Глава 6. Операции ввода/вывода

Учебник по Perl

Содержание [скрыть]

Ни одна программа не может функционировать сама по себе, не получая и не посылая информацию во внешнюю среду. Perl предоставляет несколько способов получения программой данных извне и вывода информации из выполняющегося сценария. В процессе функционирования программы может потребоваться выполнить некоторую команду операционной системы и проанализировать результаты ее выполнения, прочитать данные из внешнего файла или группы файлов, записать результаты вычислений во внешний файл или отобразить их на экране монитора – все эти действия реализуются разнообразными операциями и функциями языка Perl.

Простейшее взаимодействие с операционной системой, в которой выполняется программа Perl, реализуется операцией заключения строки данных в обратные кавычки. Содержимое такой строки передается на выполнение операционной системы, которая возвращает результат выполнения команды в эту же строку.

Для чтения из файла используется операция "ромб" o, которой в качестве операнда передается дескриптор файла. В этой главе мы не будем обсуждать ввод из файла через его дескриптор, отнеся рассмотрение этого вопроса в следующую главу, полностью посвященную работе с файлами. Здесь мы расскажем о том, как работает операция "ромб" в случае отсутствия операнда, представляющего дескриптор файла. В этом случае эта операция может читать записи из стандартного файла ввода STDIN или получать информацию, передаваемую программе через командную строку.

Для отображения в стандартный файл вывода STDOUT используется уже знакомая нам функция print, которая, однако, может выводить информацию и в файл, определенный своим дескриптором.

Операция ввода команды

Заключенная в обратные кавычки "·"· строка символов является всего лишь удобной формой записи операции ввода команды операционной системы qx{}, с которой мы уже знакомы *(см. главу 4).*

Когда интерпретатор Perl встречает строковый литерал в обратных кавычках, он осуществляет подстановку в нее значений скалярных переменных и переменных массивов и передает получившуюся

строку, как команду, на выполнение операционной системе. Последняя выполняет ее и возвращает в строковый литерал результаты вывода команды на стандартное устройство вывода, которым обычно является экран монитора. В связи с таким "поведением" строкового литерала в обратных кавычках его иногда называют псевдолитералом.

Операция ввода команды различает скалярный и списковый контексты, в которых она может выполняться. В скалярном контексте возвращается одна строка, содержащая весь вывод на экран монитора выполненной команды, включая символы новой строки в случае многострочного вывода. В списковом контексте возвращается список значений, каждое из которых содержит одну строку вывода. Пример 6.1 демонстрирует использование операции ввода команды в соответствующих контекстах.

```
#! Perl -w
$command = "dir";
$scalar = ~$command'; # Скалярный контекст. ,
@list = '$command~; # Списковый контекст.
print $scalar;
print $list[0], $list[lj;
```

При выполнении операции заключения в кавычки сначала осуществляется подстановка значения скалярной переменной $ command, а потом полученная строка передается на выполнение операционной системы. Переменная $scaiar (скалярный контекст) содержит весь вывод на экран монитора содержимого текущего каталога, поэтому при ее печати мы увидим все, что вывела команда dir. Когда результаты выполнения команды присваиваются массиву @iist (списковый контекст), то каждая строка вывода команды становится элементом массива, поэтому последний оператор печати примера 6.1 выводит первую и вторую строки.

В списковом контексте разбиение вывода команды операционной системы на элементы списка осуществляется в соответствии со значением встроенной переменной $/, которое используется в качестве разделителя. По умолчанию эта переменная содержит символ конца строки *"\n"* – поэтому и разбиение на элементы происходит по строкам. Присваивая этой переменной новое значение, мы тем самым определяем новое значение разделителя, которое будет использоваться при формировании элементов списка. Разделителем может быть любая последовательность символов. Например, в примере 6.2 в качестве разделителя задается строка "<КАТАЛОГ>".

```
§! Perl -w
$/ = "<КАТАЛОГ>";
@list = ~dirS; t Списковый контекст.
print $list[l], $list[2];
```

Теперь, в отличие от примера 6.1, элемент массива $iist[0j содержит не первую строку вывода команды dir, а вывод команды до первой встретившейся в нем последовательности символов " <КАТАЛОГ>". Аналогично, элемент $list[l] содержит вывод команды до следующей встретившейся последовательности СИМВОЛОВ "<КАТАЛОГ>" И Т. Д.

Команда, содержащаяся в псевдолитерале, выполняется всякий раз, как вычисляется этот псевдолитерал. Встроенная переменная $? содержит числовое значение состояния выполненной команды.

(Об интерпретации значений встроенной переменной $? см. главу 14.)

Хотим обратить внимание читателя еще раз на тот факт, что операция ввода команды возвращает вывод на стандартное устройство вывода операционной системы. При выполнении команды можно направить ее вывод на другое устройство, например, в файл. Для этого в строке после имени команды и всех необходимых для ее выполнения параметров следует задать символ ">", после которого ввести имя файла. В этом случае на экран монитора ничего выводиться не будет, а следовательно и ничего не будет возвращаться в псевдолитерал, т. е. после выполнения такой команды псевдолитерал будет содержать неопределенное значение (пример 6.3).

```
#! Perl -w - $/ = "<КАТАЛОГ>";
$list = 'dir >file.dat~; # Вывод осуществляется в файл file.dat print $list; # Оператор ничего не напечатает!
```

Замечание

Обобщенная форма операции заключения в обратные кавычки qx {} работает точно так же, как и операция заключения в обратные кавычки " * '".

Операция о

Для нашего читателя эта операция не является совсем уж новой. Несколько слов о ней было сказано *в главе 4;* в некоторых примерах мы ее использовали для ввода пользователем данных в программу Perl. Основное ее назначение — прочитать строку из файла, дескриптор которого является операндом этой операции. (Операнд операции о расположен внутри угловых скобок.) Мы не будем сейчас объяснять, что такое дескриптор файла, зачем он нужен и какую функцию выполняет в программах Perl. Эти вопросы будут нами подробно рассмотрены в следующей главе, посвященной работе с файлами. Здесь же мы остановимся на специальном случае использования этой операции — операции с пустым операндом о. В этом случае ввод осуществляется или из стандартного файла ввода, или из каждого файла, перечисленного в командной строке при запуске программы Perl. Но прежде чем перейти к описанию функционирования операции ввода с пустым операндом, остановимся на некоторых понятиях, необходимых для понимания дальнейшего.

Для обеспечения единообразия работы программ Perl в разных операционных системах при их запуске открывается несколько стандартных файлов. Один из них предназначен для ввода данных в программу и связан со стандартным устройством ввода – клавиатурой. Этот файл и называется стандартным файлом ввода и имеет дескриптор STDIN. Для вывода информации из программы создается стандартный файл вывода, также связанный со стандартным устройством вывода операционной системы, которым является экран монитора компьютера. Этому стандартному файлу назначается дескриптор STDOUT. Для отображения разнообразных сообщений о возникающих в процессе выполнения программы ошибках создается стандартный файл ошибок, который связан со стандартным устройством вывода. Этот файл имеет дескриптор STDERR. Эти файлы не надо создавать и открывать – они уже существуют, когда наша программа начинает выполняться. Иногда их называют предопределенными файлами ввода/вывода. Таким образом, если мы, например, говорим о том, что ввод осуществляется из стандартного файла (или стандартного устройства), мы имеем в виду стандартный файл ввода с дескриптором STDIN.

При запуске программы Perl в системе UNIX или из командной строки Windows ей можно передать параметры. Эти параметры задаются после имени файла, содержащего программу Perl, и отделяются от него и друг от друга пробелами:

```
Perl program.pl parl par2 par3
```

Параметрами могут быть ключи (обычно символ с лидирующим дефисом, например, -v), которые устанавливают определенные режимы работы программы, или имена файлов, содержимое которых должна обработать программа. Все передаваемые в программу параметры сохраняются в специальном встроенном массиве @ARGV. Если не передается ни одного параметра, то этот массив пуст.

Операция о без операнда, употребленная в циклах while и for, при первом своем вычислении проверяет, пуст ли массив @ARGV. Если он пуст, то в первый элемент этого массива $ARGV[O] заносится символ "-" и операция ожидает ввода пользователя из стандартного файла ввода STDIN. Если массив @ARGV не пуст, то он содержит параметры, переданные программе при ее запуске. Операция о трактует каждый из них как имя файла и в цикле передает в программу последовательно все строки всех файлов, указанных в командной строке. Рассмотрим простейшую программу (пример 6.4), состоящую из одного цикла while с операцией <>, и рассмотрим ее поведение при разных способах запуска.

```
i! Perl -w
while ($line = <>) {
print $line; }
```

При ее запуске без параметров она будет ожидать ввода пользователя с клавиатуры и в цикле распечатывать вводимые им строки, пока пользователь не завершит ввод комбинацией клавиш <Ctrl>+<Z>, интерпретируемой как конец файла.

Если при запуске передать ей имя файла, например, файла, содержащего саму же программу,

```
Perl examp6_4.pi examp6_4.pi
```

то программа примера 6.4 распечатает его содержимое:

```
f! Perl -w
while ($line = <>) {
print $line; }
```

Замечание

Предполагается, что программа примера 6.4 сохранена в файле с именем examp6_4.pl.

Если эту же программу запустить, задав в командной строке дважды имя файла программы,

```
Perl examp6_4.pl examp6_4.pl examp6_4.pl
```

то программа дважды распечатает свой собственный текст.

Замечание

В операционной системе Windows в именах файлов можно использовать пробелы. Для передачи в программу Perl файла с таким именем его следует заключать в двойные кавычки: "Name with blanks.dat".

При выполнении операции ввода из файла встроенная переменная $. на каждом шаге цикла хранит номер прочитанной строки файла. В случае задания нескольких имен файлов в командной строке при последовательном вводе их строк операцией о эта переменная продолжает увеличивать свое значение при переходе на чтение строк очередного файла, т. е. она рассматривает содержимое всех файлов как один-единственный файл.

Операцию о и массив @ARGV можно совместно использовать для ввода в программу содержимого нескольких файлов, не связывая их с заданием имен файлов в командной строке. В любом месте программы перед первым использованием в цикле операции ввода <> можно в массив SARGV занести имена файлов, содержимое которых необходимо обработать:

```
@ARGV = ("filel.dat", "file2.dat", "file3.dat"); for (;<>;) {
Операторы обработки строк файлов }
```

Этот фрагмент программы в цикле for последовательно обработает строки трех файлов filel.dat, file2.dat и file3.dat. Здесь же продемонстрирована еще одна интересная особенность операции ввода о. Обычно прочитанная этой операцией строка присваивается скалярной переменной, как это происходило в примере 6.4, но если эта операция одна представляет выражение условия цикла, то результат ее выполнения сохраняется в специальной встроенной переменной $_. Цикл while программы примера 6.4 можно записать и так:

```
while (<>) ( print;
}
```

Здесь также используется то обстоятельство, что функция print без параметров по умолчанию выводит содержимое переменной $_.

Если мы хотим передать в программу некоторые ключи, устанавливающие режим ее работы, то в начале программы следует поместить цикл, который проверяет содержимое массива @ARGV на наличие ключей в командной строке вызова программы. Один из способов подобной проверки приводится в примере 6.5, где предполагается, что программе могут быть переданы ключи

```
-d, -s И -e.
f! Perl -w
while ($_ = $ARGV[0], /А-/) {
if(/А-d/) { print $ARGV[0],"\n";}
if(/А-s/) { print $ARGV[0]-, "\n"; }
1£(/Л-e/) { print $ARGV[0],"\n"; }
shift; }
```

При вычислении выражения условия цикла while осуществляется присваивание переменной $_ значения первого элемента массива @ARGV и проверка присутствия дефиса "-" в качестве первого символа содержимого этой переменной (операция /Л-/). Операторы if проверяют содержимое переменной $_ на соответствие известным ключам и отображают их. (В реальных программах в этих операторах обычно определяют некоторые переменные, которые в дальнейшем используются для выполнения действий, присущих соответствующим ключам.) Функция shift удаляет из массива @ARGV первое значение, сдвигая оставшиеся в нем элементы на одну позицию влево: второй становится первым, третий вторым и т. д. Цикл повторяется до тех пор, пока переданные через командную строку параметры начинаются с дефиса. Еще одно применение операции <> связано с получением в программе имен файлов определенного каталога, удовлетворяющих заданному шаблону. Если в качестве операнда этой операции используется шаблон имен файлов, то в скалярном контексте она возвращает первое найденное имя файла в текущем каталоге, в списковом контексте – список имен файлов, удовлетворяющих заданному шаблону. (В шаблоне можно использовать метасимволы: * для произвольной цепочки символов, ? для произвольного одиночного символа.) Если в каталоге не найдены файлы с именами, удовлетворяющими шаблону, то операция возвращает неопределенное значение. Например, выполнение следующей операции

```
$first = <*.pl>;
```

приведет к сохранению в переменной $ first имени первого файла из списка всех файлов текущего каталога с расширением pi, если таковые файлы в каталоге есть, иначе эта переменная будет иметь

неопределенное значение. В списке файлы упорядочены в алфавитном порядке. Эта же операция в списковом контексте

```
gfiles = <*.pl>;
```

возвращает список всех файлов с расширением pi. После выполнения этой операции элементы массива @files содержат имена всех файлов с расширением pi.

Замечание

Имена подкаталогов текущего каталога считаются файлами без расширения. Например, в возвращаемом операцией <*. *> списке файлов будут содержаться и имена подкаталогов текущего каталога.

Если при задании шаблона файла явно указать каталог, то эта операция возвратит список файлов из указанного каталога, имена которых удовлетворяют заданному шаблону. Например, операция

```
@files = </Perl/*.pl>;
```

сохранит в массиве @flies имена всех файлов каталога /Perl с расширением pi.

Замечание

В системе Windows эта операция найдет все файлы с расширением p! в каталоге /Perl текущего диска. Для задания конкретного диска следует использовать принятый в Windows синтаксис для полного имени файла: <d: /Perl/*. *>. Эта операция возвратит список всех файлов каталога /Perl, расположенного на диске d:.

При использовании этой операции в выражении условия цикла while или for она последовательно на каждом шаге цикла возвращает очередное имя файла, удовлетворяющее заданному шаблону:

```
while ($file = <*.pl>) (
print "$file\n"; }
```

Употребленная в выражении условия самостоятельно, эта операция возвращает очередное имя файла в переменной $_. Например, предыдущий фрагмент можно переписать следующим образом:

```
while (<*.pl>) {
print $_, "\n"; }
```

Операция получения имен файлов, соответствующих заданному шаблону, реализуется с помощью внутренней функции glob, единственным параметром которой является шаблон имен файлов. Эту функцию можно использовать самостоятельно для получения соответствующих имен файлов:

```
· @scripts = glob "*.pl";
```

В скалярном контексте она возвращает имя первого файла, удовлетворяющего заданному шаблону, в списковом – список имен всех файлов. Употребленная без параметра, она использует в качестве параметра специальную переменную $_.

Функция *print*

Функция print наиболее часто используемая функция для вывода информации из сценария Perl. Ее синтаксис имеет следующий вид:

```
print ДЕСКРИПТОР СПИСОК;
```

Здесь ДЕСКРИПТОР представляет дескриптор файла, в который функция выводит строковые данные, представленные списком вывода список. Он может состоять из переменных, элементов массивов и выражений, вычисляемых как строковые данные. Дескриптор файла создается функцией open 0,0 которой мы поговорим в следующей главе. Он может быть опущен, и в этом случае вывод осуществляется в стандартный файл вывода STDOUT, если только функцией select o не выбран другой файл вывода по умолчанию. Как уже отмечалось ранее, обычно стандартное устройство вывода – экран монитора компьютера.

Функция print при выводе своего списка не завершает его символом новой строки "\n". Это означает, что следующая функция print начнет вывод на экран непосредственно после последнего выведенного предыдущей функцией print символа. Если такое поведение не желательно, то следует список вывода каждой функции print явно завершать строкой, содержащей символ новой строки, или включать его последним символом последнего элемента списка вывода. Пример 6.6 демонстрирует вывод с помощью функции print.

```
#! Perl -w
print "String 1:";
print "String 2:\n";
print "String 3:", "\n";
print STDOUT "String 4:\n";
print FILEOUT "String 4:\n";
```

Вывод первых четырех функций print примера 6.6 представлен ниже:

```
String I:String 2: String 3: String 4:
```

Вторая функция print начинает свой вывод на той же строке, на которой завершила вывод первая функция, в которой в списке вывода нет символа перехода на новую строку. В четвертой функции явно указан дескриптор стандартного файла вывода STDOUT. Относительно пятой функции скажем, что она ничего ни в какой файл, определенный дескриптором FILEOUT, не выведет, так как с этим

дескриптором не связан никакой файл. Для этого следовало бы до выполнения последней функции print открыть файл функцией open o и связать с ним дескриптор FILEOUT. Мы отложим эти вопросы до следующей главы.

Функция print, как и большинство других функций, определенных в языке Perl, является списковой операцией, и все элементы списка вывода вычисляются в списковом контексте. Это обстоятельство следует учитывать при использовании в качестве элементов списка вывода выражений с вызовами подпрограмм.

Все, что было сказано относительно списковых операций и их использования в качестве термов выражений в главе 4, относится, естественно, и к функции print. Если ее параметры, включая дескриптор файла, заключены в круглые скобки, то такая синтаксическая конструкция считается *термом* и в выражении имеет наивысший приоритет вычисления. Например, следующий оператор

```
print ($m + $n) ** 2;
```

напечатает сумму значений переменных $т и $п, а не их сумму, возведенную в квадрат. Компилятор Perl, обнаружив после лексемы print левую круглую скобку, найдет правую круглую скобку и будет рассматривать их содержимое как список параметров функции print. А так как такая конструкция есть терм, то сначала будет выполнена операция печати суммы значений переменных, а потом результат этой операции (Истина =1) будет возведен в квадрат. Добавление необязательного дескриптора стандартного файла вывода STDOUT исправит подобную ошибку:

```
print STDOUT ($m + $n) ** 2; # Выведет ($m + $n) ** 2
```

Если в функции печати print не задан список вывода, то она по умолчанию выводит содержимое специальной переменной $_ в файл, определенный параметром ДЕСКРИПТОР:

print; # Выводится содержимое переменной $_ на экран монитора, print STDOUT; # Эквивалентен предыдущему оператору, print FILEOUT; # Выводится содержимое переменной $_ в файл # с дескриптором FILEOUT

* * *

В этой главе мы познакомились с основными возможностями ввода/вывода, предоставляемыми языком Perl. Узнали, как легко и просто можно выполнить команду операционной системы и получить результаты ее вывода на экран монитора непосредственно в программу Perl. Операция <> позволяет не только считывать записи внешних файлов, но и автоматически обрабатывать содержимое нескольких файлов, заданных в командной строке при запуске сценария Perl. Эта же операция позволяет осуществлять поиск файлов, чьи имена удовлетворяют заданному шаблону. Для вывода информации из программы используется функция print o, которая может выводить ее не только на экран монитора (стандартное устройство вывода), но также и во внешний файл.

Вопросы для самоконтроля

1. Каким образом можно получить результаты выполнения команды операционной системы в программе Perl?
2. Какая операция позволяет прочитать содержимое всех файлов, переданных сценарию Perl через командную строку?
3. Где хранятся имена параметров, переданных сценарию Perl через командную строку?
4. Можно ли получить в программе Perl имена файлов определенного каталога, удовлетворяющих заданному шаблону?
5. Какая списковая операция осуществляет вывод на экран монитора?
6. Какая списковая операция осуществляет вывод во внешний файл?

Упражнения

1. Напишите программу, которая копирует один файл в другой. Имена файлов передаются в программу при ее запуске как параметры командной строки. (Подсказка: используйте системную команду copy.)
2. Напишите программу, которая отображает на экране содержимое файлов, имена которых задаются в командной строке. Отображение содержимого каждого файла должно предваряться строкой, содержащей имя файла. (Подсказка: использовать операцию <>.)
3. Напишите программу Perl, которая удаляет файлы определенного каталога. Имена файлов задаются шаблоном, который вместе с именем каталога передается в программу через командную строку при ее запуске.