку мыши в Listbox.

    sub up {
        my $self = shift;
        my $elt;

        $up = $self->curselection;,

                if $down == $up,

        # change selection list
        $elt = $sections[$down],
        splice(@sections, $down, 1),
        splice(@sections, $up, 0, $elt),
        $self->delete($down),
        $self->insert($up, $sections[$up]->[1]),
    }

    # save(widget)  сохранение списка разделов.

    sub save {
        my $self = shift;

        open(F, '> $podfile")
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
      or die "Can't open $podfile for writing  $!";
    print F map { $_->[0] } @sections,
    close F;

    exit;
}

в view(widget): вывод раздела.  Использует элемент Pod

sub view {
    my $self =  shift;
    my $temporary = "/tmp/$$-section pod";
    my $popup;

    open(F, "> $temporary")
      or warn ("Can't open $temporary : $!\n"),
    print F $sections[$down]->[0];
    close(F);
    $popup = $m->Pod('-file' => $temporary);

    $popup->bind('<Destroy>' => sub { unlink $temporary } );
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Управление процессами и межпроцессные взаимодействия 16

Это задача как раз на три трубки.
Я прошу вас минут пятьдесят
не разговаривать со мной.

Шерлок Холмс,
"Союз рыжих"

Введение

Многие из нас относятся к Perl по-своему, но большинство считает его чем-то вроде "клея", объединяющего разнородные компоненты. Эта глава посвящена командам и отдельным процессам — их созданию, взаимодействию и завершению. Итак, речь пойдет о системном программировании.

В области системного программирования на Perl, как обычно, все простое упрощается, а все сложное становится доступным. Если вы хотите работать на высоком уровне, Perl с радостью вам поможет. Если вы собираетесь закатать рукава и заняться низкоуровневым программированием, уподобившись хакерам C, — что ж, возможно и это.

Perl позволяет очень близко подобраться к системе, но при этом могут возникнуть некоторые проблемы переносимости. Из всей книги эта глава в наибольшей степени ориентирована на UNIX. Изложенный материал чрезвычайно полезен для тех, кто работает в UNIX-системах, и в меньшей степени — для всех остальных. Рассматриваемые в ней возможности не являются универсальными, как, например, строки, числа или базовая арифметика. Большинство базовых операций более или менее одинаково работает повсюду. Но если вы не работаете в системе семейства UNIX или другой POSIX-совместимой системе, многие интересные возможности у вас будут работать иначе (или не будут работать вообще). В сомнительных ситуациях обращайтесь к документации, прилагаемой к вашей версии Perl.

Создание процессов

В этой главе рассматриваются порожденные процессы. Иногда вы просто выполняете автономную команду (с помощью `system`) и оставляете созданный процесс на произвол судьбы. В других случаях приходится сохранять тесную связь с созданным процессом, скармливать ему тщательно отфильтрованные данные или уп-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

равлять его потоком вывода (`...` и конвейерные вызовы open). Наконец, даже без запуска нового процесса вызов exec позволяет заменить текущую программу чем-то совершенно новым.

Сначала мы рассмотрим самые переносимые и распространенные операции управления процессами: `...`, system, open и операции с хэшем %SIG. Здесь нет ничего сложного, но мы не остановимся на этом и покажем, что делать, когда простые решения не подходят.

Допустим, вы хотите прервать свою программу в тот момент, когда она запустила другую программу. Или вам захотелось отделить стандартный поток ошибок порожденного процесса от его стандартного вывода. Или вы собираетесь одновременно управлять и как вводом, так и выводом программы. Или вы решили воспользоваться преимуществами многозадачности и разбить свою программу на несколько одновременно работающих процессов, взаимодействующих друг с другом.

В подобных ситуациях приходится обращаться к системным функциям: pipe, fork и exec. Функция pipe создает два взаимосвязанных манипулятора, записывающий и читающий; при этом все данные, записываемые в первый, могут быть прочитаны из первого. Функция fork является основой многозадачности, но, к сожалению, она не поддерживается некоторыми системами, не входящими в семейство UNIX. Функция создает процесс-дубликат, который практически во всех отношениях идентичен своему родителю, включая значения переменных и открытые файлы. Самые заметные изменения — идентификатор процесса и идентификатор родительского процесса. Новые программы запускаются функцией fork, после чего функция exec заменяет программу порожденного процесса чем-то другим. Функции fork и exec не всегда используются вместе, поэтому наличие отдельных примитивов оказывается более выразительным и мощным по сравнению с ситуацией, когда ваши возможности ограничиваются выполнением system. На практике fork по отдельности используется чаще, чем с exec.

При уничтожении порожденного процесса его память возвращается операционной системе, но соответствующий элемент таблицы процессов не освобождается. Благодаря этому родитель может проверить статус завершения всех порожденных процессов. Процессы, которые умерли, но не были удалены из таблицы процессов, называются *зомби;* их следует своевременно удалять, чтобы они не заполнили всю таблицу процессов. Оператор `...`, а также функции system и open автоматически следят за этим и работают в большинстве систем, не входящих в семейство UNIX. При выходе за рамки этих простых переносимых функций и запуске программ с помощью низкоуровневых примитивов возникают дополнительные хлопоты. Кроме того, не стоит забывать и о сигналах.

Сигналы

Ваш процесс узнает о смерти созданного им порожденного процесса с помощью *сигнала.* Сигналы представляют собой нечто вроде оповещений, доставляемых операционной системой. Они сообщают о произошедших ошибках (когда ядро говорит: "Не трогай эту область памяти!») и событиях (смерть порожденного процесса, тайм-аут процесса, прерывание по Ctrl+C). При ручном запуске процесса обычно указывается подпрограмма, которая должна вызываться при завершении потомка.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Каждый процесс имеет стандартные обработчики для всех возможных сигналов. **Вы** можете установить свой собственный обработчик **или** изменить отношение программы к большинству сигналов. Не изменяются только SIGKILL и SIGTOP — все остальные сигналы можно игнорировать, перехватывать и блокировать.

Приведем краткую сводку важнейших сигналов.

SIGINT

Обычно возникает при нажатии Ctrl+C. Требует, чтобы процесс завершил свою работу. Простые программы (например, фильтры) обычно просто умирают, но более сложные программы — командные интерпретаторы, редакторы и программы FTP — обычно используют SIGINT для прерывания затянувшихся операций.

SIGQUIT

Обычно генерируется терминалом, как правило, при нажатии Ctrl+\. По умолчанию выводит в файл содержимое памяти.

SIGTERM

Посылается командой `kill` при отсутствии явно заданного имени сигнала. Может рассматриваться как вежливая просьба умереть, адресованная процессу.

SIGUSR1 и SIGUSR2

Никогда не вызываются системными событиями, поэтому пользовательские приложения могут смело использовать их для собственных целей.

SIGPIPE

Посылается ядром, когда ваш процесс пытается записать в канал (pipe) или сокет, а процесс на другом конце канала/сокета отсоединился (обычно потому, что он перестал существовать).

SIGALRM

Посылается при истечении промежутка времени, установленного функцией `alarm` (см. рецепт 16.21).

SIGHUP

Посылается процессу при разрыве связи (hang-up) на управляющем терминале (например, при потере несущей модемом), но также часто означает, что Программа должна перезапуститься или заново прочитать свою конфигурацию.

SIGCHLD

Вероятно, самый важный сигнал во всем низкоуровневом системном программировании. Система посылает процессу сигнал SIGSHLD в том случае, если один из его порожденных процессов перестает выполняться — или, что более вероятно, при его завершении. Дополнительные сведения о SIGCHLD приведены **в** рецепте **16.19.**

Имена сигналов существуют **лишь** для удобства программистов. С каждым сигналом связано определенное число, используемое операционной системой вместо имени. Хотя мы говорим о сигнале SIGCHLD, операционная система опозна-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ет его по номеру — например, 20 (в зависимости от операционной системы). Perl преобразует номера сигналов в имена, поэтому вы можете работать с именами сигналов.

Обработка сигналов рассматривается в рецептах 16.7, 16.15, 16.18, 16.20 и 16.21.

16.1. Получение вывода от программы

Проблема

Требуется запустить программу и сохранить ее вывод в переменной.

Решение

Воспользуйтесь либо оператором `...`:

```
$output = `ПРОГРАММА АРГУМЕНТЫ`; # Сохранение данных в одной
                                 # многострочной переменной.
@output = `ПРОГРАММА АРГУМЕНТЫ`; # Сохранение данных в массиве,
                                 # по одной строке на элемент.
```

либо решением из рецепта 16.4:

```
open(README, "ПРОГРАММА АРГУМЕНТЫ |") or die "Can't run program: $!\n";
while(<README>) {
    $output .= $_;
}
close(README);
```

Комментарий

Оператор `...` является удобным средством для запуска других программ и получения их выходных данных. Возврат из него происходит лишь после завершения вызванной программы. Для получения вывода Perl предпринимает некоторые дополнительные усилия, поэтому было бы неэффективно использовать `...` и игнорировать возвращаемое значение:

```
`fsck -y /dev/rsd1a`;      # ОТВРАТИТЕЛЬНО
```

И функция open, и оператор ` ` обращаются к командному интерпретатору для выполнения команд. Из-за этого они недостаточно безопасно работают в привилегированных программах.

Приведем низкоуровневое обходное решение с использованием pipe, fork и exec:

```
use POSIX qw(:sys_wait_h);

pipe(README, WRITEME);
if ($pid = fork) {
    # Родительский процесс
    $SIG{CHLD} = sub { 1 while ( waitpid(-1, WNOHANG)) > 0 };
    close(WRITEME);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
} else {
    die  cannot fork  $'  unless defined $pid,
    # Порожденный процесс
    open(STDOUT,  >&=WRITEME )      or die  Couldn t          STDOUT  $'
    close(README),
    exec($program, $arg1  $arg2)    or die  Couldn t run $program   $'\n ,
}

while (<README>) {
    $string  = $_,
    # or  push(@strings, $_),
}
close(REAOME),
```

Смотри также

perlfsec(1); рецепты 16.2; 16.4; 16.19; 19.6.

16.2. Запуск другой программы

Проблема

Вы хотите запустить другую программу из своей, дождаться ее завершения и затем продолжить работу. Другая программа должна использовать те же STDIN и STDOUT, что и основная.

Решение

Вызовите функцию system со строковым аргументом, который интерпретируется как командная строка:

```
$status = system( vi $myfile ),
```

Если вы не хотите привлекать командный интерпретатор, передайте system список:

```
$status = system( vi  $myfile),
```

Комментарий

Функция system обеспечивает самую простую и универсальную возможность запуска других программ в Perl. Она не возвращает выходные данные внешней программы, как или open Вместо этого ее возвращаемое значение (фактически) совпадает с кодом завершения программы. Во время работы новой программы основная приостанавливается, поэтому новая программа может взаимодействовать с пользователем посредством чтения данных из STDIN и записи в STDOUT.

При вызове с одним аргументом функция system (как и open, exec и) использует командный интерпретатор для запуска программы. Это может пригодиться для перенаправления или других фокусов·

```
system("cmd1 args | cmd2 | cmd3 >outfile'),
system( cmd args <infile >outfile 2>errfile ),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Чтобы избежать обращений к интерпретатору, вызывайте system со списком аргументов:

```
$status = system($program, $arg1, $arg),
die  $program exited funny  $?  unless $status == 0,
```

Возвращаемое значение не является обычным кодом возврата; оно включает номер сигнала, от которого умер процесс (если он был). Это же значение присваивается переменной $? функцией wait. В рецепте 16.19 рассказано о том, как декодировать tuj.

Функция system (но не !) игнорирует SIGINT и SIGQUIT во время работы порожденных процессов. Сигналы убивают лишь порожденные процессы. Если вы хотите, чтобы основная программа умерла вместе с ними, проверьте возвращаемое значение system или переменную $?:

```
if (($signo = system(@arglist)) &= 127) {
    die  program killed by signal $signo\n ,
}
```

Чтобы игнорировать SIGINT, как это делает system, установите собственный обработчик сигнала, а затем вручную вызовите fork и exec:

```
if ($pid = fork) {
    # Родитель перехватывает INT и предупреждает пользователя
    local $SIG{INT} = sub {  print  Tsk tsk,  no process interruptus\n  },
    waitpid($pid,  0)
} else {
    die  cannot  fork  $!  unless defined $pid,
    # Потомок игнорирует INT и делает свое дело
    $SIG<INT} =  IGNORE ,
    exec( summarize ,  /etc/logfiles )   or die  Can t exec  $!\n ,
}

($pid = fork)  ^ waitpid($pid,  0)   exec(@ARGV)
                                        or die  Can t exec  $!\n ,
```

Некоторые программы просматривают свое имя. Командные интерпретаторы узнают, были ли они вызваны с префиксом -, обозначающим интерактивность. Программа *expn* в конце главы 18 при вызове под именем *vrfy* работает иначе; такая ситуация возникает при создании двух ссылок на файл (см. описание *expn)*. По этой причине не следует полагать, что $0 всегда содержит имя вызванной программы.

Если вы хотите подсунуть запускаемой программе другое имя, укажите настоящий путь в виде "косвенного объекта" перед списком, передаваемым system (также работает для exec). После косвенного объекта не ставится запятая, по аналогии с вызовом printf для файлового манипулятора или вызовом методов объекта без ->.

```
$shell =  /bin/tcsh
system  $shell   -csh ,            # Прикинуться другим интерпретатором
```

Или непосредственно:

```
system  {  /bin/tcsh  }   -csh  ,     # Прикинуться другим интерпретатором
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В следующем примере настоящее имя программы передается в виде косвенногообъекта {'/home/tchrist/scripts/expn'}. **Фиктивное имя** 'vrfy' передается в виде первого настоящего аргумента функции, и программа увидит его в переменной $0.

```
# Вызвать expn как vrfy
system {'/home/tchrist/scripts/expn'} 'vrfy', @ADDRESSES;
```

. Применение косвенных объектов с system более надежно. В этом случае аргументы заведомо интерпретируются как список, даже если он состоит лишь из одного элемента. Это предотвращает расширение метасимволов командным интерпретатором или разделение слов, содержащих пропуски.

```
@args = ( "echo surprise" );

system @args;      # Если @args == 1, используются
                   # служебные преобразования интерпретатора
system { $args[0] > @args;   # Безопасно даже для одноаргументного списка
```

Первая версия (без косвенного объекта) запускает программу echo и передает ей аргумент "surprise". Вторая версия этого не делает — она честно пытается запустить программу "echo surprise", не находит ее и присваивает $? ненулевое значение, свидетельствующее об ошибке.

Смотри также

perlsec(1); описание функций waitpid, fork, exec, system и open в ***perlfunc*(1);** рецепты **16.1; 16.4; 16.19; 19.6.**

16.3. Замена текущей программы

Проблема

Требуется заменить работающую программу другой — например, после проверки параметров и настройки окружения, предшествующих выполнению основной программы.

Решение

Воспользуйтесь встроенной функцией exec. Если exec вызывается с одним аргументом, содержащим метасимволы, для запуска будет использован командный интерпретатор:

```
exec("archive *.data")
    or die "Couldn't          myself with archive: $!\n";
```

Если exec передаются несколько аргументов, командный интерпретатор не используется:

```
exec("archive", "accounting.data")
    or die "Couldn't              f with archive: $!\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При вызове с одним аргументом, не содержащим метасимволов, аргумент разбивается по пропускам и затем интерпретируется так, словно функция exec была вызвана для полученного списка:

```
exec( archive accounting data )
    or die Couldn t          myself with archive $!\n ,
```

Комментарий

Функция Perl exec обеспечивает прямой интерфейс к системной функции execlp(2), которая заменяет текущую программу другой без изменения идентификатор процесса. Программа, вызвавшая exec, стирается, а ее место в таблице процессов операционной системы занимает программа, указанная в качестве аргумента exec. В результате новая программа сохраняет тот же идентификатор ироцесса ($$), что и у исходной программы. Если указанную программу запустить не удалось, exec возвращает false, а исходная программа продолжает работу. Не забывайте проверять такую ситуацию.

При переходе к другой программе с помощью exec не будут автоматически вызваны ни блоки END, ни деструкторы объектов, как бы это произошло при нормальном завершении процесса.

Смотри также

Описание функции exec в *perlfunc*(1); страница руководства ***execlp(2)*** вашей системы (если есть); рецепт **16.2.**

16.4. Чтение или запись в другой программе

Проблема

Вы хотите запустить другую программу и либо прочитать ее вывод, либо предоставить входные данные.

Решение

Вызовите open с символом | в начале или конце строки. Чтобы прочитать вывод программы, поставьте | в конце:

```
$pid = open(README, program arguments | ) or die Couldn t fork $!\n ,
while (<README>) {
    #
}
close(README)                                or die Couldn t close $!\n ,
```

Чтобы передать данные, поставьте | в начале:

```
$pid = open(WRITEME, | program arguments ) or die Couldn t fork $!\n ,
print WRITEME "data\n",
close(WRITEME)                               or die Couldn t close $!\n ,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

При чтении происходящее напоминает , разве что на этот раз у вас имеется идентификатор процесса и файловый манипулятор. Функция open также использует командный интерпретатор, если встречает в аргументе метасимволы, и не использует в противном случае. Обычно это удобно, поскольку вы избавляетесь от хлопот с расширением метасимволов в именах файлов и перенаправлением ввода/вывода.

Однако в некоторых ситуациях это нежелательно. Конвейерные вызовы open, в которых участвуют непроверенные пользовательские данные, ненадежны при работе в режиме меченых данных или в ситуациях, требующих абсолютной уверенности. Рецепт 19.6 показывает, как имитировать эффект конвейерных вызовов open безриска, связанного с использованием командного интерпретатора.

Обратите внимание на явный вызов close для файлового манипулятора. Когда функция open используется для подключения файлового манипулятора к порожденному процессу, Perl запоминает этот факт и при закрытии манипулятора автоматически переходит в ожидание. Если порожденный процесс к этому моменту не завершился, Perl ждет, пока это произойдет. Иногда ждать приходится очень, очень долго:

```
$pid = open(F, sleep 100000| ),      # Производный процесс приостановлен "
close(F),                            # Родитель надолго задумался
```

Чтобы избежать этого, уничтожьте производный процесс по значению PID, полученному от open, или воспользуйтесь конструкцией pipe-fork-exec (см. рецепт 16.10).

При попытке записать данные в завершившийся процесс, ваш процесс получит сигнал SIGPIPE. По умолчанию этот сигнал убивает ваш процесс, поэтому программист-параноик на всякий случай установит обработчик SIGPIPE.

Если вы хотите запустить другую программу и предоставить содержимое ее STDIN, используется аналогичная конструкция:

```
$pid = open(WRITEME, | program args ),
print WRITEME hello\n ,     ft     Программа получит hello\n в STDIN
close(WRITEME),                # Программа получит EOF в STDIN
```

Символ | в начале аргумента функции open, определяющего имя файла, сообщает Perl о необходимости запустить другой процесс. Файловый манипулятор, открытый функцией open, подключается к STDIN порожденного процесса. Все, что вы запишете в этот манипулятор, может быть прочитано процессом из STDIN. После закрытия манипулятора (close) при следующей попытке чтения из STDIN порожденный процесс получит eof.

Описанная методика может применяться для изменения нормального вывода вашей программы. Например, для автоматической обработки всех данных утилитой постраничного вывода используется фрагмент вида:

```
$pager = $ENV{PAGER} || /usr/bin/less ,   # XXX  может не существовать
open(STDOUT, | $pager ),
```

Теперь все данные, направленные в стандартный вывод, будут автоматически проходить через утилиту постраничного вывода. Вам не придется исправлять другие части программы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Как и при открытии процесса для чтения, в тексте, передаваемом командному интерпретатору, происходит расширение метасимволов. Чтобы избежать обращения к интерпретатору, следует воспользоваться решением, аналогичным приведенному выше. Как и прежде, родитель должен помнить о close. При закрытии файлового манипулятора, подключенного к порожденному процессу, родитель блокируется до завершения потомка. Если порожденный процесс не завершается, то и закрытие не произойдет. Приходится либо заранее убивать порожденный процесс, либо использовать низкоуровневый сценарий pipe-fork-exec.

При использовании сцепленных открытий всегда проверяйте значения, возвращаемые open и close, не ограничиваясь одним open. Дело в том, что возвращаемое значение open не говорит о том, была ли команда успешно запущена. При сцепленном открытии команда выполняется вызовом fork для порожденного процесса. Если возможности создания процессов в системе не исчерпаны, fork немедленно возвращает PID порожденного процесса.

К тому моменту, когда порожденный процесс пытается выполнить команду exec, он уже является самостоятельно планируемым. Следовательно, если команда не будет найдена, практически не существует возможности сообщить об этом функции open, поскольку она принадлежит другому процессу!

Проверка значения, возвращаемого close, позволяет узнать, успешно ли выполнилась команда. Если порожденный процесс завершается с ненулевым кодом (что произойдет в случае, если команда не найдена), то close возвращает false, а переменной $? присваивается статус ожидания процесса. Об интерпретации содержимого этой переменной рассказано в рецепте 16.2.

Смотри также

Описание функции open в *perlfunc*(1) рецепты **16.10; 16.15; 16.19; 19.6.**

16.5. Фильтрация выходных данных

Проблема

Требуется обработать выходные данные вашей программы без написания отдельного фильтра.

Решение

Присоедините фильтр с помощью разветвляющего (forking) вызова open. Например, в следующем фрагменте вывод программы ограничивается сотней строк:

```
head(100),
while (<>) {
    print,
}

sub head {
    my $lines = shift || 20,
            if $pid   open(STDOUT,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    die "cannot fork. $!" unless defined $pid,
    while (<STDIN>) {
        print;
        last unless --$lines , .
    }
    exit;
}
```

Комментарий

Создать выходной фильтр несложно — достаточно открыть STDOUT разветвляющим вызовом open, а затем позволить порожденному процессу фильтровать STDIN в STDOUT и внести те изменения, которые он посчитает нужным. Обратите внимание: выходной фильтр устанавливается *до* генерации выходных данных. Это вполне логично — нельзя отфильтровать **вывод,** который уже покинул вашу программу.

Все подобные фильтрыдолжны устанавливаться в порядке очередности стека — последний установленный фильтр работает первым.

Рассмотрим пример, в котором используются два выходных фильтра. Первый фильтр нумерует строки; второй — снабжает их символами цитирования (как в сообщениях электронной почты). Для файла */etc/motd* результат выглядит примерно так:

```
1: > Welcome to Linux, version 2.0.33 on a i686
2: >
                                    'Windows    or better',
4: >        so I installed Linux."
```

Если изменить порядок установки фильтров, вы получите следующий результат:

```
> 1: Welcome to Linux, Kernel version 2 .0.33 on a i686
> 2:
                                    'Windows        better',
> 4:        so I installed  Linux."
```

Исходный текст программы приведен в примере 16.1.

Пример 16.1. **qnumcat**

```
#!/usr/bin/perl
# qnumcat - установка сцепленных выходных фильтров

number(),           # Установить для STDOUT нумерующий фильтр
quote(),            # Установить для STDOUT цитирующий фильтр

while (<>) {        # Имитировать /bin/cat
    print;
}

close STDOUT;       # Вежливо сообщить потомкам о завершении
exit,

sub number <
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    my $pid;
        if $pid  open(STDOUT, '|-');
    die 'cannot fork: $!' unless defined $pid,
    while (<STDIN>) { printf "%d %s", $ , $_ }
    exit;
}

sub quote {
    my $pid,
        if $pid = open(STDOUT, "|-");
    die "cannot fork $!" unless defined $pid,
    while (<STDIN>) { print "> $_" }
    exit;
}
```

Как и при любых разветвлениях, для миллиона процессов такое решение не подойдет, но для пары (или даже нескольких десятков) процессов расходы будут небольшими. Если ваша система изначально проектировалась как многозадачная (как UNIX), все обойдется дешевле, чем можно себе представить. Благодаря виртуальной памяти и копированию во время записи такие операции выполняются достаточно эффективно. Разветвление обеспечивает элегантное и недорогое решение многих (если не всех) задач, связанных с многозадачностью.

Смотри также

Описание функции open в *perlfunc*(1); рецепт 16.4.

16.6. Предварительная обработка ввода

Проблема

Ваша программа умеет работать лишь с обычным текстом в локальных файлах. Однако возникла необходимость работать с экзотическими файловыми форматами — например, сжатыми файлами или Web-документами, заданными в виде URL.

Решение

Воспользуйтесь удобными средствами Perl для работы с каналами и замените имена входных файлов каналами перед тем, как открывать их.

Например, следующий фрагмент автоматически восстанавливает архивные файлы, обработанные утилитой gzip:

```
@ARGV = map { /\.(gz|Z)$/ ? 'gzip -dc $_ |' . $_ : $_ } @ARGV,
while (<>) {
    # ... .
}
```

А чтобы получить содержимое URL перед его обработкой, воспользуйтесь программой GET из модуля LWP (см. главу 20 "Автоматизация в Web»):

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
§ARGV = map { m#^\w+ //# ^ GET $_ | $_ } @ARGV
while (o) {
    #
}
```

Конечно, вместо HTML-кода можно принять простой текст Для этого достаточно воспользоваться другой командой (например, *lynx -dump*)

Комментарий

Как показано в рецепте 16 1, встроенная функция Perl open оченьудобна каналы открываются в Perl так же, как и обычные файлы Если то, что вы открываете, похоже на канал, Perl открывает его как канал Мы используем эту особенность и включаем в имя файла восстановление архива или иную предварительную обработку Например,файл 09tails gz превращаетсяв gzcat -dc 09tails gz|

Эта методика применима и в других ситуациях Допустим, вы хотите прочитать */etc/passwd*,если компьютер не использует NIS, и вывод *ypcat passwd*в противном случае Мы определяем факт использования NIS по выходным данным программы *domainname*, после чего выбираем в качестве открываемого файла строку </etc/passwd или ypcat passwd|

```
$pwdinfo = domainname =~ /^(\(none\))?$/
                ? < /etc/passwd
                  ypcat passwd |

open(PWD $pwdinfo)                  or die can t open $pwdinfo $!
```

Но и это еще не все! Даже если вы не собирались встраивать подобные возможности в свою программу, Perl делает это за вас! Представьте себе фрагмент вида

```
print File please?
chomp($file = o)
open (FH $file)                     or die can t open $file $!
```

Пользователь может ввести как обычное имя файла, так и строку вида webget http //www perl com | — и ваша программа вдруг начинает получать выходные данные от webget! А если ввести всего один символ, дефис (-), то при открытии для чтения будет интерполирован стандартный ввод

В рецепте 7 7 эта методика использовалась для автоматизации обработки ARGV

Смотри также

Рецепты 7 7, 16 4

16.7. Чтение содержимого STDERR

Проблема

Вы хотите выполнить программу с помощью system, или open, но содержимое ее STDERR не должно выводиться в ваш STDERR Необходимо либо игнорировать содержимое STDERR, либо сохранять его отдельно

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь числовым синтаксисом перенаправления и дублирования для файловых дескрипторов. Для упрощения примеров мы не проверяем возвращаемое значение open, но выобязательно должны делать это в своих программах!

Одновременное сохранение STDERR и STDOUT:

```
$output = cmd 2>&1 ,                # Для
# или
$pid = open(PH, cmd 2>&1 | ),       # Для open
while (<PH>) { }                    Я Чтение
```

Сохранение STDOUT с игнорированием STDERR·

```
$output = cmd 2>/dev/null ,              # Для
# или
$pid = open(PH, cmd 2>/dev/null | ),     # Для open
while (<PH>) { }                         # Чтение
```

Сохранение STDERR с игнорированием STDOUT:

```
$output = cmd 2>&1 1>/dev/null ,            # Для
# или
$pid = open(PH, cmd 2>&1 1>/dev/null | ),   # Для open
while (<PH>) { }                            # Чтение
```

Замена STDOUT и STDERR команды, то есть сохранение STDERR и направление STDOUT в старый STDERR:

```
$output =' cmd 3>&1 1>&2 2>&3 3>&- ,          # Для
# или
$pid = open(PH, cmd 3>&1 1>&2 2>&3 3>&-| ), # Для open
while (<PH>) { }                            # Чтение
```

Чтобы организовать раздельное чтение STDOUT и STDERR команды, проще и надежнее всего будет перенаправить их в разные файлы, а затем прочитать из этих файлов после завершения команды:

```
system( prog args 1>/tmp/program stdout 2>/tmp/program stderr ),
```

Комментарий

При выполнении команды оператором ', сцепленным вызовом open или system для одной строки Perl проверяет наличие символов, имеющих особый смысл для командного интерпретатора. Это позволяет перенаправить файловые дескрипторы новой программы. STDIN соответствует файловому дескриптору с номером 0, STDOUT — 1, а STDERR — 2. Например, конструкция 2>файл перенаправляет STDERR в файл. Для перенаправления в файловый дескриптор используется специальная конструкция &N, где N — номер файлового дескриптора Следовательно, 2>&1 направляет STDERR в STDOUT.

Ниже приведена таблица некоторых интересных перенаправлений файловых дескрипторов.

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

Перенаправление	Значение
0</dev/null	Немедленно выдать EOF в STDIN
1>/dev/null	Игнорировать STDOUT
2>/dev/null	Игнорировать STDERR
2>&1	Направить STDERR в STDOUT
2>&-	Закрыть STDERR (не рекомендуется)
3<>/dev/tty	Связать файловый дескриптор 3 с /dev/tty в режиме чтения/записи

На основании этой таблицы мы рассмотрим самый сложный вариант перенаправления в решении:

```
$output = `cmd 3>&1 1>&2 2>&3 3>&-`;
```

Он состоит из четырех этапов.

Этап 1: 3>&1
Скопировать файловый дескриптор 1 в новый дескриптор 3. Прежнее место назначения STDOUT сохраняется в только что открытом дескрипторе.

Этап 2: 1>&2
Направить STDOUT по месту назначения STDERR. В Дескрипторе 3 остается прежнее значение STDOUT.

Этап 3: 2>&3
Скопировать файловый дескриптор 3 в дескриптор 2. Данные STDERR будут поступать туда, куда раньше поступали данные STDOUT.

Этап 4: 3>&-
Перемещение потоков закончено, и мы закрываем временный файловый дескриптор. Это позволяет избежать "утечки" дескрипторов.

Если подобные цепочки сбивают вас с толку, взгляните на них как на обычные переменные и операторы присваивания. Пусть переменная $fd1 соответствует STDOUT, а $fd2 — STDERR. Чтобы поменять значения двух переменных, понадобится временная переменная для хранения промежуточного значения. Фактически происходит следующее:

```
$fd3 = $fd1;
$fd1 = $fd2;
$fd2 = $fd3;
$fd3 = undef;
```

Когда все будет сказано и сделано, возвращаемая оператором `...` строка будет соответствовать STDERR выполняемой команды, а STDOUT будет направлен в прежний STDERR.

Во всех примерах важна последовательность выполнения. Это связано с тем, что командный интерпретатор обрабатывает перенаправления файловых дескрипторов слева направо.

```
system("prog args 1>tmpfile 2>&1");
system("prog args 2>&1 1>tmpfile");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Первая команда направляет и STDOUT и STDERR во временный файл. Вторая команда направляет в файл только STDOUT, а STDERR будет выводиться там, где раньше выводился STDOUT. Непонятно? Снова рассмотрим аналогию с переменными и присваиваниями. Фрагмент

```
# system ("prog args 1>tmpfile 2>&1");
$fd1 = "tmpfile";      ft      Сначала изменить место назначения STDOUT
$fd2 = $fd1;                 # Направить туда же STDERR
```

сильно отличается от другого фрагмента:

```
# system("prog args 2>&1 1>tmpfile");
$fd2 = $fd1;                 # Совместить. STDERR со STDOUT
$fd1 = "tmpfile";            " Изменить место назначения STDOUT
```

Смотри также

Дополнительные сведения о перенаправлении файловых дескрипторов приведены в странице руководства ***sh(1)*** вашей системы (если есть). Функция system описана в ***perlfunc(1)***.

16.8. Управление потоками ввода и вывода другой программы

Проблема

Вы хотите управлять как входными, так и выходными данными другой программы. Функция open позволяет решить одну из этих задач, но не обе сразу.

Решение

Воспользуйтесь стандартным модулем IPC::Open2:

```
use IPC::Open2;

open2(*README, *WRITEME, $program);
      WRITEME              input\n";
$output = <README>;
close(WRITEME);
close(README);
```

Комментарий

Желание управлять вводом и выводом другой программы возникает очень часто, однако за **ним** таится на удивление много опасностей. Поэтому вам не удастся вызвать open в виде:

```
open(DOUBLE_HANDLE, "| программа аргументы |")     # НЕВЕРНО
```

Большая часть трудностей связана с буферизацией. Поскольку в общем случае нельзя заставить другую программу использовать небуферизованный **вывод**, нет

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

гарантии, что операции чтения не будут блокироваться. Если блокировка ввода устанавливается одновременно с тем, как другой процесс заблокируется в ожидании вывода, возникает состояние взаимной блокировки (deadlock). Процессы входят в клинч, пока кто-нибудь не убьет их или не перезагрузит компьютер.

Если вы можете управлять буферизацией другого процесса (потому что вы сами написали программу и знаете, как она работает), возможно, вам поможет модуль IPC::Open2. Первые два аргумента функции open2, экспортируемой IPC::Open2 в ваше пространство имен, представляют собой файловые манипуляторы. Либо передавайте ссылки на typeglobs, как это сделано в решении, либо создайте собственные объекты IO::Handle и передайте их:

```
use IPC   Open2,
use IO   Handle,

        $wr1ter)        Handle->new, IO  Handle->new),
            $wr1ter, $program),
```

Чтобы передать объекты, необходимо предварительно создать их (например, функцией IO .Handle->new). Если передаваемые переменные не содержат файловых манипуляторов, функция open2 не создаст их вас.

Другой вариант — передать аргументы вида '<&OTHERFILEHANDLE или ">&OTHERFILEHANDLE", определяющие существующие файловые манипуляторы для порожденных процессов. Эти файловые манипуляторы не обязаны находиться под контролем вашей программы; они могут быть подключены к другим программам, файлам или сокетам.

Программа может задаваться в виде списка (где первый элемент определяет имя программы, а остальные элементы — аргументы программы) или в виде отдельной строки (передаваемой интерпретатору в качестве команды запуска программы). Если вы также хотите управлять потоком STDERR программы, воспользуйтесь модулем IPC::Open3 (см. следующий рецепт).

Если произойдет ошибка, возврат из open2 и open3 не происходит. Вместо этого вызывается die с сообщением об ошибке, которое начинается с open2 или open3 '. Для проверки ошибок следует использовать конструкцию eval БЛОК:

```
' eval {
                $wr1teme, @program_and_arguments),
  },
  1f ($@) {
      1f ($@ =~ /^open2/) {
          warn  open2 failed  $'\n$@\n ,

      }
      d1e,       # Заново инициировать непредвиденное исключение
  }
```

Смотри также

Документация по стандартным модулям IPC::Open2 и IPC:.Open3; рецепт 10.12; описание функции eval в *perlfunc*(1); описание переменной $@ в разделе "Special Global Variables" *perlvar*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

16.9. Управление потоками ввода, вывода и ошибок другой программы

Проблема

Вы хотите полностью управлять потоками ввода, вывода и ошибок запускаемой команды.

Решение

Аккуратно воспользуйтесь стандартным модулем IPC::Open3, возможно — в сочетании *с* модулем IO::Select (появившимся в версии 5.004).

Комментарий

Если вас интересует лишь один из потоков STDIN, STDOUT или STDOUT программы, задача решается просто. Но если потребуется управлять двумя и более потоками, сложность резко возрастает. Мультиплексирование нескольких потоков ввода/вывода всегда выглядело довольно уродливо. Существует простое обходное решение:

```
@all = '($cmd | sed -e 's/^/stdout: /' ) 2>&1';
for (@all) { push @{ s/stdout' // ^ \@outlines   \@errlines }, $_ }
print "STDOUT:\n", @outlines, '\n";
print "STDERR:\n", @errlines, "\n";
```

Если утилита *sed* не установлена в вашей системе, то в простых случаях вроде показанного можно обойтись командой *perl -pe,* которая работает практически так же.

Однако то, что здесь происходит, в действительности нельзя считать параллельными вычислениями. Мы всего лишь помечаем строки STDOUT префиксом "stdout:" и затем удаляем их после чтения всего содержимого STDOUT и STDERR, сгенерированного программой.

Кроме того, можно воспользоваться стандартным модулем IPC::Open3. Как ни странно, аргументы функции IPC::Open3 следуют в другом порядке, нежели в IPC::Open2.

```
open3(*WRITEHANDLE, *READHANDLE, *ERRHANDLE, "ЗАПУСКАЕМАЯ ПРОГРАММА");
```

Открываются широкие потенциальные возможности для создания хаоса — *еще более* широкие, чем при использовании open2. Если попытаться прочитать STDERR программы, когда она пытается записать несколько буферов в STDOUT, процесс записи будет заблокирован из-за заполнения буферов, а чтение заблокируется из-за отсутствия данных.

Чтобы избежать взаимной блокировки, можно имитировать open3 с помощью fork, open и exec; сделать все файловые манипуляторы небуферизованными и использовать syswrite select, чтобы решить, из какого доступного для чтения манипулятора следует прочитать байт. Однако ваша программа становится медленной и громоздкой, к тому же при этом не решается классическая пробле-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ма взаимной блокировки open2, при которой каждая программа ждет поступления данных от другой стороны:

```
use IPC::Open3;
$pid = open3(*HIS_IN, *HIS_OUT, *HIS_ERR, $cmd);
close(HIS_IN); # Передать порожденному процессу EOF или данные
@outlines = <HIS_OUT>;          # Читать до конца файла
@errlines = <HIS_ERR>;           # XXX: Возможная блокировка
                                 # при больших объемах
print "STDOUT:\n", @outlines, "\n";
print "STDERR:\n", @errlines, "\n";
```

Кроме того (как будто одной взаимной блокировки недостаточно), такое решение чревато нетривиальными ошибками. Существуют по крайней мере три неприятных ситуации: первая — когда и родитель и потомок пытаются читать одновременно, вызывая взаимную блокировку. Вторая — когда заполнение буферов заставляет потомка блокироваться при попытке записи в STDERR, тогда как родитель блокируется при попытке чтения из STDOUT потомка. Третья - когда заполнение буферов заставляет родителя блокировать запись в STDIN потомка, а потомок блокируется при записи в STDOUT или STDERR. Первая проблема в общем случае не решается, хотя ее можно обойти, создавая таймеры функцией alarm и предотвращая перезапуск блокирующих операций при получении сигнала SIGALRM.

Мы используем модуль IO::Select, чтобы узнать, из каких файловых манипуляторов можно прочитать данные (для этой цели можно использовать встроенную функцию select). Это решает вторую, но не третью проблему. Для решения третьей проблемы также потребуются alarm и SIGALRM.

Если вы хотите отправить программе входные данные, прочитать ее вывод и затем либо прочитать, либо проигнорировать ошибки, работы заметно прибавится (см. пример 16.2).

Пример 16.2. cmd3sel

```
#!/usr/bin/perl
# cmd3sel - управление всеми тремя потоками порожденного процесса
#           (ввод, вывод и ошибки)
use IPC::Open3;
use IO::Select;

$cmd = "grep vt33 /none/such - /etc/termcap";
$pid = open3(*CMD_IN, *CMD_OUT, *CMD_ERR, $cmd);

$SIG{CHLD} = sub {
    print "REAPER: status $? on $pid\n" if waitpid($pid, 0) > 0
};

print CMD_IN "This line has a vt33 lurking in it\n";
close(CMD_IN);

$selector = IO::Select->new();
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$selector->add(*CMD_ERR,    *CMD_OUT),

while (@ready =
            $fh (@ready)
        if (fileno($fh) == fileno(CMD_ERR))
            {print 'STDERR ', scalar <CMD_ERR>}
        else
            {print "STDOUT  ', scalar <CMD_OUT>}
        $selector->remove($fh) if eof($fh),
    }
}

close(CMD_CUT),
close(CMD_ERR),
```

Мы отправляем короткую входную строку, а затем закрываем манипулятор. Тем самым предотвращается ситуация взаимной блокировки двух процессов, каждый из которых ожидает записи данных другим процессом.

Смотри также

Документация по стандартным модулям IO::Select; IPC::Open2 и IPC::Open3; описание функции `alarm` в *perlfunc(1);* рецепты **16.8; 16.15—16.16.**

16.10. Взаимодействие между родственными процессами

Проблема

Имеются два взаимосвязанных процесса, которые должны обмениваться данными. Вам требуется более высокая степень контроля по сравнению с той, что обеспечивают `open`, `system` и `` `...` ``

Решение

Воспользуйтесь `pipe`, а затем - `fork`:

```
pipe(READER, WRITER),
if (fork) {
    # Выполнить родительский код, в котором происходит либо чтение,
    # либо запись (что-то одно)
} else {
    # Выполнить код потомка, в котором происходит либо чтение,
    # либо запись (что-то одно)
}
```

Либо используйте особую форму open:

```
if ($pid = open(CHILD, "|-")) {
        # Выполнить родительский код, передающий данные потомку
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
} else {
    die "cannot fork $!" unless defined $pid;
    # Иначе выполнить код потомка, принимающий данные от родителя
}
```

Или по-другому:

```
if ($pid = open(CHILD, "-|")) {
    # Выполнить родительский код, принимающий данные от потомка
} else {
    die "cannot fork $!" unless defined $pid
    # Иначе выполнить код потомка, передающий данные родителю
}
```

Комментарий

Канал представляет собой два файловых манипулятора, связанных так, что записанные в один файловый манипулятор данные могут быть прочитаны из другого. Функция `pipe` создает два **манипулятора**, связанных в канал; первый (приемник) предназначен для чтения, а второй (передатчик) — для записи. Хотя вы не сможете взять два существующих манипулятора и объединить их в канал, функция `pipe` часто используется при обмене данными между процессами. Один процесс создает пару манипуляторов функцией `pipe`, после чего создает потомка с помощью `fork`; в результате возникают два разных **процесса**, выполняющих одну и ту же программу, каждый из которых обладает копией связанных манипуляторов.

Неважно, какой процесс будет приемником, а какой — передатчиком; когда процесс начинает играть одну из этих ролей, его напарнику достается другая. Такой обмен данными может быть только односторонним (но не бросайте читать!)

Мы воспользуемся модулем IO::Handle, в котором нас интересует метод `autoflush()` (если вы предпочитаете более эффективные решения, воспользуйтесь решением с select, описанным в главе 7). Если этого не сделать, наша отдельная строка вывода застрянет в канале и не доберется до другого конца до закрытия канала.

Версия родителя, передающего данные потомку, приведена в примере 16.3.

Пример 16.3. pipe1

```
#!/usr/bin/perl -w
# pipe1 - применение pipe и fork для отправки данных родителем потомку

use IO::Handle;
pipe(READER, WRITER);
WRITER->autoflush(1);

if ($pid = fork) {
    close READER;
    print WRITER "$pid is sending this\n";
    close WRITER;
    waitpid($pid,0);
} else {
    die "cannot fork $!" unless defined $pid;
    close WRITER;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    chomp($line = <READER>);
    print  Child Pid $$ just      this   $line
    close READER,  # Все равно это произойдет
    exit,
}
```

В примерах этого рецепта основная проверка ошибок была оставлена читателю для самостоятельной работы. Мы так поступили для того, чтобы взаимодействие функций стало более наглядным. В реальной жизни проверяются возвращаемые значения всех вызовов системных функций.

В примере 16.4 показана версия потомка, передающего данные родителю.

Пример 16.4. pipe2

```
#!/usr/bin/perl -w
# pipe2 - применение pipe и fork для передачи данных потомком родителю

use IO  Handle,
pipe(READER  WRITER),
WRITER->autoflush(1),

if ($pid = fork) {
    close  WRITER,
    chomp($line = <READER>),
                  Pid    just      this   $line
    close READER,
    waitpid($pid,0),
} else {
    die  cannot fork  $!  unless defined $pid,
  • close READER,
    print WRITER  Child Pid $$ is sending this\n ,
    close WRITER,  # Все равно это произойдет
    exit,
}
```

Обычно обе половины входят в цикл и приемник продолжает читать до конца файла. Это происходит до техпор, пока передатчик не закроет канал или не завершится.

Поскольку манипуляторы каналов работают лишь в одном направлении, каждый процесс использует лишь один канал из пары и закрывает неиспользуемый манипулятор. Причина, по которой это делается, нетривиальна; представьте себе ситуацию, при которой принимающий процесс не закрыл передающий манипулятор. Если после этого передающий процесс завершится, пока принимающий процесс пытается что-нибудь прочитать, последний намертво "зависнет". Система не может сообщить приемнику о том, что данных для чтения больше не будет, пока не будут закрыты все копии передающего манипулятора.

Функция open, получая в качестве второго аргумента -| или |=, неявно вызывает pipe и fork. Это несколько упрощает приведенный выше фрагмент. Порожденный процесс общается с родителем через STDIN или STDOUT в зависимости от того, какая строка была использована, -| или |-.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При подобном применении open, когда родитель хочет передать данные потомку, он использует нечто похожее на пример 16.5.

Пример 16.5. pipe3

```
#!/usr/bin/perl -w
# pipe3 - применение разветвляющего вызова open
#         для передачи данных от родителя к потомку

use IO  Handle,
if ($pid = open(CHILD,  |- )) {
    CHILD->autoflush(1),
                    t Pid $$ is sending this\n ,
    close(CHILD),
} else {
    die  cannot fork  $!  unless defined $pid,
    chomp($line = <STDIN>),
    print  Child Pid $$ ]ust read  this   $line \n ,
    exit,
}
```

Поскольку STDIN потомка уже подключен к родителю, потомок может запустить через exec другую программу, читающую данные из стандартного ввода — например, *lpr*. Это полезная и часто используемая возможность.

Если потомок захочет передать данные родителю, он делает нечто похожее на пример **16.6.**

Пример 16.6. pipe4

```
#!/usr/bin/perl -w
# pipe4 - применение разветвляющего вызова open
#         для передачи данных от потомка к родителю

use IO  Handle,
if ($pid = open(CHILD,  -| )) {
    chomp($line = <CHILD>),
                Pid    just      this   $line
    close(CHILD),
} else <
    die  cannot fork  $!  unless defined $pid,
    STDOUT->autoflush(1),
    print STDOUT  Child Pid $$ is sending this\n ,
    exit,
}
```

И снова, поскольку **STDOUT** потомка уже подключен к родителю, потомок может запустить через exec другую программу, выдающую нечто интересное в его стандартный вывод. Эти данные также будут пфеданы родителю как ввод от <CHILD>.

При подобном использовании open мы не обязаны вручную вызывать waitpid, поскольку не было явного вызова fork. Однако close вызвать все же надо. В обоих случаях переменная $? содержит статус ожидания порожденного процесса (о том, как интерпретировать это значение, рассказано в рецепте 16.19).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В предыдущих примерах рассматривалась однонаправленная связь. Что делать, если вы хотите, чтобы данные передавались в обе стороны? Дважды вызовите pipe перед вызовом fork. Вам придется следить за тем, кто, что и когда передает, иначе может возникнуть взаимная блокировка (см. пример 16.7).

Пример 16.7. pipe5

```
#!/usr/bin/perl -w
# pipe5 - двусторонний обмен данными с использованием двух каналов
#         без применения socketpair
use IO Handle,
pipe(PARENT_RDR, CHILD_WTR),
pipe(CHILD_RDR,  PARENT_WTR),
CHILD_WTR->autoflush(1),
PARENT_WTR->autoflush(1),

if ($pid = fork) {
    close PARENT_RDR, close PARENT_WTR,
          CHILD_WTR        Pid    is         this\n ,
    chomp($line = <CHILD_RDR>),
    print         Pid    just      this   $line
    close CHILD_RDR, close CHILD_WTR,
    waitpid($pid,0),
} else {
    die cannot fork $! unless defined $pid,
    close CHILD_RDR, close CHILD_WTR,
    chomp($line = <PARENT_RDR>),
    print Child Pid $$ just read this   $line \n ,
    print PARENT_WTR Child Pid $$ is sending this\n ,
    close PARENT_RDR, close PARENT_WTR,
    exit,
}
```

Ситуация усложняется. Оказывается, существует специальная системная функция socketpair (см пример 16.8), которая упрощает предыдущий пример. Она работает аналогично pipe, за исключением того, что оба манипулятора могут использоваться как для приема, так и для передачи.

Пример 16.8. pipe6

```
#!/usr/bin/perl -w
# pipe6 - двусторонний обмен данными с применением socketpair

use Socket,
use IO Handle,
# Мы говорим AF_UNIX, потому что хотя константа *_LOCAL
# соответствует POSIX 1003 1g, на многих компьютерах
# она еще не поддерживается
socketpair(CHILD, PARENT, AF_UNIX, SOCK_STREAM, PF_UNSPEC)
    or die "socketpair $!";
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 16.8 (продолжение)

```
CHILD->autoflush(1)
PARENT->autoflush(1),

if ($pid = fork) {
    close PARENT,
    print              Pid    is sending this\n ,
    chomp($line = <CHILD>),
    print        Pid    just      this   $line
    close CHILD,
    waitpid($pid 0),
} else {
    die cannot fork $! unless defined $pid,
    close CHILD,
    chomp($line = <PARENT>),
    print Child Pid $$ just      this   $line
    print PARENT Child Pid $$ is sending this\n ,
    close PARENT,
    exit,
}
```

В некоторых системах каналы исторически были реализованы как два полузакрытых конца пары сокетов. Фактически реализация pipe(READER,WRITER) выглядела так:

```
socketpair(READER, WRITER, AF_UNIX, SOCK_STREAM, PF_UNSPEC),
shutdown(READER, 1),          # Запретить запись для READER
shutdown(WRITER, 0),          # Запретить чтение для WRITER
```

В ядрах **Linux** до версии **2.0.34** системная функция shutdown(2) работала неверно. Приходилось запрещать чтение для READER и запись для WRITER.

Смотри также

Описания всех использованных функций в *perlfunc(1);* документация по стандартному модулю IPC::Open2; рецепт **16.8.**

16.11. Имитация файла на базе именованного канала

Проблема

Вы хотите, чтобы процесс перехватывал все обращения к файлу. Например, файл *~/.plan* должен превратиться в программу, которая будет возвращать случайную цитату.

Решение

Воспользуйтесь именованными каналами. Сначала создайте канал (вероятно, в командном интерпретаторе):

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
% mkfifo /path/to/named pipe
```

Принимающий фрагмент выглядит так:

```
open(FIFO, < /path/to/named pipe )          or die $!,
while (<FIFO>) {
    print Got $_ ,
}
close(FIFO),
```

Передающий фрагмент выглядит так:

```
open(FIFO, '> /path/to/named pipe )         or die $1;
print FIFO 'Smoke this \n ,
close(FIFO),
```

Комментарий

Именованный канал (также встречается термин FIFO) представляет собой специальный файл, используемый в качестве буфера для взаимодействия процессов на одном компьютере. Обычные каналы также позволяют процессам обмениваться данными, но они должны наследовать файловые манипуляторы от своих родителей. Для работы с именованным каналом процессу достаточно знать его имя. В большинстве случаев процессы даже не обязаны сознавать, что они читают данные из FIFO.

Операции чтения и записи для именованных каналов выполняются точно так же, как и для обычных файлов (в отличие от сокетов UNIX, рассматриваемых в главе 17). Данные, записанные в FIFO, буферизуются операционной системой, а затем читаются обратно в порядке записи. Поскольку FIFO играет роль буфера для взаимодействия процессов, открытие канала для чтения блокирует его до техпор, пока другой процесс не откроет его для записи, и наоборот. Если отк рыть канал для чтения и записи с помощью режима +< функцииopen, блокировки (в большинстве систем) не будет, поскольку ваш процесс сможет и принимать, и передавать данные.

Давайте посмотрим, как использовать именованный канал, чтобы при каждом запуске finger люди получали разные данные. Чтобы создать именованный канал с именем *.plan* в основном каталоге, воспользуйтесь *mkfifo* или *mknod*:

```
% mkfifo ~/ plan          # Есть практическивезде
% mknod  ^/ plan p       # На случай, если у вас все же нет mkfifo
```

В некоторых системах приходится использовать *mknod*(8). Имена и местонахождение этих программ могут быть другими — обращайтесь к системной документации.

Затем необходимо написать программу, которая будет поставлять данные программам, читающим из файла *~/.plan*. Мы ограничимся выводом текущей даты и времени (см. пример 16.9).

Пример 16.9. dateplan

```
#!/usr/bin/perl -w
# dateplan - вывод текущей даты и времени в файл plan
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 16.9 (продолжение)

```
while (1) {
    open(FIFO  > $ENV{HOME}/ plan )
        or die  Couldn t open $ENV{HOME}/ plan for writing  $!\n ,
                                time is    scalar(localtime),
    close  FIFO,
    sleep 1,
}
```

К сожалению, такое решение работает не всегда, потому что некоторые варианты *finger* и соответствующие демоны проверяют разйер файла *.plan* перед тем, как пытаться читать из него. Поскольку именованные каналы в файловой системе представлены в виде специальных файлов нулевого размера, некоторые клиенты и серверы не станут открывать именованный канал и читать из него, и наш фокус не удастся.

В примере с *.plan* демон был передатчиком. Приемники-демоны тоже встречаются не так уж редко. Например, именованный канал может применяться для ведения централизованного журнала, собирающего данные от нескольких процессов. Программа-сервер читает сообщения из именованного канала и записывает их в базу данных илифайл. Клиенты передают сообщения в именованный канал. Такая схема избавляет клиентов от хлопот, связанных с логикой передачи данных, и позволяет легко внести необходимые изменения в реализацию механизма передачи.

В примере 16.10 приведена простая программа для чтения двухстрочных блоков, где первая строка определяет процесс, а вторая — текст сообщения. Все сообщения от httpd игнорируются, а сообщения от login сохраняются в */var/log/login*.

Пример 16.10, fifolog

```
#!/usr/bin/perl -w
# fifolog -  чтение и сохранение сообщений из FIFO

use IO  File,

$SIG{ALRM}  =  sub  <  close(FIFO)  },    # Переход к следующему
                                          # процессу в очереди

while  (1)  {
    alarm(0),                             # Отключить таймер
    open(FIFO,   <  /tmp/log )            or die  Can t open /tmp/log    $!\n ,
    alarm(1),                             # 1 секунда на регистрацию

    $service  =  <FIFO>,
    next  unless  defined  $service,      # Прерывание или нечего регистрировать
    chomp  $service,

    $message  =  <FIFO>,
    next  unless  defined  $message,      # Прерывание или нечего регистрировать
    chomp  $message,

    alarm(0),                             # Отключить таймеры
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                                    # для обработки сообщений

    if ($service eq "http") {
        В Игнорировать
    } elsif ($service eq "login") {
        # Сохранить в /var/log/login
        if ( open(LOG, ">> /tmp/login") ) {
            print LOG scalar(localtime), " $service $message\n";
            close(LOG),
        } else {
            warn "Couldn't log $service $message to /var/log/login : $!\n";
        }
    }
}
```

Программа получилась сложнее предыдущей по нескольким причинам. Прежде всего, мы не хотим, чтобы наш сервер ведения журнала надолго блокировал передатчики. Нетрудно представить ситуацию, при которой злонамеренный или бестолковый передатчик открывает именованный канал для записи, но не передает полного сообщения. По этой причине мы используем `alarm` и SIGALRM для передачи сигналов о нарушениях во время чтения.

При использовании именованных каналов могут возникнуть лишь два исключительных состояния: когда у приемника исчезает передатчик, и наоборот. Если процесс читает из именованного канала, а передатчик закрывает его со своего конца, то принимающий процесс получит признак конца файла (<> возвращает `undef`). Однако если приемник отключается от канала, то приследующей попытке записи передатчик получит сигнал SIGPIPE. Если игнорировать сигналы о нарушении канала конструкцией `$SIG{PIPE}='IGNORE'`, `print` возвращает `false`, а переменной $! присваивается значение EPIPE:

```
use POSIX qw(:errno_h);

$SIG{PIPE} = 'IGNORE';
# ..
$status = print FIFO "There?\n";
if (!$status && $! == EPIPE) {
    warn "My forsaken me!\n";
    next;
}
```

Возможно, у вас возник вопрос: "Если сто процессов одновременно пытаются передать данные серверу, как можно быть уверенным в том, что я получу сто разных сообщений, а не хаотическую мешанину из символов или строк разных процессов?" Хороший вопрос. Согласно стандарту **POSIX,** запись менее PIPE_BUF байт будет доставлена автоматически, то есть не перепутается с другими. Значение константы PIPE_BUF можно узнать из модуля **POSIX:**

```
use POSIX;
print _POSIX_PIPE_BUF, "\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

К счастью, стандарт POSIX также требует, чтобы значение PIPE_BUF было *не менее* 512 байт. Следовательно, остается лишь позаботиться о том, чтобы клиенты не пытались передавать более 512 байт за раз.

Но что если вам понадобилось зарегистрировать более 512 байт? Разделите каждое большое сообщение на несколько маленьких (менее 512 байт), снабдите каждое сообщение уникальным идентификатором клиента (например, идентификатором процесса) и организуйте их сборку на сервере. Нечто похожее происходит при разделении и сборке сообщений TCP/IP.

Один именованный канал не обеспечивает двухстороннего обмена данными между передатчиком и приемником, что усложняет аутентификацию и другие способы борьбы с передачей ложных сообщений (если не делает их невозможными). Вместо того чтобы упрямо втискивать эти возможности в модель, в которой они неуместны, лучше ограничить доступ к именованному каналу средствами файловой системы (на уровне прав владельца и группы).

Смотри также

Страницы руководства *mkfifo*(8) или *mknod*(8) (если они есть); рецепт 17.6.

16.12. Совместное использование переменных в разных процессах

Проблема

Требуется организовать совместный доступ к переменным в разветвлениях или неродственных процессах.

Решение

Используйте средства SysV IPC, если ваша система их поддерживает.

Комментарий

Хотя средства SysV IPC (общая память, семафоры и т. д.) реже используются в межпроцессных коммуникациях, нежели каналы, именованные каналы и сокеты, они все же обладают рядом интересных свойств. Тем не менее для совместного использования переменной несколькими процессами обычно нельзя рассчитывать на работу с общей памятью через shmget или mmap(2). Дело в том, что Perl заново выделит память под строку тогда, когда вы этого совсем не ждете.

Проблема решается с помощью с CPAN. Умный модуль tie, общая память с CPAN позволяют организовать совместный доступ к структурам данных произвольной сложности для процессов на одном компьютере. При этом процессы даже не обязаны быть родственными.

В примере 16.11 продемонстрирован несложный случай применения этого модуля.

Пример 16.11.

```
#!/usr/bin/perl
                совместный доступ к общим переменным в разветвлениях
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$handle = tie                          undef, { destroy => 1 },
$SIG{INT} = sub { die  $$ dying\n  },

for (1   10) {
    unless ($child = fork) {        # Я - потомок
        die  cannot Tork  $!  unless defined $child,
        squabble(),
        exit,
    }
    push @kids, $child,  # Если нас интересуют идентификаторы процессов
}

while (1) {
    print  Buffer is $buffer\n ,
    sleep 1,
}
die

sub squabble {
    my $1 = 0,
    while (1) {
        next if $buffer =~ /^$$\b/o,
        $handle->shlock(),
        $i++,
        $buffer =  $$ $1 ,
        $handle->shunlock(),
    }
}
```

Исходный процесс создает общую переменную, разветвляется на 10 потомков, а затем выводит значение буфера примерно каждую секунду в бесконечном цикле или до тех пор, пока вы не нажмете Ctrl+C.

Поскольку обработчик SIGINT был установлен до всех вызовов `fork`, его наследуют все потомки, которые также уничтожаются при прерывании группы процессов. Сигналы с клавиатуры передаются целой группе процессов, а не одному процессу.

Что же происходит в `squabble`? Потомки разбираются, кому из них удастся обновить общую переменную. Каждый порожденный процесс смотрит, изменилось ли состояние переменной с момента последнего визита. Если буфер начинается с его собственной сигнатуры (идентификатора процесса), процесс не трогает его. Если буфер был изменен кем-то другим, процесс блокирует общую переменную вызовом специального метода для манипулятора, полученного от `tie`, обновляет ее и снимает блокировку.

Программа заработает намного быстрее, если закомментировать строку, начинающуюся с `next`, где каждый процесс проверяет, кто последним прикасался к буферу.

Шаблон `/^$$\b/o` выглядит подозрительно, поскольку `/o` указывает на однократную компиляцию шаблона, а переменная `$$` меняется при разветвлении. Впрочем,

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

значение фиксируется не во время компиляции программы, а при первой компиляции шаблона в каждом процессе, во время жизни которого $$ остается постоянным.

также поддерживает совместное использование переменных не-родственными процессами на одном компьютере. За подробностями обращайтесь к документации.

Смотри также

Описание функций semctl, semget, semop, shmctl, shmget, shmread и shmwrite в *perlfunc(1);* документация по модулю IP с CPAN.

16.13. Получение списка сигналов

Проблема

Вы хотите знать, какие сигналы поддерживаются вашей операционной системой.

Решение

Если ваш командный интерпретатор поддерживает встроенную команду ***kill -/***, используйте ее:

```
% kill -1
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE
ALRM TERM CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM
PROF WINCH POLL PWR
```

Чтобы сделать то же самое только на Perl версии **5.004** и выше, выведите ключи хэша %SIG:

```
% perl -e 'print join(" ", keys %SIG), "\n"'
XCPU ILL QUIT STOP EMT ABRT BUS USR1 XFSZ TSTP INT IOT USR2 INFO TTOU
ALRM KILL HUP URG PIPE CONT SEGV VTALRM PROF TRAP IO TERM WINCH CHLD
FPE TTIN SYS
```

До выхода версии **5.004** приходилось использовать модуль Config:

```
% perl -MConfig -e 'print $Config{sig_name}'
ZERO HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS PIPE ALRM
TERM URG STOP TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH
INFO USR1 USR2 IOT
```

Комментарий

Если вы работаете в Perl версии младше **5.004**, для получения списка сигналов вам также придется использовать @signame и %signo модуля Config, поскольку конструкция keys %SIG в ранних версиях еще не реализована.

Следующий фрагмент извлекает имена и номера доступных сигналов из стандартного модуля ***Config.pm***. Индексирование @signame по номеру дает имя сигнала, а индексирование %signo по имени — номер сигнала.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Config,
defined $Config{sig_name} or die  No sigs? ,
$i = 0,                    # Config добавляет ложный сигнал 0
                            # с именем  ZERO
        $name (split(     $Config{sig_name}))
    $signo{$name} = $i,
    $signame[$i] = $name,
    $i++,
}
```

Смотри также

Документация по стандартному модулю Config; раздел "Signals" *perlipc(1)*.

16.14. Посылка сигнала

Проблема

Требуется послать сигнал процессу. Возможна посылка сигнала как вашему собственному процессу, так и другому процессу в той же системе. Например, вы перехватили SIGINT и хотите передать его потомкам.

Решение

Функция `kill` отправляет сигнал с заданным именем или номером процессам, идентификаторы которых перечисляются в качестве остальных аргументов:

```
kill  9     => $pid,        # Послать $pid сигнал 9
kill -1     => $pgrp,       # Послать всему заданию сигнал 1
kill  USR1  => $$,          # Послать себе SIGUSR1
kill  HUP   => @pids,       # Послать SIGHUP процессам из @pids
```

Комментарий

Функция Perl `kill` обеспечивает интерфейс к системной функции с тем же именем. Первый аргумент определяет посылаемый сигнал и задается по номеру или по имени; остальные аргументы определяют идентификаторы процессов, которым отправляется сигнал. Функция возвращает количество процессов, успешно получивших сигнал. Сигналы можно отправлять только процессам, для которых реальный или сохраненный идентификатор пользователя совпадает с вашим реальным или текущим идентификатором — если только вы не являетесь привилегированным пользователем.

Если номер сигнала отрицателен, Perl интерпретирует остальные аргументы как идентификаторы групп процессов и отправляет сигнал процессам, входящим в эти группы, с помощью системной функции `killpg`(2).

Группа процессов фактически представляет собой задание. Именно так операционная система объединяет родственные процессы. Например, когда вы с помощью командного интерпретатора сцепляете две команды, при этом запускаются два процесса, но лишь одно задание. Когда текущее задание прерывается по Ctrl+C

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

или приостанавливается по Ctrl+Z, соответствующие сигналы отправляются всему заданию, которое может состоять из нескольких процессов.

Функция `kill` также позволяет проверить, жив ли процесс. Посылка специального псевдосигнала с номером 0 сообщает, можно ли послать сигнал процессу — хотя сам сигнал при этом не передается. Если функция возвращает `true`, процесс жив. Если возвращается `false`, процесс либо сменил свой действующий идентификатор (в этом случае переменной `$!` присваивается EPERM), либо прекратил существование (`$!` присваивается ESRCH). Для процессов-зомби (см. рецепт 16.19) также возвращается ESRCH.

```
use POSIX qw( errno_h),

if (kill 0 => $minion) {
    print  $minion is alive!\n ,
} elsif ($! == EPERM) {             # Изменился UID
    print  $minion has escaped my control!\n ,
} elsif ($! == ESRCH) {
    print  $minion is deceased \n    # Или зомби
} else {
    warn  Odd,  I couldn t check on the status of $minion  $!\n ,
}
```

Смотри также

Раздел "Signals" *perlipc*(1); страницы руководства *sigaction*(2), *signal*(3) и *kill(2)* вашей системы (если есть); описание функции `kill` в *perlfunc*(1).

16.15. Установка обработчика сигнала

Проблема

Вы хотите управлять реакцией программы на сигналы. Это может понадобиться для перехвата Ctrl+C, избежания накопления завершившихся подпроцессов или предотвращения гибели вашего процесса при попытке передать данные исчезнувшему потомку.

Решение

Воспользуйтесь хэшем %SIG для установки обработчика по имени или ссылке на код:

```
$SIG{QUIT} = \&got_sig_quit,      # Вызвать &got_sig_quit
                                   # для каждого SIGQUIT
$SIG{PIPE} =  got_sig_pipe ,      # Вызвать main  got_sig_pipe
                                   # для каждого SIGPIPE
$SIG{INT}  = sub { $ouch++ },     # Увеличить $ouch для каждого SIGINT
```

Хэш %SIG также позволяет игнорировать сигнал:

```
$SIG{INT} =  IGNORE ,             # Игнорировать сигнал INT
```

Также есть возможность восстановить стандартный обработчик сигнала:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$SIG{STOP} = 'DEFAULT';          # Восстановить стандартный обработчик
                                 # сигнала STOP
```

Комментарий

Хэш %SIG используется в Perl для управления тем, что происходит при получении сигналов. Каждый ключ %SIG соответствует определенному сигналу, а значение — действию, которое должно предприниматься при его получении. В Perl предусмотрены два особых ассоциированных значения: "IGNORE" означает, что при получении сигнала не следует выполнять никаких действий, а "DEFAULT" выполняет стандартные действия UNIX для данного сигнала.

Хотя программисты на C привыкли к термину SIGINT, в Perl используется только INT. Предполагается, что имена сигналов используются только в функциях, связанных с обработкой сигналов, поэтому префикс SIG оказывается лишним. Следовательно, чтобы изменить действия вашего процесса при получении сигнала SIGCHLD, следует присвоить значение $SIG{CHLO}.

Чтобы ваш код выполнялся при получении конкретного сигнала, в хэш заносится либо ссылка на код, либо имя функции (следовательно, при сохранении строки вам не удастся использовать обработчик с именем IGNORE или DEFAULT; впрочем, для обработчика сигнала эти имена выглядят довольно странно). Если имя функции не содержит информации о пакете, Perl считает, что функция принадлежит пакету main::, а не тому пакету, в котором обработчик был установлен. Ссылка на код относится к конкретному пакету, и этот вариант считается предпочтительным.

Perl передает коду обработчика один аргумент: имя сигнала, по которому он вызывается (например, "INT" или "USR1"). При выходе из обработчика продолжается выполнение действий, выполнявшихся в момент поступления сигнала.

Perl определяет два специальных сигнала, DIE и WARN. Обработчики этих сигналов вызываются каждый раз, когда программа на Perl выводит предупреждение (warn) или умирает (die). Это позволяет нам перехватывать предупреждения и по своему усмотрению обрабатывать их или передавать дальше. На время своего выполнения обработчики die и wa'rn отключаются, поэтому вы можете спокойно вызвать die в обработчике DIE или warn в обработчике WARN, не опасаясь рекурсии.

Смотри также

Раздел "Signals" *perlipc(1);* страницы руководства *sigaction(2), signal*(3) и *kill*(2) вашей системы (если есть).

16.16. Временное переопределение обработчика сигнала

Проблема

Вы хотите установить обработчик сигнала, действующий только на время выполнения конкретной подпрограммы. Например, ваша подпрограмма перехватывает

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

сигнал SIGINT, но за ее пределами SIGINT должен обрабатываться обычными средствами.

Решение

Используйте local для временного переопределения обработчика:

```
# Обработчик сигнала
sub ding {
    $SIG{INT} = \&ding;
    warn "\aEnter your name!\n";
}

# Запросить имя с переопределением SIGINT
sub get_name {
    local $SIG{INT} = \&ding;
    my $name;

    print "Kindly Stranger, please enter your name: ";
    chomp( $name = o );
           $name;
}
```

Комментарий

Для временного сохранения одного элемента %SIG необходимо использовать local, а не my. Изменения продолжают действовать во время выполнения блока, включая все, чтоможет быть вызвано из него. В приведенном примере это подпрограмма get_name. Если сигнал будет доставлен во время работы другой функции, вызванной вашей функцией, сработает ваш обработчик сигнала — если только вызванная подпрограмма не установила собственный обработчик. Предыдущее значение элемента хэша автоматически восстанавливается при выходе из блока. Это один из немногочисленных случаев, когда динамическая область действия оказывается скорее удобной, нежели запутанной.

Смотри также

Рецепты 10.13; 16.15; 16.18.

16.17. Написание обработчика сигнала

Проблема

Требуется написать подпрограмму, которая будет вызываться программой при каждом получении сигнала.

Решение

Обработчик сигнала представляет собой обычную подпрограмму. С некоторой степенью риска в обработчике можно делать все, что допустимо в любой другой подпрограмме Perl, но чем больше вы делаете, тем больше рискуете.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В некоторых системах обработчик должен переустанавливаться после каждого сигнала:

```
$SIG{INT} = \&got_int;
sub got_int {
    $SIG{INT} = \&got_int,          # Но не для SIGCHLD!
    Я ..
}
```

Некоторые системы перезапускают блокирующие операции (например, чтение данных). В таких случаях необходимо вызвать в обработчике die и перехватить вызов eval:

```
my $interrupted = 0,

sub got_int {
    $interrupted = 1;
    $SIG{INT> = 'DEFAULT';          # или 'IGNORE'
    die,
}

eval {
    $SIG{INT} = \&got_int;
    # ... Длинный код, который нежелательно перезапускать
};

if ($interrupted) {
    # Разобраться с сигналом
}
```

Комментарий

Установка собственного обработчика сигнала напоминает игру с огнем: это очень интересно, но без исключительной осторожности вы рано или поздно обожжетесь. Создание кода Perl, предназначенного для обработки сигналов, чревато двумя опасностями. Во-первых, многие библиотечные функции нереентерабельны. Если сигнал прерывает выполнение какой-то функции (например, malloc(3) или printf(3)), а ваш обработчик сигнала снова вызовет ее, результат окажется непредсказуемым — обычно работа программы прерывается с выводом в файл содержимого памяти dump). Во-вторых, на нижних уровнях нереентерабелен сам Perl (версия 5.005 будет поддерживать облегченные процессы, называемые *нитями* но на момент издания этой книги она еще не вышла). Если сигнал прерывает Perl в момент изменения его собственных внутренних структур данных, результат тоже непредсказуем — как правило, выдаются случайные дампы.

Перед вами открываются два пути: параноидальный и практический. Параноик постарается ничего не делать внутри обработчика сигнала; примером служит код с eval и die в решении — мы присваиваем значение переменной и тут же выходим из обработчика. Но даже это покажется слишком рискованным настоящему параноику, который избегает die в обработчиках — *вдруг* система на что-нибудь обидится? Практический подход - вы говорите: "Кто не рискует, тот не выигрывает", — и делаете в обработчике все, что заблагорассудится.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Сигналы были реализованы во многих операционных системах, причем не всегда одинаково. Отличия в реализации сигналов чаще всего проявляются в двух ситуациях: когда сигнал происходит во время активности обработчика *(надежность)* и когда сигнал прерывает блокирующий вызов системной функции типа *(перезапуск)*.

Первоначальная реализация сигналов была ненадежной. Это означало, что во время работы обработчика при других поступлениях сигнала происходило некоторое стандартное действие (обычно аварийное завершение программы). Новые системы решают эту проблему (конечно, каждая — в своем, слегка особом стиле), позволяя подавлять другие экземпляры сигналов с данным номером до завершения обработчика. Если Perl обнаружит, что ваша система может использовать надежные сигналы, он генерирует соответствующие вызовы системных функций, чтобы программы вели себя более логично и безопасно. Система сигналов POSIX позволяет запретить доставку сигналов и в другие моменты времени (см. рецепт 16.20).

Чтобы получить по-настоящему переносимый код, программист-параноик заранее предполагает самое худшее (ненадежные сигналы) и вручную переустанавливает обработчик сигналов, обычно в самом начале функции:

```
$SIG{INT} = \&catcher;
sub catcher {
    # ...
    $SIG{INT} = \&catcher;
}
```

Особый случай перехвата SIGCHLD описан в рецепте 16.19. System V ведет себя очень странно и может сбить с толку.

Чтобы узнать, располагаете ли вы надежными сигналами, воспользуйтесь модулем Config:

```
use Config;
print "Hurrah!\n" if $Config{d_sigaction};
```

Наличие надежных сигналов еще не означает, что вы автоматически получаете надежную программу. Впрочем, без них программа заведомо окажется ненадежной.

Первые реализации сигналов прерывали медленные вызовы системных функций, которые требовали взаимодействия со стороны других процессов или драйверов устройств. Если сигнал поступает во время выполнения этих функций, они (и их аналоги в Perl) возвращают признак ошибки и присваивают коду ошибки EINTR, "Interrupted system call". Проверка этого условия настолько усложняет программу, что во многих случаях это вообще не делается, поэтому при прерывании сигналом медленных системных функций программа начинает вести себя неверно или аварийно завершается. Большинство современных версий UNIX позволяет изменить ход событий. Perl всегда делает системные функции перезапускаемыми, если эта возможность поддерживается системой. В системах POSIX можно управлять перезапуском с помощью модуля POSIX (см. рецепт 16.20).

Чтобы узнать, будет ли прерванная системная функция автоматически перезапущена, загляните в заголовочный файл signal.h вашей системы:

```
'S[AV]_(RESTART|INTERRUPT)' /usr/include/*/signal.h
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Два сигнала не перехватываются и не игнорируются: SIGKILL и SIGSTOP. Полная информация о сигналах вашей системы и об их значении приведена в странице руководства *signal*(3).

Смотри также

Раздел "Signals" *perlipc(1);* страницы руководства *sigaction*(2), *signal*(3) и *kill*(2) вашей системы (если есть).

16.18. Перехват Ctrl+C

Проблема

Требуется перехватить нажатие Ctrl+C, приводящее к остановке работы программы. Вы хотите либо игнорировать его, либо выполнить свою собственную функцию при получении сигнала.

Решение

Установите обработчик для SIGINT. Присвойте ему "IGNORE", чтобы нажатие Ctrl+C игнорировалось:

```
$SIG{INT} = 'IGNORE';
```

Или установите собственную подпрограмму, которая должна реагировать на Ctrl+C:

```
$SIG{INT} = \&tsktsk;

sub tsktsk {
    $SIG{INT} = \&tsktsk;      # См. "Написание обработчика сигнала"
    warn "\aThe long habit of living indisposeth us for dying.\n";
}
```

Комментарий

Ctrl+C не влияет на вашу программу напрямую. Драйвер терминала, обрабатывающий нажатия клавиш, опознает комбинацию Ctrl+C (или другую комбинацию, заданную вами в качестве символа прерывания при настройке параметров терминала) и посылает SIGINT каждому процессу активной группы *(активного задания)* данного терминала. Активное задание обычно состоит из всех программ, запущенных отдельной строкой в командном интерпретаторе, а также всех программ, запущенных этими программами. За подробностями обращайтесь к разделу введения "Сигналы".

Символ прерывания — не единственный служебный символ, интерпретируемый драйвером терминала. Текущие параметры терминала можно узнать с помощью команды stty -a:

```
% stty -a
speed 9600 baud; 38 rows; 80 columns;
lflags: icanon isig iexten echo echoe -echok echoke -echonl echoctl
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        -echoprt -altwerase -noflsh -tostop -flusho pendin -nokerninfo
        -extproc
iflags: -istrip icrnl -inlor -igncr ixon -ixoff ixany imaxbel -ignbrk
        brkint -inpok -ignpar -parmrk
oflags: opost onlcr oxtabs
        d cs8          -parodd hupcl   looal -cstopb -crtscts -dsrflow
        -dtrflow -mdmbuf
cchars: discard = ^0; dsusp = ^Y; eof = ^D; eol • <undef;>
        eol2 = <undef; erase = ^H; intr = ^C; kill * ^U; lnext = ^V;>
        min                                start  ^Q; status  <undef;>
        stop = ^S; susp = ^Z; time • 0; werase * ^W;
```

В последней секции, cchars:, перечисляются служебные символы. В рецепте 15.8 показано, как изменить в сценарии без вызова программы stty.

Смотри также

Страница руководства *stty*(1) вашей системы (если есть); рецепты 15.8; 16.17.

16.19. Уничтожение процессов-зомби

Проблема

Программа создает порожденные процессы с помощью fork. Зомби накапливаются, забивают таблицу процессов и раздражают системного администратора.

Решение

Если вам не нужно регистрировать завершившихся потомков, используйте:

```
$SIG{CHLD} = 'IGNORE',
```

Чтобы следить за умирающими потомками, установите обработчик SIGCHLD с вызовом waitpid:

```
use POSIX  sys_wait_h ,

$SIG{CHLD} = \&REAPER,
sub REAPER {
    my $stiff,
    while ($stiff * waitpid(-1, &WNOHANG) > 0) {
        # Обработать $stiff, если нужно
    }
    $SIG{CHLD} = \&REAPER,       # Установить "после* вызова waitpid
}
```

Комментарий

Когда процесс завершается, система оставляет его в таблице процессов, чтобы родитель мог проверить его статус, то есть узнать, как завершился потомок, нормально или аварийно. Определение статуса потомка (после которого он получает воз-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

можность навсегда покинуть систему) называется "чисткой" этом рецепте приведены различные рекомендации по чисткезомби. В процессе чистки используется вызов wait или waitpid. Некоторые функции Perl (конвейерные вызовы open, system и `...`) автоматически вычищают созданных ими потомков, но при запуске другого процесса с помощью fork вам придется дожидаться его завершения.

Чтобы избежать накопления зомби, достаточно сообщить системе, что они вас не интересуют. Для этого $SIG{CHLD} присваивается значение "IGNORE". Если вы хотите узнать, когда скончался тот или иной потомок, необходимо использовать waitpid.

Функция waitpid вычищает один процесс. Ее первый аргумент определяет идентификатор процесса (-1 означает любой процесс), а второй — набор флагов. Флаг WNOHANG заставляет waitpid немедленно вернуть 0, если нет ни одного мертвого потомка. Флаг 0 поддерживается всеми системами и означает блокирующий вызов. Вызов waitpid в обработчике SIGCHLD (см. решение) вычищает потомков сразу после их смерти.

Функция wait тоже вычищает потомков, но она вызывается только в блокирующем режиме. Если случайно вызвать ее при наличии работающих потомков, ни один из которых не умер, программа приостанавливается до появления зомби.

Поскольку ядро следит за недоставленными сигналами посредством битового вектора (по одному биту на сигнал), если до перехода вашего процесса в активное состояние умрут два потомка, процесс все равно получит один сигнал SIGCHLD. Чистка в обработчике SIGCHLD всегда выполняется в цикле, поэтому wait использовать нельзя.

И wait и waitpid возвращают идентификатор только что вычищенного процесса и присваивают $? его статус ожидания. Код статуса в действительности состоит из двух 8-разрядных значений, объединенных в одном 16-разрядном числе. Старший байт определяет код возврата процесса. Младшие 7 бит определяют номер сигнала, убившего процесс, а 8-й бит показывает, произошла ли критическая ошибка. Составляющие можно выделить следующим образом:

```
$exit_value  = $? >> 8;
$signal_num  = $? & 127;
$dumped_core = $? & 128;
```

Стандартный модуль POSIX содержит специальные макросы для выделения составляющих статуса: WIFEXITED, WEXITSTATUS, WIFSIGNALLED и WTERMSIG. Как ни странно, POSIX не содержит макроса для определения того, произошла ли критическая ошибка.

При использовании SIGCHLD необходимо помнить одвух обстоятельствах. Во-первых, сигнал SIGCHLD посылается системой не только при завершении потомка; сигнал также посылается при остановке. Процесс может остановиться по многим причинам — он может ожидать перехода в активное состояние для выполнения терминального ввода/вывода, получить сигнал SIGSTOP (после чего будет ожидать SIGCONT для продолжения работы) или быть приостановленным с терминала. Проверьте статус функцией WIFEXITED[1] модуля POSIX, чтобы убедиться, что процесс действительно умер, а не был остановлен:

[1] Но не SPOUSEXITED, даже на PC.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use POSIX qw( signal_h  errno_h),

$SIG{CHLD}  =  \&REAPER,
sub REAPER {
    my $pid,

    $pid = waitpid(-1, &WNOHANG),

    if ($pid == -1) {
        # Ожидающих потомков нет  Игнорировать
    } elsif (WIFEXITED($?)) {
        print  Process $pid exited \n ,
    } else {
        print  False alarm on $pid \n ,
    }
    $SIG{CHLD} = \&REAPER,          # На случай ненадежных сигналов
}
```

Вторая ловушка, связанная с SIGCHLD, относится к Perl, а не к операционной системе. Поскольку system, open и ' ' запускают подпроцессы через fork, а операционная система отправляет процессу SIGCHLD при выходе из любого подпроцесса, вызов обработчика может быть и непредвиденным. Встроенные операции сами ожидают завершения потомков, поэтому иногда SIGCHLD прибывает до того, как вызов close для манипулятора заблокирует его для чистки. Если первым до него доберется обработчик сигнала, то к моменту нормального закрытия зомби уже не будет. В результате close вернет false и присвоит $! значение No child processes . Если вызов close первым доберется до умершего потомка, waitpid возвращает 0.

В большинстве систем поддерживается неблокирующий режим waitpid. Об этом можно узнать из стандартного модуля Perl *Config.pm*:

```
use Config,
$has_nonblocking = $Config{d_waitpid} eq  define   ||
                   $Config{d_wait4}   eq  define  ,
```

System V определяет сигнал SIGCLD, который имеет тот же номер, что и SIGCHLD, но слегка отличается по семантике. Чтобы избежать путаницы, используйте SIGCHLD.

Смотри также

Раздел "Signals" *perlipc*(1); описание функций wait и waitpid в *perlfunc(1)*\ документация по стандартному модулю POSIX; страницы руководства *sigaction*(2), *signal*(3) и *kill*(2) вашей системы (если есть); рецепт 16.17.

16.20. Блокировка сигналов

Проблема

Требуется отложить прием сигнала — например, чтобы предотвратить непредсказуемые последствия от сигналов, которые могут прервать программу в любой момент.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь интерфейсом модуля **POSIX** к системной функции `sigprocmask` (только в POSIX-совместимых системах).

Блокировка сигнала на время выполнения операции выполняется так:

```
use POSIX qw( signal_h),

$sigset = POSIX  SigSet->new(SIGINT),  # Определить блокируемые сигналы
$old_sigset = POSIX  SigSet->new,      # Для хранения старой маски

unless (defined sigprocmask(SIG_BLOCK, $sigset, $old_sigset)) {
    die  Could not block SIGINT\n ,
}
```

Снятие блокировки выполняется так:

```
unless (defined sigprocmask(SIG_UNBLOCK, $old_sigset)) {
    die  Could not unblock SIGINT\n ,
}
```

Комментарий

В стандарт **POSIX** входят функции `sigaction` и `sigprocmask`, которые позволяют лучше управлять доставкой сигналов. Функция `sigprocmask` управляет отложенной доставкой сигналов, а `sigaction` устанавливает обработчики. При изменении `%SIG` Perl по возможности использует `sigaction`.

Чтобы использовать `sigprocmask`, сначала постройте набор сигналов методом `POSIX  SigSet->new`. В качестве аргумента передается список номеров сигналов. Модуль **POSIX** экспортирует функции, возвращающие номера сигналов; имена функций совпадают с именами сигналов:

```
use POSIX qw( signal_h),

$sigset = POSIX  SigSet->new( SIGINT, SIGKILL ),
```

Передайте объект POSIX::SigSet функции `sigprocmask` с нужным флагом. Флаг SIG_BLOCK откладывает доставку сигнала. Флаг SIG_UNBLOCK восстанавливает нормальную доставку сигналов, а **SIG_GETMASK** блокирует только сигналы, содержащиеся в POSIX::SigSet. Самые отчаянные перестраховщики блокируют сигналы при вызове `fork`, чтобы предотвратить вызов обработчика сигнала в порожденном процессе перед тем, как Perl обновит его переменную **$$** (идентификатор процесса). Если обработчик сигнала вызывается немедленно и сообщает значение **$$,** то вместо своего собственного **$$** он может использовать родительское значение. Такая проблема возникает очень редко.

Смотри также

Страница руководства *sigprocmask*(2) вашей системы (если есть); документация по стандартному модулю **POSIX.**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

16.21. Тайм-аут

Проблема

Вы хотите гарантировать, что продолжительность некоторой операции не превышает заданный промежуток времени. Допустим, вы проводите архивацию файловой системы и хотите прервать ее, если она затянется более чем на час. Или вы хотите, чтобы через час произошло некоторое событие.

Решение

Чтобы прервать затянувшуюся операцию, используйте обработчик SIGALRM и вызовите в нем die. Установите таймер функцией alarm и включите код в eval:

```
$SIG{ALRM} = sub { die "timeout" };

eval {
    alarm(3600);
    # Продолжительные операции
    alarm(0);
};

if ($@) {
    if ($@ =~ /timeout/) {
                        # Тайм-аут; сделайте то, что считаете нужным
    } else {
        die;            # Передать дальше неожиданное исключение
    }
}
```

Комментарий

Функция alarm получает один аргумент: целое число секунд, после истечения которых ваш процесс получит SIGALRM. В сильно загруженных системах с разделением времени сигнал может быть доставлен позже указанного времени. По умолчанию SIGALRM завершает программу, поэтому вы должны установить собственный обработчик сигнала.

Функции alarm нельзя (с пользой) передать дробное число секунд; если вы попытаетесь это сделать, число секунд будет округлено до целого. Создание более точных таймеров рассматривается в рецепте 3.9.

Смотри также

Раздел «Signals» *perlipc*(1); описание функции alarm в *perlfunc(1)*; рецепт 3.9.

16.22. Программа: sigrand

Следующая программа выдает случайные подписи с применением именованных каналов. Предполагается, что файл подписей хранится в формате программы fortune — то есть каждый многострочный блок завершается последовательностью "%%\n". Приведем пример:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
Make is like Pascal  everybody likes it, so they go in and change it
                                        --Dennis Ritchie
%%
I eschew embedded capital letters in names, to my prose-oriented eyes,
                          comfortably  They jangle like bad typography
                                        --Rob Pike
%%
God made the integers, all else is the work of Man
                                        --Kronecker
%%
I d rather have  rofix than const        --Dennis Ritchie
%%
If you want to program in C, program in C  It's a nice language
I use it occasionally       -)             --Larry Wall
%%
Twisted cleverness is my only skill as a programmer
                                        --Elizabeth Zwicky
%%
Basically, avoid comments  If your code needs a comment to be understood,
it would be                  it    it   easier to understand
                                        --Rob Pike
%%
Comments on data are usually much      helpful than on algorithms
                                        --Rob Pike
%%
Programs that write programs are the happiest programs in the world
                                        --Andrew Hume
%%
```

Мы проверяем, не была ли программа запущена ранее — для этого используется файл с идентификатором процесса. Если посылка сигнала с номером 0 показывает, что идентификатор процесса все еще существует (или, что случается редко — что им воспользовался кто-то другой), программа просто завершается. Также мы проверяем текущую отправку Usenet и решаем, следует ли искать специализированные файлы подписей для конкретных конференций. В этом случае можно завести разные подписи для каждой конференции, в которую вы пишете. Для большего разнообразия глобальный файл подписей иногда применяется даже при наличии специализированного файла.

Программа *sigrand* может использоваться даже в системах без именованных каналов — достаточно удалить код создания именованного канала и увеличить паузу перед обновлениями файла. После этого ***.signature*** может быть обычным файлом. Другая проблема переносимости возникает при переходе программы в фоновый режим (при котором она почти становится демоном). Если функция `fork` недоступна, просто закомментируйте ее.

Полный текст программы приведен в примере 16.12.

Пример 16.12. sigrand

```
#!/usr/bin/perl -w
# sigrand - выдача случайных подписей для
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 16.12 (продолжение)

```
use strict,

# Конфигурационные переменные
use vars qw( $NG_IS_DIR$MKNOD $FULLNAME $FIFO $ART $NEWS $SIGS $SEMA
          $GLOBRAND $NAME ),

# Глобальные имена
use vars qw( $Home $Fortune_Path @Pwd ),

##################################################################
# Начало секции конфигурации
# В действительности следует читать из ~/ sigrandrc

gethome(),

                                humor funny
$NG_IS_DIR       = 1,

$MKNOD           =  /bin/mknod ,
$FULLNAME        =  $Home/ fullname
$FIFO            =  $Home/
$ART             =  $Home/ article ,
$NEWS            =  $Home/News ,
$SIGS            =  $NEWS/SIGNATURES ,
$SEMA            =  $Home/ sigrandpid ,
$GLOBRAND        = 1/4,  #  Вероятность использования глобальных
                         # подписей при наличии специализированного файла

# $NAME следует  (1) оставить неопределенным,  чтобы программа
#  попыталась угадать адрес подписи  (возможно,  заглянув
# в ^/ fullname,  (2) присвоить точный адрес,  или  (3)  присвоить
# пустую строку,  чтобы отказаться от использования  имени

$NAME            =    ,            # Означает,  что имя не используется
##  $NAME         =  me\@home org\n  ,

# Конец секции конфигурации -- HOME и FORTUNE
# настраиваются автоматически
##################################################################

setup(),                 #  Выполнить инициализацию
justme(),                 # Убедиться,  что программа еще не работает
fork && exit,             # Перейти в фоновый режим

open  (SEMA,  > $SEMA )   or  die  can t  write $SEMA  $' ,
print  SEMA  $$\n  ,
close(SEMA)    or  die  can t close  $SEMA  $' ,

# В бесконечном цикле записывать подпись в FIFO
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Если именованные каналы у вас не поддерживаются, измените
# паузу в конце цикла (например, 10, чтобы обновление
# происходило только каждые 10 секунд)
for (,,) {
    open (FIFO, > $FIFO )        or die can t write $FIFO $! ,
    my $sig = pick_quote(),
    for ($sig) {
        s/^(( ?[^\n]*\n){4}) *$/$1/s,   # Ограничиться 4 строками
        s/^( {1,80}) *? *$/$1/gm,       # Обрезать длинные строки
    }
    # Вывести подпись с именем, если оно присутствует,
    # и дополнить до 4 строк
    if ($NAME) {
        print FIFO $NAME, \n x (3 - ($sig =^ tr/\n//)),$sig,
    } else {
        print FIFO $sig,
    }
    close FIFO,

    # Без небольшой паузы приемник не закончит чтение к моменту,
    # когда передатчик снова попытается открыть FIFO,
    # поскольку приемник существует, попытка окажется успешной
    № В итоге появятся сразу несколько подписей
    # Небольшая пауза между открытиями дает приемникам возможность
    # завершить чтение и закрыть канал

    select(undef, undef, undef, 0 2),   # Выждать 1/5 секунды
}
die XXX NOT REACHED ,          # На эту строку вы никогда не попадете

##################################################################

# Игнорировать SIGPIPE на случай, если кто-то открыл FIFO и
# снова закрыл, не читая данных, взять имя пользователя из файла
# fullname Попытаться определить полное имя хоста Следить за
# амперсандами в паролях Убедиться, что у нас есть подписи или
# цитаты При необходимости построить FIFO

sub setup {
    $SIG{PIPE} = IGNORE ,

    unless (defined $NAME) {            Я Если $NAME не определено
        if (-e $FULLNAME) {            # при конфигурации
          $NAME = cat $FULLNAME ,
          die $FULLNAME should contain only 1 line, aborting
              if $NAME =~ tr/\n// > 1,
        } else {
          my($user, $host),
          chop($host = hostname ),
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 16.12 (продолжение)

```
            ($host) = gethostbyname($host) unless $host =^ /\ /,
            $user = $ENV{USER} || $ENV{LOGNAME} || $Pwd[0]
                or die "intruder alert",
            ($NAME = $Pwd[6]) =- s/, */;
            $NAME =^ s/&/\u\L$user/g, # До сих пор встречается
            $NAME = "\t$NAME\t$user\@$host\n",
          }
      }

      check_fortunes() if '-e $SIGS;

      unless (-p $FIFO) {       # -p проверяет, является ли операнд
                                 # именованным каналом
          if (!-e _) {
             system('$MKNOD $FIFO p") && die "can't mknod $FIFO';
             warn          $FIFO as a named pipe\n',
          } else {
             die "$0 won t overwrite file signature\n',
          }
      } else {
          warn "$0. using existing named pipe $FIFO\n";
      }

      8 Получить хорошее начальное значение для раскрутки генератора.
      # Не нужно в версиях 5 004 и выше
      srand(time() ^ ($$ + ($$ " 15))),
  }

  # Выбрать случайную подпись
  sub pick_quote {
      my $sigfile = signame();
      if ('-e $sigfile) {
                 fortune();
      >
      open (SI6S, "< $sigfile" )        or die "can't open $sigfile"
      local $/ = '%%\n",
      local $_;
      my $quip;
      rand($.) < 1 && ($quip = $_) while <SIGS>;
      close SIGS,
      chomp $quip;
             $quip || ENOSIG: This           file is empty \n";
  }

  " Проверить, содержит ли "/ article строку Newsgroups Если содержит,
  # найти первую конференцию и узнать, существует ли для нее
  # специализированный набор цитат, в противном случае вернуть глобальный
  # набор. Кроме того, время от времени возвращать глобальный набор
  # для внесения большего разнообразия в подписи.
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub signame {   >
     (rand(1 0) > ($GLOBRAND) && open ART)  ||         $SIGS,
     local $/  =
     local $_  = <ART>,
     my($ng)   = /Newsgroups \s*([^,\s]*)/,
     $ng =^ s'\ '/'g if $NG_IS_DIR,      # if rn -/,   or SAVEDIR=%p/%c
     $ng =  $NEWS/$ng/SIGNATURES ,
                    ^        $SIGS,
}

# Вызывать программу fortune с параметром -s до тех пор,
# пока мы не получим достаточно короткую цитату или не
# превысим лимит попыток
sub fortune {
   local $_,
   my $tries = 0,
   do {
       $_ =  $Fortune_Path -s ,
   } until tr/\n// < 5 ||  $tries++ > 20,
   s/^/ /mg,
   $_ ||   SIGRAND  deliver random signals to all processes \n  ,
}

# Проверить наличие программы fortune   Определить полный путь
# и занести его в глобальную переменную
sub check_fortunes {
           if $Fortune_Path,          найден
    for my $dir (split(/ /  $ENV<PATH}),  /usr/games ) {
                if    ($Fortune_Path =  $dir/fortune ),
    }
    die  Need either $SIGS or a fortune program, bailing out ,
}

# Определение каталога
sub gethome {
    @Pwd = getpwuid($<),
    $Home = $ENV{HOME} ||  $ENV{LOGDIR} ||  $Pwd[7]
         or die  no home            r user $<  ,
}

#  Останется только один   -- из фильма  Горец
sub justme {
    if (open SEMA) {
        my $pid,
        chop($pid = <SEMA>),
        kill(0, $pid)     die            running (pid $pid), bailing out ;
        close  SEMA,
    }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

17 Сокеты

Глендаур:Я духов вызывать могу из бездны.
Хотспер: И я могу, и каждый это может,
Вопрос лишь, явятся ль они на зов.

Вильям Шекспир, "Генрих IV"

Введение

Сокеты являются "конечными пунктами" в процессе обмена данными. Одни типы сокетов обеспечивают надежный обмен данными, другие почти ничего не гарантируют, зато обеспечивают низкий расход системных ресурсов. Обмен данными через сокеты может осуществляться на одном компьютере или через Интернет.

В этой главе мы рассмотрим два самых распространенных типа сокетов: *потоковые* и *датаграммные*. Потоковые сокеты обеспечивают двусторонние, последовательные и надежные коммуникации; они похожи на каналы (pipes). Датаграммные сокеты не обеспечивают последовательную, надежную доставку, но они гарантируют, что в процессе чтения сохранятся границы сообщений. Ваша система также может поддерживать сокеты других типов; за подробностями обращайтесь к man-странице socket(2) или эквивалентной документации.

Сокеты делятся по областям (domain): сокеты Интернета и сокеты UNIX. Имя сокета Интернета содержит две составляющие: хост (IP-адрес в определенном формате) и номер порта. В мире UNIX сокеты пре дставляют собой файлы (например, */tmp/mysock*).

Кроме области и типа, с сокетом также ассоциируется определенный *протокол*. Протоколы не имеют особого значения для рядового программиста, поскольку для конкретного сочетания области и типа сокета редко используется более одного протокола.

Области и типы обычно идентифицируются числовыми константами (которые возвращаются функциями, экспортируемыми модулями Socket и IO::Socket). Потоковые сокеты имеют тип SOCK_STREAM, а датаграммные — SOCK_DGRAM. Области Интернета соответствует константа PF_INET, а области UNIX — константа PF_UNIX (в POSIX вместо PF_UNIX используется PF_LOCAL, но PF_UNIX почти всегда допустима просто потому, что используется в огромном количестве

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

существующих программ). Используйте символические имена вместо числовых значений, поскольку последние могут измениться (что неоднократно происходило).

Имена протоколов (например, tcp и udp) тоже соответствуют числам, используемым операционной системой. Встроенная функция Perl getprotobyname возвращает номер по имени протокола. Если функциям сокетов передается значение 0, система выберет подходящий протокол по умолчанию.

Perl содержит встроенные функции для создания сокетов и управления ими; они в основном дублируют свои прототипы на C. Хотя это удобно для получения низкоуровневого, прямого доступа к системе, большинство предпочитает работать с более удобными средствами. На помощь приходят классы IO::Socket::INET и IO::Socket::UNIX — они обеспечивают высокоуровневый интерфейс к низкоуровневым системным функциям.

Начнем с рассмотрения встроенных функций. В случае ошибки все они возвращают undef и присваивают $! соответствующее значение. Функция socket создает сокет, bind — назначает ему локальное имя, connect — подключает локальный сокет к другому (возможно, удаленному). Функция listen готовит сокет к подключениям со стороны других сокетов, а accept последовательно принимает подключения. При обмене данными с потоковыми сокетами можно использовать как print и <>, так и syswrite и sysread, а при обмене с датаграммными сокетами — send и

Типичный сервер вызывает socket, bind и listen, после чего в цикле вызывает accept в блокирующем режиме, ожидая входящих подключений (см. рецепты 17.2 и 17.5). Типичный клиент вызывает socket и connect (см. рецепты 17.1 и 17.4). Датаграммные клиенты ведут себя особым образом. Они не обязаны вызывать connect для передачи данных, поскольку могут указать место назначения в качестве аргумента send.

При вызове bind, connect или send для конкретного приемника необходимо указать имя сокета. Имя сокета Интернета состоит из хоста (IP-адрес, упакованный функцией inet_aton) и порта (числа), объединенных в C-подобную структуру функцией sockaddr_in:

```
use Socket;

$packed_ip   = inet_aton("208.146.240.1");
$socket_name = sockaddr_in($port, $packed_ip),
```

Имя сокета UNIX представляет собой имя файла, упакованное в структуру C функцией sockaddr_un:

```
use Socket;

$socket_name = sockaddr_un("/tmp/mysock");
```

Чтобы преобразовать упакованное имя сокета и снова получить имя файла или пару "хост/порт", вызовите sockaddr_un или sockaddr_in в списковом контексте:

```
($port, $packed_ip) = sockaddr_in($socket_name),  # Для сокетов PF_INET
($filename)         = sockaddr_un($socket_name);  " Для сокетов PF_UNIX
```

Функция inet_ntoa преобразует упакованный IP-адрес в ASCII-строку.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$ip_address = inet_ntoa($packed_ip),
$packed_ip  = inet_aton( 204 148 40 9 ),
$packed_ip  = inet_aton( www oreilly com ),
```

В большинстве рецептов используются сокеты Интернета, однако практически все сказанное в равной мере относится и к сокетам UNIX. В рецепте 17.6 объясняются отличия и возможные расхождения.

Сокеты являются основой для работы сетевых серверов. Мы рассмотрим три варианта построения серверов: в первом для каждого входящего подключения создается порожденный процесс (рецепт 17.11), во втором сервер создает порожденные процессы заранее (рецепт 17.12), а в третьем процесс-сервер вообще не создает порожденные процессы (рецепт 17.13)

Некоторые серверы должны одновременно вести прослушивание по многим IP-адресам (см. рецепт 17.14). Хорошо написанный сервер деинициализируется и перезапускается при получении сигнала HUP; в рецепте 17.16 показано, как реализовать такое поведение в Perl. Кроме того, вы узнаете, как идентифицировать оба конца соединения (см. рецепты 17.7 и 17.8).

17.1. Написание клиента TCP

Проблема

Вы хотите подключиться к сокету на удаленном компьютере.

Решение

Следующее решение предполагает, что связь осуществляется через Интернет. TCP-подобные коммуникации на одном компьютере рассматриваются в рецепте 17.6.

Либо воспользуйтесь стандартным (для версии 5.004) классом IO::Socket::INET:

```
use IO  Socket,

$socket = IO  Socket  INET->new(PeerAddr => $remote_host,
                                PeerPort =>
                                Proto    =>  tcp ,
                                Type     => SOCK_STREAM)
    or die  Couldn t connect to $remote_host $remote_port   $@\n ,

#     Сделать что-то с сокетом
print $socket  Why don t you call me  anymore?\n,

$answer  =  <$socket>,

# Отключиться после завершения
close($socket),
```

либо создайте сокет вручную, чтобы лучше упраэлять его поведением:

```
use Socket,

# Создать сокет
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
socket(SERVER, PF_INET, SOCK_STREAM, getprotobyname( tcp )),

# Построить адрес удаленного компьютера
$internet_addr = inet_aton($remote_host)
    or die Couldn t convert $remote_host into an Internet          $!\n
$paddr = sockaddr_in($remote_port, $internet_addr),

#  Подключиться
connect(TO_SERVER, $paddr)
    or die Couldn t connect to                                  $!\n
                                t
9      Сделать что-то с сокетом
print TO_SERVER Why don t you call me anymore?\n ,

" И отключиться после завершения
close(TO_SERVER),
```

Комментарий

Ручное кодирование состоит из множества действий, а класс IO:.Socket::INET объединяет их все в удобном конструкторе. Главное, что необходимо знать, — куда вы направляетесь (параметры PeerAddr и PeerPort) и каким образом (параметр Type). По переданной информации IO::Socket::INET пытается узнать все остальное. Так, протокол по возможности вычисляется по типу и номеру порта; если это не удается сделать, предполагается протокол tcp.

Параметр PeerAddr содержит строку с именем хоста (www oreilly com) или его IP-адресом (204 148 40 9). PeerPort —целое число, номер порта для подключения. Номер порта можно включить в адрес в виде www Параметр Type определяет тип создаваемого сокета: SOCK_DGRAM для датаграммного сокета или SOCK_STREAM для потокового.

Чтобы подключиться через SOCK_STREAM к порту конкретного компьютера, не поддерживающего других возможностей, передайте IO Socket INET->new одну строку с именем хоста и портом, разделенными двоеточием:

```
$client = IO Socket INET->new( www yahoo com 80 )
    or die $@,
```

При возникновении ошибки IO::Socket::INET возвращает undef, а переменной $@ (не $!) присваивается сообщение об ошибке.

```
$s = IO Socket INET->new(PeerAddr => Does not Exist ,
                         Peerport => 80,
                         Type     => SOCK_STREAM )
    or die $@,
```

Если ваши пакеты бесследно исчезают в глубинах сети, вероятно, невозможность подключения к порту будет обнаружена лишь через некоторое время. Вы можете уменьшить этот промежуток, передавая параметр Timeout при вызове IO Socket INET->new():

```
$s = IO Socket INET->new(PeerAddr => 'bad host com",
                         PeerPort => 80,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                                Type      => SOCK_STREAM,
                                Timeout   => 5 )
    or die $@,
```

Но в этом случае вы уже не сможете использовать $! или $@, чтобы узнать причину неудачи — невозможность подключения или тайм-аут. Иногда бывает удобнее установить тайм-аут вручную, без использования модуля.

INADDR_ANY — специальный адрес, означающий "прослушивание на всех интерфейсах". Если вы хотите ограничить его конкретным IP-адресом, включите параметр LocalAddr в вызов IO::Socket::INET->new. При ручном кодировании это делается так:

```
$inet_addr = inet_aton( 208.146.240.1 ),
$paddr     = sockaddr_in($port, $inet_addr),
bind(SOCKET, $paddr)          or die "bind: $! ",
```

Если вам известно только имя, действуйте следующим образом:

```
$inet_addr = gethostbyname( "www.yahoo.com" )
                                  die "                    yahoo.com: $!"
$paddr     = sockaddr_in($port, $inet_addr),
bind(SOCKET, $paddr)          or die "bind: $! ",
```

Смотрит также

Описание функций socket, bind, connect и gethostbyname в *perlfunc*(1); документация по стандартным модулям Socket, IO::Socket и Net::hostent; раздел "Internet TCP Clients and Servers» *perlipc*(1); рецепты 17.2—17.3.

17.2. Написание сервера TCP

Проблема

Вы хотите написать сервер, который ожидает подключения клиентов по сети к определенному порту.

Решение

Следующее решение предполагает, что связь осуществляется через Интернет. TCP-подобные коммуникации на одном компьютере рассматриваются в рецепте 17.6.

Воспользуйтесь стандартным (для версии 5.004) классом IO::Socket.::INET:

```
use IO::Socket;

$server = IO::Socket::INET->new(LocalPort => $server_port,
                                Type      => SOCK_STREAM,
                                Reuse     => 1,
                                Listen    => 10 )   # or SOMAXCONN
    or die "Couldn't be a tcp server on port $server_port: $@\n"
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
while ($client = $server->accept()) {
    # $client - новое подключение
}

close($server);
```

Или создайте сокет вручную, что позволит получить полный контроль над ним:

```
use Socket;

# Создать сокет
socket(SERVER, PF_INET, SOCK_STREAM, getprotobyname( tcp ));

# Чтобы мы могли быстро перезапустить сервер
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1);

# Построить свой адрес сокета
$my_addr = sockaddr_in($server_port, INADDR_ANY);
bind(SERVER, $my_addr)
    or die 'Couldn t bind to port $server_port   $!\n ;

# Установить очередь для входящих соединений
listen(SERVER, SOMAXCONN)
    or die 'Couldn t listen on port $server_port   $!\n ;

# Принимать и обрабатывать подключения
while (accept(CLIENT, SERVER)) {
    # Сделать что-то с CLIENT
}

close(SERVER);
```

Комментарий

Написать сервер сложнее, чем клиент. Необязательная функция `listen` сообщает операционной системе, сколько подключений могут находиться в очереди к серверу, ожидая обслуживания. Функция setsockopt, использованная в решении, позволяет избежать двухминутного интервала после уничтожения сервера перед его перезапуском (полезна при тестировании). Функция `bind` регистрирует сервер в ядре. Наконец, функция accept последовательно принимает входящие подключения.

Числовой аргумент listen определяет количество не принятых функцией accept подключений, которые будут поставлены в очередь операционной системой перед тем, как клиенты начнут получать ошибки "отказ в обслуживании". Исторически максимальное значение этого аргумента было равно 5, но и сегодня многие операционные системы тайно устанавливают максимальный размер очереди равным примерно 20. Сильно загруженные Web-серверы стали распространенным явлением, поэтому многие поставщики увеличивают это значение. Максималь-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ный размер очереди для вашей системы хранится в константе SOMAXCONN модуля Socket.

Функции accept передаются два аргумента: файловый манипулятор, подключаемый к удаленному клиенту, и файловый манипулятор сервера. Она возвращает IP-адрес и порт клиента, упакованные inet_ntoa:

```
use Socket,

while ($client_address = accept(CLIENT, SERVER)) {
    ($port, $packed_ip) = sockaddr_in($client_address),
    $dotted_quad = inet_ntoa($packed_ip),
    # Обработать
}
```

В классах IO::Socket accept является методом, вызываемым для манипулятора сервера:

```
while (($client $client_address) = $server->accept()) {
    #
}
```

Если ожидающих подключений нет, программа блокируется на вызове accept до того, как появится подключение. Если **вы** хотите гарантировать, что вызов accept не будет блокироваться, воспользуйтесь неблокирующими сокетами:

```
use Fcntl qw(F_GETFL F_SETFL O_NONBLOCK),

$flags = fcntl($SERVER, F_GETFL, 0)
            or die Can t get flags for the socket $!\n ,

$flags = fcntl($SERVER, F_SETFL, $flags | O_NONBLOCK)
            or die Can t set flags for the socket $!\n ,
```

Если теперь при вызове accept не окажется ожидающих подключений, accept вернет undef и присвоит $! значение EWOULDBLOCK.

Может показаться, что при возвращении нулевых флагов от **F_GETFL** будет вызвана функция die, как и при неудачном вызове, возвращающем undef. Это не так — неошибочное возвращаемое значение fcntl, как и для ioctl, преобразуется Perl в специальное значение '0 but true'. Для этой специальной строки даже не действуют надоедливые предупреждения флага **-w** о нечисловых величинах, поэтому вы можете использовать ее в своих функциях, когда возвращаемое значение равно 0 и тем не менее истинно.

Смотри также

Описание функций socket, bind, listen, accept, fcntl и setsockopt в *perlfunc*(1); страницы руководства *fcntl*(2), *socket*(2), *setsockopt*(2) вашей системы (если они есть); документация по стандартным модулям Socket, IO::Socket и Net::hostent; раздел "Internet TCP Clients and Servers» *perlipc*(1); рецепты 7.13—7.14; 17.1; 17.3; 17.7.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

17.3. Передача данных через TCP

Проблема

Требуется передать или принять данные по TCP-соединению.

Решение

Следующее решение предполагает, что связь осуществляется через Интернет. TCP-подобные коммуникации на одном компьютере рассматриваются в рецепте 17.6.

Первый вариант — print или <>:

```
print SERVER  What is your name?\n ,
                    <SERVER>),
```

Второй вариант — функции send и

```
defined (send(SERVER  $data_to_send,  $flags))
    or die  Can t send   $!\n

                          $maxlen, $flags)
       die                 $!\n
```

Третий вариант — соответствующие методы объекта IO::Socket:

```
use IO  Socket,

$server->send($data_to_send, $flags)
    or die  Can t send  $!\n ,

                          $flags)
       die              $!\n
```

Чтобы узнать, могут ли быть получены или приняты данные, воспользуйтесь функцией select, для которой в классе IO::Socket также предусмотрена удобная оболочка:

```
use IO  Select,

$select = IO  Select->new(),
$select->add(*FROM_SERVER),
$select->add($to_client),

@read_from = $select->can_read($timeout),
        $socket  (@read_from) {
    8 Прочитать ожидающие данные из $socket
}
```

Комментарий

Сокеты используются в двух принципиально различных типах ввода/вывода, каждый из которых обладает своими достоинствами и недостатками. Стандартные функции ввода/вывода Perl, используемые для файлов (кроме seek и sysseek),

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

работают и для потоковых сокетов, однако для датаграммных сокетов необходимы системные функции send и работающие с целыми записями.

При программировании сокетов очень важно помнить о буферизации. Хотя буферизация и была спроектирована для повышения быстродействия, она может повлиять на интерактивное поведение некоторых программ. Если при вводе данных с помощью <> будет обнаружен разделитель записей, программа может попытаться прочитать из сокета больше данных, чем доступно в данный момент. И print и <> используют буферы stdio, поэтому без включения автоматической очистки буфера (см. введение главы 7 "Доступ к файлам") для манипулятора сокета данные не отправятся на другой конец в момент их передачи функцией print. Вместо этого они будут ждать заполнения буфера.

Вероятно, для клиентов и серверов с построчным обменом данных это подходит — при условии, что вы не забыли включить автоматическую очистку буфера. Новые версии IO::Socket делают это автоматически для анонимных файловых манипуляторов, возвращаемых IO Socket->new.

Но стандартный ввод/вывод — не единственный источник буферизации. Операции вывода (print, printf, syswrite — или send для сокета TCP) буферизуются на уровне операционной системы по так называемому *алгоритму Нейгла*. Если пакет данных отправлен, но еще не подтвержден, другие передаваемые данные ставятся в очередь и отправляются либо после набора следующего полного пакета, либо при получении подтверждения. В некоторых ситуациях (события мыши в оконных системах, нажатия клавиш в приложениях реального времени) такая буферизация оказывается неудобной или попросту неверной. Буферизация Нейгла отключается параметром сокета TCP_NODELAY:

```
use Socket,
        sys/socket ph      # Для &TCP_NODELAY

setsockopt(SOCKET, SOL_SOCKET, &TCP_NODELAY, 1)
    or die Couldn t disable Nagle s algorithm $'\n ,
```

Ее повторное включение происходит так:

```
setsockopt(SOCKET, SOL_SOCKET &TCP_NODELAY, 0)
    or die Couldn t enable Nagle s algorithm $'\n ,
```

Как правило, TCP_NODELAY все же лучше не указывать. Буферизация TCP существует не зря, поэтому не отключайте ее без крайней необходимости — например, если ваше приложение работает в режиме реального времени с крайне интенсивным обменом пакетов.

TCP_NODELAY загружается из *sys/socket.ph* — этот файл не устанавливается автоматически вместе с Perl, но может быть легко построен. Подробности приведены в рецепте 12.14.

Буферизация чрезвычайно важна, поэтому в вашем распоряжении имеется функция select. Она определяет, какие манипуляторы содержат непрочитанный ввод, в какие манипуляторы возможна запись и для каких имеются необработанные "исключительные состояния". Функция select получает три строки, интерпретируемые как двоичные данные; каждый бит соответствует файловому манипулятору. Типичный вызов select выглядит так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$rin = '';                          # Инициализировать маску
vec($rin, fileno(SOCKET), 1) = 1;   # Пометить SOCKET в $rin
# Повторить вызовы vec() для каждого проверяемого сокета

$timeout = 10;                      # Подождать 10 секунд

$nfound = select($rout = $rin, undef, undef, $timeout),
if (vec($rout, fileno(socket),1)){
    # В SOCKET имеются данные для чтения
}
```

Функция select вызывается с четырьмя аргументами. Три из них представляют собой битовые маски: первая проверяет в манипуляторах наличие непрочитанных данных в манипуляторах; вторая — возможность безопасной записи без блокировки; третья — наличие в них исключительных состояний. Четвертый аргумент определяет максимальную длительность ожидания в секундах (может быть вещественным числом).

Функция модифицирует передаваемые ей маски, поэтому при выходе из нее биты будут установлены лишь для манипуляторов, готовых к вводу/выводу. Отсюда один стандартный прием — входная маска ($rin в предыдущем примере) присваивается выходной ($rout), чтобы вызов select изменил только $rout и оставил $rin в прежнем состоянии.

Нулевой тайм-аут определяет режим ***опроса*** (проверка без блокировки). Некоторые начинающие программисты не любят блокировки, и в их программах выполняется "занятое ожидание" (busy-wait) — программа в цикле выполняет опрос, снова и снова. Когда программа блокируется, операционная система понимает, что процесс ждет ввода, и передает процессорное время другим программам до появления входных данных. Когда программа находится в "занятом ожидании", система не оставляет ее в покое, поскольку программа всегда что-то делает — проверяет ввод! Иногда опрос действительно является правильным решением, но гораздо чаще это не так. Тайм-аут, равный undef, означает отсутствие тайм-аута, поэтому ваша программа терпеливо блокируется до появления ввода.

Поскольку select использует битовые маски, которые утомительно создавать и трудно интерпретировать, в решении используется стандартный модуль IO::Select. Он обходит работу с битовыми масками и, как правило, более удобен.

Полное объяснение исключительных состояний, проверяемых третьей маской select, выходит за рамки настоящей книги.

Другие флаги send и recv перечислены в страницах руководства этих системных функций.

Смотри также

Описание функций send, fileno, setsockopt в *perlfunc*(1); разделы "I/O Operators" и "Bitwise String Operators" в *perlop*(1); страница руководства *setsockopt*(2) вашей системы (если есть); документация по стандартным модулям Socket и IO::Socket; раздел "Internet TCP Clients and Servers" *perlipc*(1); рецепты 17.1—17.2.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

17.4. Создание клиента UDP

Проблема

Вы хотите обмениваться сообщениями с другим процессом, используя **UDP** (**датаграммы**).

Решение

Чтобы создать манипулятор для сокета UDP, воспользуйтесь либо низкоуровневым модулем Socket для уже существующего манипулятора:

```
use Socket,
socket(SockHandle, PF_INET, SOCK_DGRAM, getprotobyname( udp ))
    or die socket $! ,
```

либо модулем IO::Socket, возвращающим анонимный манипулятор:

```
use IO Socket
$handle = IO Socket INET->new(Proto => udp )
    or die socket $@ ,      # Да, здесь используется $@
```

Отправка сообщения на компьютер с именем $HOSTNAME и адресом порта $PORTNO выполняется так:

```
$ipaddr   = inet_aton($HOSTNAME),
$portaddr = sockaddr_in($PORTNO, $ipaddr),
send(SockHandle, $MSG, 0, $portaddr) == length($MSG)
        or die cannot send to $HOSTNAME($PORTNO) $' ,
```

Получение сообщения, длина которого не превышает $MAXLEN:

```
$portaddr =                $MSG, $MAXLEN,            die        $!
($portno, $ipaddr) = sockaddr_in($portaddr),
$host = gethostbyaddr($ipaddr, AF_INET),
print $host($portno) said $MSG\n ,
```

Комментарий

Датаграммные сокеты не похожи на потоковые. Поток создает иллюзию постоянного соединения. Он напоминает телефонный звонок — установка связи обходится дорого, но в дальнейшем связь надежна и проста в использовании. Датаграммы больше похожи на почту — если ваш знакомый находится на другом конце света, дешевле и проще отправить ему **письмо**, чем дозвониться по телефону. **Датаграммы** потребляют меньше системных ресурсов, чем потоки. Вы пересылаете небольшой объем информации, по одному сообщению за раз. Однако доставка сообщений не гарантируется, и они могут быть приняты в неверном порядке. Если очередь получателя переполнится, как маленький почтовый ящик, то дальнейшие сообщения теряются.

Если датаграммы настолько ненадежны, зачем же ими пользоваться? Просто некоторые приложения наиболее логично реализуются с применением датаграмм. Например, при пересылке аудиоданных важнее сохранить поток в целом, чем гарантировать прохождение каждого пакета, особенно если потеря пакетов вызва-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

на недостаточной пропускной способностью. Датаграммы также часто применяются в широковещательной рассылке (аналог массовой рассылки рекламных объявлений по почте). В частности, широковещательные пакеты используются для отправки в локальную подсеть сообщений типа: "Есть здесь кто-нибудь, кто хочет быть моим сервером?"

Поскольку **датаграммы** не создают иллюзии постоянного соединения, в работе с ними вы располагаете несколько большей свободной. Вам не придется вызывать connect для подключения сокета к удаленной точке, с которой вы обмениваетесь данными. Вместо этого каждая **датаграмма** адресуется отдельно при вызове send. Предполагая, что `$remote_addr` является результатом вызова `sockaddr_in`, поступите следующим образом:

```
send(MYSOCKET, $msg_buffer, $flags,
    or die Can t send $!\n ,
```

Единственный часто используемый флаг, MSG_OOB, позволяет отправлять и принимать внеполосные (out-of-band) данные в нетривиальных приложениях.

Удаленный адрес (`$remote_addr`) должен представлять собой комбинацию порта и адреса Интернета, возвращаемую функцией `sockaddr_in` модуля Socket. Если хотите, вызовите connect для этого адреса — в этом случае последний аргумент при вызове send можно опускать, а все сообщения будут отправлены этому получателю. В отличие от потоковых коммуникаций, один **датаграммный сокет** позволяет подключиться к другому компьютеру.

В примере 17.1 приведена небольшая программа, использующая протокол UDP. Она устанавливает связь с портом времени UDP на **компьютере**, имя которого задается в командной **строке**, или по умолчанию на локальном компьютере. Программа работает не на всех компьютерах, но при наличии сервера UDP вы получите 4-байтовое целое число, байты которого упакованы в сетевом порядке; число равно количеству секунд с 1900 года по данным этого компьютера. Чтобы передать это время функции преобразования `localtime` или `gmtime`, необходимо вычесть из него количество секунд от 1900 до 1970 года.

Пример 17.1. **clockdrift**

```
#!/usr/bin/perl
# clockdrift - сравнение текущего времени с другой системой
use strict,
use Socket,

my ($host, $him, $src, $port, $ipaddr, $ptime, $delta),
my $SECS_of_70_YEARS        = 2_208_988_800,

socket(MsgBox, PF_INET SOCK_DGRAM, getprotobyname( udp ))
    or die socket $! ,
$him = sockaddr_in(scalar(getservbyname( time , udp )),
    inet_aton(shift || 127 1 )),
defined(send(MsgBox, 0, 0, $him))
    or die send $! ,
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.1 (продолжение)

```
defined($src = recv(MsgBox, $ptime, 4, 0))
    or die "recv: $!";
($port, $ipaddr) = sockaddr_in($src);
$host = gethostbyaddr($ipaddr, AF_INET);
my $delta = (unpack("N", $ptime) - $SECS_of_70_YEARS) - time();
print "Clock on $host is $delta seconds ahead of this one \n";
```

Если компьютер, с которым вы пытаетесь связаться, не работает или ответ потерян, программа застрянет при вызове ожидании ответа, который никогда не придет.

Смотри также

Описание функций send, recv, gethostbyaddr и unpack в *perlfunc*(1); документация по стандартным модулям Socket и IO::Socket; раздел «UDP: message passing" *perlipc(1);* рецепт 17.5.

17.5. Создание сервера UDP

Проблема

Вы хотите написать сервер UDP.

Решение

Сначала вызовите функцию bind для номера порта, по которому будет осуществляться связь с вашим сервером. С модулем IO::Socket это делается просто:

```
use IO..Socket;
$server = IO.::Socket::INET->new(LocalPort => $server_port,
                                 Proto     => "udp')
    or die "Couldn't be a udp server on port $server_port • $@\n",
```

Затем в цикле принимайте сообщения:

```
while ($him = $server->recv($datagram, $MAX_TO_READ, $flags)) {
    # Обработать сообщение
}
```

Комментарий

Программирование для UDP намного проще, чем для TCP. Вместо того чтобы последовательно принимать клиентские подключения и вступать в долгосрочную связь с клиентом, достаточно просто принимать сообщения от клиентов по мере их поступления. Функция recv возвращает адрес отправителя, подлежащий декодированию.

В примере 17.2 показан небольшой сервер UDP, который просто ожидает сообщений. Каждый раз, когда приходит очередное сообщение, мы выясняем, кто его послал, и отправляем ответ-сообщение с принятым текстом, после чего сохраняем новое сообщение.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.2. udpqotd

```
#!/usr/bin/perl -w
# udpqotd - сервер сообщении UDP
use strict,
use IO  Socket,
my($sock, $oldmsg, $newmsg, $hisaddr, $hishost  $MAXLEN, $PORTNO),
$MAXLEN = 1024,
$PORTNO = 5151,
$sock = IO  Socket  INET->new(LocalPort => $PORTNO, Proto =>  udp  )
    or die  socket  $@ ,
print  Awaiting UOP messages on port $PORTNO\n ,
$oldmsg =  This is the starting message  ,
while ($sock->recv($newmsg, $MAXLEN)) {
    my($port, $ipaddr) = sockaddr_in($sock->peername),
    $hishost = gethostbyaddr($ipaddr, AF_INET),
    print  Client $hishost said  $newmsg  \n  ,
    $sock->send($oldmsg),
    $oldmsg =  [$hishost] $newmsg ,
}
die        $!
```

С использованием модуля IO::Socket программа получается проще, чем с низкоуровневым модулем Socket. Нам не приходится указывать, куда отправить сообщение, поскольку библиотека сама определяет отправителя последнего сообщения и сохраняет его в объекте $sock. Метод peername извлекает данные для декодирования.

Программа telnet не подходит для общения с этим сервером; для этого необходим специальный клиент. Один из вариантов приведен в примере 17.3.

Пример 17.3. udpmsg

```
#!/usr/bin/perl -w
# udpmsg - отправка сообщения серверу udpquotd

use IO  Socket,
use strict,

my($sock, $server_host, $msg  $port, $ipaddr, $hishost,
   $MAXLEN, $PORTNO, $TIMEOUT),

$MAXLEN  = 1024,
$PORTNO  = 5151,
$TIMEOUT = 5,

$server_host = shift,
$msg         =  @ARGV ,
$sock = IO  Socket  INET->new(Proto      =>  udp ,
                              PeerPort   => $PORTNO,
                              PeerAddr   => $server_host)
```

продолжение ⇨

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.3 (продолжение)

```
        die            socket: $!\n";
$sock->send($msg) or die "send: $!";

eval {
    local $SIG{ALRM} = sub { die "alarm timeout" };
    alarm $TIMEOUT;
    $sock->recv($msg, $MAXLEN)      or die "recv: $!";
    alarm 0;
    1;  Я Нормальное возвращаемое значение eval
} or die "recv from $server_host timed out after $TIMEOUT seconds.\n";

($port, $ipaddr) = sockaddr_in($sock->peername);
$hishost = gethostbyaddr($ipaddr, AF_INET);
print "Server $hishost            ''$msg''\n";
```

При создании сокета мы с самого начала указываем хост и номер порта, что позволяет опустить эти данные при вызовах send.

Тайм-аут (alarm) был добавлен на случай, если сервер не отвечает или вообще не работает. Поскольку recv является блокирующей системной функцией, выход из которой может и не произойти, мы включаем ее в стандартный блок eval для прерывания блокировки по тайм-ауту.

Смотри также

Описание функций alarm в *perlfunc*(1); документация по стандартным модулям Socket и IO::Socket; раздел «UDP: message passing" *perlipc*(1); рецепты 16.21; 17.4.

17.6. Использование сокетов UNIX

Проблема

Вы хотите обмениваться данными с другими процессами, находящимися исключительно на локальном компьютере.

Решение

Воспользуйтесь сокетами UNIX. При этом можно использовать программы и приемы из предыдущих рецептов для сокетов Интернета со следующими изменениями:

- Вместо socketaddr_in используется socketaddr_un.
- Вместо IO::Socket::UNIX используется IO::Socket::INET.
- Вместо PF_INET используется PF_UNIX, а при вызове socket в качестве аргумента передается PF_UNSPEC.
- Клиенты SOCK_STREAM не обязаны вызывать bind для локального адреса перед вызовом connect.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Имена сокетов UNIX похожи на имена файлов в файловой системе. Фактически в большинстве систем они реализуются в виде специальных файлов; именно это и делает оператор Perl -S — он проверяет, я вляется ли файл сокетом UNIX.

Передайте имя файла в качестве адресного аргумента IO 'Socket 'UNIX->new или закодируйте его функцией sockaddr_un и передайте его connect. Посмотрим, как создаются серверные и клиентские сокеты UNIX в модуле IO::Socket::UNIX:

```
use IO  Socket,

unlink  /tmp/mysock ,
$server = IO  Socket  UNIX->new(LocalAddr =>  /tmp/mysock ,
                                Type      => SOCK_DGRAM,
                                Listen    => 5 )
    or die $@,

$client = IO  Socket  UNIX->new(PeerAddr  =>  /tmp/mysock ,
                                Type      => SOCK_DGRAM,
                                Timeout   => 10 )
    or  die  $@,
```

Пример использования традиционных функций для создания потоковых сокетов выглядит так:

```
use Socket,

socket(SERVER,  PF_UNIX,  SOCK_STREAM,  0),
unlink  /tmp/mysock ,
bind(SERVER,  sockaddr_un(  /tmp/mysock  ))
       die                          $'

socket(CLIENT  PF_UNIX,  SOCK_STREAM,  0),
connect(CLIENT,  sockaddr_un(  /tmp/mysock  ))
    or die  Can t  connect  to  /tmp/mysock   $' ,
```

Если вы не уверены полностью в правильном выборе протокола, присвойте аргументу Proto при вызове IO : Socket UNIX->new значение 0 для сокетов PF_UNIX. Сокеты UNIX могут быть как датаграммными (SOCK_DGRAM), так и потоковыми (SOCK_STREAM), сохраняя при этом семантику аналогичных сокетов Интернета. Изменение области не меняет характеристик типа сокета.

Поскольку многие системы действительно создают специальный файл в файловой системе, вы должны удалить этот файл перед попыткой привязки сокета функцией bind. Хотя при этом возникает опасность перехвата (между вызовами unlink и bind кто-то может создать файл с именем вашего сокета), это не вызывает особых погрешностей в системе безопасности, поскольку bind не перезаписывает существующие файлы.

Смотри также

Рецепты 17.1-17.5.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

17.7. Идентификация другого конца сокета

Проблема

Имеется сокет. Вы хотите идентифицировать компьютер, находящийся на другом конце.

Решение

Если вас интересует только IP-адрес удаленного компьютера, поступите следующим образом:

```
use Socket;

$other_end          = getpeername(SOCKET)
    or die "Couldn't identify other end: $!\n";
($port, $iaddr)     = unpack_sockaddr_in($other_end);
$ip_address         = inet_ntoa($iaddr);
```

Имя хоста определяется несколько иначе:

```
use Socket;

$other_end          = getpeername(SOCKET)
    or die "Couldn't identify other end: $!\n";
($port, $iaddr)     = unpack_sockaddr_in($other_end);
$actual_ip          = inet_ntoa($iaddr);
$claimed_hostname = gethostbyaddr($iaddr, AF_INET);
@name_lookup        = gethostbyname($claimed_hostname)
    or die "Could not look up $claimed_hostname . $!\n";
@resolved_ips     = map { inet_ntoa($_) }
    @name_lookup[ 4 .. $#ips_for_hostname ],
```

Комментарий

В течение долгого времени задача идентификации подключившихся компьютеров считалась более простой, чем на самом деле. Функция `getpeername` возвращает IP-адрес удаленного компьютера в упакованной двоичной структуре (или undef в случае ошибки). Распаковка выполняется функцией `inet_ntoa`. Если вас интересует имя удаленного компьютера, достаточно вызвать `gethostbyaddr` и поискать его в таблицах DNS, не так ли?

Не совсем. Это лишь половина решения. Поскольку поиск по имени выполняется на сервере DNS владельца имени, а поиск по IP-адресу — на сервере DNS владельца адреса, приходится учитывать возможность, что компьютер, к которому вы подключились, выдает неверные имена. Например, компьютер *evil.crackers.org* может принадлежать злобным киберпиратам, которые сказали своему серверу DNS, что их IP-адрес (1.2.3.4) следует идентифицировать как *trusted.dod.gov*. Если ваша программа доверяет *trusted.dod.gov*, то при подключении с *evil.crackers.org* функция `getpeername` вернет правильный IP-адрес (1.2.3.4), однако `gethostbyaddr` вернет ложное имя.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Чтобы справиться с этой проблемой, мы берем имя (возможно, ложное), полученное от gethostbyaddr, и снова вызываем для него функцию gethostbyname. В примере с *evil.crackers.org* поиск для *trusted.dod.gov* будет выполняться на сервере DNS *dod.gov* и вернет настоящий IP-адрес (адреса) *trusted.dod.gov*. Поскольку многие компьютеры имеют несколько IP-адресов (очевидный пример — распределенные Web-серверы), мы не можем использовать упрощенную форму gethostbyname:

```
$packed_ip = gethostbyname($name) or die "Couldn't look up $name: $!\n",
$ip_address = inet_ntoa($packed_ip),
```

До настоящего момента предполагалось, что мы рассматриваем приложение с сокетами Интернета. Функцию getpeername также можно вызвать для сокета UNIX. Если на другом конце была вызвана функция bind, вы получите имя файла, к которому была выполнена привязка. Однако если на другом конце функция bind *не вызывалась*, то getpeername может вернуть пустую (неупакованную) строку, упакованную строку со случайным мусором, или undef как признак ошибки.., или ваш компьютер перезагрузится (варианты перечислены по убыванию вероятности и возрастанию неприятностей). В нашем компьютерном деле это называется "непредсказуемым поведением".

Но даже этого уровня паранойи и перестраховки недостаточно. При желании можно обмануть сервер DNS, не находящийся в вашем непосредственном распоряжении, поэтому при идентификации и аутентификации не следует полагаться на имена хостов. Настоящие параноики и мизантропы обеспечивают безопасность с помощью криптографических методов.

Смотри также

Описание функций gethostbyaddr, gethostbyname и getpeername в *perlfunc*(1); описание функции inet_ntoa в стандартном модуле Socket; документация по стандартным модулям IO::Socket и Net::hostnet.

17.8. Определение вашего имени и адреса

Проблема

Требуется узнать ваше (полное) имя хоста.

Решение

Сначала получите свое (возможно, полное) имя хоста. Воспользуйтесь либо стандартным модулем Sys::Hostname:

```
use Sys::Hostname;

$hostname = hostname()
```

либо функцией uname модуля **POSIX:**

```
use POSIX qw(uname);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
($kernel, $hostname, $release, $version, $hardware) = uname();

$hostname = (uname)[1];
```

Затем превратите его в IP-адрес и преобразуйте в каноническую форму:

```
use Socket;                          # Для AF_INET
            gethostbyname($hostname)
    or die "Couldn't        $hostname    $!";
$hostname = gethostbyaddr($address, AF_INET)
    or die "Couldn't            $hostname    $!";
```

Комментарий

Для улучшения переносимости модуль Sys::Hostname выбирает оптимальный способ определения имени хоста, руководствуясь сведениями о вашей системе. Он пытается получить имя хоста несколькими различными способами, но часть из них связана с запуском других программ. Это может привести к появлению меченых данных (см. рецепт 19.1).

С другой стороны, `POSIX::uname` работает только в POSIX-системах и не гарантирует получения полезных данных в интересующем нас поле `nodename`. Впрочем, на многих компьютерах это значение все же *приносит* пользу и не страдает от проблем меченых данных в отличие от Sys::Hostname.

Однако после получения имени хоста следует учесть возможность того, что в нем отсутствует имя домена. Например, Sys::Hostname вместо *guanaco.camelids.org* может вернуть просто *guanaco*. Чтобы исправить ситуацию, преобразуйте имя в IP-адрес функцией `gethostbyname`, а затем — снова в имя функцией geth ostbyaddr. Привлечение DNS гарантирует получение полного имени.

Смотри также

Описание функций gethostbyaddr и `gethostbyname` в *perlfunc*(1); документация по стандартным модулям Net::hostnet и Sys::Hostname.

17.9. Закрытие сокета после разветвления

Проблема

Ваша программа разветвилась, и теперь другому концу необходимо сообщить о завершении отправки данных. Вы попытались вызвать close для сокета, но удаленный конец не получает ни EOF, ни SIGPIPE.

Решение

Воспользуйтесь функцией shutdown:

```
shutdown(SOCKET, 0);    # Прекращается чтение данных
shutdown(SOCKET, 1);    " Прекращается запись данных
shutdown(SOCKET, 2);    # Прекращается работа с сокетом
```

Используя объект IO::Socket, также можно написать:

```
$socket->shutdown(0);  # Прекращается чтение данных
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

При разветвлении (forking) процесса потомок получает копии всех открытых файловых манипуляторов родителя, включая сокеты. Вызывая close для файла или сокета, вы закрываете только копию манипулятора, принадлежащую текущему процессу. Если в другом процессе (родителе или потомке) манипулятор остался открытым, операционная система не будет считать файл или сокет закрытым.

Рассмотрим в качестве примера сокет, в который посылаются данные. Если он открыт в двух процессах, то один из процессов может закрыть его, и операционная система все равно не будет считать сокет закрытым, поскольку он остается открытым в другом процессе. До тех пор пока он не будет закрыт *другим* процессом, процесс, читающий из сокета, не получит признак конца файла. Это может привести к недоразумениям и взаимным блокировкам.

Чтобы избежать затруднений, либо вызовите close для незакрытых манипуляторов, либо воспользуйтесь функцией shutdown. Функция shutdown является более радикальной формой close — она сообщает операционной системе, что, даже несмотря на наличие копий манипулятора у других процессов, он должен быть помечен как закрытый, а другая сторона должна получить признак конца файла при чтении или SIGPIPE при записи.

Числовой аргумент shutdown позволяет указать, какие стороны соединения закрываются. Значение 0 говорит, что чтение данных закончено, а другой конец сокета при попытке передачи данных должен получить SIGPIPE. Значение 1 говорит о том, что закончена запись данных, а другой конец сокета при попытке чтения данных должен получать признак конца файла. Значение 2 говорит о завершении как чтения, так и записи.

Представьте себе сервер, который читает запрос своего клиента до конца файла и затем отправляет ответ. Если клиент вызовет close, сокет станет недоступным для ввода/вывода, поэтому ответ от сервера не доберется до клиента. Вместо этого клиент должен вызвать shutdown, чтобы закрыть соединение наполовину.

```
print SERVER "my request\n";    # Отправить данные
shutdown(SERVER, 1);            # Отправить признак конца данных;
                                в запись окончена.
$answer = <SERVER>;             " Хотя чтение все еще возможно.
```

Смотри также

Описание функций close и shutdown в *perlfunc*(1); страница руководства *shutdown*(2) вашей системы (если есть).

17.10. Написание двусторонних клиентов

Проблема

Вы хотите написать полностью интерактивного клиента, в котором можно ввести строку, получить ответ, ввести другую строку, получить новый ответ и т. д. — словом, нечто похожее на telnet.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

После того как соединение будет установлено, разветвите процесс. Один из близнецов только читает ввод и передает его серверу, а другой — читает выходные данные сервера и копирует их в поток вывода.

Комментарий

В отношениях "клиент/сервер" бывает трудно определить, чья сейчас очередь "говорить". Однозадачные решения, в которых используется версия select с четырьмя аргументами, трудны в написании и сопровождении. Однако нет причин игнорировать многозадачные решения, а функция fork кардинально упрощает эту проблему.

После подключения к серверу, с которым вы будете обмениваться данными, вызовите fork. Каждый из двух идентичных (или почти идентичных) процессов выполняет простую задачу. Родитель копирует все данные, полученные из сокета, в стандартный вывод, а потомок одновременно копирует все данные из стандартного ввода в сокет.

Исходный текст программы приведен в примере 17.4.

Пример 17.4. biclient

```
#!/usr/bin/perl -w
# biclient - двусторонний клиент с разветвлением
    use strict;
use IO::Socket;
my ($host, $port, $kidpid, $handle, $line);

unless (@ARGV == 2) { die "usage: $0 host port\n"; }
($host, $port) = @ARGV;

# Создать tcp-подключение для заданного хоста и порта
$handle = IO::Socket::INET->new(Proto     => "tcp",
                                PeerAddr  => $host,
                                PeerPort  => $port)
       or die "can't connect to port $port on $host: $!\n";

$handle->autoflush(1);          # Запретить буферизацию
print STDERR "[Connected to $host $port]\n";

# Разделить программу на два идентичных процесса
die "can't fork: $!\n" unless defined($kidpid = fork());

if ($kidpid) {
    # Родитель копирует сокет в стандартный вывод
    while (defined ($line = <$handle>)) {
        print STDOUT $line;
    }
    kill("TERM" => $kidpid);        # Послать потомку SIGTERM
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
else {
    # Потомок копирует стандартный ввод в сокет
    while (defined ($line = <STDIN>)) {
        print $handle $line;
    }
}
exit;
```

Добиться того же эффекта с одним процессом намного труднее. Проще создать два процесса и поручить каждому простую задачу, нежели кодировать выполнение двух задач в одном процессе. Стоит воспользоваться преимуществами мультизадачности и разделить программу на несколько подзадач, как многие сложнейшие проблемы упрощаются на глазах.

Функция `kill` в родительском блоке `if` нужна для того, чтобы послать сигнал потомку (в настоящее время работающему в блоке else), как только удаленный сервер закроет соединение со своего конца. Вызов `kill` в конце родительского блока ликвидирует порожденный процесс с завершением работы сервера.

Если удаленный сервер передает данные по байтам и вы хотите получать их немедленно, без ожидания перевода строки (которого вообще может не быть), замените цикл `while` родительского процесса следующей конструкцией:

```
my $byte;
while (sysread($byte, 1) == 1) {
    print STDOUT $byte
}
```

Смотри также

Описание функций `fork` в *perlfunc*(1); документация по стандартному модулю IO::Socket; рецепты 16.5; 16.10; 17.11.

17.11. Разветвляющие серверы

Проблема

Требуется написать сервер, который для работы с очередным клиентом ответвляет специальный подпроцесс.

Решение

Ответвляйте подпроцессы в цикле accept и используйте обработчик `$SIG{CHLD}` для чистки потомков.

```
# Создать сокет SERVER, вызвать bind и прослушивать
use POSIX qw(sys_wait_h);

sub REAPER {
    1 until (-1 == waitpid(-1, WNOHANG));
    $SIG{CHLD} = \&REAPER;                  # если $] >= 5.002
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
}

$SIG{CHLD} = \&REAPER;

while ($hisaddr = accept(CLIENT, SERVER)) {
    next if $pid = fork;                    И Родитель
    die "fork: $!" unless defined $pid;     " Неудача
    # otherwise child
    close(SERVER);                          8 Не нужно для потомка
    # ... Сделать что-то
    exit;                                   # Выход из потомка
} continue {
    close(CLIENT);                          # Не нужно для родителя
}
```

Комментарий

Подобный подход очень часто используется в потоковых (SOCK_STREAM) серверах на базе сокетов Интернета и UNIX. Каждое входящее подключение получает собственный дубликат сервера. Общая модель выглядит так:

1. Принять потоковое подключение.
2. Ответвить дубликат для обмена данными с этим потоком.
3. Вернуться кп. 1.

Такая методика не используется с **датаграммными** сокетами (SOCK_ DGRAM) из-за особенностей обмена данными в них. Из-за **времени,** затраченного на разветвление, эта модель непрактичнадля UDP-серверов. Вместо продолжительных **соединений,** обладающих определенным состоянием, серверы SOCK_DGRAM работают с непредсказуемым набором **датаграмм,** обычно без определенного состояния. В этом варианте наша модель принимает следующий вид:

1. **Принять датаграмму.**
2. Обработать датаграмму.
3. Вернуться к п. 1.

Новое соединение обрабатывается порожденным процессом. Поскольку сокет SERVER никогда не будет использоваться этим процессом, мы немедленно закрываем его. Отчасти это делается из стремления к порядку, но в основном — для того, чтобы серверный сокет закрывался при завершении родительского (серверного) процесса. Если потомки не будут закрывать сокет SERVER, операционная система будет считать его открытым даже после завершения родителя. За подробностями обращайтесь к рецепту 17.9.

%SIG обеспечивает чистку таблицы процессов после завершения потомков (см. главу 16).

Смотри также

Описание функций fork и accept в *perlfunc*(1)рецепты 16.15; 16.19; 17.12—17.13.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

17.12. Серверы с предварительным ветвлением

Проблема

Вы хотите написать сервер, параллельно обслуживающий нескольких клиентов (как и в предыдущем разделе), однако подключения поступают так быстро, что ветвление слишком сильно замедлит работу сервера.

Решение

Организуйте пул заранее разветвленных потомков, как показано в примере 17.5.

Пример 17.5. **preforker**

```
#!/usr/bin/perl
#              сервер с предварительным ветвлением
use IO  Socket,
use Symbol,
use POSIX

# Создать сокет SERVER, вызвать bind и прослушивать порт
$server = IO  Socket  INET->new(LocalPort => 6969,
                                Type      => SOCK_STREAM,
                                Proto     =>  tcp  ,
                                Reuse     => 1,
                                Listen    => 10 )
  or die  making socket  $@\n ,

# Глобальные переменные
$PREFORK                = 5,    в Количество поддерживаемых потомков
$MAX_CLIENTS_PER_CHILD  = 5,    " Количество клиентов, обрабатываемых
                                0 каждым потомком
%children               = (),   " Ключами являются текущие
                                # идентификаторы процессов-потомков
                                # Текущее число потомков

sub REAPER {                    # Чистка мертвых потомков
    $SIG{CHLD} = \&REAPER,
    my $pid = wait,


}

sub HUNTSMAN (                      # Обработчик сигнала SIGINT
    local($SIG{CHLD}) =  IGNORE ,   " Убиваем своих потомков
    kill  INT  => keys
    exit,                           # Корректно завершиться
}
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.5 (продолжение)

```
# Создать потомков
for (1    $PREFORK) {
    make_new_child(),
}

# Установить обработчики сигналов
$SIG{CHLD}  =  \&REAPER,
$SIG{INT}   =  \&HUNTSMAN,

# Поддерживать численность процессов
while (1) {
    sleep                           # Ждать сигнала (например,
                                     # смерти потомка)
    for ($i = $children, $1 < $PREFORK, $i++) {
        make_new_child(),           # Заполнить пул потомков
    }
}

sub make_new_child {
    my $pid,
    my $sigset,

    # Блокировать сигнал для fork
    $sigset = POSIX  SigSet->new(SIGINT),
    sigprocmask(SIG_BLOCK, $sigset)
        or die  Can t block SIGINT for fork  $!\n ,

    die  fork  $!  unless defined ($pid = fork),

    if ($pid) {
        # Родитель запоминает рождение потомка и возвращается
        sigprocmask(SIG_UNBLOCK, $sigset)
            or die  Can t unblock SIGINT for fork  $!\n ,
        $children{$pid} = 1,
        $children++,

    } else {
        # Потомок *не может* выйти из этой подпрограммы
        $SIG{INT} =  DEFAULT ,    # Пусть SIGINT убивает процесс,
                                  # как это было раньше

        # Разблокировать сигналы
        sigprocmask(SIG_UNBLOCK, $sigset)
            or die  Can t unblock SIGINT for fork  $!\n ,

        # Обрабатывать подключения, пока их число не достигнет
        # $MAX_CLIENTS_PER_CHILD
        for ($i=0, $1 < $MAX_CLIENTS_PER_CHILD, $i++) {
            $client = $server->accept()      or  last,
            в Сделать что-то с соединением
        }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            # Корректно убрать мусор и завершиться

            # Этот выход ОЧЕНЬ важен, в противном случае потомок начнет
            # плодить все больше и больше потомков, что в конечном счете
            # приведет к переполнению таблицы процессов
            exit;
        }
    }
```

Комментарий

Программа получилась большой, но ее логика проста: родительский процесс никогда не работает с клиентами сам, а вместо этого ответвляет $PREFORK потомков. Родитель следит за количеством потомков и своевременно плодит процессы, чтобы заменить мертвых потомков. Потомки завершаются после обработки $MAX_CLIENTS_PER_CHILD клиентов.

Пример 17.5 более или менее прямолинейно реализует описанную логику. Единственная проблема связана с обработчиками сигналов: мы хотим, чтобы родитель перехватывал SIGINT и убивал своих потомков, и устанавливает для этого свой обработчик сигнала &HUNTSMAN. Но в этом случае нам приходится соблюдать меры предосторожности, чтобы потомок не унаследовал тот же обработчик после ветвления. Мы используем сигналы POSIX, чтобы заблокировать сигнал на время ветвления (см. рецепт 16.20).

Используя этот код в своих программах, проследите, чтобы в make_new_child никогда не использовался выход через случае потомок вернется, станет родителем и начнет плодить своих собственных потомков. Система переполнится процессами, прибежит разъяренный системный администратор — и вы будете долго и мучительно жалеть, что не обратили должного внимания на этотабзац.

В некоторых операционных системах (в первую очередь — Solaris) несколько потомков не могут вызывать accept для одного сокета. Чтобы гарантировать, что лишь один потомок вызывает accept в произвольный момент времени, придется использовать блокировку файлов.

Смотри также

Описание функции select *в perlfunc(1);* страница руководства *fcntl*(2) вашей системы (если есть); документация по стандартным модулям Fcntl, Socket, IO::Select, IO::Socket и Tie::RefHash; рецепты 17.11-17.12.

17.13. Серверы без ветвления

Сервер должен обрабатывать несколько одновременных подключений, но вы не хотите ответвлять новый процесс для каждого соединения.

Решение

Создайте массив открытых клиентов, воспользуйтесь select для чтения информации по мере ее поступления и работайте с клиентом лишь после получения полного запроса от него, как показано в примере 17.6.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.6. nonforker

```
#!/usr/bin/perl -w
# nonforker - мультиплексный сервер без ветвления
use POSIX;
use IO::Socket;
use IO::Select;
use Socket;
use Fcntl;
use Tie::RefHash;

$port = 1685;                й Замените по своему усмотрению

# Прослушивать порт.
$server = IO::Socket::INET->new(LocalPort => $port,
                                Listen    => 10 )
  or die "Can't make server socket: $@\n";

" Начать с пустыми буферами
%inbuffer  = ();
%outbuffer = ();
%ready     = ();

            'Tie::RefHash';

nonblock($server);
$select = IO::Select->new($server);

# Главный цикл: проверка чтения/принятия, проверка записи,
# проверка готовности к обработке
while (1) {
    my $client;
    my $rv;
    my $data;

    # Проверить наличие новой информации на имеющихся подключениях

    # Есть ли что-нибудь для чтения или подтверждения?
    foreach $client

        if ($client == $server) {
            # Принять новое подключение

            $client = $server->accept();
            $select->add($client);
            nonblock($client);
        } else {
            # Прочитать данные
            $data = '';
            $rv   = $client->recv($data, POSIX::BUFSIZ, 0);

            unless (defined($rv) && length $data) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
            # Это должен быть конец файла, поэтому закрываем клиента
            delete $inbuffer{$client},
            delete $outbuffer{$client},
            delete $ready{$client},

            $select->remove($client),
            close $client,
            next,
        }

        $inbuffer{$client} = $data,

        # Проверить, говорят ли данные в буфере или только что
        # прочитанные данные о наличии полного запроса ожидающего
        # выполнения Если да - заполнить
        # запросами, ожидающими обработки
        while ($inbuffer{$client} =" s/( *\n)//) {
            push( @{$ready{$client}}, $1 ),
        .   }
    }
}

# Есть ли полные запросы для обработки?
foreach $client (keys %ready) {
    handle($client),
}

# Сбрасываемые буферы ^
        $client ($select->can_write(1))
    # Пропустить этого клиента, если нам нечего сказать
    next unless exists $outbuffer{$client},

    $rv = $client->send($outbuffer{$client}, 0)
    unless (defined $rv) {
        tt Пожаловаться, но следовать дальше
        warn I was told I could write, but I can t \n
        next,
    }
    if ($rv == length $outbuffer{$client} ||
        {$' == POSIX EWOULDBLOCK) {
        substr($outbuffer{$client}, 0, $rv) =
        delete $outbuffer{$client} unless length $outbuffer{$client},
    } else {
        # Не удалось записать все данные и не из-за блокировки
        # Очистить буферы и следовать дальше
        delete $inbuffer{$client},
        delete $outbuffer{$client},
        delete

        $select->remove($client),
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.6 (продолжение)

```
            close($client),
            next;
        }
    }

    # Внеполосные данные?
            $client ($select->has_exception(0))     аргумент - тайм-аут
        # Обработайте внеполосные данные, если хотите.
    }
}

# handle($socket) обрабатывает все необработанные запросы
# для клиента $client
sub handle {
    # Запрос находится в $ready{$client}
    # Отправить вывод в $outbuffer{$client}
    my $client = shift;


                    (@{$ready{$client}})
                    екст запроса
        # Занести текст ответа в $outbuffer{$client}
    }
    delete $ready{$client};
}

# nonblock($socket) переводит сокет в неблокирующий режим
sub nonblock {
    my $socket = shift;
    my $flags,


    $flags = fcntl($socket, F_GETFL, 0)
            or die "Can't get flags for socket: $!\n";
    fcntl($socket, F_SETFL, $flags | O_NONBLOCK)
            or die "Can't make socket nonblocking: $!\n";
}
```

Комментарий

Как видите, одновременно обрабатывать несколько клиентов в одном процессе сложнее, чем ответвлять специальные процессы-дубликаты. Приходится выполнять много работы за операционную систему — например, делить время между разными подключениями и следить, чтобы чтение осуществлялось без блокировки.

Функция select сообщает, в каких подключениях есть данные, ожидающие чтения, какие подключения позволяют записать данные или имеют непрочитанные внеполосные данные. Мы могли бы использовать встроенную функцию Perl select, но это усложнит работу с манипуляторами. Поэтому мы используем стандартный (для версии 5.004) модуль IO::Select.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Функции getsockopt и setsockopt включают неблокирующий режим для серверного сокета. Иначе заполнение буферов сокета одного клиента привело бы к приостановке работы сервера до очистки буферов. Однако применение неблокирующего ввода/вывода означает, что нам придется разбираться с неполными операциями чтения/записи. Мы не сйожем просто использовать оператор о, блокирующий до того, как станет возможным чтение всей записи, или print для вывода всей записи. Буфер %inbuffer содержит неполные команды, полученные от клиентов, %outbuffer — неотправленные данные, а массивы необработанных сообщений.

Чтобы использовать этот код в своей программе, выполните три действия. Во-первых, измените вызов IO::Socket::INET и включите в него порт своего сервера. Во-вторых, измените код, который переносит записи из inbuffer в очередь настоящее время'каждая строка (текст, заканчивающийся \n) рассматривается как запрос. Если ваши запросы не являются отдельными строками, внесите необходимые изменения.

```
while ($inbuffer{$client} =^ s/(.*\n)//) {
    push( @{$ready{$client}}, $1 ),
}
```

Наконец, измените середину цикла в handler так, чтобы в ней действительно создавался ответ на запрос. В простейшей программе эхо-вывода это выглядит так

```
$outbuffer{$client}
```

Обработка ошибок предоставляется читателю в качестве упражнения для самостоятельной работы. На данный момент предполагается, что любая ошибка при чтении или записи завершает подключение клиента. Вероятно, это слишком сурово, поскольку "ошибки" вроде EINTR или EAGAIN не должны приводить к разрыву соединения (впрочем, при использовании select вы *никогда* не должны получать EAGAIN).

Смотри также

Описание функции select в *perlfunc(1);* страница руководства *fcntl*(2) вашей системы (если есть); документация по стандартным модулям Fcntl, Socket, IO::Select, IO::Socket и Tie::RefHash; рецепты 17.11-17.12.

17.14. Написание распределенного сервера

Проблема

Требуется написать сервер дЛя компьютера с несколькими IP-адресами, чтобы он мог выполнять различные операции для каждого адреса.

Решение

Не привязывайте сервер к определенному адресу. Вместо этого вызовите bind с аргументом INADDR_ANY. После того как подключение будет принято, вызов getsockname для клиентского сокета позволяет узнать, к какому адресу он подключился:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use Socket,

socket(SERVER, PF_INET, SOCK_STREAM, getprotobyname( tcp )),
setsockopt(SERVER, SOL_SOCKET SO_REUSEADDR, 1),
bind(SERVER, sockaddr_in($server_port, INADDR_ANY))
    or die Binding $!\n ,

# Цикл принятия подключений
while (accept(CLIENT, SERVER)) {
                        getsockname(CLIENT),
    ($port, $myaddr)   = sockaddr_in($my_socket_address),
}
```

Комментарий

Если функция getpeername (см. рецепт 17.7) возвращает адрес удаленного конца сокета, то функция getsockname возвращает адрес локального конца. При вызове bind с аргументом INADDR_ANY принимаются подключения для всех адресов данного компьютера, поэтому для определения адреса, к которому подключился клиент, можно использовать функцию getsockname.

При использовании модуля IO.::Socket::INET программа будет выглядеть так:

```
$server = IO Socket INET->new(LocalPort => $server_port,
                              Type      => SOCK_STREAM,
                              Proto     => tcp ,
                              Listen    => 10)
                        server socket $@\n ,

while ($client = $server->accept()) {
    $my_socket_address = $client->sockname(),
    ($port, $myaddr)   = sockaddr_in($my_socket_address),
    й
}
```

Если не указать локальный порт при вызове IO 'Socket INET->new, привязка сокета будет выполнена для INADDR_ANY.

Если **вы** хотите, чтобы при прослушивании сервер ограничивался ***конкретным*** виртуальным хостом, **не** используйте INADDR_ANY. Вместо этого следует вызвать bind для конкретного адреса хоста:

```
use Socket,

$port = 4269,                    # Порт
$host = specific host com ,      й Виртуальный хост

socket(Server, PF_INET, SOCK_STREAM, getprotobyname( tcp ))
    or die socket $! ,
bind(Server, sockaddr_in($port, inet_aton($host)))
    or die bind $! ,
while                   accept(Client, Server)) {
    #
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Описание функции `getsockname` в *perlfunc*(1); документация по стандартным модулям Socket и IO::Socket; раздел "Sockets" *в perlipc(1).*

17.15. Создание сервера-демона

Проблема

Вы хотите, чтобы ваша программа работала в качестве демона.

Решение

Если вы — параноик с правами привилегированного пользователя, для начала вызовите `chroot` для безопасного каталога:

```
chroot( /var/daemon )
    or die  Couldn t chroot to /var/daemon  $!  ,
```

Вызовите `fork` и завершите родительский процесс.

```
$pid = fork,
exit if $pid,
die  Couldn t fork  $!  unless defined($pid),
```

Разорвите связь с управляющим терминалом, с которого был запущен процесс, — при этом процесс перестает входить в группу процессов, к которой он принадлежал.

```
use  POSIX,

POSIX  setsid()
    or  die  Can t start a new session  $!  ,
```

Перехватывайте фатальные сигналы и устанавливайте флаг, означающий, что мы хотим корректно завершиться:

```
$time_to_die = 0,

sub  signal_handler  {
    $time to die = 1,
}

$SIG{INT} = $SIG{TERM} = $SIG<HUP} = \&signal_handler,
И Перехватить или игнорировать $SIG{PIPE>
```

Настоящий код сервера включается в цикл следующего вида:

```
until ($time_to_die) {
    Я
}
```

Комментарий

До появления стандарта POSIX у каждой операционной системы были свои средства, с помощью которых процесс говорил системе: "Я работаю в одиночку;

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

пожалуйста, не мешайте мне". Появление POSIX внесло в происходящее относительный порядок. Впрочем, это не мешает вам использовать любые специфические функции вашей операционной системы.

К числу этих функций принадлежит `chroot`, которая изменяет корневой каталог процесса (/). Например, после вызова `chroot '/var/daemon` при попытке прочитать файл */etc/passwd* процесс в действительности прочитает файл */var/daemon/etc/passwd*. Конечно, при вызове функции `chroot` необходимо скопировать все файлы, с которыми работает процесс, в новый каталог. Например, процессу может потребоваться файл */var/daemon/bin/csh*. По соображениям безопасности вызов `chroot` разрешен только привилегированным пользователям. Он выполняется на серверах FTP при анонимной регистрации. На самом деле становиться демоном необязательно.

Операционная система предполагает, что родитель ожидает смерти потомка. Для нашего процесса-демона это не нужно, поэтому мы разрываем наследственные связи. Для этого программа вызывает `fork` и `exit`, чтобы потомок не был связан с процессом, запустившем родителя. Затем потомок закрывает все файловые манипуляторы, полученные от родителя (STDIN, STDERR и STDOUT), и вызывает POSIX `setsid`, чтобы обеспечить полное отсоединение от родительского терминала.

Все почти готово. Сигналы типа SIGINT не должны немедленно убивать наш процесс (поведение по умолчанию), поэтому мы перехватываем их с помощью %SIG и устанавливаем флаг завершения. Далее главная программа работает по принципу: "Пока не убили, что-то делаем".

Сигнал SIGPIPE — особый случай. Получить его нетрудно (достаточно записать что-нибудь в манипулятор, закрытый с другого конца), а по умолчанию он ведет себя довольно сурово (завершает процесс). Вероятно, его желательно либо проигнорировать (`$SIG{PIPE} = IGNORE`), либо определить собственный обработчик сигнала и организовать его обработку.

Смотри также

Страницы руководства *setsid*(2) и *chroot*(1) вашей системы (если есть); описание функции `chroot` в *perlfunc*(1).

17.16. Перезапуск сервера по требованию

Проблема

При получении сигнала HUP сервер должен перезапускаться, по аналогии с `inetd` или `httpd`.

Решение

Перехватите сигнал SIGHUP и перезапустите свою программу:

```
$SELF = /usr/local/libexec/myd ,    # Моя программа
@ARGS = qw(-l /var/log/myd -d),     # Аргументы

$SI6{HUP} = \&phoenix,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
sub phoenix {
    # Закрыть все соединения, убить потомков и
    # приготовиться к корректному возрождению.
    exec($SELF, @ARGS)            or d]e                    $!\n";
}
```

Комментарий

Внешне все выглядит просто ("Получил сигнал HUP — перезапустись"), но на самом деле проблем хватает. Вы должны знать имя своей программы, а определить его непросто. Конечно, можно воспользоваться переменной $0 модуля FindBin. Для нормальных программ этого достаточно, но важнейшие системные утилиты должны проявлять большую осторожность, поскольку правильность $0 не гарантирована. Имя программы и аргументы йожно жестко закодировать в программе, как это сделано в нашем примере. Однако такое решение не всегда удобно, поэтому имя и аргументы можно читать из внешнего файла (защищая подлинность его содержимого на уровне файловой системы).

Обработчик сигнала обязательно должен устанавливаться *после* определения $SELF и @ARGS, в противном случае может возникнуть ситуация перехвата — SIGHUP потребует перезапуска, а вы не будете знать, что запускать. Это приведет к гибели вашей программы.

Некоторые серверы при получении SIGHUP не должны перезапускаться — они всего лишь заново читают свой конфигурационный файл:

```
$CONFIG_FILE = "/usr/local/etc/myprog/server_conf.pl";
$SIG{HUP} = \&read_config;
sub read_config {
    do $CONFIG_FILE;
}
```

Некоторые умные серверы даже автоматически перезагружают свои конфигурационные файлы в случае их обновления. Вам даже не придется ни о чем сигнализировать.

Смотри также

Описание функции exec в *perlfunc*(1); рецепты 8.16—8.17; 16.15.

17.17. Программа: backsniff

Программа *backsniff* регистрирует попытки подключения к портам. Она использует модуль Sys::Syslog, а ему, в свою очередь, нужна библиотека *syslog.ph,* которая не обязательно присутствует в вашей системе. Попытка подключения регистрируется с параметрами LOG_NOTICE и LOG_DAEMON. Функция getsockname идентифицирует порт, к которому произошло подключение, а getpeername — компьютер, установивший соединение. Функция getservbyport преобразует локальный номер порта (например, 7) в название службы (например, "echo").

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

В системном журнале появляются записи:

```
May 25 15:50:22 ooprolith sniffer: Connection from 207.46.131.141 to
207.46.130.164:echo
```

В файл *inetd.conf* включается строка следующего вида:

```
echo            tcp  nowait   nobody  /usr/scripts/snfsqrd  sniffer
```

Исходный текст программы приведен в примере 17.7.

Пример 17.7. **backsniff**

```
#!/usr/bin/perl -w
# backsniff - регистрация попыток подключения к определенным портам

use Sys::Syslog;
use Socket;

# Идентифицировать порт и адрес
$sockname          = getsockname(STDIN)
                     or die "Couldn't identify myself: $!\n";
($port, $iaddr)    = sockaddr_in($sockname);
$my_address        = inet_ntoa($iaddr);

# Получить имя службы
$service = (getservbyport($port, "tcp"))[0] || $port;
#
$sockname          = getpeername(STDIN)
                         or die "Couldn't identify other end: $!\n";
($port, $iaddr)    = sockaddr_in($sockname);
                     inet_ntoa($iaddr);

# Занести информацию в журнал
openlog("sniffer", "ndelay", "daemon");
syslog("notice", "Connection from %s to %s %s\n",

        $my_address, $service);
closelog();
exit;
```

17.18. Программа: fwdport

Предположим, у вас имеется защитный брандмауэр. Где-то в окружающем мире есть сервер, к которому обращаются внутренние компьютеры, но доступ к серверу разрешен лишь процессам, работающим на брандмауэре. Вы не хотите, чтобы при каждом обращении к внешнему серверу приходилось заново регистрироваться на компьютере брандмауэра.

Например, такая ситуация возникает, когда Интернет-провайдер вашей компании позволяет читать новости при поступлении запроса с брандмауэра, но отвергает все подключения NNTP с остальных адресов. Вы как администратор брандмауэра не хотите, чтобы на нем регистрировались десятки пользователей — лучше разрешить им читать и отправлять новости со своих рабочих станций.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Программа *fwdport* из примера 17.8 содержит общее решение этой проблемы. Вы можете запустить любое количество экземпляров, по одному для каждого внешнего запроса. Работая на брандмауэре, она общается с обоими мирами. Когда кто-то хочет воспользоваться внешней службой, он связывается с нашим прокси-сервером, который далее действует по его поручению. Для внешней службы подключение устанавливается с брандмауэра и потому является допустимым. Затем программа ответвляет двапроцесса: первый читает данные с внешнего сервера и передает их внутреннему клиенту, а второй читает данные от внутреннего клиента и передает их внешнему серверу.

Например, командная строка может выглядеть так:

```
% fwdport -s nntp -1 fw.oursite.com -r news.bigorg.com
```

Это означает, что программа выполняет функции сервера NNTP, прослушивая локальные подключения на порте NNTP компьютера *fw.oursite.com*. При поступлении запроса она связывается с *news.bigorg.com* (на том же порте) и организует обмен данными между удаленным сервером и локальным клиентом.

Рассмотрим другой пример:

```
% fwdport -1 myname:9191 -r news.bigorg.com:nntp
```

На этот раз мы прослушиваем локальные подключения на порте 9191 хоста myname и связываем клиентов с удаленным сервером *news.bigorg.com* через порт NNTP.

В некотором смысле *fwdport* действует и как сервер, и как клиент. Для внешнего сервера программа является клиентом, а для компьютеров за брандмауэром — сервером. Эта программа завершает данную главу, поскольку в ней продемонстрирован практически весь изложенный материал: серверные операции, клиентские операции, удаление зомби, разветвление и управление процессами, а также многое другое.

Пример 17.8. fwdport

```
#!/usr/bin/perl -w
# fwdport - прокси-сервер для внешних служб

use strict;                  И Обязательные объявления
use Getopt::Long;             # Для обработки параметров
use Net::hostent;            # Именованный интерфейс для информации о хосте
use IO·:Socket;               # Для создания серверных и клиентских сокетов
use POSIX ":sys_wait_h";     # Для уничтожения зомби

my (
    %Children,               # Хэш порожденных процессов
    $REMOTE,                  # Внешнее соединение
    $LOCAL,                  '# Для внутреннего прослушивания
    $SERVICE,               ' # Имя службы или номер порта
    $proxy_server,            # Сокет, для которого вызывается accept()
    $ME,                     # Базовое имя программы
);

($ME = $0) =^ s,.*/,,;       # Сохранить базовое имя сценария
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.8 (продолжение)

```
check_args(),                # Обработать параметры
start_proxy(),               # Запустить наш сервер
service_clients(),           # Ждать входящих подключений
die  NOT REACHED  ,          # Сюда попасть невозможно

# Обработать командную строку с применением расширенной версии
# библиотеки getopts
sub check_args {
    GetOptions(
        remote=s      => \$REMOTE,
        local=s       => \$LOCAL,
        service=s     => \$SERVICE,
    ) or die "EOUSAGE,
    usage  $0 [ --remote host ] [  --local interface ] [  --service service ]
EOUSAGE
    die                                unless $REMOTE,
    die  Need local or service       unless $LOCAL  || $SERVICE,
}

# Запустить наш сервер
sub start_proxy {
    my @proxy_server_config = (
      Proto     =>  tcp  ,
      Reuse     => 1,
      Listen    => SOMAXCONN,
    ),
    push @proxy_server_config, LocalPort => $SERVICE if $SERVICE,
    push @proxy_server_config, LocalAddr => $LOCAL   if $LOCAL,
    $proxy_server = IO  Socket  INET->new(@proxy_server_config)
                      die              proxy server  $@',
    print  [Proxy server on  , ($LOCAL || $SERVICE),   initialized ]\n  ,
}

sub service_clients {
    my (
        $local_client,     # Клиент, обращающийся к внешней службе
        $lc_info,          И Имя/порт локального клиента
        $remote_server,    # Сокет для внешнего соединения
        @rs_config,        # Временный массив параметров удаленного сокета
        $rs_info,          # Имя/порт удаленного сервера
        $kidpid,           # Порожденный процесс для каждого подключения
    ),

    $SIG{CHLD} = \&REAPER,    # Уничтожить зомби

    accepting(),

    # Принятое подключение означает, чтО внутренний клиент
    # хочет выйти наружу
    while  ($local_client = $proxy_server->accept())  {
        $lc_info = peerinfo($local_client),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
set_state( servicing local $lc_info ),
printf  [Connect from $lc_info]\n ,

@rs_config = (
    Proto      =>  tcp  ,
    PeerAddr   => $REMOTE,
),
push(@rs_config, PeerPort => $SERVICE) if $SERVICE,

print  [Connecting to $REMOTE
set_state( connecting to $REMOTE ),                    # См  ниже
                  IO Socket INET->new(@rs_config)
                    die        server  $@ ,
print  done]\n ,

$rs_info = peerinfo($remote_server),
set_state( connected to $rs_info ),

$kidpid = fork(),
die  Cannot fork  unless defined $kidpid,
if ($kidpid) {
,   $Children{$kidpid} = time()       # Запомнить время запуска
    close                                        главному процессу
    close $local_client,              # Тоже
    next,                             # Перейти к другому клиенту


# В этой точке программа представляет собой ответвленный
# порожденный процесс, созданный специально для входящего
# клиента, но для упрощения ввода/вывода нам понадобится близнец

close  $proxy_server,                  # Не нужно потомку

$kidpid = fork(),
die  Cannot fork  unless defined $kidpid,

# Теперь каждый близнец сидит на своем месте и переправляет
# строки данных  Видите, как многозадачность упрощает алгоритм?

# Родитель ответвленного процесса, потомок главного процесса
if ($kidpid) {
    set_state( $rs_info --> $lc_info ),
    select($local_client), $| = 1,
    print while
    kill( TERM ,  $kidpid),      # Работа  закончена,
    }                            # убить близнеца
# Потомок потомка, внук главного процесса
else {
    set_state( $rs_info <-- $lc_info ),
    select($remote_server), $| = 1,
    print while <$local_client>,
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 17.8 (продолжение)

```
            kill('TERM', getppid()),     # Работа закончена,
            }                            # убить близнеца
        exit;                            # Тот, кто еще жив, умирает
    } continue {
        accepting(),
    }
}

# Вспомогательная функция для получения строки в формате ХОСТ.ПОРТ
sub peerinfo {
    my $sock = shift;
    my $hostinfo = gethostbyaddr($sock->peeraddr);
            sprintf("%s
                     $hostinfo->name || $sock->peerhost,
                     $sock->peerport),
}

# Сбросить $0, при этом в некоторых системах "ps" выдает
# нечто интересное. строку, которую мы присвоили $0'
sub set_state { $0 = "$ME [@_]"}

# Вспомогательная функция для вызова set_state
sub accepting {
    set_state("accepting proxy for "   ($REMOTE || $SERVICE)),
>

# Кто-то умер Уничтожать зомби, пока они остаются.
# Проверить время их работы.
sub REAPER {
    my $child;
    my $start,
    while (($child= waitpid(-1,WNOHANG)) > 0) {
        if ($start = $Children{$child}) {
            my $runtime = time() - $start;
            printf "Child $child ran %dm%ss\n",
                $runtime / 60, $runtime % 60;
            delete $Children{$child};
        } else {
            print          kid $child exited $?\n";
        }
    }
    # Если бы мне пришлось выбирать между System V и 4 2,
    # я бы уволился. - Питер Ханиман
    $SIG{CHLD} = \&REAPER;
};
```

Смотри также

Getopt::Long(3), Net::hostent(3), IO::Socket(3), POSIX(3), глава 16, раздел "Написание двусторонних клиентов" этой главы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Протоколы Интернета 18

Так называемый "телефон" обладает слишком многими недостатками, что не позволяет серьезно рассматривать его как средство связи. Для нас это устройство совершенно бесполезно.

Служебная записка Western Union, 1876 г.

Введение

Правильная работа с сокетами — лишь часть программирования сетевых коммуникаций. Даже если вы организовали обмен данными между двумя программами, все равно вам понадобится определенный *протокол*. С помощью протокола каждая сторона узнает, когда передаются или принимаются данные и кто именно отвечает за данный аспект службы.

Наиболее распространены следующие протоколы Интернета.

Протокол	Расшифровка	Описание
FTP	File Transfer Protocol	Копирование файлов между удаленными компьютерами
telnet		Удаленное подключение к компьютеру
rsh и rcp	Remote shell and Remote Copy	Удаленная регистрация и копирование файлов
NNTP	Network News Transfer Protocol	Чтение и отправка новостей Usenet
HTTP	Hypertext Transfer Protocol	Пересылка документов по Web
SMTP	Simple Mail Transfer Problem	Отправка почты
POP3	Post Office Protocol	Чтение почты

Даже такая относительно простая задача, как подключение к удаленному компьютеру, требует довольно сложных переговоров между клиентом и сервером и многочисленных параметров с динамической настройкой. Если бы при каждой попытке воспользоваться сетевой службой вам приходилось писать код Perl с реализацией этих протоколов, ничего хорошего бы не вышло — программы содержали бы неимоверное количество ошибок.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

К счастью, на CPAN имеются модули для всех протоколов. Большинство модулей реализует клиентскую, а не серверную сторону протокола. Следовательно, программа сможет использовать эти модули для отправки почты, но не для выполнения функций почтового сервера, к которому подключаются другие клиенты. Она может читать и отправлять новости, но не являться сервером новостей для других клиентов; обмениваться файлами с сервером FTP, но не быть сервером FTP; и т. д.

Большинство этих модулей принадлежит иерархии Net::. Модуль Net::FTP используется для отправки и приема файлов по FTP; модуль Net::NNTP — для чтения и отправки новостей Usenet; модуль Net::Telnet — для имитации подключения к другому компьютеру; модуль Net::Whois — для получения данных об имени домена; модуль Net::Ping — для проверки связи с компьютером, а модули Net::POP3 и Mail::Mailer — для отправки и получения почты. Протокол CGI рассматривается в главе 19 "Программирование CGI», а протокол HTTP — в главе 20 "Автоматизация в Web".

Большинство этих модулей написал Грэхем Барр, автор модулей IO::Socket, использовавшихся в низкоуровневых сетевых коммуникациях в главе 17 "Сокеты». Он написал Net::FTP, Net::NNTP, Net::POP3 и Mail::Mailer. Джей Роджерс (Jey Rogers) написал Net.:Telnet, а Чип Зальценберг (Chip
Благодаря им вам не придется заново изобретать велосипед!

18.1. Простой поиск в DNS

Проблема

Требуется определить IP-адрес хоста или преобразовать IP-адрес в имя. Сетевые серверы решают эту задачу в процессе аутентификации своих клиентов, а клиенты — когда пользователь вводит имя хоста, но для библиотеки сокетов Perl нужен IP-адрес. Более того, многие серверы регистрируют в файлах журналов IP-адреса, но аналитическим программам и людям удобнее работать с именами хостов.

Решение

Для получения всех IP-адресов по имени хоста (например, *www.perl.com*) воспользуйтесь функцией `gethostbyname`:

```
use Socket,

            gethostbyname($name)      die                $name  $'\n ,
@addresses          inet_ntoa($_)   @addresses[4
# @addresses - список IP-адресов ( 208 201 239 48 ,  208 201 239 48 )
```

Если вам нужен только первый адрес, воспользуйтесь функцией `inet_aton`:

```
use Socket,

          inet_ntoa(inet_aton($name)),
                IP-адрес ( 208 201 239 48 )
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Для получения имени хоста по строке с IP-адресом (например, "208.201.239.48"), воспользуйтесь следующим фрагментом:

```
use Socket,

$name = gethostbyaddr(inet_aton($address), AF_INET)
            or die "Can't                    $!\n";
# $name - имя хоста ("www.perl.com")
```

Комментарий

Наша задача усложняется тем, что функции Perl являются простыми оболочками для системных функций C, поэтому IP-адреса приходится преобразовывать из ASCII-строк ("208. 201. 239.48") в структуры C. Стандартный модуль Socket содержит функцию inet_aton для перехода от ASCII к упакованному числбвому формату и функцию inet_ntoa, выполняющую обратное преобразование:

```
use Socket;
                    inet_aton("208.146 140.1");
$ascii_address  = inet_ntoa($packed_address),
```

Функция gethostbyname получает строку, содержащую имя хоста (или IP-адрес). В скалярном контексте она возвращает IP-адрес удаленного хоста, который можно передать inet_ntoa (или undef в случае ошибки). В списковом контексте она возвращает список, состоящий по крайней мере из пяти элементов (или пустой список в случае ошибки). Список состоит из следующих элементов.

Индекс	Значение
0	Официальное имя хоста
1	Синонимы (строка, разделенная пробелами)
2	Тип адреса (обычно AF_INET)
3	Длина структуры адреса (не имеет значения)
4,5...	Структурыадресов

Имени хоста может соответствоватьнесколько IP-адресов; в частности, это происходит на сильно загруженных Web-серверах, гдедля снижения загрузки на разных компьютерах размещаются идентичные Web-страницы. В подобных ситуациях сервер DNS, предоставляющий адреса, чередует их, обеспечивая сбалансированную нагрузку на сервер. Если вы хотите выбрать IP-адрес для подключения, просто возьмите первый адрес в списке (а если он не работает, попробуйте остальные адреса):

```
$packed = gethostbyname($hostname)
            or die "Couldn't                    $hostname  $!\n";
           inet_ntoa($packed);
print "I will use          as the          for $hostname\n";
```

Используя имена хостов для разрешения или отказа в обслуживании, будьте осторожны. Любой желающий может настроить свой сервер DNS так, чтобы его компьютер идентифицировался как *www.whitehouse.gov*, *www.yahoo.com* или

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

this.is.not.funny. Нельзя сказать, действительно ли ему принадлежит то имя, на которое он претендует, пока вы не вызовете `gethostbyname` и не проверите исходный адрес по адресному списку для данного имени.

```
#           - проверяемый IP-адрес (например, "128.138.243.20")
use Socket;
$name     = gethostbyaddr(inet_aton($address), AF_INET)
                or die "Can't look up          : $!\n";
@addr     = gethostbyname($name)
                or die "Can't look up$name : $!\n";
$found    =      {          eq inet_ntoa($_) } @addr[4..$#addr];
```

Оказывается, даже такой алгоритм недает полной уверенности в полученном имени, поскольку существуют разнообразные обходные пути. Даже IP-адрес, из которого вроде бы поступают пакеты, может быть поддельным, и в процессе аутентификации не следует полагаться на сетевой уровень. В действительно важных ситуациях всегда выполняйте аутентификацию сами (с помощью паролей или криптографических методов), поскольку сеть IPv4 не проектировалась для соблюдения безопасности.

Информация о хосте не ограничивается адресами и синонимами. Чтобы полноценно работать с дополнительными данными, воспользуйтесь модулем Net::DNS с CPAN. Программа 18.1 показывает, какполучить записи MX (mail exchange) для произвольного хоста.

Пример 18.1. mxhost

```
#!/usr/bin/perl
# mxhost - поиск записей mx для хоста
use Net::DNS;

$host = shift;
     = Net::DNS::Resolver->new();
@mx = mx($res, $host)
   or die "Can't find MX              $host  (".$res->errorstring,")\n";

                (@mx) {
    print                        " ", $record->exchange, "\n";
}
```

Примерный вывод выглядит так:

```
% mxhost cnn.com
10   mail.turner.com
30   alfw2.turner.com
```

Функция `inet_aton`, как и `gethostbyname`, получаетстроку с именем хоста или IP-адресом, однако она возвращает только первый IP-адрес для данного хоста. Чтобы узнать все IP-адреса, приходится писать дополнительный код. Модуль Net::hostent поддерживает соответствующие средства доступа по имени. Пример 18.2 показывает, как это делается.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 18.2. **hostaddrs**

```
#!/usr/bin/perl
# hostaddrs - канонизация имени и вывод адресов
use Socket;
use Net::hostent;
$name = shift;
if ($hent = gethostbyname($name)) {
    $name      = $hent->name;            # Если отличается
               = $hent->addr_list;
    @addresses = map { inet_ntoa($_) } @$addr_ref;
}
print "$name => @addresses\n";
```

Примерный результат выглядит так:

```
% hostaddrs www.ora.com
helio.ora.com  ->  204.148.40.9

% hostaddrs www.whitehouse.gov
www.whitehouse.gov  ">  198.137.240.91   198.137.240.92
```

Смотри также

Описание функций `gethostbyname` и `gethostbyaddr` *в perlfunc(1);* документация по модулю Net::DNS с CPAN; документация по стандартным модулям Socket и Net::hostent.

18.2. Клиентские операции FTP

Проблема

Вы хотите подключиться к серверу FTP, чтобы отправить или принять с него файлы. Например, **вы** решили автоматизировать разовую пересылку многих файлов или автоматически создать зеркальную копию целого раздела сервера **FTP.**

Решение

Воспользуйтесь модулем Net::FTP с CPAN.

```
use Net::FTP;

$ftp = Net::FTP->new("ftp.host.com")   or die "Can't connect: $@\n";
$ftp->login($username, $password)      or die "Couldn't login\n";
                                              "Couldn't change directory\n";
$ftp->get($filename)                   or die "Couldn't get $filename\n";
$ftp->put($filename)                   or die "Couldn't put $filename\n";
```

Комментарий

Работа с модулем Net::FTP состоит из трех шагов: *подключение* к серверу, идентификация и *аутентификация* и *пересылка* файлов. Все операции с сервером

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

FTP реализуются методами объекта Net::FTP. При возникновении ошибки методы возвращают undef в скалярном контексте и пустой список в списковом контексте.

Подключение осуществляется конструктором new. В случае ошибки переменной $@ присваивается сообщение об ошибке, а new возвращает undef. Первый аргумент определяет имя хоста сервера FTP и может сопровождаться необязательными параметрами:

```
$ftp = Net FTP->new( ftp host com ,
                     Timeout => 30,
                     Debug   => 1)
    or die Can t connect $@\n ,
```

Параметр Timeout определяет промежуток времени в секундах, после которого любая операция считается неудачной. Параметр Debug устанавливает уровень отладки (при значении, отличном от нуля, копии всех команд отправляются в STDERR). Строковый параметр Firewall определяет компьютер, являющийся прокси-сервером FTP. Параметр Port задает альтернативный номер порта (по умолчанию используется значение 21, стандартный номер порта FTP). Наконец, если параметр Passive равен true, все пересылки выполняются в пассивном режиме (требование некоторых брандмауэров и прокси-серверов). Параметры Passive переопределяют переменные окружения FTP_FIREWALL и FTP_PASSIVE.

Следующим после подключения шагом является аутентификация. Обычно функция login вызывается с тремя аргументами: именем пользователя, паролем и учетной записью (account).

```
$ftp->login()
    or die Couldn t authenticate \n ,

$ftp->login($username)
    or die Still couldn t authenticate \n ,

$ftp->login($username, $password)
    or die Couldn t authenticate, even with explicit username
           and password \n ,

$ftp->login($username, $password, $account)
    or die No dice It hates me \n ,
```

Если вызвать login без аргументов, Net::FTP с помощью модуля Net::Netrc определяет параметры хоста, к которому вы подключились. Если данные не найдены, делается попытка анонимной регистрации (пользователь anonymous, пароль username@hostname). Если при имени пользователя anonymous пароль не задан, в качестве пароля передается почтовый адрес пользователя. Дополнительный аргумент (учетная запись) в большинстве систем не используется. При неудачной аутентификации функция login возвращает undef.

После завершения аутентификации стандартные команды FTP выполняются с помощью методов, вызываемых для объекта Net::FTP. Методы get и put принимают и отправляют файлы. Отправка файла выполняется так:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$ftp->put($localfile, $remotefile)
    or die Can t send $localfile $!\n ,
```

Если второй аргумент отсутствует, имя удаленного файла совпадает с именем локального файла. Передаваемые данные также можно брать из файлового манипулятора (в этом случае имя удаленного файла передается в качестве второго аргумента):

```
$ftp->put(*STDIN, $remotefile)
    or die Can t send from STDIN $!\n ,
```

Если пересылка прерывается, удаленный файл не удаляется автоматически. Метод put возвращает удаленное имя файла при успешном завершении или undef в случае ошибки.

Метод get, используемый для приема файлов, возвращает локальное имя файла или undef вслучае ошибки:

```
$ftp->get($remotefile, $localfile)
    or die Can t fetch $remotefile $!\n ,
```

Метод get тоже может использоваться для приема файла в манипулятор; он возвращает манипулятор (или undef в случае ошибки):

```
$ftp->get($remotefile, *STDOUT)
    or die Can t fetch $remotefile $!\n ,
```

Необязательный третий аргумент get, смещение в удаленном файле, инициирует пересылку с заданного смещения. Принятые байты дописываются в конец локального файла.

Метод type изменяет режим трансляции файла. Если передать ему строку (A , I , E или L), возвращается предыдущий режим трансляции. Методы ascii, binary, ebcdic и byte вызывают type с соответствующей строкой. При возникновении ошибок (например, если сервер FTP не поддерживает EBCDIC) type и вспомогательные методы возвращают undef.

Методы cwd($remotedir) и pwd используются для установки и определения текущего удаленного каталога. Оба метода возвращают true в случае успеха и false в противном случае. При вызове cwd(") вызывается метод cdup для перехода в родительский каталог текущего каталога. Вызов cwd без аргументов выполняет переход в корневой каталог.

```
$ftp->cwd( /pub/perl/CPAN/images/g-rated ),
print m in , $ftp->pwd(), \n
```

Методы mkdir($remotedir) и rmdir($remotedir) создают и, соответственно, удаляют каталоги на удаленном компьютере. Для создания и удаления каталогов на локальном компьютере применяются встроенные функции mkdir и rmdir. Чтобы создать промежуточные каталоги на пути к указанному, передайте mkdir второй аргумент, равный true. Например, чтобы создать каталоги /pub, /pub/gnat и pub/gnat/perl, воспользуйтесь следующим вызовом:

```
$ftp->mkdir("/pub/gnat/perl", 1)
or die Can t /pub/gnat/perl $!\n
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если функция `mkdir` вызывается успешно, возвращается полный путь к только что созданному каталогу. При неудаче `mkdir` возвращает `undef`.

Методы `ls` и `dir` возвращают список содержимого удаленного каталога. Традиционно `dir` выводит более подробную информацию, чем `ls`, но в обоих случаях стандартного формата не существует. Большинство серверов FTP выводит результаты команд `&` и `&-l`, нонельзя гарантировать, что это правило будет соблюдаться всеми серверами. В списковом контексте эти методы возвращают список строк, возвращенных сервером. В скалярном контексте возвращается ссылка на массив.

```
@lines = $ftp->ls("/pub/gnat/perl")
    or die "Can't get a list of files in /pub/gnat/perl: $!";
                $ftp->dir("/pub/perl/CPAN/src/latest.tar.gz")
    or die "Can't check status of latest.tar.gz: $!\n";
```

Для корректного завершения работы с **FTP** используется метод `quit`:

```
$ftp->quit()    or warn "Couldn't quit.  Oh well.\n";
```

Другие методы переименовывают удаленные файлы, меняют владельца и права доступа, проверяют размер удаленных файлов и т. д. За подробностями обращайтесь к документации по модулю Net::FTP.

Если вы хотите организовать зеркальное копирование файлов между компьютерами, воспользуйтесь превосходной программой mirror, написанной на Perl Ли Маклафлином (Lee McLoughlin) (*http://sunsite.doc.ic.ac.uk/packages/mirror/*).

Смотри также

Страницы руководства *ftp*(1) и *ftpd*(8) вашей системы (если есть); документация по модулю Net::FTP с CPAN.

18,3. Отправка почты

Проблема

Ваша программа должна отправлять почту. Некоторые программы следят за системными ресурсами — например, свободным местом на диске — и рассылают сообщения о том, что ресурс достигопасного предела. Авторы сценариев CGI часто делают так, чтобы при нарушениях работы базы данных программа не сообщала об ошибке пользователю, а отправляла сообщение о проблеме администратору базы данных.

Решение

Воспользуйтесь модулем Mail::Mailer с CPAN:

```
use Mail::Mailer;

$mailer = Mail::Mailer->new();
$mailer->open({ From    => $from_address,
                To      => $to_address,
                Subject => $subject,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    or die "Can't open: $!\n";
print $mailer $body;
$mailer->close();
```

Кроме того, можно воспользоваться программой *sendmail*:

```
open(SENDMAIL, "|/usr/lib/sendmail -oi -t -odq")
                    or die "Can't fork forsendmail: $!\n";
print SENDMAIL <<"EOF";
From: User Originating Mail <me\@host>
To: Final Destination <you\@otherhost>
Subject: A          subject line

Body of the message goes        in as many lines as you like.
EOF
close(SENDMAIL)     or warn "sendmail didn't close nicely";
```

Комментарий

Существуют три варианта отправки почты из программы. Во-первых, можно воспользоваться внешней программой, которая обычно применяется пользователями для отправки почты — например, Mail или mailx; такие программы называются "пользовательскими почтовыми агентами" (MUA, Mail User Agents). Во-вторых, существуют почтовые программы системного уровня (например, *sendmail*); они называются "транспортными почтовыми агентами" (MTA, Mail Transport Agents). Наконец, можно подключиться к серверу SMTP (Simple Mail Transfer Protocol). К сожалению, стандартной программы пользовательского уровня не существует, для *sendmail* не определено стандартного местонахождения, а протокол SMTP довольно сложен. Модуль Mail::Mailer от CPAN избавляет вас от этих сложностей.

При установке модуль Mail::Mailer ищет mail, Mail и другие имена, под которыми обычно скрываются программы отправки почты. Кроме того, он просматривает некоторые распространенные каталоги, где может находиться *sendmail*. При создании объекта Mail::Mailer вы получаете удобный доступ к этим программам (и почтовым серверам SMTP), не заботясь о структуре аргументов или о возвращаемых ошибках.

Создайте объект Mail::Mailer конструктором `Mail::Mailer->new`. При вызове конструктора без аргументов используется метод отправки почты по умолчанию (вероятно, с помощью внешней программы типа mail). Аргументы new позволяют выбрать альтернативный способ отправки сообщений. Первый аргумент определяет способ отправки ("`mail`" для пользовательских почтовых агентов UNIX, "`sendmail`" для программы sendmail и "`smtp`" для подключения к серверу SMTP). Необязательный второй аргумент определяет путь к программе.

Например, следующая команда приказывает Mail::Mailer использовать *sendmail* вместо способа отправки, принятого по умолчанию:

```
$mailer = Mail::Mailer->new("sendmail");
```

В следующем примере вместо mail используется почтовая программа /u/gnat/bin/funkymailer:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$mailer = Mail::Mailer->new( mail , '/u/gnat/bin/funkymailer' ),
```

Подключение к серверу SMTP *mail.myisp.com* выполняется так:

```
$mailer = Mail::Mailer->new( smtp , 'mail.myisp.com' ),
```

При возникновении ошибки в любой части Mail::Mailer вызывается die. Следовательно, для проверки ошибок следует включить код отправки почты в блок eval, после чего проверить переменную $@:

```
eval {
    $mailer = Mail::Mailer->new( bogus , 'arguments' ),
    #
}
if ($@) {
    # Неудачный вызов eval
    print "Couldn't send mail: $@\n" ,
} else {
    # Успешный вызов eval
    print "The authorities have been notified \n" ,
}
```

Если конструктор new не понимает переданные аргументы или не имеет метода по умолчанию при отсутствии аргументов, он инициирует исключение. Модуль Mail::Mailer запускает почтовую программу или подключается к серверу **SMTP** лишь после вызова метода open для заголовков сообщения:

```
$mailer->open( 'From'    => 'Nathan Torkington <gnat@frii.com>' ,
               'To'      => 'Tom Christiansen* <tchrist@perl.com>' ,
               'Subject' => 'The Perl Cookbook' ),
```

Если попытка запустить программу или подключиться к серверу завершилась неудачно, метод open инициирует исключение. После успешного вызова open переменную $mailer можно интерпретировать как файловый манипулятор и вывести в нее основной текст сообщения:

```
print $mailer <<EO_SIG,
Мы когда-нибудь закончим эту книгу?
Жена грозится уйти от меня
Она говорит, что я люблю EMACS больше, чем ее
Что делать?

Нат
EO_SIG
```

Завершив отправку текста, вызовите функцию close для объекта Mail::Mailer:

```
close($mailer)        or die "can't close mailer: $!" ,
```

Впрочем, с программой *sendmail* можно общаться и напрямую:

```
open(SENDMAIL, "|/usr/sbin/sendmail -oi -t -odq ")
            or die "Can't fork for sendmail: $!\n" ,
print SENDMAIL " EOF ,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
From  Tom Christiansen <tchrist\@perl com>
To  Nathan Torkington <gnat\@frii com>
Subject  Re  The Perl Cookbook

(1) Мы никогда не закончим эту книгу
(2) Тот, кто работает с EMACS, не заслуживает любви
(3) Переходи на vi

Том
EOF
close(SENDMAIL),
```

Перед нами тривиальное использование функции open для запуска другой программы (см. рецепт 16.4). Нам приходится указывать полный путь к *sendmail*, поскольку местонахождение этой программы меняется от компьютера к компьютеру. Обычно она находится в каталоге */usr/lib* или */usr/sbin*. Флаги, передаваемые *sendmail*, говорят о том, что программа не должна завершаться при чтении строки, состоящей из одной точки (*-oi*); что адресат сообщения определяется по данным заголовка (*-t*); *а* также о том, чтовместо немедленной доставки сообщение должно помещаться в очередь (*-odq*). Последний параметр важен лишь при отправке больших объемов почты — безнего компьютер быстро захлебнется в многочисленных процессах *sendmail*. Чтобы сообщение доставлялось немедленно (например, во время тестирования или при срочной доставке почты), удалите *-odq* из командной строки.

Мы выводим функцией print все сообщение — заголовки и основной текст, разделяя их пустой строкой. Не существует специальных служебных символов для вставки новых строк (как в некоторых пользовательских почтовых программах), поэтому весь текст интерпретируется буквально. *Sendmail* добавляет заголовки Date и Message-ID (которые все равно пришлось бы генерировать вручную).

Некоторые версии Perl (особенно для Windows и Mac) не имеют аналогов *sendmail* или *mail*. В таких случаях отправка почты осуществляется через сервер SMTP.

Смотри также

Описание функции open в *perlfunc*(1); рецепты 16.4; 16.10; 16.19; 19.6; определение протокола SMTP в документе RFC 821, а также дополнения в последующих RFC.

18.4. Чтение и отправка новостей Usenet

Проблема

Вы хотите подключиться к серверу новостей Usenet для чтения и отправки новостей. Ваша программа может периодически отправлять материалы, собирать статистику по конференции или идентифицировать новичков, чтобы отправить им приветственное сообщение.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь модулем **Net::NNTP** с CPAN:

```
use Net::NNTP;

$server = Net::NNTP->new("news.host.dom")
    or die "Can't connect to news server: $@\n";
($narticles, $first, $last, $name) = $server->group( "misc.test" )
    or die "Can't select misc.test\n";
$headers = $server->head($first)
    or die "Can't get headers from article $first in $name\n";
$bodytext = $server->body($first)
    or die "Can't get body from article $first in $name\n";
$article = $server->article($first)
    or die "Can't get article $first from $name\n";

$server->postok()
    or warn "Server didn't tell me I could post.\n";

$server->post( [ @lines ] )
    or die "Can't post: $!\n";
```

Комментарий

Usenet представляет собой распределенную систему конференций. Обмен сообщениями между серверами обеспечивает одинаковое содержимое всех находящихся на них конференций. Каждый сервер устанавливает собственный **критерий,** определяющий максимальный срок хранения сообщений. Клиентские программы подключаются к выделенному серверу (обычно принадлежащему их организации, Интернет-провайдеру или университету),получая возможность читать существующие или отправлять новые сообщения.

Каждое сообщение состоит из набора заголовков и основного текста, разделенных пустой строкой. Сообщения идентифицируются двумя способами: заголовком *идентификатора сообщения* и *номером сообщения* в конференции. Идентификатор сообщения хранится внутри самого сообщения. Он заведомо остается уникальным независимо от того, с какого сервера Usenet было прочитано сообщение. Если сообщение ссылается на другие сообщения, оно также использует их идентификаторы. Идентификатор сообщения представляет собой строку вида:

```
<0401@jpl-devvax.JPL.NASA.GOV>
```

Также возможна идентификация сообщений по конференции и номеру внутри конференции. Каждый сервер Usenet присваивает своим сообщениям собственные **номера,** поэтому правильность ссылок гарантирована лишь для того сервера Usenet, с которого они были получены.

Конструктор Net::NNTP подключается к заданному серверу Usenet. Если соединение не удается установить, он возвращает `undef` и присваивает переменной **$@** сообщение об ошибке. Если соединение было успешно установлено, new **возвращает** новый объект **Net::NNTP:**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$server = Net::NNTP->new("news.mycompany.com")
    or die "Couldn't connect to news.mycompany.com: $@\n";
```

После установки соединения метод list возвращает список конференций в виде ссылки на хэш, ключи которого соответствуют именам конференций. Ассоциированные значения представляют собой ссылки на массивы, содержащие первый допустимый номер сообщения в конференции, последний допустимый номер сообщения в конференции и строку флагов. Флаги обычно равны "y" (отправка разрешена), но также могут быть равны "m" (модерируемая конференция) или =ИМЯ (данная конференция дублирует конференцию ИМЯ). На сервере могут храниться свыше 17 000 конференций, поэтому выборка всего списка требует некоторого времени.

```
$grouplist = $server->list()
    or die "Couldn't fetch group list\n";

                    %$grouplist)
    if ($grouplist->{$group}->[2] eq 'y') {
        # Отправка в $group разрешена
    }
}
```

По аналогии с концепцией текущего каталога в FTP, протокол NNTP (NetNews Transfer Protocol) поддерживает концепцию текущей конференции. Назначение текущей конференции выполняется методом group:

```
($nartiqles, $first, $last, $name) = $server->group("comp.lang.perl.misc")
    or die "Can't select comp.lang.perl.misc\n";
```

Метод group возвращает список из четырех элементов: количество сообщений в конференции, номер первого сообщения, номер последнего сообщения и название конференции. Если конференция не существует, возвращается пустой список.

Содержимое сообщений можно получить двумя способами: вызвать метод article с идентификатором сообщения или выбрать конференцию методом group, а затем вызвать article с номером сообщения. В скалярном контексте метод возвращает ссылку на массив строк. В списковом контексте возвращается список строк. При возникновении ошибки article возвращает false:

```
@lines = $server->article($message_id)
    or die "Can't fetch article $article_number: $!\n";
```

Для получения заголовка и основного текста сообщения используются соответственно методы head и body. Как и article, они вызываются для идентификатора или номера сообщения и возвращают список строк или ссылку на массив:

```
@group = $server->group("comp.lang.perl.misc")
    or die "Can't select group comp.lang.perl.misc\n";
@lines = $server->head($group[1])
    or die "Can't get headers from first article in comp.lang.perl.misc\n";
```

Метод post отправляет новое сообщение. Он получает список строк или ссылку на массив строк и возвращает true при успешной отправке или false в случае неудачи.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
$server->post(@message)
    or die  Can t post\n ,
```

Метод postok позволяет узнать, разрешена ли отправка сообщений на данный сервер Usenet:

```
unless ($server->postok()) {
    warn  You may not post \n ,
}
```

Полный список методов приведен в man-странице модуля Net::NNTP.

Смотри также

Документация по модулю Net::NNTP от CPAN; RFC 977, "Network News Transport Protocol"; страницы руководства *trn*(1) и *innd*(8) вашей системы (если есть).

18.5. Чтение почты на серверах POP3

Проблема

Требуется принять почту с сервера POP3. Например, программа может получать данные о непрочитанной почте, перемещать ее с удаленного сервера в локальный почтовый ящик или переключаться между Интернетом и локальной почтовой системой.

Решение

Воспользуйтесь модулем Net::POP3 с CPAN:

```
$pop = Net  POP3->new($mail_server)
    or die  Can t open connection to $mail_server   $!\n ,
$pop->login($username, $password)
    or die  Can t authenticate  $!\n ,
$messages = $pop->list
    or die  Can t get list of undeleted messages  $!\n ,
foreach $msgid (keys %$messages) {
    $message = $pop->get($msgid),
    unless (defined $message) {
        warn  Couldn t fetch $msgid from server  $!\n ,
        next,
    }
    tt $message - ссылка на массив строк
    $pop->delete($msgid),
}
```

Комментарий

Традиционно в доставке почты участвовали три стороны: MTA (транспортный почтовый агент - системная программа типа *sendmail*) доставляет почту в накопитель (spool), а затем сообщения читаются с помощью MUA (пользователь-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

ские почтовые агенты — программы типа *mail*). Такая схема появилась в те времена, когда почта хранилась на больших серверах, а пользователи читали сообщения на простейших терминалах. По мере развития PC и сетевых средств появилась потребность в MUA (таких, как *Pine*), которые бы работали на пользовательских компьютерах (а не на том компьютере, где находится накопитель). Протокол POP (Post Office Protocol) обеспечивает эффективное чтение и удаление сообщений во время сеансов TCP/IP.

Модуль Net::POP3 от CPAN обслуживает клиентскую сторону POP. Иначе говоря, он позволяет программе на Perl выполнять функции MUA. Работа с Net::POP3 начинается с создания нового объекта Net::POP3. Конструктору new передается имя сервера POP3:

```
$pop = Net::POP3->new( 'pop.myisp.com' )
    or die "Can't connect to pop.myisp.com: $!\n",
```

При возникновении ошибок все функции Net::POP3 возвращают undef или пустой список в зависимости от контекста вызова. При этом переменная $! может содержать осмысленное описание ошибки (а может и не содержать).

Кроме того, конструктору new можно передать дополнительные аргументы и определить тайм-аут (в секундах) для сетевых операций:

```
$pop = Net::POP3->new( 'pop.myisp.com',
                       Timeout => 30 )
    or die "Can't connect to pop.myisp.com: $!\n",
```

Метод login выполняет аутентификацию на сервере POP3. Он получает два аргумента — имя пользователя и пароль, но оба аргумента являются необязательными. Если пропущено имя пользователя, используется текущее имя. Если пропущен пароль, Net::POP3 пытается определить пароль с помощью модуля Net::Netrc:

```
$pop->login( 'gnat', 'S33kr1T Pa55wOrD' )
    or die "Hey, my username and password didn't work!\n",

$pop->login( 'midget' )      # Искать пароль с помощью Net::Netrc
    or die "Authentication failed\n",

$pop->login()                # Текущее имя пользователя и Net::Netrc
    or die "Authentication failed: Miserably\n",
```

При вызове метода login пароль пересылается по сети в виде обычного текста. Это нежелательно, поэтому при наличии модуля MD5 от CPAN можно воспользоваться методом apop. Он полностью идентичен login за исключением того, что пароль пересылается в текстовом виде:

```
$pop->apop( $username, $password )
    or die "Couldn't authenticate: $!\n",
```

После аутентификации методы list, get и delete используются для работы с накопителем. Метод list выдает список неудаленных сообщений, хранящихся в

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

накопителе. Он возвращает хэш, где ключом является номер сообщения, а ассоциированное значение — размер сообщения в байтах:

```
%undeleted = $pop->list();
foreach $msgnum (keys %undeleted) {
    print "Message $msgnum is $undeleted{$msgnum} bytes long.\n";
}
```

Чтобы принять сообщение, вызовите метод get с нужным номером. Метод возвращает ссылку на массив строк сообщения:

```
print "Retrieving $msgnum : ",
$message = $pop->get($msgnum);
if ($message) {
    # succeeded
    print "\n";
    print @$message;                    # вывести сообщение
} else {
        # failed
    print "failed ($!)\n";
}
```

Метод delete помечает сообщение как удаленное. При вызове метода quit, завершающего сеанс POP3, помеченные сообщения удаляются из почтового ящика. отменяет все вызовы delete, сделанные во время сеанса. Если сеанс завершается из-за того, что объект Net::POP3 уничтожен при выходе из области действия, метод будет вызван автоматически.

Возможно, вы заметили, что мы ничего не сказали об *отправке* почты. POP3 поддерживает только чтение и удаление существующих сообщений. Новые сообщения приходится отправлять с помощью программ типа *mail* или *sendmail* или протокола SMTP. Другими словами, рецепт 18.3 все равно пригодится.

Основная задача POP3 — подключение почтовых клиентов к почтовым серверам — также выполняется протоколом IMAP. IMAP обладает более широкими возможностями и чаще используется на очень больших узлах.

Смотри также

Документация по модулю Net::POP3 с CPAN; RCS 1734, "POP3 AUTHentication command"; RFC 1957, "Some Observations on Implementations of the Post Office Protocol».

18.6. Программная имитация сеанса telnet

Проблема

Вы хотите обслуживать подключение *telnet* в своей программе — регистрироваться на удаленном компьютере, вводить команды и реагировать на них. Такая задача имеет много практических применений — от автоматизации на компьютерах с доступом *telnet*, но без поддержки сценариев или *rsh*, до обычной проверки работоспособности демона *telnet* на другом компьютере.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Воспользуйтесь модулем Net::Telnet с CPAN:

```
use Net::Telnet;

$t = Net::Telnet->new( Timeout => 10,
                       Prompt  => '/%/',
                       Host    => $hostname );

$t->login($username, $password);
@files = $t->cmd("ls");
$t->print("top");
(undef, $process_string) = $t->waitfor('/\d+ processes/');
$t->close;
```

Комментарий

Модуль Net::Telnet поддерживает объектно-ориентированный интерфейс к протоколу *telnet.* Сначала вы устанавливаете соединение методом `Net::Telnet->new`, а затем взаимодействуете с удаленным компьютером, вызывая методы полученного объекта.

Метод new вызывается с несколькими параметрами, передаваемыми в хэш-подобной записи (параметр => значение). Мы упомянем лишь некоторые из многих допустимых параметров. Самый важный, Host, определяет компьютер, к которому вы подключаетесь. По умолчанию используется значение `localhost`. Чтобы использовать порт, отличный от стандартного порта *telnet,* укажите его в параметре Port. Обработка ошибок выполняется функцией, ссылка на которую передается в параметре `Errmode`.

Еще один важный параметр — `Prompt`. При регистрации или выполнении команды модуль Net::Telnet по шаблону Prompt определяет, завершилась ли регистрация или выполнение команды. По умолчанию Prompt совпадает со стандартными приглашениями распространенных командных интерпретаторов:

```
/[\$%#>] $/
```

Если на удаленном компьютере используется нестандартное приглашение, вам придется определить собственный шаблон. Не забудьте включить в него символы /.

Параметр Timeout определяет продолжительность (в секундах) тайм-аута при сетевых операциях. По умолчанию тайм-аут равен 10 секундам.

Если в модуле Net::Telnet происходит ошибка или **тайм-аут,** по умолчанию инициируется исключение. Если не перехватить его, исключение выводит сообщение в STDERR и завершает работу программы. Чтобы изменить это поведение, передайте в параметре `Errmode` ссылку на подпрограмму. Если вместо ссылки `Errmode` содержит строку то при возникновении ошибок методы возвращают `undef` (в скалярном контексте) или пустой список (в списковом контексте); при этом сообщение об ошибке можно получить с помощью метода `errmsg`:

```
$telnet = Net::Telnet->new( Errmode => sub { main::log(@_) }, ... );
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Метод login передает имя пользователя и пароль на другой компьютер. Успешное завершение регистрации определяется по шаблону Prompt; если хост не выдал приглашения, происходит тайм-аут:

```
$telnet->login($username, $password)
    or die Login failed @{[ $telnet->errmsg() ]}\n ,
```

Для запуска программы и получения ее вывода применяется метод cmd. Он получает командную строку и возвращает выходные данные программы. В списковом контексте возвращается список, каждый элемент которого соответствует отдельной строке. В скалярном контексте возвращается одна длинная строка. Перед возвратом метод ожидает выдачи приглашения, определяемого по шаблону Prompt.

Пара методов, print и waitfor, позволяет отделить отправку команды от получения ее выходных данных, как это было сделано в решении. Метод waitfor получает либо набор именованных аргументов, либо одну строку с регулярным выражением Perl:

```
$telnet->waitfor(
```

Параметр Timeout определяет новый тайм-аут, отменяя значение по умолчанию. Параметр Match содержит оператор совпадения (см. выше), а String — искомую строковую константу:

```
$telnet->waitfor(String          smoke , Timeout => 30)
```

В скалярном контексте waitfor возвращает true, если шаблон или строка были успешно найдены. В противном случае выполняется действие, определяемое параметром Errmode. В списковом контексте метод возвращает две строки: весь текст до совпадения и совпавший текст.

Смотри также

Документация по модулю Net::Telnet с CPAN; RFC **854-856** и дополнения в последующих RFC.

18.7. Проверка удаленного компьютера

Проблема

Требуется проверить доступность сетевого компьютера. Сетевые и системные программы часто используют для этой цели программу ***ping***.

Решение

Воспользуйтесь стандартным модулем Net::Ping:

```
use Net  Ping,

$p = Net  Ping->new()
      die                  ping object  $!\n
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
print "$host is alive" if $p->ping($host);
$p->close;
```

Комментарий

Проверить работоспособность компьютера сложнее, чем кажется. Компьютер может реагировать на команду *ping* даже при отсутствии нормальной фунциональности; это не только теоретическая возможность, но, как ни печально, распространенное явление. Лучше рассматривать утилиту *ping* как средство для проверки доступности компьютера, а не выполнения им своих функций. Чтобы решить последнюю задачу, вы должны попытаться обратиться к его демонам (telnet, FTP, Web, NFS и т. д.).

В форме, показанной в решении, модуль Net::Ping пытается подключиться к эхо-порту UDP (порт 7) на удаленном компьютере, отправить датаграмму и получить эхо-ответ. Компьютер считается недоступным, если не удалось установить соединение, если отправленная датаграмма не была получена или если ответ отличался от исходной датаграммы. Метод `ping` возвращает `true`, если компьютер доступен, и `false` в противном случае.

Чтобы использовать другой протокол, достаточно передать его имя при вызове new. Допустимыми являются протоколы tcp, `udp` и `icmp` (записываются в нижнем регистре). При выборе протокола TCP программа пробует подключиться к эхо-порту TCP (порт 7) удаленного компьютера и возвращает `true` при успешной установке соединения и `false` в противном случае (в отличие от UDP пересылка данных не выполняется). При выборе ICMP будет использован протокол ICMP, как в команде *ping*(8). На компьютерах UNIX протокол ICMP может быть выбран только привилегированным пользователем:

```
# Использовать ICMP при наличии привилегий и TCP в противном случае
$pong = Net: Ping->new( $> ? "tcp" : "icmp" ),

(defined $pong)
    or die "Couldn't Net::Ping object: $!\n",

if ($pong->ping('kingkong.com')) {
    print "The giant ape lives!\n",
} else {
    print "All hail mighty Gamera, friend of children!\n",
}
```

Ни один из этих способов не является абсолютно надежным. Маршрутизаторы некоторых узлов отфильтровывают протокол ICMP, поэтому Net::Ping сочтет такие компьютеры недоступными даже при возможности подключения по другим протоколам. Многие компьютеры запрещают эхо-сервис TCP и UDP, что приводит к неудачам при опросе через TCP и UDP. Запрет службы или фильтрацию протокола невозможно отличить от неработоспособности компьютера.

Смотри также

Документация по модулю Net::Ping; страницы руководства *ping*(8), *tcp*(3), *udp*(4) и *icmp*(4) вашей системы (если есть); RFC 792 и 950.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

18.8. При менение whois для получения данных от InterNIC

Проблема

Вы хотите узнать, кому принадлежит домен (по аналогии с командой UNIX whois).

Решение

Воспользуйтесь модулем Net::Whois с CPAN:

```
use Net::Whois;

$domain_obj = Net::Whois::Domain->new($domain_name)
    or die "Couldn't get information on $domain_name: $!\n";

# Вызвать методы объекта $domain_obj
# для получения имени, тега, адреса и т. д.
```

Комментарий

Сервис whois предоставляется службой регистрации доменных имен и предназначается для идентификации владельца имени. Исторически в системах семейства UNIX эти данные получались с помощью программы ***whois***(1), которая возвращала около 15 строк информации, включая имена, адреса и телефоны административных, технических и финансовых контактных лиц домена.

Модуль Net::Whois, как и ***whois***(1), является клиентом службы whois. Он подключается к серверу whois (по умолчанию используется ***whois.internic.net***, главный сервер доменов ".com", ".org.", ".net" и ".edu"). Доступ к данным осуществляется с помощью методов, вызываемых для объекта.

Чтобы получить информацию о домене, создайте объект Net::Whois::Domain. Например, для получения данных о ***perl.org*** объект создается так:

```
$d = Net::Whois:.Domain->new( "perl.org" )
    or die "Can't get information on perl.org\n";
```

Гарантируется только получение имени домена и тега — уникального идентификатора домена в учетных записях NIC:

```
print "The domain is called ", $d->domain, "\n";
print "Its tag is ", $d->tag, "\n";
```

Также могут присутствовать следующие данные: ***название*** организации, которой принадлежит домен (например, "The Perl Institute"); ***адрес*** компании в виде списка строк (например, ("221B Baker "London")) и страна (например, "United Kingdom" или двухбуквенное сокращение "uk").

```
print "Mail for ", $d->name, " should be sent to:\n";
print map { "\t$_\n" }
print "\t", $d->country, "\n";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Кроме информации о самом домене также можно получить сведения о контактных лицах домена. Метод contact возвращает ссылку на хэш, в котором тип контакта(например, Billing или Administrative)ассоциируетсясмассивомстрок.

```
$contact_hash = $d->contacts,
if ($contact_hash) {
    print Contacts \n ,
            $type (sort keys %$contact_hash) {
        print    $type \n ,
                 $line (@{$contact_hash->{$type}})
            print      $line\n ,
        }
    }
} else {
    print No contact information \n ,
```

Смотри также

Документация по модулю Net::Whois с CPAN; man-страница ***whois*(8)** вашей системы (если есть); RFC 812 и 954.

18.9. Прогр амма: expn и vrfy

Программа *expn* напрямую общается с сервером SMTP и проверяет адрес с помощью команд EXPN и VRFY. Она неидеальна, поскольку ее работа зависит от получения достоверной информации с удаленного сервера командами EXPN и VRFY. Программа использует модуль Net::DNS, если он присутствует, но может работать и без него.

Программа узнает, как она была вызвана, с помощью переменной $0 (имя программы). При запуске под именем *expn* используется команда EXPN; при запуске под именем *vrfy* используется команда VRFY. Установка команды под двумя разными именами осуществляется с помощью ссылок:

```
% cat > expn
#!/usr/bin/perl -w

^D
% ln expn vrfy
```

Передайте программе адрес электронной почты, и она сообщит результаты проверки адреса командой EXPN или VRFY. Если у вас установлен модуль Net::DNS, программа проверяет все хосты пересылки почты (mail exchangers), перечисленные в записи DNS данногоадреса.

Без Net::DNS результаты выглядят так:

```
% expn gnat@frii com
Expanding gnat at frll.com (gnat@frii.com):
call3to.frll.com Hello coprollth.frll.com [207.46.130.14],
    pleased to meet you
<gnat@mail.frii.com>
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При установленном модуле Net::DNS получен следующий результат:

```
% expn gnat@frii.com
Expanding gnat at mail.frii.net (gnat@frii.com):
deimos.frii.com Hello coprolith.frii.com [207.46.130.14],
    pleased to meet you
Nathan Torkington <gnat@deimos.frii.com>

Expanding gnat at mx1.fni.net (gnat@frii.com):
phobos.frii.com Hello coprolith.frn.com [207.46.130.14],
    pleased to meet you
<gnat@mail.frii.com>

Expanding gnat at mx2.frn.net (gnat@frn.com):
europa.frii.com Hello coprolith frii.com [207.46.130.14],
    pleased to meet you
<gnat@mail.frii.com>

Expanding gnat at mx3.frii.net (gnat@frii.com):
ns2.winterlan.com Hello coprolith.frii.com [207.46.130 14],
    pleased to meet you
550 gnat... User unknown
```

Исходный текст программы приведен в примере 18.3.

Пример 18.3. expn

```
#!/usr/bin/perl -w
# expn - расширение адресовчерез SMTP
use strict,
use IO Socket,
use Sys Hostname,

my $fetch_mx = 0,
# Попытаться загрузить модуль, но не огорчаться в случае неудачи
eval {
                DNS,
    Net DNS->import( mx ),
    $fetch_mx = 1,
},

my $selfname = hostname(),
die usage $0                    unless @ARGV,

# Определить имя программы - vrfy или expn
my $VERB = ($0 =^ /ve?ri?fy$/i) ^ VRFY     EXPN
my $multi = @ARGV > 1,


# Перебрать адреса, указанные в командной строке
          $combo (@ARGV)
    my ($name, $host) = split(/\@/, $combo),
    my @hosts,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $host ||= 'localhost';
    @hosts = map { $_->exchange } mx($host)    if $fetch_mx;
    @hosts = ($host)  unless @hosts;

    foreach my $host (@hosts) {
        print $VERB eq 'VRFY' ? "Verify" : "Expand",
            "ing $name at $host ($combo):";

        $remote = IO::Socket::INET->new(
             Proto    => "tcp",
             PeerAddr => $host,
             PeerPort => "smtp(25)",
         );

        unless ($remote) {
            warn "cannot connect to $host\n";
            next;
        }
        print "\n";



        # Использовать сетевые разделители строк CRLF
        print $remote "HELO $selfname\015\012";
        print $remote "$VERB $name\015\012";
        print         "quit\015\012";
        while (<$remote>) {
          /^220\b/ && next;
          /^221\b/ && last;
          s/250\b[\-\s]+//;
          print;
        }
                                    die "can't close socket: $!"
        print "\n";
    }
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Программирование CGI 19

Хорош тот инструмент, который может использоваться для целей, неожиданных для его создателя.

Стивен Джонсон

Введение

Резкие изменения окружающей среды приводят к тому, что некоторые виды лучше других добывают пропитание или избегают хищников. Многие ученые полагают, что миллионы лет назад при столкновении кометы с Землей в атмосферу поднялось огромное облако пыли. За этим последовали радикальные изменения окружающей среды. Некоторые организмы — например, динозавры — не смогли справиться с ними, что привело к их вымиранию. Другие виды (в частности, млекопитающие) нашли новые источники пищи и места обитания и продолжили борьбу за существование.

Подобно тому, как комета изменила среду обитания доисторических животных, развитие Web изменило ситуацию в современных языках программирования и открыло новые горизонты. Хотя некоторые языки так и не прижились в "новом мировом порядке", Perl выдержал испытание. Благодаря своим сильным сторонам — обработке текстов и объединению системных компонентов — Perl легко приспособился к задачам пересылки информации с использованием текстовых протоколов.

Архитектура

В основе Web лежит обычный текст. Web-серверы общаются с броузерами с помощью текстового протокола HTTP (Hypertext Transfer Protocol). Многие пересылаемые документы кодируются специальной разметкой, которая называется HTML (Hypertext Markup Language). Текстовая ориентация внесла немалый вклад в гибкость, широту возможностей и успех Web. Единственным исключением на этом фоне является протокол SSL (Secure Sockets Layer), который шифрует другие протоколы (например, HTTP) в двоичные данные, защищенные от перехвата.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Web-страницы идентифицируются по так называемым URL (Universal Resource Locator). URL выглядят так:

http://www.perl.com/CPAN/
http://www.perl.com:8001/bad/mojo.html
ftp://gatekeeper.dec.com/pub/misc/netlib.tar2
ftp://anonymous@myplace:gatekeeper.dec.com/pub/misc/netlib.tar.Z
file:///etc/motd

Первая часть (*http*, *ftp*, *file*) называется *схемой* и определяет способ получения файла. Вторая (*://*) означает, что далее следует имя хоста, интерпретация которого зависит от схемы. За именем хоста следует *путь*, идентифицирующий документ. Путь также называется *частичным URL*.

Web является системой "клиент/сервер". Клиентские броузеры (например, Netscape или Lynx) запрашивают документы (идентифицируемые по частичным URL) у Web-серверов — таких, как Apache. Диалог броузера с сервером определяется протоколом HTTP. В основном сервер просто пересылает содержимое некоторого файла. Однако иногда Web-сервер запускает другую программу для отправки документа, который может представлять собой HTML-текст, графическое изображение или иной тип данных. Диалог сервера с программой определяется протоколом CGI (Common Gateway Interface), а запускаемая сервером программа называется *программой CGI* или *сценарием CGI*.

Сервер сообщает программе CGI, какая страница была затребована, какие значения были переданы в HTML-формах, откуда поступил запрос, какие данные использовались при аутентификации и многое другое. Ответ программы CGI состоит из двух частей: заголовка, говорящего "Я передаю документ HTML», "Я передаю изображение формата GIF" или "Я вообще ничего не передаю, обращайся на такую-то страницу", и тела документа (возможно, содержащего данные GIF, обычный текст или код HTML).

Правильно реализовать протокол CGI нелегко, а ошибиться проще простого, поэтому мы рекомендуем использовать превосходный модуль CGI.pm Линкольна Штейна (Lincoln Stein). Модуль содержит удобные функции для обработки информации, полученной от сервера, и подготовки ответов CGI, ожидаемых сервером. Это чрезвычайно полезный модуль был включен в стандартную поставку Perl версии 5.004 вместе с вспомогательными модулями (например, CGI::Carp или CGI::Fast). Использование модуля демонстрируется в рецепте 19.1.

Некоторые Web-серверы содержат встроенный интерпретатор Perl, что позволяет генерировать документы на Perl без запуска нового процесса. Системные издержки на чтение неизменившейся страницы пренебрежимо малы для страниц с редкими обращениями (даже порядка нескольких обращений в секунду). Однако вызовы CGI существенно замедляют компьютер, на котором работает Web-сервер. В рецепте 19.5 показано, как работать с *mod_perl*, встроенным интерпретатором Perl Web-сервера Apache, чтобы пользоваться преимуществами программ CGI без издержек, связанных с ними.

За кулисами

Программы CGI вызываются каждый раз, когда Web-серверу требуется сгенерировать динамический документ. Необходимо понимать, что программа CGI

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

не работает постоянно с обращениями к ее различным частям со стороны броузера. При каждом запросе частичного URL, соответствующего программе, запускается ее новая копия. Программа генерирует страницу для данного запроса и завершается.

Броузер может запросить документ несколькими способами, которые называются *методами* (не путайте методы HTTP с методами объектно-ориентированного программирования!). Чаще всего встречается метод GET, который обозначает простой запрос документа. Метод HEAD используется в том случае, если броузер хочет получить сведения о документе без фактической загрузки. Метод POST применяется при передаче заполненных форм.

Значения полей форм также могут кодироваться в методах GET и POST. В методе GET значение кодируется прямо в URL, что приводит к появлению уродливых URL следующего вида:

http://mox.perl.com/cgi-bin/program ?name=Johann &born=1685

В методе POST значения находятся в другой части запроса HTTP — не той, которую броузер отправляет серверу. Если бы в приведенном выше URL значения полей отсылались методом POST, то пользователь, сервер и сценарий CGI видели бы обычный URL:

http://mox.perl.com/cgi-bin/program

Методы GET и POST также отличаются свойством *идемпотентности*. Проще говоря, однократный или многократный запрос GET для некоторого URL должен давать одинаковые результаты. Это объясняется тем, что в соответствии со спецификацией протокола HTTP запрос GET может кэшироваться броузером, сервером или промежуточным прокси-сервером. Запросы POST не могут кэшироваться, поскольку каждый запрос считается самостоятельным и независимым от других. Как правило, запросы POST влияют на состояние сервера или зависят от него (обращение или обновление базы данных, отправка почты).

Большинство серверов регистрирует запросы к файлам (ведут *журнал обращений)* для их последующего анализа Web-мастером. Ошибки в программах CGI тоже по умолчанию не передаются броузеру. Вместо этого они регистрируются в файле *(журнал ошибок),* а броузер Просто получает сообщение "500 Server Error", которое означает, что программа CGI не справилась со своей задачей.

Сообщения об ошибках полезны в процессе отладки любой программы, но особенно полезны они в сценариях CGI. Однако авторы программ CGI не всегда имеют доступ к журналу ошибок или не знают, где он находится. Перенаправление ошибок рассматривается в рецепте 19.2, а исправление — в рецепте 19.3.

В рецепте 19.9 показано, как узнать, что в действительности говорят друг другу броузер с сервером. К сожалению, некоторые броузеры не реализуют спецификацию HTTP в полной мере. Рецепт поможет выяснить, что является причиной возникших трудностей — программа или броузер.

Безопасность

Сценарии CGI позволяют запускать программы на вашем компьютере кому угодно. Конечно, программу выбираете вы, но анонимный пользователь может

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

передать ей неожиданные значения и обмануть ее, заставляя сделать нечто нехорошее. Безопасности в Web уделяется большое внимание.

Некоторые узлы решают проблему, попросту отказываясь от программ CGI. Там, где без силы и возможностей программ CGI не обойтись, приходится искать средства обезопасить их. В рецепте 19.4 приведен список рекомендаций по написанию безопасных сценариев CGI, а также кратко рассмотрен механизм пометки, защищающий от случайного применения ненадежных данных. В рецепте 19.6 показано, как организовать безопасный запуск других программ из сценария CGI.

HTML и формы

Теги HTML позволяют создавать экранные формы. В этих формах пользователь вводит значения, передаваемые серверу. Формы состоят из *элементов* (widgets) — например, текстовых полей и флажков. Программы CGI обычно возвращают HTML-код, поэтому в модуле CGI предусмотрены вспомогательные функции создания HTML-кода для чего угодно, от таблиц до элементов форм.

В дополнение к рецепту 19.7 в этой главе также имеется рецепт 19.11. В нем показано, как создать форму, сохраняющую свои значения между вызовами. В рецепте 19.12 продемонстрировано создание одного сценария CGI, который создает и обрабатывает целый набор страниц — например, в системе приема заказов по каталогу.

Ресурсы Web

Разумеется, лучшую информацию о программировании Web можно найти непосредственно в Web.

Безопасность Web

http://www.w3.org/Security/Faq/

Общие сведения о Web

http://www.boutell.com/faq/

CGI

http://www.webthing.com/tutorials/cgifaq.html

Спецификация HTTP

http://www.w3.org/pub/WWW/Protocols/HTTP/

Спецификация HTML

http://www.w3.org/TR/REC-html40/
http://www.w3.org/pub/WWW/MarkUp/

Спецификация CGI

http://www.w3.org/CGI/

Безопасность CGI

http://www.go2net.com/people/paulp/cgi-security/safe-cgi.txt

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

19.1. Написание сценария CGI

Проблема

Требуется написать сценарий CGI для обработки содержимого HTML-формы. В частности, вы хотите работать со значениями полей формы и выдавать нужные выходные данные.

Решение

Сценарий CGI представляет собой программу, работающую на сервере и запускаемую Web-сервером для построения динамического документа. Он получает кодированную информацию от удаленного клиента (пользовательского броузера) через STDIN и переменные окружения и выводит в STDOUT правильные заголовки и тело запросов HTTP. Стандартный модуль CGI (см. пример 19.1) обеспечивает удобное преобразование ввода и вывода.

Пример 19.1. hiweb

```
#!/usr/bin/perl -w
# hiweb - загрузить модуль CGI для расшифровки
# данных, полученных от Web-сервера
use strict;

use CGI qw(:standard escapeHTML);

# Получить параметр от формы
my $value = param('PARAM_NAME');

# Вывести документ
print header(), start_html("Howdy there!"),
      p("You typed: ", tt(escapeHTML($value))),
      end_html();
```

Комментарий

CGI — всего лишь протокол, формальное соглашение между Web-сервером и отдельной программой. Сервер кодирует входные данные клиентской формы, а программа CGI декодирует форму и генерирует выходные данные. В спецификации протокола ничего не сказано о языке, на котором должна быть написана программа. Программы и сценарии, соответствующие протоколу CGI, могут быть написаны в командном интерпретаторе, на C, Rexx, C++, VMS DCL, Smalltalk, Tcl, Python и, конечно, на Perl.

Полная спецификация CGI определяет, какие данные хранятся в тех или иных переменных окружения (например, входные параметры форм) и как они кодируются. Теоретически декодирование входных данных в соответствии с протоколом не должно вызывать никаких проблем, но на практике задача оказывается на удивление хитрой. Именно поэтому мы *настоятельно* рекомендуем использовать превосходный модуль CGI Линкольна Штейна. Вся тяжелая работа по пра-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

вильной обработке требований CGI выполнена заранее; вам остается лишь написать содержательную часть программы без нудных сетевых протоколов.

Сценарии CGI вызываются двумя основными способами, которые называются *методами,* — но не путайте методы HTTP с методами объектов Perl! Метод GET используется для получения документов в ситуациях, когда идентичные запросы должны давать идентичные результаты — например, при поиске в словаре. В методе GET данные формы хранятся внутри URL. Это позволяет сохранить запрос на будущее, но ограничивает общий размер запрашиваемых данных. Метод POST отправляет данные отдельно от запроса. Он не имеет ограничений на размер, но не может сохраняться. Формы, обновляющие информацию на сервере (например, при отправке ответного сообщения или модификации базы данных), должны использовать POST. Клиентские броузеры и промежуточные прокси-серверы могут кэшировать и обновлять результаты запросов GET незаметно для пользователя, но запросы POST не кэшируются. Метод GET надежен лишь для коротких запросов, ограничивающихся чтением информации, тогда как метод POST надежно работает для форм любого размера, а также подходит для обновления и ответов в схемах с обратной связью. По умолчанию модуль CGI использует POST для всех форм.

За небольшими исключениями, связанными с правами доступа к файлам и режимами повышенной интерактивности, сценарии CGI делают практически все то же, что и любая другая программа. Они могут возвращать результаты во многих форматах: обычный текст, документы HTML, звуковые файлы, графика и т. д. в зависимости от заголовка HTTP. Помимо вывода простого текста или HTML-кода, они также могут перенаправлять клиентский броузер в другое место, задавать серверные cookies, требовать аутентификации или сообщать об ошибках.

Модуль CGI поддерживает два интерфейса —процедурный для повседневного использования и объектно-ориентированный для компетентных пользователей с нетривиальными потребностями. Практически все сценарии CGIдолжны использовать процедурный интерфейс, но, к сожалению, в большей частидокументации по CGI.pm приведены примеры для исходного объектно-ориентированного подхода. Еcди вы хотите использовать упрощенный процедурный интерфейс, то для обеспечения обратной совместимости вам придется явно запросить его с помощью тега `:standard`. О тегах рассказано в главе 12 "Пакеты, библиотеки и модули".

Чтобы прочитать входные данные пользовательской формы, передайте функции `param` имя нужного поля. Если на форме имеется поле с именем "favorite", то вызов `param("favorite")` вернет его значение. В некоторых элементах форм (например, в списках) пользователь может выбрать несколько значений. В таких случаях `param` возвращает список значений, который можно присвоить массиву.

Например, следующий сценарий получает значения трех полей формы, последнее из которых может возвращать несколько значений:

```
use CGI qw(:standard);
$who   = param("Name");
$phone = param("Number");
@picks = param("Choices");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

При вызове без аргументов `param` возвращает список допустимых параметров формы в списковом контексте или количество параметров формы в скалярном контексте.

Вот и все, что нужно знать о пользовательском вводе. Делайте с ним, что хотите, а потом генерируйте выходные данные в нужном формате. В этом тоже нет ничего сложного. Помните, что, в отличие от обычных программ, выходные данные сценария CGI должны форматироваться определенным образом: сначала набор заголовков, за ними — пустая строка и лишь потом нормальный вывод.

Как видно из решения, модуль CGI упрощает не только ввод, но и вывод данных. Он содержит функции для генерации заголовков HTTP и HTML-кода. Функция `header` строит текст заголовка. По умолчанию она генерирует заголовки для документов `text/html`, но вы можете изменить тип содержимого и передать другие необязательные параметры:

```
print header( -TYPE    => 'text/plain ,
              -EXPIRES => '+3d' ),
```

Модуль CGI.pm также применяется для генерации HTML-кода. Звучит тривиально, но модуль CGI проявляется во всем блеске при создании динамических форм с сохранением состояния (например, страниц, предназначенных для оформления заказов). В модуле CGI даже имеются функции для генерации форм и таблиц.

При выводе элементов формы символы &, <, > и " в выходных данных HTML автоматически заменяются своими эквивалентами. В пользовательских выходных данных этого не происходит. Именно поэтому в решении импортируется и используется функция `escapeHTML` — даже если пользователь введет специальные символы, это не вызовет ошибок форматирования в HTML.

Полный список функций вместе с правилами вызова приведен в документации по модулю CGI.pm, хранящейся в формате POD внутри самого модуля.

Смотри также

Документация по стандартному модулю CGI; *http://www.w3.org/CGI/*; рецепт 19.7.

19.2. Перенаправление сообщений об ошибках

Проблема

У вас возникли трудности с отслеживанием предупреждений и ошибок вашего сценария, или вывод в STDERR из сценария приводит сервер в замешательство.

Решение

Воспользуйтесь модулем CGI::Carp из стандартной поставки Perl, чтобы все сообщения, направляемые в STDERR, снабжались префиксом — именем приложения и текущей датой. При желании предупреждения и ошибки также можно сохранять в файле или передавать броузеру.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Задача отслеживания сообщений в сценариях CGI пользуется дурной славой. Даже если вам удалось найти на сервере журнал ошибок, вы все равно не сможете определить, когда и от какого сценария поступило то или иное сообщение. Некоторые недружелюбные Web-серверы даже прерывают работу сценария, если он неосторожно выдал в STDERR какие-нибудь данные до генерации заголовка Content-Type — а это означает, что флаг *-w* может навлечь беду.

На сцене появляется модуль CGI::Carp. Он замещает `warn` и `die`, а также функции `carp`, `croak` и `confess` обычного модуля Carp более надежными и содержательными версиями. При этом данные по-прежнему отсылаются в журнал ошибок сервера.

```
use CGI::Carp;
warn "This is a complaint";
die "But this one is serious";
```

В следующем примере использования CGI::Carp ошибки перенаправляются в файл по вашему выбору. Все это происходит в блоке BEGIN, что позволяет перехватывать предупреждения на стадии компиляции:

```
BEGIN {
    use CGI::Carp qw(carpout);
    open(LOG, ">>/var/local/cgi-logs/mycgi-log")
        or die "Unable to append to mycgi-log: $!\n";
    carpout(*LOG);
}
```

Фатальные ошибки могут даже возвращаться клиентскому броузеру — это удобно при отладке, но может смутить рядового пользователя.

```
use CGI::Carp qw(fatalsToBrowser);
die "Bad error";
```

Даже если ошибка произойдет до вывода заголовка HTTP, модуль попытается избежать ужасной ошибки 500 Server Error. Нормальные предупреждения по-прежнему направляются в журнал ошибок сервера (или туда, куда вы отправили их функцией `carpout`) с префиксом из имени приложения и текущего времени.

Смотри также

Документация по стандартному модулю CGI::Carp; описание `BEGIN` в рецепте 12.3.

19.3. Исправление ошибки 500 Server Error

Проблема

Сценарий CGI выдает ошибку 500 Server Error.

Решение

Воспользуйтесь приведенным ниже списком рекомендаций. Советы ориентированы на аудиторию UNIX, однако общие принципы относятся ко всем системам.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Убедитесь, что сценарий может выполняться Web-сервером.
Проверьте **владельца** и права доступа командой *ls -/*. Чтобы сценарий мог выполняться сервером, для него должны быть установлены необходимые права чтения и исполнения. Сценарий должен быть доступен для чтения и исполнения для всех пользователей (или по крайней мере для того, под чьим именем сервер выполняет сценарии). Используйте команду *chmod 0755 имя-сценария,* если сценарий принадлежит вам, или *chmod 0555 имя-сценария,* если он принадлежит анонимному пользователю Web, а вы работаете как этот или привилегированный пользователь. Бит исполнения также должен быть установлен для всех каталогов, входящих в путь.

Проследите, чтобы сценарий идентифицировался Web-сервером как сценарий. Большинство Web-серверов имеет общий для всей системы каталог cgi-bin, и все файлы данного каталога считаются сценариями. Некоторые серверы идентифицируют сценарий CGI как файл с определенным расширением — например, *.cgi* или *.pcx*. Параметры некоторых серверов разрешают доступ только методом GET, а не методом POST, который, вероятно, используется вашей формой. Обращайтесь к документации по Web-серверу, конфигурационным файлам, Web-мастеру и (если ничего не помогает) в службу технической поддержки.

Если вы работаете в UNIX, проверьте, правильно ли задан путь к исполняемому файлу Perl в строке `#!`. Она должна быть первой в сценарии, перед ней даже не разрешаются пустые строки. Некоторые операционные системы устанавливают смехотворно низкие ограничения на размер этой строки — в таких случаях следует использовать ссылки (допустим, из *fliome/richh/perl* на */opt/installed/third-party/software/perl-5.004/bin/perl* взят вымышленный патологический пример).

Если вы работаете в Win32, посмотрите, правильно ли связаны свои сценарии с исполняемым файлом Perl.

Проверьте наличие необходимых прав у сценария

Проверьте пользователя, с правами которого работает сценарий, с помощью простого фрагмента из примера 19.2.

Пример 19.2. webw hoami

```
#!/usr/bin/perl
# webwhoami - show web users id
print Content-Type text/plain\n\n ,
print Running as , scalar getpwuid($>), \n ,
```

Сценарий выводит имя пользователя, с правами которого он работает.

Определите ресурсы, к которым обращается сценарий. Составьте список файлов, сетевых соединений, системных функций и т. д., требующихособых привилегий. Затем убедитесь в их доступности для пользователя, с правами которого работает сценарий. Действуют ли дисковые или сетевые квоты? Обеспечивает ли защита файла доступ к нему? Не пытаетесь ли вы получить зашифрованный пароль с помощью `getpwent` в системе со скрытыми паролями (обычно скрытые пароли доступны только для привилегированного пользователя)?

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Для всех файлов, в которые сценарий выполняет запись, установите права доступа 0666, а еще лучше — 0644, если они принадлежат тому пользователю, с чьими правами выполняется сценарий. Если сценарий создает новые файлы или перемещает/удаляет старые, потребуются также права записи и исполнения для каталога *с* ними.

Не содержит ли сценарий ошибок Perf?

Попытайтесь запустить его в командной строке. Модуль CGI.pm позволяет запускать и отлаживать сценарии в командной строке или из стандартного ввода. В следующем фрагменте ^D — вводимый вами признак концафайла:

```
% perl -wc cgi-script                    # Простая компиляция

% perl -w cgi-script                     # Параметры из stdin
(offline mode: enter name=value pairs on standard input)
name=joe
number=10
^D

% perl -w cgi-script name=joe number=10   # Запустить с входными
                                         # данными формы
% perl -d cgi-script name=joe number=10   # То же в отладчике

# Сценарий с методом POST в csh
% (setenv HTTP_METHOD POST, perl -w cgi-script name=joe number=10)
# Сценарий с методом POST в sh
% HTTP_METHOD=POST perl -w cgi-script name=joe number=10
```

Проверьте журнал ошибок сервера. Большинство Web-серверов перенаправляет поток STDERR для процессов CGI в файл. Найдите его (попробуйте */usr/local/etc/httpd/logs/error_log*, */usr/local/www/logs/error_log* или спросите у администратора) и посмотрите, есть ли в нем предупреждения или сообщения об ошибках.

Не устарела ли ваша версия Perl? Ответ даст команда *perl -v*. Если у вас не установлена версия 5.004 или выше, вам или вашему администратору следует подумать об обновлении, поскольку 5.003 и более ранние версии не были защищены от переполнения буфера, из-за чего возникали серьезные проблемы безопасности.

Не используете ли вы старые версии библиотек? Выполните команду *version* для библиотечного файла (вероятно, находящегося в */usr/lib/perl5*, */usr/local/lib/perl5/*, */usr/lib/perl5/site_perl* или похожем каталоге). Для CGI.pm (а фактически — для любого модуля) версию можно узнать и другим способом:

```
% perl -MCGI -le print CGI->VERSION
2.40
```

Используете ли вы последнюю версию Web-сервера? Хотя такое происходит редко, но все же в Web-серверах иногда встречаются ошибки, мешающие работе сценариев.

Используете ли вы флаг *-w*? С этим флагом Perl начинает жаловаться на неинициализированные переменные, чтение из манипулятора, предназначенного только для записи, и т. д.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Используете ли вы флаг -7? Если Perl жалуется на небезопасные действия, возможно, вы допустили какие-то неверные предположения относительно входных данных и рабочей среды вашего сценария. Обеспечьте чистоту данных, и вы сможете спокойно спать по ночам, а заодно и получите рабочий сценарий (меченые данные и их последствия для программ рассматриваются в рецепте 19.4 и на странице руководства perlsec; в списке FAQ по безопасности CGI описаны конкретные проблемы, которых следует избегать).

Используете ли вы директиву use strict? Она заставляет объявлять переменные перед использованием и ограничивать кавычками строки, чтобы избежать возможной путаницы с подпрограммами, и при этом находит множество ошибок.

Проверяете ли вы возвращаемые значения всех системных функций? Многие люди наивно полагают, что любой вызов open, system, rename или unlink всегда проходит успешно. Они возвращают значение, по которому можно проверить результат их работы, — так проверьте!

Находит ли Perl используемые вами библиотеки? Напишите маленький сценарий, который просто выводит содержимое @INC (список каталогов, в которых ищутся модули и библиотеки). Проверьте права доступа к библиотекам (должно быть разрешено чтение для пользователя, с правами которого работает сценарий). Не пытайтесь копировать модули с одного компьютера на другой — многие из них имеют скомпилированные и автоматически загружаемые компоненты, находящиеся за пределами библиотечного каталога Perl. Установите их с нуля.

Выдает ли Perl предупреждения или сообщения об ошибках? Попробуйте использовать CGI::Carp (см. рецепт 19.2), чтобы направить предупреждения и ошибки в броузер или доступный файл.

Соблюдает ли сценарий протокол CGI?

Перед возвращаемым текстом или изображением должен находиться заголовок HTTP. Не забывайте о пустой строке между заголовком и телом сообщения. Кроме того, STDOUT в отличие от STDERR не очищается автоматически. Если ваш сценарий направляет в STDERR предупреждения или ошибки, Web-сервер может увидеть их раньше, чем заголовок HTTP, и на некоторых серверах это приводит к ошибке. Чтобы обеспечить автоматическую очистку STDOUT, вставьте в начало сценария следующую команду (после строки #!):

```
$| = 1;
```

Никогда не пытайтесь декодировать поступающие данные формы, самостоятельно анализируя окружение и стандартный ввод — возникает слишком много возможностей для ошибок. Воспользуйтесь модулем CGI и проводите время за творческим программированием или чтением Usenet, вместо того чтобы возиться с поиском ошибок в доморощенной реализации мудрейого протокола.

Справочная информация

Обратитесь к спискам FAQ и другим документам, перечисленным в конце введения этой главы. Возможно, вы допустили какую-нибудь распространенную ошибку для своей системы — прочитайте соответствующий FAQ, и вам не придется краснеть за вопросы типа: "Почему моя машина не ездит без бензина и масла?"

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Спросите других. Почти у каждого есть знакомый специалист, к которому можно обратиться за помощью. Вероятно, ответ найдется намного быстрее, чем при обращении в Сеть.

Если ваш вопрос относится к сценариям CGI (модуль CGI, декодирование cookies, получение данных о пользователе и т. д.), пишите в *comp.infosystems. www.authoring.misc.*

Смотри также

Рецепт 19.2; сведения о буферизации во введении к главе 8 "Содержимое файлов"; CGI FAQ по адресу *http://www.webthing.com/tutoriah/cgifaq.html.*

19.4. Написание безопасных программ CGI

Проблема

Поскольку сценарий CGI позволяет внешнему пользователю запускать программы на недоступном для него компьютере, любая программа CGI представляет потенциальную угрозу для безопасности. Вам хотелось бы свести эту угрозу к минимуму.

Решение

- Воспользуйтесь режимом пометки (флаг `-T` в строке `#!`).
- Не снимайте пометку с данных (см. ниже).
- Проверяйте все, в том числе возвращаемые значения всех элементов формы, даже скрытые элементы и значения, сгенерированные кодом JavaScript. Многие наивно полагают — раз они приказали JavaScript проверить значения полей формы перед отправкой данных, то значения действительно будут проверены. Ничего подобного! Пользователь может тривиально обойти ограничения — запретить JavaScript в своем броузере, загрузить форму и модифицировать JavaScript или общаться на уровне HTTP без броузера (см. главу 20 "Автоматизация в Web»).
- Проверяйте значения, возвращаемые системными функциями.
- Помните о возможности перехвата (см. ниже).
- Используйте флаг `-w` и директиву use `strict`, чтобы застраховаться от неправильных допущений со стороны Perl.
- Никогда не запускайте сценарий со сменой прав, если только это не вызвано абсолютной необходимостью. Подумайте, не будет ли достаточно сменить идентификтор группы. Любой ценой избегайте запуска с правами администратора. Если вам приходится использовать `setuid` или `setgid`, используйте командный интерпретатор, если только на вашей машине нельзя безопасно запускать сценарии Perl с `setuid` и выточно знаете, что это такое.
- Всегда шифруйте пароли, номера кредитных карт, номера социального страхования и все остальное, что обычно не печатается на первых страницах местных газет. При работе с такими данными следует использовать безопасный протокол SSL.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Многие из этих рекомендаций подходят для **любых** программ — флаг **-w** и проверка значений, возвращаемых системными функциями, пригодятся и **в** тех ситуациях, когда безопасность не является первоочередной заботой. Флаг *-w* заставляет Perl выводить предупреждения о сомнительных конструкциях (например, когда неопределенная переменная используется так, словно ей присвоено законное значение, или при попытке записи **в манипулятор,** доступный только для чтения).

Самая распространенная угроза безопасности (не считая непредвиденных вызовов командного интерпретатора) кроется в передаче форм. Кто угодно может сохранить исходный текст формы, отредактировать HTML-код и передать измененную форму. Даже если вы уверены, что поле может возвращать только "yes" или "по", его всегда можно отредактировать и заставить возвращать "maybe". Даже скрытые поля, имеющие тип HIDDEN, не защищены от вмешательства извне. Если программа на другом конце слепо полагается на значения полей, ее можно заставить удалять файлы, создавать новые учетные записи пользователей, выводить информацию из баз данных паролей или кредитных карт и совершать множество других злонамеренных действий. Вот почему нельзя слепо доверять данным (например, информации о цене товара), хранящимся в скрытых полях при написании приложений CGI для электронных магазинов.

Еще хуже, если сценарий CGI использует значение поля формы как основу для выбора открываемого файла или выполняемой команды. Ложные значения, переданные сценарию, заставят его открывать произвольные файлы. Именно из-за таких ситуаций в Perl появился режим помеченных данных. Если программа выполняет setuid или имеет активный флаг *-T,* то любые данные, получаемые ею в виде аргументов, переменных окружения, списков каталогов или файлов, считаются ненадежными и не могут прямо или косвенно воздействовать на внешний мир.

Р ⊃ "ом режиме Perl настаивает на том, чтобы переменная пути задавалась заранее, даже если при запуске программы указывается полный путь. Дело в том, что нельзя быть уверенным, что выполняемая команда не вызовет другую программу по относительному имени. Кроме того, вы должны снимать пометку со всех внешних данных.

Например, при выполнении в режиме пометки фрагмента:

```
#!/usr/bin/perl -T
open(FH, "> $ARGV[0]") or die;
```

Perl выдает следующее предупреждение:

```
Insecure dependency in open while running with -T switch at ...
```

Это объясняется тем, что значение $ARGV[0] (поступившее в программу извне) считается не заслуживающим доверия. Единственный способ снять пометку с ненадежных данных — использовать обратные ссылки в регулярных выражениях:

```
$file = $ARGV[0];                          # $file помечена
unless ($file =" m#^([\w.-]+)$#) {         # С $1 снята пометка
    die "filename '$file' has invalid characters.\n";
}
$file = $1,                                # С $file снята пометка
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Помеченные данные могут поступать из любого источника, находящегося вне программы, — например, из аргументов или переменных окружения, из результатов чтения файловых или каталоговых манипуляторов, команды stat или данных о локальном контексте. К числу операций, которые считаются ненадежными с помеченными данными, относятся: system(СТРОКА), exec(СТРОКА), ' , glob, openВ любомрежиме,кроме"толькодлячтения", unlink, mkdir, rmdir, chown, chmod, umask, link, symlink, флаг командной строки -s, kill, eval, truncate, ioctl, fcntl, socket, socketpair, bind, connect, chdir, chroot, setgrp, setpriority и syscall.

Один из распространенных видов атаки связан с так называемой ***ситуацией перехвата*** (race condition). Ситуация перехвата возникает тогда, когда нападающий вмешивается между двумя вашими действиями и вносит какие-то изменения, нарушающие работу программу. Печально известная ситуация перехвата возникала при работе setuid-сценариев в старых ядрах UNIX. Между тем как ядро читало файл и выбирало нужный интерпретатор и чтением файла интерпретатором после setuid злонамеренный чужак мог подставить свой собственный сценарий.

Ситуации перехвата возникают даже во внешне безобидных местах. Допустим, у вас одновременно выполняется не одна, а сразу несколько копий следующего кода:

```
unless (-e $filename) {                    # НЕВЕРНО!
    open(FH, > $filename )
    #
}
```

Между проверкой существования файла и его открытием для записи возникает возможность перехвата. Что еще хуже, если файл заменится ссылкой на что-нибудь важное (например, на ваш личный конфигурационный файл), предыдущий фрагмент сотрет этот файл. Правильным решением является неразрушающее создание функцией sysopen (см.рецепт 7.1).

Setuid-сценарий CGI работает с другими правами, нежели Web-сервер. Так он получает возможность работать с ресурсами (файлами, скрытыми базами данных паролей и т. д.), которые иначе были бы для него недоступны. Это может быть удобно, но может быть и опасно. Из-за недостатков setuid-сценариев хакеры могут получить доступ не только к файлам, доступным для Web-сервера с его низкими привилегиями, но и к файлам, доступным для пользователя, с правами которого работает сценарий. Плохо написанный сценарий, работающий с правами системного администратора, позволит кому угодно изменить пароли, удалить файлы, прочитать данные кредитных карт и совершить иные злодеяния. По этой причине программа всегда должна работать с минимальным возможным уровнем привилегий, как правило — со стандартными дляWeb-сервера правами nobody.

Наконец, принимайте во внимание физический путь вашего сетевого трафика (возможно, это самая трудная из всех рекомендаций). Передаете ли вы незашифрованные пароли? Не перемещаются ли они по ненадежной сети? Поле формы PASSWORD защищает лишь от тех, кто подглядывает из-за плеча. При работе с паролями всегда используйте SSL. Если вы серьезно думаете о безопасности, беритесь за броузер и программу перехвата пакетов, чтобы узнать, легко ли расшифровать ваш сетевой трафик.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Смотри также

Perlsec(1); спецификации CGI и HTTP, а также список FAQ по безопасности CGI, упомянутые во введении этой главы; раздел "Avoiding Denial of Service Attacks" в стандартной документации по модулю CGI; рецепт 19.6.

19.5. Повышение эффективности сценариев CGI

Проблема

От частых вызовов вашего сценария CGI снижается производительность сервера. Вы хотите уменьшить нагрузку, связанную с работой сценария.

Решение

Используйте модуль `mod_perl` Web-сервера Apache и включите в файл *httpd.conf* следующую секцию:

```
Alias /perl/ /real/path/to/perl/scripts/      /

<Location /perl>
SetHandler perl-script
PerlHandler Apache Registry
Options ExecCGI
</Location>

PerlModule Apache Registry
PerlModule CGI
PerlSendHeader On
```

Комментарий

Модуль *mod_perl* Web-сервера Apache позволяет писать код Perl, который может выполняться на любой стадии обработки запроса. Вы можете написать свои собственные процедуры регистрации и аутентификации, определить виртуальные хосты и их конфигурацию и написать собственные обработчики для некоторых типов запросов.

Приведенный выше фрагмент сообщает, что URL, начинающиеся *c/perl/,* в действительности находятся в и обрабатываются Apache.:Registry. В результате они будут выполняться в среде CGI. Строка PerlModule CGIвыполняетпредварительнуюзагрузкумодуляCGI,аPerlSendHandlerOnпозволяет большинству сценариев CGI работать с *mod_perl.*

/perl/ работает аналогично `/cgi-bin/`. Чтобы суффикс *.perl* являлся признаком сценариев CGI `mod_perl`, подобно тому, как суффикс *.cgi* является признаком обычных сценариев CGI, включите в конфигурационный файл Apache следующий фрагмент:

```
<Files * perl>
SetHandler perl-script
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
PerlHandler Apache::Registry
Options ExecCGI
</Files>
```

Поскольку интерпретатор Perl, выполняющий сценарий CGI, не выгружается из памяти при завершении сценария (что обычно происходит, когда Web-сервер выполняет сценарий как отдельную программу), не следует полагаться на то, что при запуске программы глобальные переменные имеют неопределенные значения. Флаг *-w* и use `strict` проверяют многие недостатки в сценариях такого рода. Существуют и другие потенциальные ловушки — обращайтесь к странице руководства *mod_perl_traps*.

Не беспокойтесь о том, насколько снизится быстродействие Web-сервера от предварительной загрузки всех сценариев. Все равно когда-нибудь придется загружать их в память; желательно, чтобы это произошло до того, как Apache начнет плодить потомков. В этом случае каждый сценарий будет находиться в памяти в единственном экземпляре, поскольку влюбой современной операционной системе потомки используют общие страницы памяти. Иначе говоря, предварительная загрузка только на первый взгляд увеличивает расходы памяти — на самом деле она их уменьшает!

По адресу *http://www.perl.com/CPAN-local/modules/by-modules/Netscape/nsapi_perl-0.24.tar.gz* имеется интерфейс к серверу Netscape, который также повышает производительность за счет отказа от порождения новых процессов.

Смотри также

Документация по модулям Bundle::Apache, Apache, Apache::Registry от CPAN; *http://perl.apache.org/*, *http://perl.apache.org/faqa/*; man-страницы *mod_perl*(3) и *cgi_to_mod_perl*(1) (если есть).

19.6. Выполнение команд без обращений к командному интерпретатору

Проблема

Пользовательский ввод должен использоваться как часть команды, но вы не хотите, чтобы пользователь заставлял командный интерпретатор выполнять другие команды или обращаться к другим файлам. Если просто вызвать функцию system или `` `...` `` с одним аргументом (командной строкой), то для выполнения может быть использован командный интерпретатор, а это небезопасно.

Решение

В отличие от одноаргументной версии, списковый вариант функции system надежно защищен от обращений к командному интерпретатору. Если аргументы команды содержат пользовательский ввод от формы, никогда не используйте вызовы вида:

```
system("command $input @files");            # НЕНАДЕЖНО
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Воспользуйтесь следующей записью:

```
system('command', $input, @files);          # НАДЕЖНЕЕ
```

Комментарий

Поскольку Perl разрабатывался как "язык-клей", в нем легко запустить другую программу — в некоторых ситуациях даже слишком легко.

Если **вы** просто пытаетесь выполнить команду оболочки без сохранения ее вывода, вызвать system в многоаргументной версии достаточно просто. Но что делать, если вы используете команду в '...' или она является аргументом функции open? Возникают серьезные трудности, поскольку эти конструкции в отличие от system непозволяют передавать несколько аргументов. Возможное решение — вручную создавать процессы с помощью fork и exec. Работы прибавится, но, по крайней мере, непредвиденные обращения к командному интерпретатору не будут портить вам настроение.

Обратные апострофы используются в сценариях CGI лишь в том случае, если передаваемые аргументы генерируются внутри самой программы:

```
chomp($now = `date`);
```

Но если команда в обратных апострофах содержит пользовательский ввод — например:

```
@output = `$input @files`;
```

приходится действовать намного осторожнее.

```
die "cannot fork: $!" unless defined ($pid = open(SAFE_KID, '|-')).
if ($pid == 0) {
                $input, @files) or die "can't exec grep: $!",
} else {
    @output = <SAFE_KID>;
    close SAFE_KID,              # $? содержит информацию состояния
}
```

Такое решение работает, поскольку exec, как и system, допускает форму вызова, свободную от обращений к командному интерпретатору. При передаче списка интерпретатор не используется, что исключает возможные побочные эффекты.

При выполнении команды функцией open также потребуется немного потрудиться. Начнем с открытия функцией open конвейера для чтения. Вместо ненадежного кода:

```
open(KID_TO_READ, "$program @options @args |"),     # НЕНАДЕЖНО
```

используется более сложный, но безопасный код:

```
# Добавить обработку ошибок
die "cannot fork: $!" unless defined($pid = open(KID_TO_READ, "-|"));

if ($pid) {   # Родитель
   while (<KID_TO_READ>) {
       # Сделать что-то интересное
   }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    close(KID_TO_READ)                or warn  kid exited $? ,

} else {      # Потомок
    # Переконфигурировать затем
    exec($program @options @args) or die can t exec program $! ,
}
```

Безопасный конвейерный вызов open существует и для записи. Ненадежный вызов:

```
open(KID_TO_WRITE, |$program $options @args ),    # НЕНАДЕЖНО
```

заменяется более сложным, но безопасным кодом:

```
$pid = open(KID_TO_WRITE, |- ),
die cannot fork $! unless defined($pid = open(KID_TO_WRITE, |- )),
$SIG{ALRM} = sub { die whoops, $program pipe broke },

if ($pid) {  # Родитель
    for (@data) { print KID_TO_WRITE $_ }
    close(KID_TO_WRITE)               or warn  kid exited $?

} else {      # Потомок
    # Переконфигурировать, затем
    exec($program, @options, @args) or die can t exec program $! ,
}
```

Там, где комментарий гласит "Переконфигурировать", предпринимаются дополнительные меры безопасности. Вы находитесь в порожденном процессе, и вносимые изменения не распространяются на родителя. Можно изменить переменные окружения, сбросить временный идентификатор пользователя или группы, сменить каталог или маску umask и т. д.

Разумеется, все это не поможет в ситуации, когда вызов system запускает программу с другим идентификатором пользователя, Например, почтовая программа ***sendmail*** является setuid-программой, часто запускаемой из сценариев CGI Вы должны хорошо понимать риск, связанный с запуском ***sendmail*** или любой другой setuid-программы.

Смотри также

Описание функций system, exec и open в *perlfunc*(1); *perlsec*(1); рецепты **16.1—16.3.**

19.7. Форматирование списков и таблиц средствами HTML

Проблема

Требуется сгенерировать несколько списков и таблиц. Нужны вспомогательные функции, которые **бы** упростили вашу работу.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Решение

Модуль CGI содержит вспомогательные функции HTML, которые получают ссылку на массив и автоматически применяются к каждому элементу массива:

```
print ol( li([
<OL><LI>red</LI>  <LI>blue</LI>  <LI>green</LI></OL>
@names = qw(Larry Moe Curly),
print ul( li({ -TYPE => disc  },\@names)  ),
<UL><LI  TYPE="disc">Larry</LI>  <LI  TYPE="disc">Moe</LI>
      <LI  TYPE="disc">Curly</LI></UL>
```

Комментарий

Свойство дистрибутивности функций CGI.pm, генерирующих HTML-код, заметно упрощает процесс генерации списков и таблиц. При передаче простой строки эти функции простовыдают HTML-код для даннойстроки. Но при передаче ссылки на массив они применяются ко всем строкам.

```
print  li( alpha ),
      <LI>alpha</LI>
print  li( [  alpha ,  omega ] ),
      <LI>alpha</LI>  <LI>omega</LI>
```

Вспомогательные функции для списков загружаются при использовании тега standard, но дляполучения вспомогательных функций для работы с таблицами придется явно запросить html3. Кроме того, возникает конфликт между тегом <TR>, которому должна соответствовать функция tr(), и встроенным оператором Perl tr///. Следовательно, для построения строк таблицы следует использовать функцию Tr().

Следующий пример генерирует таблицу HTML похэшу массивов. Ключи хэша содержат заголовки строк, а массивы значений — столбцы.

```
use CGI qw( standard html3),

%hash = (
     Wisconsin   => [  Superior ,  Lake Geneva ,  Madison  ],
     Colorado    => [  Denver ,  Fort Collins ,  Boulder  ],
     Texas       => [  Plano ,  Austin ,  Fort Stockton  ],
     California  => [  Sebastopol ,  Santa Rosa ,  Berkeley  ],
 )

$\ =  \n ,

print  <TABLE> <CAPTION>Cities I Have Known</CAPTION>
print Tr(th [qw(State Cities)]),
for $k (sort keys %hash) {
    print Tr(th($k), td( [ sort @{$hash{$k}} ] )),
}
print  </TABLE> ,
```

Генерируется следующий текст.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
<TABLE> <CAPTION>Cities I Have Known</CAPTION>
      <TR><TH>State</TH> <TH>Cities</TH></TR>
      <TR><TH>California</TH> <TD>Berkeley</TD> <TD>Santa Rosa</TD>
         <TD>Sebastopol</TD> </TR>
      <TR><TH>Colorado</TH> <TD>Boulder</TD> <TD>Denver</TD>
         <TD>Fort Collins</TD> </TR>
      <TR><TH>Texas</TH> <TD>Austin</TD> <TD>Fort Stockton</TD>
         <TD>Plano</TD></TR>
      <TR><TH>Wisconsin</TH> <TD>Lake Geneva</TD> <TD>Madison</TD>
         <TD>Superior</TDX/TR>
</TABLE>
```

Те же результаты можно получить всего одной командой print, хотя это несколько сложнее, поскольку вам придется создавать неявный цикл с помощью map. Следующая команда print выдает результат, идентичный приведенному выше:

```
print table
      caption( Cities I have Known ),
      Tr(th [qw(State Cities)]),
      map { Tr(th($_), td( [ sort @{$hash{$_}} ] )) } sort keys %hash,
```

Эти функции особенно удобны при форматировании результатов запроса к базе данных, как показано в примере 19.3 (см. главу 14 "Базы данных»).

Пример 19.3. salcheck

```
#!/usr/bin/perl
# salcheck - проверка жалованья
use DBI,
use CGI qw( standard html3),

$limit = param( LIMIT )

print header(), start_html( Salary Query ),
      h1( Search )
      start_form()
      p( Enter minimum salary  textfield( LIMIT )),
      submit(),
      end_form(),

if (defined $limit) {
    $dbh = DBI->connect( dbi mysql somedb server host dom 3306 ,
        username ,  password )
       or die  Connecting  $DBI  errstr ,
    $sth =                SELECT name salary FROM employees
       WHERE salary > $limit )
          die              , $dbh->errstr
    $sth->execute
       or die  Executing    , $sth->errstr

    print h1( Results ),  <TABLE BORDER=1>
```

продолжение ☞

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 19.3 (продолжение)

```
    while (@row = $sth->fetchrow()) {
          print Tr( td( \@row ) ),
    }

    print </TABLE>\n ,
    $sth->finish,
    $dbh->disconnect,
}

print end_html(),
```

Смотри также

Документация по стандартному модулю CGI; рецепт 14.10.

19.8. Перенаправление клиентского броузера

Проблема

Требуется сообщить клиентскому броузеру о том, что страница находится в другом месте.

Решение

Вместо обычного заголовка выведите перенаправление и завершите программу. Не забудьте о дополнительной пустой строке в конце заголовка:

```
$url = http //www perl com/CPAN/ ,
print Location $url\n\n ,
exit
```

Комментарий

Иногда программа CGI не генерирует документ сама. Она лишь сообщает клиенту о том, что ему следует получить другой документ. В этом случае заголовок HTTP содержит слово Location, за которым следует новый URL. Обязательно используйте абсолютный, а не относительный URL.

Прямолинейного решения, показанного выше, обычно вполне хватает. Но если модуль CGI уже загружен, воспользуйтесь функцией примере 19.4 эта возможность применяется при построении cookie.

Пример 19.4.

```
#!/usr/bin/perl -w
                тановить cookie и перенаправить броузер
use CGI qw( cgi),

        cookie( -NAME    => filling ,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
                -VALUE   => "vanilla crème",
                -EXPIRES => '+3M',     # M - месяц, m - минута
                -DOMAIN  => '.perl.com');

$whither  = "http://somewhere.perl.com/nonesuch.html";

print           -URL     => $whither,
                -COOKIE  =>
```

Результат выглядит так:

```
Status:  302  Moved  Temporarily
Set-Cookie:   filling=vanilla%20cr%E4me;   domain=.perl.com;
                21-Jul-1998 11:58:55 QMT
Date: Tue, 21 Apr 1998 11:55:55 GMT
Location:     http://somewhere.perlcom/nonesuch.html
Content-Type:  text/html
B<<blank line
```

В примере 19.5 приведена законченная программа, которая определяет имя клиентского броузера и перенаправляет его на страницу "Файла жаргона" Эрика Реймонда, где говорится о соответствующей операционной системе. Кроме того, в программе хорошо продемонстрирован альтернативный подход к созданию конструкций switch в Perl.

Пример 19.5. os_snipe

```
#!/usr/bin/perl
# os_snipe - перенаправить в статью Файла жаргона,
#            посвященную текущей операционной системе
$dir  =  "http://www.wins.uva.nl/%7Emes/jargon";
for ($ENV{HTTP_USER_AGENT}) {
                /Mac/              &&  "m/Macintrash.html"
              |  /Win(dows )?NT/   &&  "e/evilandrude.html"
              |  /Win|MSIE|WebTV/  &&  "m/MicroslothWindows.html"
              |  /Linux/           &&  "l/Linux.html"
              |  /HP-UX/           &&  "h/HP-SUX.html"
              |  /SunOS/           &&  "s/ScumOS.html"
              |                       "a/AppendixB.html";
}
print "Location:  $dir/$page\n\n";
```

В программе ***os_snipe*** использовано динамическое перенаправление, поскольку разные пользователи отсылаются на разные страницы. Если перенаправление всегда ведет к одному месту, разумнее включить статическую строку **в** конфигурационный файл сервера — это обойдется дешевле, чем запуск сценария CGI для каждого перенаправления.

Сообщить клиентскому броузеру, что **вы** не собираетесь выдавать никаких данных — далеко не то же самое, что перенаправить его "**в** никуда":

```
use CGI qw(:standard);
print header(
```

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

Результат выглядит так:

```
Status:
Content-Type:   text/html
<blank line
```

Например, этот вариант используется в ситуации, когда от пользователя приходит запрос, а вы не хотите, чтобы его страница изменилась или даже просто обновилась.

Выглядит немного глупо — сначалаМы указываем тип содержимого, а потом говорим, что содержимого не будет, — но модуль поступает именно так. При ручном кодировании это бы не понадобилось.

```
#!/bin/sh

cat "EOCAT
Status: 204 No

EOCAT
```

Смотри также

Документация по стандартному модулю CGI.

19.9. Отладка на уровне HTTP

Проблема

Сценарий CGI странно ведет себя с броузером. Вы подозреваете, что в заголовке HTTP чего-то не хватает. Требуется узнать, что именно броузер посылает серверу в заголовке HTTP.

Решение

Создайте фиктивный Web-сервер (см. пример 19.6) и подключитесь к нему в своем броузере.

Пример 19.6. dummyhttpd

```
#!/usr/bin/perl -w
# dummyhttpd - запустить демона HTTP и выводить данные,
#              получаемые от клиента

use strict;
use HTTP::Daemon;   # Требуется LWP-5.32 и выше

my $server = HTTP::Daemon->new(Timeout => 60);
print "Please contact me at: <URL:", $server->url, ">\n";

while (my $client = $server->accept) {
  CONNECTION:
    while (my $answer = $client->get_request) {
        print $answer->as_string;
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
        $client->autoflush;
      RESPONSE:
        while (<STDIN>) {
            last RESPONSE  if $_ eq ".\n";
            last CONNECTION if $_ eq "..\n";
            print $client $_;
        }
        print "\nEOF\n";
    }
    print "CLOSE: ", $client->reason, '\n';
    $client->close;
    undef $client;
}
```

Комментарий

Трудно уследить за тем, какие версии тех или иных броузеров все еще содержат ошибки. Фиктивная программа-сервер может спасти от многодневных напряженных раздумий, поскольку иногда неправильно работающий броузер посылает серверу неверные данные. На своем опыте нам приходилось видеть, как броузеры теряли cookies, неверно оформляли URL, передавали неверную строку состояния и совершали менее очевидные ошибки.

Фиктивный сервер лучше всего запускать на том же компьютере, что и настоящий. При этом броузер будет отправлять ему все cookies, предназначенные для этого домена. Вместо того чтобы направлять броузер по обычному URL:

воспользуйтесь альтернативным портом, указанным в конструкторе new. При использовании альтернативного порта необязательно быть привилегированным пользователем, чтобы запустить сервер.

http://somewhere.com:8989/cgi-bin/whatever

Если вы решите, что клиент ведет себя правильно, и захотите проверить сервер, проще всего воспользоваться программой telnet для непосредственного общения с удаленным сервером.

```
% telnet www perl com 80
GET /bogotic HTTP/1.0
<blank line here>
HTTP/1.1 404 File Not Found
Date: Tue, 21 Apr 1998 11:25:43 GMT
Server: Apache/1.2.4
Connection: close
Content-Type: text/html

<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
</HEAD><BODY>
<H1>File Not Found</H1>
The requested URL /bogotic was not found on this server.<P>
</BODY></HTML>
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Если в вашей системе установлены модули LWP, вы сможете использовать синоним GET для программы *lwprequest*. При этом будут отслеживаться все цепочки перенаправлений, что может пролить свет на вашу проблему. Например:

```
% GET -esuSU http://mox.perl.com/perl/bogotic
QET   http://language.perl.com/bogotic
Host:  mox.perl.com
User-Agent:   lwp-request/1.32

QET  http://mox.perl.com/perl/bogotlc  ->  302  Moved  Temporarily
QET  http://www.perl.com/perl/bogotlc  ->  302  Moved  Temporarily
QET  http://language.perl.com/bogotic  ->  404  File  Not  Found
Connection;  close
Date:  Tue,  21  Apr  1998  11:29:03  QMT
Server:   Apache/1.2.4
Content-Type:   text/html
Client-Date:  Tue,  21  Apr  1998  12:29:01  GMT
Client-Peer:    208.201.239.47:80
Title:  Broken  perl.com  Links

<HTML>
<HEAD><TITLE>An   Error   Occurred</TITLE></HEAD>
<BODY>
<H1>An   Errqr
404  File  Not  Found
</BODY>
</HTML>
```

Смотри также

Документация по стандартному модулю CGI; рецепт 14.10.

19.10. Работа с cookies

Проблема

Вы хотите получить или создать cookie для хранения параметров сеанса или настроек пользователя.

Решение

В модуле CGI.pm получение существующей cookie выполняется так:

```
cookie("preference name");
```

Cookie создаются следующим образом:

```
$packed_cookie = cookie( -NAME    =>              name",
                         -VALUE   => "whatever you'd like",
                         -EXPIRES => "+2y");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Чтобы сохранить **cookie в** клиентском броузере, необходимо включить **ее** в заголовок HTTP (вероятно, с помощью функций `header` или `redirect`):

```
print header(-COOKIE => $packed_oookie);
```

Комментарий

Cookies используются для хранения информации о клиентском броузере. Если вы работаете с Netscape в UNIX, просмотрите файл *~/.netscape/cookies*, хотя в **нем** содержатся не все **cookies**, а лишь те, которые присутствовали на момент последнего выхода из броузера. Cookies можно рассматривать как пользовательские настройки уровня приложения или как средство упростить обмен данными. Преимущества cookies заключаются в **том**, что они могут совместно использоваться несколькими разными программами и даже сохраняются между вызовами броузера.

Однако cookies также применяются и для сомнительных штучек типа анализа трафика. Нервные пользователи начинают гадать, кто и зачем собирает их личные данные. Кроме того, cookies привязаны к одному компьютеру. Если вы работаете с броузером у себя дома или в другом офисе, в нем не будет cookies из броузера, находящегося у вас на работе. По этой причине не следует ожидать, что каждый броузер примет cookies, которые вы ему даете. А если этого покажется недостаточно, броузеры могут уничтожать cookies по своему усмотрению. Ниже приведена выдержка из чернового документа "Механизм управления состоянием HTTP" (HTTP State Management Mechanism") по адресу *~dmk/cookie-2.81-3.1.txt*.

"Поскольку пользовательские агенты обладают ограниченным пространством для хранения cookies, они могут удалять старые cookies, чтобы освободить место для новых — например, используя алгоритм удаления по сроку последнего использования в сочетании с ограничением максимального числа cookies, создаваемых каждым сервером."

Cookies ненадежны, поэтому на них не стоит чрезмерно полагаться. Используйте их для простых транзакций с конкретным состоянием. Избегайте анализа трафика, это может быть принято за вмешательство в личные дела пользователей.

В примере 19.7 приведена законченная программа, которая запоминает последний выбор пользователя.

Пример 19.7. ic_cookies

```
#!/usr/bin/perl -w
# ic_cookies - пример сценария CGI с использованием cookie
use CGI qw(:standard);

use strict;

my $cookname = "favorite ice
my $favorite = param("flavor");
my $tasty    = cookie($cookname) || "mint";

unless ($favorite) {
    print header(), start_html("Ice Cookies"), h1("Hello Ice
        hr(), start_form(),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 19.7 (продолжение)

```
            p( Please select a flavor    , textfield( flavor',$tasty)),
              end_form(), hr(),
    exit,
}

my $cookie = cookie(
                -NAME    => $cookname,
                -VALUE   => $favorite,
                -EXPIRES =>  +2y  ,
            ),

print header(-COOKIE => $cookie),
      start_html( Ice Cookies, #2 ),
      h1( Hello Ice
      p( You chose as your favorite flavor  $favorite   ),
```

Смотри также

Документация по стандартному модулю CGI.

19.11. Создание устойчивых элементов

Проблема

Вы хотите, чтобы по умолчанию в полях формы отображались последние использованные значения. Например, вы хотите создать форму для поисковой системы наподобие AltaVista (*http://www.altavista.com*), где над результатами отображаются искомые ключевые слова.

Решение

Создайте форму с помощью вспомогательных функций HTML, которые автоматически заносят в поле предыдущее значение:

```
print textfield( SEARCH ),       # Предыдущее значение SEARCH
                                 # используется по умолчанию
```

Комментарий

В примере 19.8 приведен простой сценарий для получения информации о пользователях, зарегистрированных в настоящее время.

Пример 19.8. who.cgi

```
#!/usr/bin/perl -wT
# who cgi - вызвать who(1) для пользователя и отформатировать результат

$ENV{IFS}=  ,
$ENV{PATH}= /bin /usr/bin ,

use CGI qw( standard),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
# Вывести поисковую форму
print header(), start_html('Query Users"), h1('Search");
print start_form(), p("Which user?", textfield("WHO")); submit(), end_form();

# Вывести результаты поиска
$name = param("WHO");
if ($name) <
    print h1("Results");
    $html = '';

    # Вызвать who и построить текст ответа                '
            ('who') {
        next unless /^$name\s/o;   # Только строки, совпадающие с $name
        s/&/&amp;/g;
        s/</&lt;/g,
        s/>/&gt;/g;
        $html .= $_;
    }
    # Если пользователь не найден, вывести сообщение
    $html = $html || "$name is not logged in";

    print pre($html),


print end_html();
```

Функция `textfield` генерирует HTML-код для текстового поля с именем параметра `WHO`. После вывода формы мы проверяем, было ли присвоено значение параметру `WHO`, и если было — ищем в результатах *who* строки данного пользователя.

Смотри также

Документация по стандартному модулю CGI; рецепты **19.4; 19.7.**

19.12. Создание многостраничного сценария CGI

Проблема

Требуется написать сценарий CGI, который бы возвращал броузеру несколько страниц. Допустим, вы хотите написать сценарий CGI для работы с базой данных продуктов. Он должен выводить несколько форм: общий список продуктов, формы для добавления новых и удаления существующих продуктов, для редактирования текущих атрибутов продуктов и т. д. Многостраничный сценарий CGI образует простейший вариант "электронного магазина".

Решение

Сохраните информацию о текущей странице в скрытом поле.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Модуль CGI позволяет легко генерировать устойчивые скрытые поля Функция hidden возвращает HTML-код скрытого элемента и использует его текущее значение в том случае, если ей передается только имя элемента:

```
use CGI qw( standard),
print hidden( bacon ),
```

Отображаемая страница ("Общий список продуктов», "Список заказанных продуктов", "Подтверждение заказа" и т. д.) выбирается по значению скрытого поля. Мы назовем его State, чтобы избежать возможных конфликтов с именами других полей. Для перемещения между страницами используются кнопки, которые присваивают State имя новой страницы. Например, кнопка для перехода к странице "Checkout" создается так:

```
print submit(-NAME=> State , -VALUE => Checkout ),
```

Для удобства можно создать вспомогательную функцию:

```
sub to_page {        submit( -NAME => State -VALUE => shift ) >
```

Чтобы выбрать отображаемый код, достаточно проверить параметр State:

```
$page = param( State ) || Default
```

Код, генерирующий каждую страницу, размещается в отдельной подпрограмме. Вообще говоря, нужную подпрограмму можно выбирать длинной конструкцией if elsif elsif:

```
if ($page eq Default ) {
    front_page(),
} elsif ($page eq Checkout ) {
    checkout(),
} else {
    no_such_page(), № Если State ссылается на несуществующую страницу
}
```

Получается некрасивое, громоздкое решение. Вместо этого следует использовать **хэш,** ассоциирующий имя страницы с подпрограммой. Это еще один **из** вариантов реализации С-подобной конструкции switch на Perl.

```
%States = (
    Default      => \&front_page
    Shirt        => \&shirt,
    Sweater      => \&sweater,
    Checkout     => \&checkout,
    Card         =>
    Order        => \&order,
    Cancel       => \&front_page,
),

if ($States{$page}) {
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    $States{$page}->();    # Вызвать нужную подпрограмму
) else {
    no_such page();
}
```

На каждой странице найдется несколько устойчивых элементов. Например, страница для заказа футболок должна сохранить количество заказанных товаров, даже если пользователь переходит на страницу для заказа кроссовок. Для этого подпрограмма, генерирующая страницу, вызывается с параметром, который определяет, является ли данная страница активной. Если страница не является активной, возвращаются лишь значения скрытых полей для любых устойчивых данных:

```
while (($state, $sub) = each %States) {
    $sub->( $page eq $state );
}
```

Оператор сравнения eq возвращает true, если страница является активной, и false в противном случае. Подпрограмма, генерирующая страницу, принимает следующий вид:

```
sub t_shirt {
    my $active = shift;

    unless ($active) {
        print hidden("size"), hidden("color");

    }

    print p("You want to buy a t-shirt?");
    print p("Size: ", popup_menu("size", [ qw(XL L M S XS) ]));
    print p("Color:", popup_menu("color", [ qw(Black White) ]));

    print p( to page("Shoes"), to page("Checkout") );
}
```

Поскольку все подпрограммы генерируют HTML-код, перед вызовом необходимо вывести заголовок HTTP и начать HTML-документ и форму. Это позволит использовать стандартные колонтитулы для всех страниц, если мы захотим. Следующий фрагмент предполагает, что у нас имеются процедуры standard_header и standard_footer для вывода верхних и нижних колонтитулов страниц:

```
print header("Program Title"), begin_html();
print standard_header(), begin_form();
while (($state, $sub) = each %States) {
    $sub->( $page eq $state );
}
print standard_footer(), end_form(), end_html();
```

Кодирование цены в форме будет ошибкой. Вычисляйте цены на основании значений скрытых элементов и как можно чаще проверяйте информацию. Например, сравнение со списком существующих продуктов позволяет отбросить явно неразумные заказы.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Скрытые данные обладают большими возможностями, чем cookies, поскольку вы не можете твердо рассчитывать на поддержку cookies или на то, что броузер согласится принять их. Более полная информация приведена в рецепте 19.10.

В конце главы приведена программа *chemiserie* — простейшее приложение для обслуживания электронного магазина.

Смотри также

Документация по стандартному модулю CGI.

19.13. Сохранение формы в файле или канале

Проблема

Сценарий CGI должен сохранить все содержимое формы в файле или передать его в канал.

Решение

Для сохранения формы воспользуйтесь функцией `save_parameters` или методом `save` модуля CGI; их параметром является файловый манипулятор. Сохранение в файле выполняется так:

```
Я Сначала открыть и монопольно заблокировать файл
open(FH,  >>/tmp/formlog )        or die  can t append to formlog  $!  ,
flock(FH,  2)                     or die  can t flock formlog  $!  ,

#  Используется процедурный интерфейс
use CGI qw( standard),
save_parameters(*FH),                        я CGI  save

# Используется объектный интерфейс
use CGI,
$query = CGI->new(),
$query->save(*FH),

close(FH)                         or die  can t close formlog  $!  ,
```

Или форма сохраняется в канале — например, соединенном с процессом *sendmail*:

```
use CGI qw( standard),
open(MAIL,  |/usr/lib/sendmail -oi -t )
                                  or die  can t fork sendmail  $!  ,
print  MAIL  "EOF,
From  $0  (your cgi script)
To  hisname\@hishost com
Subject  mailed form submission

EOF
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
save_parameters(*MAIL),
close(MAIL)                    or die  can t close  sendmail  $'
```

Комментарий

Иногда данные формы сохраняются для последующего использования. Функция save_parameters и метод save модуля CGI pm записывают параметры формы в открытый манипулятор. Манипулятор может быть связан с открытым файлом (желательно — открытым в режиме дополнения и заблокированным, как в решении) или каналом, другой конец которого подключен к почтовой программе.

Данные сохраняются в файле в виде пар переменная=значение, служебные символы оформляются по правилам URL. Записи разделяются строками, состоящими из единственного символа =. Как правило, чтение осуществляется методом CGI->new с аргументом-манипулятором, что обеспечивает автоматическое восстановление служебных символов (см. ниже).

Если вы хотите перед сохранением включить *в* запрос дополнительную информацию, вызовите функцию param (или метод, если используется объектно-ориентированный интерфейс) с несколькими аргументами и установите нужное значение (или значения) параметра формы Например, текущее время и состояние окружения сохраняется следующим образом:

```
param( _timestamp   scalar localtime),
param( _environs   %ENV),
```

После сохранения формы в файле дальнейшая работа с ней ведется через объектно-ориентированный интерфейс

Чтобы загрузить объект-запрос из файлового манипулятора, вызовите метод new с аргументом-манипулятором. При каждом вызове возвращается законченная форма. При достижении конца файла будет возвращена форма, не имеющая параметров Следующий фрагмент показывает, как это делается. Он накапливает сумму всех параметров лишь **в** том случае, если форма поступила не с сайта *perl com.* Напомним, что параметры _environs и _timestamp были добавлены при записи файла.

```
use CGI
open(FORMS,  < /tmp/formlog )        or die  can t      formlog  $'
flock(FORMS, 1)                     or die  can t lock formlog  $' ,
while ($query = CGI->new(*FORMS)) {
    last unless $query->param(),      # Признак конца файла
    %his_env = $query->param( _environs ),
    $count  += $query->param( items
                unless $his env{REMOTE_HOST} =^ /(^|\ )perl\ com$/
}
print  Total orders  $count\n
```

Как всегда при создании файлов в сценариях CGI, важную роль играют права доступа и права владельца файла.

Смотри также

Рецепты **18.3; 19.3.**

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

19.14. Программа: chemiserie

Сценарий CGI из примера **19.9** предназначен для заказа футболок и свитеров через Web. В нем использованы приемы, описанные в рецепте **19.12.** Вывод не отличается особой элегантностью или красотой, но продемонстрировать многостраничную работу в короткой программе слишком сложно, чтобы заботиться об эстетике.

Подпрограммы shirt и sweater проверяют значения соответствующих элементов формы. Если цвет или размер окажется неправильным, в элемент заносится первое значение из списка допустимых цветов или размеров.

Пример 19.9. chemiserie

```
#!/usr/bin/perl -w
# chemiserie - простой сценарий CGI для заказа футболок и свитеров

use strict;
use CGI qw( standard);
use CGI   Carp qw(fatalsToBrowser);

my %States;            # Хэш состояний - связывает страницы
                       ft с функциями
                       # Текущий экран

# Хэш страниц и функций

%States = (
     Default       =>  \&front_page,
     Shirt         => \&shirt,
     Sweater       =>  \&sweater,
     Checkout      =>  \&checkout,
     Card          => \&credit_card,
     Order         => \&order,
     Cancel        => \&front_page,
);

$Current_Screen = param(  State ) ||  Default ;
                                      unless

# Сгенерировать текущую страницу

standard_header();

while  (my($screen_name, $function) = each  %States) {
    $function->($screen_name eq $Current_Screen);
}
standard_footer();
exit;

################################
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
  Колонтитулы формы, функции меню


sub standard_header {
    print header(), start_html(-Title => Shirts , -BGCOLOR=> White ),
    print start form(), # start multipart_form() if file upload
}

sub standard_footer { print end_form(), end_html() }

sub shop_menu {
    print p(defaults( Empty My Shopping Cart ),
        to_page( Shirt ),
        to_page( Sweater ),
        to_page( Checkout )),
)


# Подпрограммы для каждого экрана


# Страница по умолчанию                                    ,
sub front_page {
    my $active = shift,
            unless $active,

    print <H1>Hi'</H1>\n ,
    print Welcome to our Shirt Shop' Please make your selection from
    print the menu below \n ,

    shop_menu(),
}

8 Страница для заказа футболок
sub shirt {
    my $active = shift,
    my @sizes = qw(XL L M S),
    my @colors = qw(Black White),

    my ($size, $color, $count) =
       (param('shirt_size'), param( shirt_color ), param( shirt_count )),

    я Проверка
    if ($count) {
        $color = $colors[0]                    $color } @colors,
        $size = $sizes[0]                      $size } @sizes,
        param('shirt_color , $color),
        param( shirt_size , $size),
    }
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 19.9 (продолжение)

```
    unless ($active) {
        print hidden("shirt_size") if $size;
        print hidden("shirt_color") if $color;
        print hidden("shirt_count") if $count;

    }

    print h1("T-Shirt"),
    print p("What a shirt! This baby is decked out with all the options ",
        "It comes with full luxury interior, cotton trim, and a collar",
        "to make your eyes water! Unit price: $33.00"),
    print h2("Options");
    print p("How Many?", textfield("shirt_count"));
    print p("Size?", popup_menu("shirt_size", \@sizes ),
        "Color?", popup_menu("shirt_color", \@colors));

    shop_menu();
}

fl Страница для заказа свитеров.
sub sweater {
    my $active = shift;
    my @sizes = qw(XL L M);
    my @colors = qw(Chartreuse Puce Lavender);

    my ($size, $color, $count) =
       (param("sweater_size"), param("sweater_color"), param("sweater_count"));

    # Проверка
    if ($count) {
        $color = $colors[0] unless grep { $_ eq $color } @colors,
        $size = $sizes[0] unless grep { $_ eq $size } @sizes;
        param("sweater_color", $color);
        param("sweater_size", $size);
    }

    unless ($active) {
        print hidden("sweater_size") if $size;
        print hidden("sweater_color") if $color;
        print hidden("sweater_count") if $count;

    }

    print h1("Sweater");
    print p("Nothing implies elegance than this fine",
        "sweater, Made by peasant workers from black market silk,",
        "it slides onto your lean form and cries out "Take me,",
        "for I am a god!" Unit price \$49.99.");
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    print h2( Options ),
    print p( How Many?   textfield( sweater_count ))
    print p( Size? ,  popup_menu( sweater_size ,  \@sizes)),
    print p( Color? , popup_menu( sweater_color , \@colors)),

    shop_menu(),
}

# Страница для подтверждения текущего заказа
sub checkout {
    my $active = shift,

           unless $active,

    print h1( Order Confirmation ),
    print p( You             following  ),
    print  order_text(),
    print p( Is this right?  Select  Card  to pay for the  items ,
         or  Shirt  or  Sweater  to continue shopping  ),
    print  p(to_page( Card  ),
        to_page( Shirt  ),
        to_page( Sweater )),
}

# Страница для ввода данных кредитной карты
sub credit_card {
    my $active = shift,
    my @widgets = qw(Name                    City Zip State Phone Card Expiry),

    unless ($active) {
        print map { hidden($_) } @widgets,

    }

    print        Name              textfield(  Name  )),
                        , textfield(
    P(                  , textfield(
    p( City             , textfield( City )),
    p( Zip              , textfield( Zip )),
    p( State            , textfield( State )),
    p( Phone            , textfield( Phone )),
                        ,  textfield( Card  )),
    p( Expiry           ,  textfield( Expiry  )))

    print p( Click on  Order  to order the items   Click on  Cancel


    print  p(to_page( Order  )   to_page( Cancel  )),
}
```

продолжение

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 19.9 (продолжение)

```
# Страница для завершения заказа.
sub order {
    my $active = shift;

    unless ($active) {

    }

    # Проверка данных кредитной карты

    print h1("Ordered!");
    print p("You have              following toppings.");
    print order_text();

    print p(defaults("Begin Again")),
}

# Возвращает HTML-код текущего заказа ('Вы заказали ...")
sub order_text {
    my $html = '';

    if (param("shirt_count")) {
        $html   p("You                  param("shirt_count"),
         " shirts of size",  param("shirt_size"),
         " and color ", param("shirt_color"),  ".");
    }
    if (param("sweater_count")) {
        $html .= p("You have          ",  param("sweater_count"),
         " sweaters of size  ', param("sweater_size"),
         " and color ", param("sweater_color"),  "  ");
    }
    $html = p("Nothing!") unless $html;
    $html  = p("For a total cost of  ", calculate_price());
           $html;
}

sub calculate_price {
    my $shirts   = param("shirt_count")    || 0,
    my $sweaters = param("sweater_count")  || 0;
            sprintf("\$%.2f",  $shirts*33   $sweaters  •  49.99),
}

sub to_page { submit(-NAME => ".State", -VALUE => shift) }
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Автоматизация в Web 20

...Сеть одновременно чувственная и логическая, элегантная и изобилующая смыслом — это стиль, это основа литературного искусства.

Роберт Льюис Стивенсон, "0 некоторых технических элементах стиля в литературе»

Введение

В главе 19 "Программирование CGI" основное внимание уделяется ответам на запросы броузеров и генерации документов с применением CGI. В этой главе программирование для Web рассматривается с другой стороны: вместо того чтобы общаться с броузером, вы сами притворяетесь броузером, генерируете запросы и обрабатываете возвращаемые документы. Для упрощения этого процесса мы будем широко использовать модули, поскольку правильно реализовать низкоуровневые сетевые протоколы и форматы документов непросто. Поручая всю трудную работу модулям, вы концентрируетесь на самом интересном — вашей собственной программе.

Упоминаемые модули находятся по следующему URL:

http://www.perl.com/CPAN/modules/by -category/15_ World_ Wide_ Web_HTML_ HTTP_CGI/

Здесь хранятся модули для вычисления контрольных сумм кредитных карт, взаимодействия с API Netscape или сервера Apache, обработки графических карт (image maps), проверки HTML и работы с MIME. Однако самые большие и важные модули этой главы входят в комплекс модулей libwww-perl, объединяемых общим термином LWP. Ниже описаны лишь некоторые модули, входящие в LWP.

Модули HTTP:: и LWP:: позволяют запрашивать документы с сервера. В частности, модуль LWP::Simple обеспечивает простейший способ получения документов. Однако LWP::Simple не хватает возможности обращаться к отдельным компонентам ответов HTTP. Для работы с ними используются модули HTTP::Request, HTTP::Response и HTTP::UserAgent. Оба набора модулей демонстрируются в рецептах 20.1-20.2 и 20.10.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Имя модуля	Назначение
LWP::UserAgent	Класс пользовательских агентов WWW
LWP::RobotUA	Разработка приложений-роботов
LWP::Protocol	Интерфейс для различных схем протоколов
LWP::Authen::Basic	Обработка ответов 401 и 407
LWP::MediaTypes	Конфигурация типов MIME (text/html и т. д.)
LWP::Debug	Отладочный модуль
LWP::Simple	Простой процедурный интерфейс для часто используемых функций
LWP::UserAgent	Отправка HTTP::Request и возвращение HTTP::Response
HTTP::Headers	Заголовки стилей MIME/RFC822
HTTP::Message	Сообщение в стиле HTTP
HTTP::Request	Запрос HTTP
HTTP::Response	Ответ HTTP
HTTP::Daemon	Класс сервера HTTP
HTTP::Status	Коды статуса HTTP (200 OK и т. д.)
HTTP::Date	Модуль обработки даты для форматов дат HTTP
HTTP::Negotiate	Обсуждение содержимого HTTP
URI::URL	URL
WWW::RobotRules	Анализ файлов robots.txt
File::Listing	Анализ списков содержимого каталогов

Модули HTML:: находятся в близкой связи с LWP, но не распространяются в составе этого пакета. Они предназначены для анализа HTML-кода. На них основаны рецепты 20.3—20.7, про граммы *htmlsub* и *hrefsub*.

В рецепте 20.12 приведено регулярное выражение для декодирования полей в файлах журналов Web-сервера, а также показано, как интерпретировать эти поля. Мы используем это регулярное выражение с модулем Logfile::Apacheв рецепте 20.13, чтобы продемонстрировать два подхода к обобщению данных в журналах Web-серверов.

20.1. Выборка URL из сценария Perl

Проблема

Требуется обратиться из сценария по некоторому URL.

Решение

Воспользуйтесь функцией get модуля LWP::Simple от CPAN, входящего в LWP.

```
use LWP::Simple;
$content = get($URL);
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Правильный выбор библиотек заметно упрощает работу. Модули LWP идеально подходят для поставленной задачи.

Функция get модуля LWP::Simple в случае ошибки возвращает undef, поэтому ошибки следует проверять так:

```
use LWP Simple,
unless (defined ($content = get $URL)) {
    die 'could not get $URL\n',
}
```

Однако в этом случае вы не сможете определить причину ошибки. В этой и других нетривиальных ситуациях возможностей LWP::Simple оказывается недостаточно.

В примере 20.1 приведена программа выборки документа по URL. Если попытка оказывается неудачной, программа выводит строку с кодом ошибки. В противном случае печатается название документа и количество строк в его содержимом. Мы используем четыре модуля от LWP.

LWP::UserAgent

Модуль создает виртуальный броузер. Объект, полученный при вызове конструктора new, используется для дальнейших запросов. Мы задаем для своего агента имя «Schmozilla/v9.14 Platinum», чтобы Web-мастер мучился от зависти при просмотре журнала.

HTTP::Request

Модуль создает запрос, но не отправляет его. Мы создаем запрос GET и присваиваем фиктивный URL для запрашивающей страницы.

HTTP::Response

Тип объекта, возвращаемый при фактическом выполнении запроса пользовательским объектом. Проверяется на предмет ошибок и для получения искомого содержимого.

URI::Heuristic

Занятный маленький модуль использует Netscape-подобные алгоритмы для расширения частичных URL. Например:

Частичный URL	Предположение
perl	http://www.perl.com
ftp.funet.fi	ftp://ftp.funet.fi
/etc/passwd	file:/etc/passwd

Хотя строки в левом столбце не являются правильными URL (их формат не отвечает спецификации URI), Netscape пытается угадать, каким URL они соответствуют. То же самое делают и многие другие броузеры.

Исходный текст программы приведен в примере 20.1.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Пример 20.1. titlebytes

```
ft'/usr/bin/perl   -w
Я titlebytes - определение названия и размера документа
use LWP UserAgent,
use HTTP Request,
use HTTP Response,
use URI Heuristic,
my $raw_url = shift                or die usage $0 url\n ,
my $url = URI Heuristic uf_urlstr($raw_url),
$| = 1,          # Немедленный вывод следующей строки
printf %s =>\n\t , $url,
my $ua = LWP UserAgent->new(),
$ua->agent( Schmozilla/v9 14 Platinum ),
                Request->new(GET => $url),
               http //wizard yellowbrick
                   # Чтобы озадачить программы анализа журнала

if ($response->is_error()) {
     printf %s\n , $response->status_line,
 } else {
     my $count,
     my $bytes,

     $bytes = length $content,
     $count = ($content =~ tr/\n/\n/),
     printf %s (%d lines, %d bytes)\n , $response->title(), $count, $bytes, }
```

Программа выдает результаты следующего вида:

```
% titlebytes http //www tpj com/
http.//www.tpj.com/  =>
     The Perl Journal (109 lines, 4530 bytes)
```

Обратите внимание: вместо правильного английского используется вариант написания Ошибка была допущена разработчиками стандарта при выборе имени HTTP_REFERER.

Смотри также

Документация по модулю LWP::Simple с CPAN и страница руководства *lwpcook*(1), прилагаемая к LWP; документация по модулям LWP::UserAgent, HTTP::Request, HTTP::Response и URI::Heuristic; рецепт 20.2.

20.2. Автоматизация подачи формы

Проблема

Вы хотите передать сценарию CGI значения полей формы из своей программы.

Решение

Если значения передаются методом GET, создайте URL и закодируйте форму методом `query_form`:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use LWP::Simple;
use URI::URL;

my $url = url( http://www perl com/cgi-bin/cpan_mod ),
$url->query_form(module => 'DB_File ,
$content = get($url),
```

Если вы используете метод POST, создайте собственного пользовательского агента и закодируйте содержимое:

```
use HTTP::Request Common qw(POST),
use LWP UserAgent;

$ua = LWP UserAgent->new();
                    //www perl com/cgi-bin/cpan_mod',
                         "DB_File",
$content = $ua->request($req)->as_string,
```

Комментарий

Для простых операций хватает процедурного интерфейса модуля LWP::Simple. Для менее тривиальных ситуаций модуль LWP::UserAgent предоставляет объект виртуального броузера, работа с которым осуществляется посредством вызова методов.

Строка запроса имеет следующий формат:

```
ПОЛЕ1=ЗНАЧЕНИЕ1&ПОЛЕ2=ЗНАЧЕНИЕ2&ПОЛЕ3=ЗНАЧЕНИЕ3
```

В запросах GET информация кодируется в запрашиваемом URL:

```
http //www site com/path/to/
script cgi?field1=value1&field2=value2&field3=value3
```

Служебные символы в полях должны быть соответствующим образом преобразованы, поэтому присваивание параметру arg строки 'this isn t <EASY> and <FUN>' выглядит так:

```
http //www site com/path/to/
script.cgi?arg=%22this+isn%27t+%3CEASY%3E+%26+%3CFUN%3E%22
```

Метод query_form, вызываемый для объекта URL, оформляет служебные символы формы за вас. Кроме того, можно вызвать URI::Escape::url_escape или CGI::escape_html по собственной инициативе. В запросах POST строка параметров входит в тело HTML-документа, передаваемого сценарию CGI.

Для передачи данных в запросе GET можно использовать модуль LWP::Simle, однако для запросов POST не существует аналогичного интерфейса LWP::Simple. Вместо этого функция POST модуля HTTP::Request::Common создает правильно отформатированный запрос с оформлением всех служебных символов.

Если запрос должен проходить через прокси-сервер, сконструируйте своего пользовательского агента и прикажите ему использовать прокси:

```
$ua->proxy([ http , ftp ] => http //proxy myorg com 8081 ),
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Это означает, что запросы HTTP и FTP для данного пользовательского агента должны маршрутизироваться через прокси на порте 8081 по адресу *proxy.myorg.com*.

Смотри также

Документация по модулям LWP::Simple, LWP::UserAgent, HTTP::Request::Common, URI::Escape и URI::URL с CPAN; рецепт 20.1.

20.3. Извлечение URL

Проблема

Требуется извлечь все URL из HTML-файла.

Решение

Воспользуйтесь модулем HTML::LinkExtor из LWP:

```
use HTML .LinkExtor,

$parser = HTML::LinkExtor->new(undef, $base_url);
$parser->parse_file($filename);
@links = $parser->links;
        $linkarray (@links)
    my @element = @$linkarray,
    my $elt_type = shift @element,          # Тип элемента

    # Проверить, тот ли это элемент, который нас интересует
    while (@element) {
        # Извлечь следующий атрибут и его значение
        my ($attr_name, $attr_value) = splice(@element, 0, 2);
        # ... Сделать что-то ..
    }
}
```

Комментарий

Модуль HTML::LinkExtor можно использовать двумя способами: либо вызвать `links` для получения списка всех URL в документе после его полного разбора, либо передать ссылку на функцию в первом аргументе new. Указанная функция будет вызываться для каждого URL, найденного во время разбора документа.

Метод `links` очищает список ссылок, поэтому для каждого анализируемого документа он вызывается лишь один раз. Метод возвращает ссылку на массив элементов. Каждый элемент сам по себе представляет ссылку на массив, в начале которого находится объект HTML::Element, а далее следует список пар "имя атрибута/значение". Например, для следующего HTML-фрагмента:

```
<A HREF="http://www.perl.com/">Home page</A>
<IMG SRC="images/big.gif" LOWSRC='images/big-lowres gif">
```

возвращается структура данных:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
[
                                    perl com/  ],
  [ img, src     => images/big gif ,
         lowsrc =>  images/big-lowres gif  ]
]
```

В следующем фрагменте демонстрируется пример использования $elt_type и $attr_name:

```
if ($elt_type eq  a  && $attr_name eq  href ) { /
    print  ANCHOR  $attr_value\n
        if$attr_value->scheme =~/http|ftp/,
}
if ($elt_type eq  img  && $attr_name eq  src ) {
    print  IMAGE   $attr_value\n ,
}
```

Программа из примера 20.2 получает в качестве аргументов URL (например, *file:///tmp/testing.html* или *http://www.ora.com/*) и выдает в стандартный вывод отсортированный по алфавиту список уникальных ссылок из него.

Пример 20.2. xurl

```
#!/usr/bin/perl -w
Я xurl - получение отсортированного списка ссылок с URL
use HTML LinkExtor,
use LWP  Simple,

$base_url = shift,
$parser = HTML  LinkExtor->new(undef, $base_url),
$parser->parse(get($base_url))->eof,
@links = $parser->links,
        $linkarray (@links)
    my @element  = @$linkarray,
    my $elt_type = shift @element,
    while (@element) {
        my ($attr_name , $attr_value) = splice(@element, 0, 2),
        $seen<$attr_value}++,
    }
}
for (sort keys %seen) { print $_,  \n  }
```

У программы *xurl* имеется существенный недостаток: если в get или $base_url используется перенаправление, все ссылки будут рассматриваться для исходного, а не для перенаправленного URL. Возможное решение: получите документ с помощью LWP::UserAgent и проанализируйте код ответа, чтобы узнать, произошло ли перенаправление. Определив URL после перенаправления (если он есть), конструируйте объект HTML::LinkExtor.

Примерный результат выглядит так:

```
%xurl http //www perl com/CPAN
ftp://ftp@ftp.perl.com/CPAN/CPAN.html
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
http://language.perl.com/misc/CPAN.cgi
http://language.perl.com/misc/cpan_module
http://language.perl.com/misc/getcpan
http://language.perl.com/index.html
http://language.perl.cora/gif3/lcb.xbm
```

В почте и сообщениях Usenet часто встречаются вида:

< *URL:http://www.perl.com*>

Это упрощает выборку URL из сообщений:

```
@URLs = ($message =^ /<URL:(.*?)>/g);
```

Смотри также

Документация по модулям **LWP::Simple**, HTML::LinkExtor и HTML::Entities; рецепт 20.1.

20,4. Преобразование ASCII в HTML

Проблема

Требуется преобразовать ASCII-текст в HTML.

Решение

Воспользуйтесь простым кодирующим фильтром из примера 20.3.

Пример 20.3. text2html

```
#!/usr/bin/perl -w -p00
# te :t2html - простейшее html-кодирование обычного текста
8 -p означает, что сценарий применяется для каждой записи.
# -00 означает, что запись представляет собой абзац

use HTMl::Entities;
$_=encode_entities($_, "\200-\377");

if (/^\s/){
    # Абзацы, начинающиеся с пропусков, заключаются в <PRE>
    s{(.*)$}          {<PRE>\n$1</PRE>\n}s;      "Оставитьотступы
} else {
    s{^(>.*)}         {$1<BR>}gm;                # quoted text
    s{<URL:(.*?)>}    {<A HREF="$1">$1</A>}gs    # Внутренние URL(хорошо)
                      II
    s{(http:\S+)}     {<A HREF="$1">$1</A>}gs;   # Предполагаемые URL(плохо)
    s{\*(\S+)\*}      <<STRONG>$1</STRONG>}g;    # *Полужирный*
    s{\b_(\S+)\_\b}   {<EM>$1</EM>}g;            # _Курсив_
    s{^}              {<P>\n};                   " Добавить тег абзаца
}
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Комментарий

Задача преобразования произвольного текста в формат HTML не имеетобщего решения, поскольку существует много разных, конфликтующих друг с другом способов форматирования обычного текста. Чем больше вам известно о входных данных, тем лучше вы их отформатируете.

Например, если вы знаете, что исходным текстом будет почтовое сообщение, можно добавить следующий блок для форматирования почтовых заголовков:

```
BEGIN {
    print <TABLE> ,
    $_ = encode_entities(scalar o),
    s/\n\s+/ /g,   # Строки продолжения
    while ( /^(\S+? )\s*( *)$/gm ) {          # Анализ заголовков
        print <TR><TH ALIGN= LEFT >$1</TH><TD>$2</TD></TR>\n ,
    }
    print </TABLE><HR> ,
}
```

Смотри также

Документация по модулю HTML::Entities от CPAN.

20.5. Преобразование HTML в ASCII

Проблема

Требуется преобразовать HTML-файл в отформатированный ASCII-Текст.

Решение

Если у вас есть внешняя программа форматирования (например, *lynx*), воспользуйтесь ей:

```
$ascii = lynx -dump $filename ,
```

Если вы хотите сделать все в своей программе и не беспокоитесь отом, что таблицы и фреймы:

```
use HTML FormatText,
use HTML Parse,

$html = parse_htmlfile($filename),
$formatter = HTML FormatText->new(leftmargin => 0, rightmargin => 50),
$ascii = $formatter->format($html),
```

Комментарий

В обоих примерах предполагается, что HTML-текст находится в файле. Если он хранится в переменной, то для применения *lynx* необходимо записать его в файл. При работе с HTML::FormatText воспользуйтесь модулем HTML::TreeBuilder:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
use HTML::TreeBuilder;
use HTML::FormatText;

$html = HTML::TreeBuilder->new();
$html->parse($document);

$formatter = HTML::FormatText->new(leftmargin => 0, rightmargin => 50);

$ascii = $formatter->format($html);
```

Если вы используете Netscape, команда Save As с типом Text отлично справляется с таблицами.

Смотри также

Документация по модулям HTML::Parse, HTML::FormatText; man-страница *lynx*(1) вашей системы; рецепт 20.6.

20.6. Удаление тегов HTML

Проблема

Требуется удалить из строки теги HTML и оставить в ней обычный текст.

Решение

Следующее решение встречается часто, но работает неверно (за исключением простейшего HTML-кода):

```
($plain_text = $html_text) =~ s/<[^>]*>//gs;      # НЕВЕРНО
```

Правильный, но медленный и более сложный способ связан с применением модуля LWP:

```
use HTML::Parse;
use HTML::FormatText;
$plain_text = HTML::FormatText->new->format(parse_html($html_text));
```

Комментарий

Как всегда, поставленную задачу можно решить несколькими способами. Каждое решение пытается соблюдать баланс между скоростью и универсальностью. Для простейшего HTML-кода работает даже самая элементарная командная строка:

```
% perl -pe "s/<[^>]*>//g" ФАЙЛ
```

Однако это решение не подходит для файлов, в которых теги пересекают границы строк:

```
<IMG SRC = "foo.gif"
     ALT = "Flurp!">
```

Поэтому иногда встречается следующее решение:

```
% perl -0777 -pe "s/<[^>]*>//gs" ФАЙЛ
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

или его сценарный эквивалент:

```
{
    local $/,              # Временный режим чтения всего файла
    $html = <FILE>,
    $html =^ s/<[^>]*>//gs,
}
```

Но даже этот вариант работает лишь для самого примитивного HTML-кода, не содержащего никаких "изюминок". В частности, он пасует перед следующими примерами допустимого HTML-кода (не говоря о многих других):

```
<IMG SRC = foo gif ALT = A > B >

<!- <Комментарий> ->

<script>if (a<b && a>c)</script>

<# Просто данные #>

<![INCLUDE CDATA [ ********** ]]>
```

Проблемы возникают и в том случае, если комментарии HTML содержат другие теги:

```
<!- Раздел закомментирован
    <B>Меня не видно!</B>
->
```

Единственное надежное решение — использовать алгоритмы анализа HTML-кода из LWP. Эта методика продемонстрирована во втором фрагменте, приведенном в решении.

Чтобы сделать анализ более гибким, субклассируйте HTML::Parser от LWP и записывайте только найденные текстовые элементы:

```
package MyParser,
use HTML Parser,
use HTML Entities qw(decode_entities),

@ISA = qw(HTML Parser),

sub text {
    my($self, $text) = @_,
    print decode_entities($text),
}

package main,
MyParser->new->parse_file(*F),
```

Если вас интересуют лишь простые теги, не содержащие вложенных тегов, возможно, вам подойдет другое решение. Следующий пример извлекает название несложного HTML-документа:

```
($title) = ($html =~ m#<TITLE>\s*( *?)\s*</TITLE>#is),
```

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

Как говорилось выше, подход с регулярными выражениями имеет свои недостатки. В примере 20.4 показано более полное решение, в котором HTML-код обрабатывается с использованием LWP.

Пример 20.4. htitle

```
#!/usr/bin/perl
# htitle - Получить название HTML-документа для URL

die  usage  $0 url    \n  unless  @ARGV,
        LWP,                                                          ,

            (@ARGV)
    $ua = LWP  UserAgent->new(),
                              Request->new(GET  =>  $url)),
    print  $url    if @ARGV  > 1,
    if ($res->is_success) {
        print $res->title,  \n ,
    } else {
        print
 ,
    >
>
```

Приведем пример вывода:

```
% htitle http //www ora com
www.oreilly.com  -  Welcome  to  O'Reilly  &  Associates!

% htitle  http  //www perl  com/  http  //www perl  com/nullvoid
http://www.perl.com/:  The  www.perl.com  Home  Page
http://www.perl.com/nullvoid:  404  File  Not  Found
```

Смотри также

Документация по модулям HTML::Entities и LWP::UserAgent с CPAN; рецепт 20.5.

20.7. Поиск устаревших ссылок

Проблема

Требуется узнать, содержит ли документ устаревшие ссылки.

Решение

Воспользуйтесь методикой, описанной в рецепте 20.3, для получения всех ссылок и проверьте их существование функцией head модуля LWP::Simple.

Комментарий

Следующая программа является прикладным примером методики извлечения ссылок из HTML-документа. На этот раз мы не ограничиваемся простым выво-

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

дом ссылок и вызываем для нее функцию head модуля LWP::Simple. Метод HEAD получает метаданные удаленного документа и определяет его **статус,** не загружая самого документа. Если вызов закончился неудачно, значит, ссылка не работает, и мы выводим соответствующее сообщение.

Поскольку программа использует функцию get из LWP::Simple, она должна получать URL, а не имя файла. Если вы хотите поддерживать обе возможности, воспользуйтесь модулем URI::Heuristic (см. рецепт 20.1).

Пример 20.5. churl

```
#!/usr/bin/perl -w
# churl - проверка URL

use HTML::LinkExtor;
use LWP::Simple qw(get head);

$base_url = shift
    or die "usage: $0 <start_url>\n";
$parser = HTML::LinkExtor->new(undef, $base_url);
$parser->parse(get($base_url));
@links = $parser->links;
print "$base_url: \n";
        $linkarray (@links) {
    my @element = @$linkarray;
    my $elt_type = shift @element;
    while (@element) {
        my ($attr_name , $attr_value) = splice(@element, 0, 2);
        if ($attr_value->scheme =' /\b(ftp|https?|file)\b/) {
            print " $attr_value: ", head($attr_value)? "OK" : "BAD", "\n";
        }
    }
}
```

Для программы действуют те же ограничения, что и для программы, использующей **HTML::LinkExtor,** из рецепта 20.3.

Смотри также

Документация по модулям **HTML::LinkExtor,** LWP::Simple, LWP::UserAgent и **HTTP::Response** с CPAN; рецепт 20.8.

20.8. Поиск свежих ссылок

Проблема

Имеется список URL. Вы хотите **узнать,** какие из них изменялись позже других.

Решение

Программа из примера 20.6 читает URL из стандартного ввода, упорядочивает их по времени последней модификации и выводит в стандартный вывод с префиксами времени.

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

Пример 20.6. surl

```
#!/usr/bin/perl -w
# surl - сортировка URL по времени последней модификации

use LWP::UserAgent;
use HTTP::Request;
use URI::URL qw(url);

my($url, %Date);
my $ua = LWP::UserAgent->new();

while ( $url = <>) ) {
    my($req, $ans);
    next unless $url->scheme =~ /^(file|https?)$/;
    $ans = $ua->request(HTTP::Request->new( HEAD , $url));
    if ($ans->is_success) {
        $Date{$url} = $ans->last_modified || 0;  # unknown
    } else {
        print STOERR $url Error [ , $ans->code, ]  ,
            $ans->message, '\n ,
    }
}

        $url ( sort { $Date{$b} <=>$Date{$a} } keys %Date ) {
    printf %-25s %s\n , $Date{$url} ^ (scalar localtime $Date{$url})
                                      <NONE SPECIFIED> , $url,
}
```

Комментарий

Сценарий *surl* больше похож на традиционную программу-фильтр. Он построчно читает URL из стандартного ввода (на самом деле данные читаются из <ARGV>, что по умолчанию совпадает с STDIN при пустом массиве @ARGV). Время последней модификации каждого URL извлекается с помощью запроса HEAD. Время сохраняется в хэше, где ключами являются URL. Затем простейшая сортировка хэша по значению упорядочивает URL по времени. При выводе внутренний формат времени преобразуется в формат localtime.

В следующем примере программа *xurl* из предыдущего рецепта извлекает список URL, после чего выходные данные этой программы передаются на вход *surl*.

```
%xurl http://www.perl.com/ | surl | head
Mon Apr 20 06:16:02 1998   http://electriclichen.com/linux/srom.html
Fri Apr 17 13:38:51 1998   http://www.oreilly.com/
Fri Mar 13 12:16:47 1998   http://www2.binevolve.com/
Sun Mar 8 21:01:27 1998    http://www.perl.org/
Tue Nov 18 13:41:32 1997   http://www.perl.com/universal/header.map
Wed Oct 1 12:55:13 1997    http://www.songline.com/
Sun Aug 17 21:43:51 1997   http://www.perl.com/graphics/perlhome_header.jpg
Sun Aug 17 21:43:47 1997   http://www.perl.com/graphics/perl_id_313c.gif
Sun Aug 17 21:43:46 1997   http://www.perl.com/graphics/ora_logo.gif
Sun Aug 17 21:43:44 1997   http://www.perl.com/graphics/header-nav.gif
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Маленькие программы, которые выполняют свою узкую задачу и могут объединяться в более мощные конструкции, — верный признак хорошего программирования. Более того, можно было бы заставить *xurl* работать с файлами и организовать фактическую выборку содержимого URL в Web другой программой, которая бы передавала свои результаты *xurl, churl* или *surl*. Вероятно, эту программу следовало бы назвать *gurl*, но программа с таким именем уже существует: в комплекс модулей LWP входит программа с синонимами HEAD, GET и POST для выполнения этих операций в сценариях командного интерпретатора.

Смотри также

Документация по модулям LWP::UserAgent, HTTP::Request и URI::URL с CPAN; рецепт 20.7.

20.9. Создание шаблонов HTML

Проблема

Вы хотите сохранить параметризованный шаблон во внешнем файле, прочитать его в сценарий CGI и подставить собственные переменные вместо заполнителей, находящихся в тексте. Это позволяет отделить программу от статических частей документа.

Решение

Если вы ограничиваетесь заменой ссылок на переменные, используйте функцию template:

```
sub template {
    my ($filename, $fillings) = @_,
    my $text,
    local $/,                    # Режим поглощающего ввода (undef)
    local *F,                   И Создать локальный манипулятор
    open(F, < $filename )       ||
    $text = <F>,                # Прочитать весь файл
    close(F),                   # Игнорировать код возврата
    # Заменить конструкции %%   %% значениями из хэша %$fillings
    $text =^ s{ %% ( *? ) %% }
              { exists( $fillings->{$1} )
                     ^ $fillings->{$1}

              }gsex,
          $text,
}
```

В этом случае используемые данные выглядят так:

```
<!- simple template для внутренней функции template() ->
<HTML><HEAD><TITLE>Report for %%username%%</TITLE></HEAD>
<BODY><H1>Report for %%username%%</H1>
%%username%% logged inft%count%%times, for a total of %%total%% minutes
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Для расширения полноценных выражений используйте модуль Text::Template с CPAN, если вы можете гарантировать защиту данных от постороннего вмешательства. Файл данных для Text::Template выглядит так:

```
<!- fancy.template for Text::Template ->
<HTML><HEAD><TITLE>Report for {$user}</TITLE></HEAD>
<BODY><H1>Report for {$user}</H1>
{ lcfirst($user) } logged in {$count} times, for a total of
{ int($seconds / 60) } minutes.
```

Комментарий

Параметризованный ввод в сценариях CGI хорош по многим причинам. Отделение программы от данных дает возможность другим людям (например, **дизайнерам**) изменять код HTML, не трогая программы. Еще лучше **то,** что две программы могут работать с одним **шаблоном,** поэтому стилевые изменения шаблона **немедленно** отразятся на обеих программах.

Предположим, вы сохранили в файле первый шаблон из решения. Ваша программа CGI содержит определение функции template (см. выше) и соответствующим образом задает значения переменных **$whats_his_name,** $login_count и **$minute_used.** Шаблон заполняется просто:

```
%fields = (
            username => $whats_his_name,
            count    => $login_count,
            total    => $minute_used,
);
print template("/home/httpd/templates/simple.template", \%fields);
```

Файл шаблона содержит ключевые слова, окруженные двойными символами % **(%%КЛЮЧЕВОЕ-СЛОВО%%).** Ключевые слова ищутся в хэше **%$fillings,** ссылка на который передается **template** в качестве второго аргумента. В примере 20.7 приведен более близкий к реальности пример, использующий базу данных **SQL.**

Пример 20.7.

```
#!/usr/bin/perl -w
            вывод данных о продолжительности работы пользователей
в с применением базы данных SOL

use DBI;
use CGI qw(:standard);

# Функция template() определена в решении (см. выше)
$user = param("username")                  or die "No username";

$dbh = DBI->connect("dbi:mysql:connections:mysql.domain.com:3306",
    "connections", "seekritpassword")      or die "Couldn't connect\n";
$sth = $dbh->prepare(<<"END_OF_SELECT")    or die "Couldn't         SQL";
    SELECT COUNT(duration),SUM(duration)
    FROM logins WHERE username='$user'
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
END_OF_SELECT

if (@row = $sth->fetchrow()) {
    ($count, $seconds) = @row;
} else {
    ($count, $seconds) = (0,0);
}

$sth->finish();
$dbh->disconnect;

print header();
print template("report.tpl", {
    'username' => $user,
    'count'    => $oount,
    'total'    => $total
});
```

Если вам потребуется более изощренное и гибкое решение, рассмотрите второй шаблон решения, основанный на модуле Text:Template с CPAN. Содержимое пар фигурных скобок, обнаруженных в файле шаблона, вычисляется как код Perl. Как правило, расширение сводится к простой подстановке переменных:

```
You owe: {$total}
```

но в фигурных скобках также могут находиться полноценные выражения:

```
The average was {$count ? ($total/$count) : 0}.
```

Возможное применение этого шаблона продемонстрировано в примере 20.8.

```
#!/usr/bin/perl -w
            вывод данных о продолжительности работы пользователей
# с применением базы данных SQL

use Text::Template;
use DBI;
use CGI qw(:standard);

$tmpl = "/home/httpd/templates/fancy.template";
$template = Text::Template->new(-type => "file", -source => $tmpl);
$user = param("username")                    or die "No username";

$dbh = DBI->connect("dbi:mysql:connections:mysql.domain.com:3306",
    "connections",        passwd")            "Couldn't db connect\n";
$sth = $dbh->prepare(<<"END_OF_SELECT") or die "Couldn't          SQL";
    SELECT COUNT(duration),SUM(duration)
    FROM logins WHERE username='$user'
END_OF_SELECT

$sth->execute()             or die "Couldn't execute SQL";
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
if (@row = $sth->fetchrow()) {
    ($count, $total) = @row;
} else {
    $count = $total = 0;
}

$sth->finish();
$dbh->disconnect;

print header();
print $template->fill_in();
```

При более широких возможностях этого подхода возникают определенные проблемы безопасности. Любой, кому разрешена запись в файл шаблона, сможет вставить в него код, выполняемый вашей программой. В рецепте 8.17 рассказано о том, как снизить этот риск.

Смотри также

Документация по модулю Text::Template с CPAN; рецепты **8.16; 14.10.**

20.10. Зеркальное копирование Web-страниц

Проблема

Вы хотите поддерживать локальную копию Web-страницы.

Решение

Воспользуйтесь функцией `mirror` модуля LWP::Simple:

```
use LWP Simple;
mirror($URL, $local_filename);
```

Комментарий

Несмотря на тесную связь с функцией get, описанной в рецепте 20.1, функция `mirror` не выполняет безусловной загрузки файла. В создаваемый ей запрос GET включается заголовок `If-Modified-Since`, чтобы сервер передавал лишь недавно обновленные файлы.

Функция `mirror` копирует только одну страницу, а не целое дерево. Для копирования набора страниц следует использовать ее в сочетании с рецептом **20.3.** Хороший вариант зеркального копирования целого удаленного дерева приведен в программе *w3mir*, также находящейся на CPAN.

Будьте осторожны! Можно (и даже просто) написать программу, которая сходит с ума и начинает перекачивать все Web-страницы подряд. Это не только дурной тон, но и бесконечный труд, поскольку некоторые страницы генерируются

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

динамически. Кроме того, у вас могут возникнуть неприятности с теми, кто не желает, чтобы их страницы загружались *en masse*.

Смотри также

Документация по модулю LWP::Simple с CPAN; спецификация HTTP по адресу *http://www.w3.or/pub/WWW/Protocols/HTTP/*.

20.11. Создание робота

Проблема

Требуется написать сценарий, который самостоятельно работает в Web (то есть робота), При этом желательно уважать правила работы удаленных узлов.

Решение

Вместо модуля LWP::UserAgent используйте в роботе модуль LWP::RobotUA:

```
use LWP:'RobotUA;
$ua = LWP:'RobotUA->new('websnuffler/0.1', 'me@wherever.com');
```

Комментарий

Чтообы жадные роботы не перегружали серверы, на узлах рекомендуется создавать файл с правилами доступа *robots.txt*. Если ваш сценарий получает лишь один документ, ничего страшного, но при получении множества документов с одного сервера вы легко перекроете пропускную способность узла.

Создавая собственные сценарии для работы в Web, важно помнить о правилах хорошего тона. Во-первых, не следует слишком часто запрашиватьдокументы с одного сервера. Во-вторых, соблюдайте правила, описанные в файле *robots.txt*.

Самый простой выход заключается в создании агентов с применением модуля LWP::RobotUA вместо LWP::UserAgent. Этот агент автоматически "снижаетобороты" при многократных обращениях к одному серверу. Кроме того, он просматривает файл *robots.txt* каждого узла и проверяет, не пытаетесь ли вы принять файл, размер которого превышает максимально допустимый. В этом случае возвращается ответ вида:

```
403 (Forbidden) Forbidden by robots.txt
```

Следующий пример файла *robots.txt* получен программой *GET,* входящей в комплекс модулей LWP:

```
% GET http://www.webtechniques.com/robots txt
User-agent: *
      Disallow: /stats
      Disallow: /db
      Disallow: /logs

      Disallow: /forms
      Disallow: /gifs
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
Disallow:  /wais-src
Disallow:  /scripts
Disallow:  /config
```

Более интересный и содержательный пример находится по адресу *http://www.cnn.com/robots.txt*. Этот файл настолько велик, что его даже держат под контролем RCS!

```
% GET http //www.cnn  com/robots txt | head
# robots,  scram
ff $I d  :  robots.txt,v 1.2  1998/03/10  18:27:01 mreed Exp $
User-agent:  *
Disallow:  /
  |
User-agent:        Mozilla/3.01  (hotwired-test/0.1)
Disallow:     /ogi-bin
Disallow:     /TRANSCRIPTS
Disallow:     /development
```

Смотри также

Документация по модулю LWP::RobotUA(3) с CPAN; описание правил хорошего тона для роботов по адресу *http://info.webcrawler.com/mak/projects/robots/robot.html*.

20.12. Анализ файла журнала Web-сервера

Проблема

Вы хотите извлечь из файла журнала Web-сервера лишь интересующую вас информацию.

Решение

Разберите содержимое файла журнала следующим образом:

```
while (<LOGFILE>) {
  my ($client, $identuser, $authuser, $date, $time, $tz, $method,
      $url, $protocol, $status, $bytes) =
  /^(\S+) (\S+) (\S+) \[([^.]+)·(\d+.\d+:\d+) ([^\]]+) "(\S+) (.*?) (\S+)"
      (\S+) (\S+)$/;
  #  .
}
```

Комментарий

Приведенное выше регулярное выражение разбирает записи формата Common Log Format — неформального стандарта, которого придерживается большинство Web-серверов. Поля имеют следующий смысл:

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

client
IP-адрес или имя домена для броузера.

identuser
Результаты команды IDENT (RFC 1413), если она использовалась.

authuser
Имя пользователя при аутентификации по схеме "имя/пароль".

date
Дата поступления запроса (01/Mar/1997).

time
Время поступления запроса (12:5536).

tz
Часовой пояс (-0700).

method
Метод запроса: GET, POST, PUT.

url
Запрашиваемый ukL (/~user/index.html).

protocol
HTTP/1.0 или HTTP/1.1.

status
Возвращаемый статус (200 — все в порядке, 500 — ошибка сервера).

bytes
Количество возвращаемых байт (может быть равно "-" для ошибок, перенаправлений и операций, не сопровождаемых пересылкой документа).

В другие форматы также включаются данные о внешней ссылке и агенте. Ценой минимальных изменений можно заставить этот шаблон работать с другим форматом журнала. Обратите внимание: пробелы в URL *не оформляются* служебными символами. Это означает, что для извлечения URL нельзя использовать \S* — . * заставит регулярное выражение совпасть с целой строкой, а затем возвращаться до тех пор, пока не будет найдено соответствие для остатка шаблона. Мы используем . *? и фиксируем шаблон в конце строки с помощью $, чтобы механизм поиска не устанавливал совпадения и последовательно добавлял символы до тех пор, пока не будет найдено совпадение для всего шаблона.

Смотри также

Спецификация CLF по адресу *http://www.w3.org/Daemon/User/Config/Logging.html.*

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

20.13. Обработка серверных журналов

Проблема

Требуется обобщить данные в серверном журнале, но у вас нет специальной программы с возможностью настройки параметров.

Решение

Анализируйте журнал с помощью регулярных выражений или воспользуйтесь модулями Logfile с CPAN.

Комментарий

В примере 20.9 приведен образец генератора отчетов для журнала Apache.

Пример 20.9. sumwww

```
#!/usr/bin/perl -w
# sumwww - обобщение данных об операциях Web-сервера

$lastdate =
daily_logs(),
summary(),
exit,

# Читать файлы CLF и запоминать обращения с хоста и на URL
sub daily_logs {
    while (<>) {
        ($type, $what) = / (GET|POST)\s+(\S+?) \S+ / or next,
        ($host, undef, undef $datetime) = split,
        ($bytes) = /\s(\d+)\s"$/ or next,
        ($date)  =  ($datetime =~ /\[([^ ]•)/),
        $posts  += ($type eq POST),
        $home++ if m, / ,,
        if ($date ne $ lastdate) {
            if ($lastdate) { write_report()     }
            else           { $lastdate = $date  }
        }
        $count++,
        $hosts{$host}++,
        $what{$what}++,
        $bytesum += $bytes,
    >
                    if
}

й Ускорить копирование за счет создания синонимов
# глобальных переменных вида *typeglob
sub summary  {
    $lastdate =  Grand Total ,
    *count   = *sumcount,
```

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг


```
    *bytesum = *bytesumsum,
    *hosts   = *allhosts,
    *posts   = *allposts,
    *what    = *allwhat,
    *home    = *allhome,
    write,
}

# Вывести сведения по хостам и URL с применением специального формата
sub write_report {
    write,

    # add to summary data
    $lastdate     = $date,
    $sumcount    += $count,
    $bytesumsum += $bytesum,
    $allposts    += $posts,
    $allhome     += $home,

    # Сбросить данные за день
    $posts = $count = $bytesum = $home = 0,
    @allwhat{keys %what}   = keys %what,
    @allhosts{keys %hosts} = keys %hosts,
    %hosts = %what = (),
}

format STDOUT_TOP =
@||||||||||  @||||||  @|||||||  @|||||||  @||||||  @||||||  @|||||||||||||  .
 Date ,      Hosts ,  Accesses ,  Unidocs ,  POST ,  Home ,   Bytes



format STDOUT =
@>>>>>>>>>>> @>>>>>> @""">" @""">" @>>>>>> @>>>>>> @"""""""">"
$lastdate,   scalar(keys %hosts),
             $count, scalar(keys %what),
                              $posts,  $home,  $bytesum
```

Пример вывода выглядит так:

Date	Hosts	Accesses	Unidocs	POST	Home	Bytes
19/May/1998	353	6447	3074	352	51	16058246
20/May/1998	1938	23868	4288	972	350	61879643
21/May/1998	1775	27872	6596	1064	376	64613798
22/May/1998	1680	21402	4467	735	285	52437374
23/May/1998	1128	21260	4944	592	186	55623059
Grand Total	6050	100849	10090	3715	1248	250612120

Модуль Logfile::Apache с CPAN (см. пример 20.10) позволяет написать аналогичную, но менее специализированную программу. Этот модуль распространяет-

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

ся вместе с другимимодулями Logfile в единой поставке Logfile (на момент написания книги — *Logfile-0.115.tar.gz*).

Пример 20.10.

```
#!/usr/bin/perl -w
            отчет по журналам Apache

use Logfile::Apache;

$1 = Logfile::Apache->new(
    File  => "-",                    # STDIN
    Group => [ Domain, File ]);

$l->report(Group  => Domain,  Sort  => Records);
                  => File,    List  => [Bytes,Records]);
```

Конструктор new читает файл журнала и строит индексы. В параметре File передается имя файла, а в параметре Group — индексируемые поля. Возможные значения — Date (дата), Hour (время получения запроса), File (запрашиваемый файл), User (имя пользователя, извлеченное из запроса), Host (имя хоста, запросившего документ) и Domain (Host, преобразованный в строку типа "France", "Germany" и т. д.).

Вывод отчета в STDOUT осуществляется методом В параметре Group передается используемый индекс, а также дополнительно — способ сортировки (Records — поколичеству обращений, Bytes — поколичеству переданных байт) и способ дальнейшей группировки данных (по количеству байт или количеству обращений).

Приведем примеры вывода:

```
Domain

US Commercial          222  38.47%
US Educational         115  19.93%
Network                 93  16.12%
Unresolved              54   9.36%
Australia               48   8.32%
Canada                  20   3.47X
Mexico                   8  1.39X
United Kingdom           6   1. 04%

File                                  Bytes          Records

/                                   13008  0.89X          6  1.04X
/cgi-bin/MxScreen                   11870  0.81X          2  0.35X
/cgi-bin/pickcards                  39431  2.70X         48  8.32X
/deckmaster                        143793  9.83X         21  3.64X
/deckmaster/admin                   54447  3.72X          3  0.52X
```

Смотри также

Документация по модулю Logfile::Apache с CPAN; *perlform*(1).

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

20.14. Программа: htmlsub

Следующая программа выполняет подстановку в HTML-файле так, что изменения происходят только в обычном текст. Предположим, у вас имеется файл *index.html* следующего содержания:

```
<HTML><HEAD><TITLE>Hi!</TITLE></HEAD><BODY>
<H1>Welcome to Scooby World!</H1>
I have <A HREF="pictures.html">pictures</A> of the crazy dog
himself.
<IMG SRC="scooby.jpg" ALT="Good doggy!"><P>
<BLINK>He's my hero!</BLINK>  I would like to meet him some day,
                    taken with him.<P>
P.S. I am deathly ill.  <A HREF="shergold.html">Please send
cards</A>.
</BODY></HTML>
```

Программа ***htmlsub*** заменяет каждый экземпляр слова документа на "photo". Новый документ выводится в STDOUT:

```
% htmlsub        photo scooby.html
<HTML><HEAD><TITLE>Hi!</TITLE></HEAD><BODY>
<H1>Welcome to Scooby World!</H1>
I have <A HREF="pictures.html">photos</A> of the orazy dog
himself.  Here's one!<P>
<IMG SRC="scooby.jpg" ALT="Good doggy!"><P>
<BLINK>He's my hero!</BLINK>  1 would like to meet him some day,
and get my photo taken with him.<P>
P.S. I am deathly ill.  <A HREF="shergold.html">Please send
cards</A>.
</BODY></HTHL>
```

Исходный текст программы приведен в примере 20.11.

Пример 20.11. htmlsub

```
#!/usr/bin/perl -w
# htmlsub - замена обычного текста в HTML-файле
# Автор - Джайсл Аас <gisle@aas.no>

sub usage { die "Usage: $0 <from> <to> <file>...\n" }

my $from = shift or usage;
my $to   = shift or usage;
usage unless @ARGV;

# Субклассировать HTML::Filter для выполнения подстановок.

package MyFilter;
        HTML::Filter;
@ISA=qw(HTML::Filter);
```

продолжение ⇨

The electronic books BOOKS-SHOP.com Книжный магазин электронных книг

Пример 20.11 (продолжение)

```
use HTML Entities qw(decode_entities encode_entities);

sub text
{
   my $self = shift;
   my $text = decode_entities($_[0]);
   $text =~ s/\Q$from/$to/go;      # Самая важная строка
   $self->SUPER::text(encode entities($text));
}

# Now use the class

package main;
foreach (@ARGV) {
    MyFilter->new->parse_file($_);
}
```

20.15. Программа:

Программа выполняет подстановки в HTML-файлах так, что изменения относятся только к тексту в полях HREF тегов <A HREF=" ">. Например, если в предыдущем примере *scooby.html* файл *shergold.html* был переименован в *cards.html*, достаточно сказать:

```
% hrefsub shergold html cards html scooby html
<HTML><HEAD><TITLE>Hi!</TITLE></HEAD><BODY>
<H1>Welcome to Scooby World!</H1>
I have <A HREF="pictures.html">pictures</A> of the crazy dog
himself.          one!<P>
<IMG SRC="scooby.jpg" ALT="Good doggy!"><P>
<BLINK>He's roy hero!</BLINK> I would like to meet him some day,
    get                  with him.<P>
P.S. I am de athly ill.  <a href="cards.html">Please send
cards</A>.
</BODY></HTML>
```

В странице руководства HTML::Filter есть раздел BUGS, в котором сказано:

"Комментарии в объявлениях удаляются, а затем вставляются в виде отдельных комментариев после объявления. Если включить strict_comment(), то комментарии с внутренними '-\|-" делятся на несколько комментариев".

Данная версия при выполнении подстановки всегда преобразует <a> и имена атрибутов в теге в нижний регистр. Если строка $foo содержит несколько слов, то текст, передаваемый MyFilter->text, может быть разбит так, что эти слова разделятся и подстановка не сработает. Вероятно, в HTML::Parser следует предусмотреть новый параметр, чтобы текст возвращался лишь после чтения всего сегмента. Кроме того, кое-кто не любит, когда 8-битные символы кодировки Latin-1 замещаются уродливыми эквивалентами, поэтому *hrefsub* справляется и с этой проблемой.

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

Алфавитный указатель

Символы

$', переменная, 184
$+, переменная, 184
$/, переменная, 255
$^W, переменная, 465
$_, переменная, 490
$|, переменная, 242
%, хэши, 151
&&, оператор, 32
-, 377,451
.. и ..., операторы, 197
/ (корневой каталог), 324
:символ, 362
:служебные символы, 53
<, 241,253
<&= и <&, режимы открытия, 277
<=>, 137
=, 152
=-, оператор, 30
>, 92
*, при сохранении манипуляторов, 240, 272
', qx(), 116
', расширение, 46
“, оператор, 560
@_, массив, 349
||, оператор, 31
||=, оператор, 31
-, расширение в именах файлов, 248

A

accept(), 605,609
alarm(), 598
AND, в регулярных выражениях, 216
Apache
  mod_perl, модуль, 680
  журналы, 724
atan2(), 82
atime, поле, 324
autoflush(), 265
AUTOLOAD, механизм, 371
AutoLoader, модуль, 428

B

B-деревья, 506
basename(), 342
Berkeley DB, библиотека, 498
bind(), 605
binmode(), 307
bless(), 450,473

C

C, написание модулей, 436
cacheout(), 275
caller(), 355
can(), 470
Carp, модуль, 430

ceil(), 72
CGI, программирование
  методы HTTP, 671
  многостраничные сценарии, 693
  общие сведения, 667
CGI.pm, модуль, 667
CGI::Carp, модуль, 672
chr(), 34
Class::Struct, модуль, 464
close(), 241
closedir(), 327
cmp(), 138
color() (Term::ANSIColor), 529
(Term::ANSIColor),
Common Log Format, стандарт, 722
confess(), 430
Config, модуль, 586
copy(), 330
cos(), 82
CPAN
  общие сведения, 410
  построение и установка модулей, 442
croak(), 430
ctime, поле, 324
Curses, модуль, 538
Cwd, модуль, 429
сборка мусора, 449

D

Data::Dumper, модуль, 397
DateCalc(), 110
Day_of_week(), 103
Day_of_Yearweek(), 103
DBD, модуль, 515
DBI, модуль, 515
DBM, библиотеки, 497

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

DBM, файлы
блокировка, 504
преобразование, 502
сортировка, 506
хранение сложных данных, 511
dbmclose(), 499
dbmopen(), 499
dclone(), 398
delete(), 156
die, функция, 364
dirname(), 342
DNS, поиск, 644
do(), 314

E

each(), 167
eval(), 413
exec(), 557,682
exists(), 154
Expect, модуль, 540
Exporter, модуль, 406

F

fcntl(), 264,361
fdopen()(IO::Handle), 277
FETCH(), 488
FIFO, 581
File::Basename, модуль, 342
File::Copy, модуль, 330
File::Find, модуль, 336
File::KGlob, модуль, 335
File::LockDir, модуль, 280
File::stat, модуль, 299
FileCache, модуль, 275
FileHandler, модуль, 240
fileparse(), 342
FindBin(), модуль, 422
finddepth(), 339
, flock(), 262,283
< floor(), 72

fork()
закрытие сокета, 626
зомби, 594
неразветвляющие серверы, 629
общие сведения, 557, 573, 682
разветвляющие серверы, 625
FTP, клиенты, 647

G

GET, метод, 671
get(LWP::Simple), 704
gethostbyaddr(), 620
gethostbyname(), 621
getopt()(Getopt::Std), 523
GetOptions()(Getopt::Long), 524
getopts()(Getopt::Std), 524
getsockopt(), 633
GetTerminalSize(), 528
gmtime(), 95

H

h2ph, утилита, 434
h2xs, утилита, 425
HEAD, метод, 668
head()(LWP::Simple), 714
hex(), 87
hidden(), 694
HotKey, модуль, 533
HTML, формы, 669
HTML::FormatText, модуль, 711
HTML::LinkExtor, модуль, 708
HTML::TreeBuilder, модуль, 711

I

If-Modified-Since, заголовок, 720
inet_ntoa(), 610
int(), 72
IO::File, модуль, 240, 243
IO::Pty, модуль, 541
IO::Select, модуль, 268
IO::Socket, модуль, 604, 614
IO::Stty, модуль, 541
ioctl(), 270,361
IPC::Open2, модуль, 571
IPC::Open3, модуль, 573
IPC::Shareable, модуль, 584
isa()(UNIVERSAL), 471

K

keys(), 156,158,168
kill, команда, 587

L

last(), 134
lc(), 43
lcfirst(), 44
Lingua::EN, 90
listen(), 605
local(), 366
localtime(), 95
Logfile::Apache, 725

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

logn()(Math::Complex), 84
LWP, модули, 690

M

m//, оператор, 30
Mail::Mailer, модуль, 650
map(), 141
Math::Complex, модуль, 84
Math::Random, модуль, 78
Math::TrulyRandom, модуль, 78
mirror()(LWP::Simple), 720
MLDBM, модуль, 401, 511
mod_perl, модуль, 680
mtime, поле, 324
my, ключевое слово, 351

N

Net::DNS, модуль, 646
Net::FTP, модуль, 647
Net::NNTP, модуль, 654
Net::Ping, модуль, 660
Net::Telnet, модуль, 659
Net::Whois, модуль, 662
new(), 451
NOFILE,КОНСТаНТа, 275
NOT, в регулярных выражениях, 216

O

oct(), 87
open(), 241
opendir(), 335
OR, в регулярных выражениях, 216
ord(), 34
output(), 175

P

pack(), 35,309
ParseDate(), 104
PDL, модули, 84
pipe(), 557
places(), 485
pod, документация, 439
pop(), 143
POP3, серверы, 656
POSIX, модуль, 72,592
POST, метод, 671
print(), 605
printf(), 35
push(), 131,383

Q

q(), 116
qq(), 116
qw(), 116

R

rand(), 76
read(), 269,309

retrieve() (Storable), 399

rmdir(), 340

S

s///, оператор, 30
seek(), 269,292
seekdir(), 334
select(), 109
SelfLoader, модуль, 427
setsockopt(), 633
shift(), 133
shutdown(), 622
SIGALRM, сигнал, 558
SIGCHLD, сигнал, 558
SIGHUP, сигнал, 558
SIGINT, сигнал, 558
SIGPIPE, сигнал, 558
SIGQUIT, сигнал, 558
SIGTERM, сигнал, 558
SIGUSR1 и SIGUSR2, сигналы, 558
sin(), 82
SOCK_, константы, 604
sockaddr_in(), 615
sockaddr_un(), 618
socket(), 605
sort(), 138
SQL, базы данных, 718
srand(), 77
stat(), 316
STDERR, манипулятор, 240
STDIN, манипулятор, 240
STDOUT, манипулятор, 240
STORE(), 488
struct()(Class::Struct), 465
sysread(), 269
sysseek(), 269
system(), 560

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

T

TCP, протокол, 606
tell(), 269
telldir(), 334
Term::ANSIColor, модуль, 529
Term::Cap, модуль, 526
Term::ReadKey, модуль, 52
Term::ReadLine, модуль, 536
Term::ReadLine, 536
tie(), 488
Tie::IxHash, модуль, 161
timegm(), 96
timelocal(), 96
Tk, пакет, 522
tr///, оператор, 30, 44
truncate(), 291

U

uc(), 43
ucfirst(), 44
UDP, протокол, 615
umask(), 246
uname(), 621
Unicode, 28
unlink(), 329
unpack(), 35
unshift(), 143
untie(), 499
URI::Heuristic, 715
URL, 667
utime(), 328

W

wait(), 595
waitpid(), 595
wantarray(), 355
Week_Number(), 103
whois, служба, 662

А

анонимные данные, 378
аргументы
  общие сведения, 349
  передача в именованных параметрах, 358
  передача по ссылке, 362
  прототипы функций, 362
асинхронный ввод/вывод, 267
атрибуты, 454, 476

Б

базовый класс, 454, 473
базы данных
  запросы, 685
  преобразование DBM-файлов, 502
  сортировка, 506
  устойчивые данные, 513
  хранение сложных данных, 511
библиотеки, 409
бинарные деревья, 402
блокировка сигналов, 596, 597
брандмауэры, 638

В

визуальный сигнал, 531
вложенные подпрограммы, 372
вложенные теги HTML, 713
временные файлы, 250
встроенные документы, 27
выражения, интерполяция, 46

Г

глобальные величины, сохранение, 36
глубокое копирование, 398

Д

данные классов, 463
данные объектов, 463
дата и время
  Date, 101,103,110
  вычисления, 97
  общие сведения, 94
  таймеры, 107
датаграммные сокеты, 604, 614
двоичные файлы, 307, 309
двоичные числа, 72
двусторонние клиенты, 623
демоны, 635
дескрипторы, файловые, 277
деструкторы, 452, 480
динамическая область действия, 368

Ж

журнал, Web-сервера, 668

З

замыкания, 387

И

идемпотентность, 668
именованные каналы, 581

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

индексные узлы, 323
исключения
обработка, 364
перехват, 371
истинность, логическая, 27

К

каскадные команды, 544
каталоги
копирование файлов, 330
модулей, 423
общие сведения, 323
переименование файлов, 340
сортировка, 334
удаление файлов, 339
классы
генерация методов с помощью AUTOLOAD, 475
доступ к переопределенным методам, 474
как структуры, 465
наследование, 449
общие сведения, 449
клиенты
FTP, 647
TCP, 606
UDP, 614
двусторонние, 623
комментарии
в документации pod, 441
в регулярных выражениях, 188
комплексные числа, 84
конструкторы, 451,474
конфигурационные файлы, 314
копирование
манипуляторов, 279
структур данных, 398
файлов, 330

Л

логарифмы, 83
локальный контекст, 207

М

манипуляторы
копирование, 279
кэширование, 275
общие сведения, 239
связанные, 488
сохранение, 240
маска доступа, 246
массивы
анонимные, 378
изменение размера, 119
инициализация, 115
многомерные, 115
общиесведения, 114
объединение, 129
/переборэлементов, 121
последнийиндекс, 119
случайные перестановки, 144
сортировка, 138
сравнение со списками, 114
умножение матриц, 84
хэши массивов, 380
методы, 449
методы класса, 451
многомерные массивы, 115
модули
AUTOLOADER, 428
SelfLoader, 427
загрузка, 414
написание на С, 436
общие сведения, 405
проектирование интерфейса, 406
монопольная блокировка, 262

Н

натуральные логарифмы, 83
Нейгла, алгоритм, 612
неразветвляющиеся серверы, 629
неформальный поиск, 209

О

объединение массивов, 129
объекты
конструирование, 451
общие сведения, 449
определенность, 27
ошибка, признак, 361

П

пакеты, 386
общие сведения, 405
перегрузка операторов, 482
переменные, 40
поверхностное копирование, 398
построение модулей от CPAN, 442
потоковые сокеты, 604

The electronic books
BOOKS-SHOP.com
Книжный магазин
электронных книг

преобразование
ASCII-символов и кодов, 34
DBM-файлов, 502
между ASCIIиHTML 710
системы счисления, 72
формат pod, 440
приведение, 450
производные классы, 473
прототипы, 362
прямые ссылки, 327

Р

регулярные выражения
комментарии, 188
неформальный поиск, 57
общие сведения, 89
список, 236

С

сглаживание списков, 131
синонимы
для манипуляторов, 278
для элементов списков, 123
скалярные переменные, 26
сокеты
TCP, 611
UDP, 614
UNIX, 604
датаграммные, 604
двусторонние клиенты, 623
идентификация компьютеров, 620
Интернета, 604
общие сведения, 604
потоковые, 604
соединение через брандмауэр, 638
списки
инициализация, 115
общиесведения, 114
сглаживание, 131
циклические, 143
списковый контекст, 363
сравнение
вещественных чисел, 68
ключей в хэшах, 168
ссылки
на массивы, 378
на скаляры, 377
на функции, 385
на хэши, 377
общие сведения, 376
строки
общие сведения, 26
поиск по шаблону, 184
поиск слов, 187
последовательная обработка символов, 36
преобразование регистра, 44
форматирование абзацев, 51
субъекты, 376

Т

текстовые файлы, 307

У

условная блокировка, 262

Х

хэши
анонимные, 378
добавление элементов, 153
инвертирование, 164
инициализация, 152
массивов, 380
общие сведения, 123
объединение, 166
сортировка, 165
срезы, 123
удаление элементов, 156

Ц

циклические списки, 143
циклические структуры данных, 479