Глава 7. Работа с файлами

Учебник по Perl

Содержание [скрыть]

Когда в программе мы создаем переменные и храним в них разнообразные данные, мы теряем их по завершении работы программы. Если нам необходимо сохранить данные и использовать их в разрабатываемых программах, мы создаем файл, записываем в него данные и сохраняем его на диске. Практически любой язык программирования предоставляет программисту средства манипулирования файлами и хранимыми в них данными.

Доступ к файлам в программе Perl осуществляется через специально создаваемые дескрипторы, которые можно рассматривать как некоторый особый вид переменных. Один дескриптор в каждый момент времени может быть связан с одним и только одним файлом, хотя на протяжении всей программы один и тот же дескриптор можно последовательно связывать с разными файлами.

Более того, дескриптор можно связать не только с файлом, но и с программным каналом, обеспечивающим связь между процессами. В этой главе мы не будем касаться вопросов взаимодействия программ с другими процессами, а рассмотрим только работу с файлами и их содержимым. Поэтому дескрипторы мы иногда будем называть дескрипторами файлов.

Дескрипторы файлов

Дескриптор – это символическое имя, которое используется в программе Perl для представления файла, устройства, сокета или программного канала. При создании дескриптора он "присоединяется" к соответствующему объекту данных и представляет его в операциях ввода/вывода. Мы дали наиболее полное определение дескриптора, чтобы читатель понимал, что дескриптор позволяет работать не только с данными файлов, но и с данными других специальных программных объектов, реализующих специфические задачи получения и передачи данных. Когда дескриптор присоединен к файлу, мы будем называть его дескриптором файла.

Замечание

> При открытии файла в системе UNIX ему также назначается файловый деск-, риптор, или дескриптор файла, который ничего общего не имеет с файловым дескриптором Perl. В UNIX дескриптор файла является целым числом, тогда как в Perl это символическое имя, по которому мы можем ссылаться на файл. Чтобы получить числовой файловый дескриптор в программе Perl, можно воспользоваться функцией f ileno ().

В программе дескриптор файла чаще всего создается при открытии файла функцией open (), которой передаются два параметра – имя дескриптора и строка с именем файла и режимом доступа:

```
open( LOGFILE, "> /temp/logfile.log");
```

Этот оператор создает дескриптор с именем LOGFILE и присоединяет его к файлу с указанным именем, который открывается в режиме записи (строка второго параметра начинается с символа ">"). В этом разделе мы не будем касаться вопросов, связанных с режимом открытия файла, а сконцентрируем наше внимание на дескрипторах. В следующем разделе режимы открытия файла будут рассмотрены нами подробнее.

Дескриптор, как указывалось, является символическим именем файла и представляет собой правильный идентификатор, который не может совпадать с зарезервированными словами Perl. В нашем примере создается дескриптор LOGFILE, "замещающий" в операциях ввода/вывода файл, к которому он присоединен (/temp/logfile.log). Например, известной нам функцией print о мы можем теперь записать в этот файл значение какой-либо переменной:

```
print LOGFILE $var;
```

Любой созданный дескриптор попадает в символьную таблицу имен Perl, в которой находятся также имена всех переменных и функций. Однако дескриптор не является переменной, хотя некоторые авторы и называют его файловой переменной. В имени дескриптора не содержится никакого префикса, присущего переменным Perl ($, @ или %). Поэтому его нельзя непосредственно использовать в операции присваивания и сохранить в переменной или передать в качестве параметра в функцию. Для подобных целей приходится использовать перед его именем префикс *, который дает ссылку на глобальный тип данных. Например, предыдущий оператор печати в файл, определенный дескриптором LOGFILE, можно осуществить с помощью следующих операторов, предварительно сохранив ссылку на дескриптор в переменной $iogf:

```
$logf = *LOGFILE; print $logf $var;
```

В операции print первая переменная $iogf замещает дескриптор файла LOGFILE, в который выводится значение второй переменной $var.

(Ссылки на глобальные имена более подробно рассматриваются в главе 9.)

> **Замечание**

> В программах Perl принято в именах дескрипторов использовать прописные буквы. Подобная практика позволяет легко обнаруживать их в программе и не приводит к конфликтам с зарезервированными именами функций, которые обычно определены строчными буквами.

В любой программе Perl всегда существуют три предопределенные дескриптора (STDIN, STDOUT и STDERR), которые связаны со стандартными устройствами ввода/вывода и используются некоторыми функциями Perl в качестве умалчиваемых дескрипторов файлов ввода или вывода. Как мы уже знаем, дескриптор STDIN связан со стандартным устройством ввода (обычно клавиатура), STDOUT и STDERR – со стандартным устройством вывода (обычно экран монитора). Стандартное устройство ввода используется операцией о, если в командной строке вызова сценария Perl не задан список файлов. Дескриптор STDOUT ПО уМОЛЧаНИЮ ИСПОЛЬЗуеТСЯ ФУНКЦИЯМИ print И die, а STDERR – функцией warn. Другие функции также используют предопределенные дескрипторы файлов для вывода своей информации.

При вызове программ в среде Unix и DOS можно перенаправлять стандартный ввод и вывод в другие файлы, задавая в командной строке их имена с префиксами > для файла вывода и < для файла ввода:

```
perl program.pl <in.dat >out.dat
```

При выполнении программы program.pi все исходные данные должны быть подготовлены в файле in.dat. Вывод будет сохранен в файле out.dat, а не отображаться на экране монитора.

Перенаправление стандартного ввода и вывода, а также стандартного отображения ошибок, можно осуществлять непосредственно в программе Perl. Для этого следует функцией open() связать соответствующий предопределенный дескриптор с некоторым дисковым файлом:

```
open(STDIN, "in.dat"); open(STDOUT, ">out.dat"); open(STDERR, ">err.dat");
```

Теперь весь стандартный ввод/вывод будет осуществляться через указанные в операторах open о файлы. Обратите внимание, что при переопределении стандартных файлов вывода и ошибок перед именами файлов стоит префикс ">", указывающий на то, что файлы открываются в режиме записи.

> **Замечание**
>
> Перенаправление стандартного ввода/вывода в программе можно производить только один раз. Это переназначение действует с момента перенаправления ввода/вывода и до конца программы, причем функцией open () нельзя вернуть первоначальные установки для дескрипторов STDIN, STDOUT и STDERR.

Доступ к файлам

Как мы уже знаем, для доступа к файлу из программы Perl необходим дескриптор. Дескриптор файла создается функцией open (), которая является списковой операцией Perl:

```
open ДЕСКРИПТОР, ИМЯ_ФАЙЛА; open ДЕСКРИПТОР;
```

При выполнении операции open с заданным в параметрах именем файла открывается соответствующий файл и создается дескриптор этого файла. В качестве дескриптора файла в функции open () можно использовать выражение – его значение и будет именем дескриптора. Имя файла задается непосредственно в виде строкового литерала или выражения, значением которого является строка. Операция open без имени файла открывает файл, имя которого содержится в скалярной переменной $ДЕСКРИПТОР, которая не может быть лексической переменной, определенной функцией ту(). Пример 7.1 демонстрирует использование операции open () для открытия файлов.

```
#! perl -w
$var = "out.dat";
$FILE4 = "file4.dat";
open FILE1, "in.dat"; # Имя файла задано строкой
open FILE2, $var; # Имя файла'задано переменной
open FILE3, "/perlourbook/01/".$var; # Имя файла вычисляется в выражении
open FILE4; # Имя файла в переменной $FILE4
```

Замечание

Если задано не полное имя файла, то открывается файл с указанным именем и расположенный в том же каталоге, что и программа Perl. Можно задавать полное имя файла (см. третий оператор open примера 7.1), однако следует иметь в виду, что оно зависит от используемой операционной системы. Например, в Windows следует обязательно задавать имя диска: d: /perlourbook/01/Chapterl. doc.

Замечание

В системе UNIX можно открыть достаточно много файлов, тогда как в DOS и Windows количество открытых файлов зависит от установленного значения переменной окружения FILE и варьируется от 20 до 50 одновременно открытых файлов.

Любой файл можно открыть в одном из следующих режимов: чтения, записи или добавления в конец файла. Это осуществляется присоединением соответствующего префикса к имени файла: < (чтение), > (запись), » (добавление). Если префикс опущен, то по умолчанию файл открывается в режиме чтения. Запись информации в файл, открытый в режиме записи (префикс >), осуществляется в начало файла, что приводит к уничтожению содержащейся в нем до его открытия информации. Информация, содержащаяся в файле, открытом в режиме добавления (префикс »), не уничтожается, новые записи добавляются в конец файла. Если при открытии файла в режиме записи или добавления не существует файла с указанным именем, то он создается, что отличает эти режимы открытия файла от режима

чтения, при котором файл должен существовать. В противном случае операция открытия завершается с ошибкой и соответствующий дескриптор не создается.

Perl позволяет открыть файл еще в одном режиме — режиме чтения/записи. Для этого перед префиксом чтения <, записи > или добавления » следует поставить знак плюс +. Отметим различия между тремя режимами чтения/записи +<, +> и +». Первый и третий режимы сохраняют содержимое открываемого файла, тогда как открытие файла с использованием второго режима (+>) сначала очищает содержимое открываемого файла. Третий режим отличается от первых двух тем, что запись в файл всегда осуществляется в конец содержимого файла.

Замечание

Некоторые операционные системы требуют устанавливать указатель чтения/записи файла при переключении с операций чтения на операции записи. В Perl для этого предназначена функция seek (), описание которой будет дано несколько позже в этом же параграфе.

Открытие файла и создание для него дескриптора функцией open () охватывает все практически важные режимы работы с файлом. Однако возможности этой функции не позволяют задать права доступа для создаваемых файлов, а также вообще решить, следует ли создавать файл, если его не существует. Для подобного "тонкого" открытия файлов можно использовать функцию sysopeno, которая позволяет программисту самому задать отдельные компоненты режима работы с файлом: чтение, запись, создание, добавление, очистка содержимого и т. д. Синтаксис этой функции таков:

```
sysopen ДЕСКРИПТОР, ИМЯ_ФАЙЛА, ФЛАГ [, РАЗРЕШЕНИЕ];
```

Здесь параметр ИМЯ_ФАЙЛА представляет имя файла без префиксов функции open (), определяющих режим открытия файла. Последний задается третьим параметром ФЛАГ — числом, представляющим результат операции побитового ИЛИ (|) над *константами режимов,* определенными в модуле Fcnti. Состав доступных констант зависит от операционной системы. В табл. 7.1 перечислены константы режима, встречающиеся практически во всех операционных системах.

Таблица 7.1. *Константы режима доступа к файлу*

Константа	Значение
0_RDONLY	Только чтение
0_WRONLY	Только запись
O_RDWR	Чтение и запись
O_CREAT	Создание файла, если он не существует

O EXCL	Завершение с ошибкой, если файл уже существует
0_APPEND	Добавление в конец файла %

Права доступа (необязательный параметр РАЗРЕШЕНИЕ) задаются в восьмеричной системе и при их определении учитывается текущее значение маски доступа к процессу, задаваемого функцией umasko. Если этот параметр не задан, то Perl использует значение o666.

(О правах доступа читайте документацию Perl для установленной на вашем компьютере операционной системе.)

Совет

Если возникают затруднения с установкой прав доступа, то придерживайтесь следующего правила: для обычных файлов передавайте 0666, а для каталогов и исполняемых файлов 0777.

В примере 7.2 собраны операции открытия файлов функцией open (} и эквивалентные ИМ ОТКРЫТИЯ С ПОМОЩЬЮ фуНКЦИИ sysopen () .

```
use Fcnti;
# Только чтение
open FF, "< file.txt";
sysopen FF, "file.txt", O_RDONLY;
# Только запись (создается, если не существует,
# и очищается содержимое, если существует)
open FF, "> file.txt";
sysopen FF, "file.txt", 0_WRONLY | 0_CREAT | OJTRUNC;
# Добавление в конец (создается, если не существует)
open FF, "» file.txt";
sysopen FF, "file.txt", OJJRONLY I 0_CREAT I O_APPEND;
# Чтение/запись (файл должен существовать) open FF, "+< file.txt"; sysopen FF, "file.txt",
O_RDWR;
# Чтение/запись (файл очищается)
open FF, "+> file.txt";
sysopen FF, "file.txt", O_RDWR | 0_CREAT I OJTRUNC;
```

При открытии файла функции open o и sysopen o возвращают значение o, если открытие файла с заданным режимом произошло успешно, и неопределенное значение undef в противном случае. Всегда следует проверять успешность выполнения операции открытия файла, прекращая выполнение программы функцией die (). Эта функция отображает список передаваемых ей параметров и завершает выполнение сценария Perl:

```
open(FF, "+< $file") or'die "Нельзя открыть файл $file: $!";
```

Обратите внимание, в сообщении функции die () используется специальная переменная $!, в которой хранится системное сообщение или код ошибки. Эта информация помогает обнаружить и исправить ошибки в программе. Например, если переменная $fiie содержит имя не существующего файла, то при выполнении предыдущего оператора пользователь может увидеть сообщение следующего вида:

```
Нельзя открыть файл file.txt: No such file or directory at D:\PERL\EX2.PL line 4.
```

Английский текст этого сообщения представляет информацию, содержащуюся в Переменной $!.

Для полноты описания работы с функцией open о следует сказать, что если имя файла представляет строку "-", то открываемый файл соответствует стандартному вводу STDIN. Это означает, что ввод с помощью созданного дескриптора файла осуществляется со стандартного устройства ввода. Если имя файла задано в виде строки ">-", то это соответствует выводу на стандартное устройство вывода, представленное в программе дескриптором STDOUT.

Замечание

Если стандартный ввод или вывод были перенаправлены *(см. раздел 7.1),* то ввод/вывод с помощью дескрипторов, соответствующих файлам "-" и ">-", будет осуществляться в файл, определенный в операции перенаправления стандартного ввода или вывода.

Последнее, что нам хотелось бы осветить в связи с дескрипторами файлов, — это создание дескриптора-дубликата. Если в строке имени файла после префикса режима открытия следует амперсанд "&", то ее оставшаяся часть рассматривается как имя дескриптора файла, а не как имя открываемого файла. В этом случае создается независимая копия этого дескриптора с именем, заданным первым параметром функции open <). Оба дескриптора имеют общий указатель текущей позиции файла, но разные буферы ввода/вывода. Закрытие одного из дескрипторов не влияет на работу другого. В программах Perl возможность создания копии дескриптора в основном применяется для восстановления стандартных файлов ввода/вывода после их перенаправления на другие файлы (пример 7.3).

```
#! perl -w
# Создание копии дескриптора STDOUT open(OLDOUT, ">&STDOUT");
# Перенаправление стандартного вывода open(STDOUT, "> file.out") or die "Невозможно
перенаправить STDOUT: $!";
# Печать в файл file.out print "Информация в перенаправленный STDOUTXn";
# Закрытие перенаправленного дескриптора стандартного вывода close(STDOUT) or die "Невозможно
закрыть STDOUT: $!";
# Восстановить файл стандартного вывода open(STDOUT, ">&OLDOUT") or die "Невозможно
восстановить STDOUT: $!";
```

```
# Закрыть копию дескриптора стандартного вывода STDOUT close(OLDOUT) or die "Невозможно закрыть OLDOUT: $!";
# Печать в восстановленный файл стандартного вывода print "Информация в восстановленный STDOUTXn";
```

Замечание

В программах следует избегать работу с одним файлом через несколько дескрипторов-копий.

По завершении работы с файлом он закрывается функцией close o. Единственным необязательным параметром этой функции является дескриптор, ассоциированный с файлом:

```
close ДЕСКРИПТОР;
```

Эта функция возвращает значение Истина, если успешно очищен буфер ввода/вывода и закрыт системный дескриптор файла. Вызванная без параметра, функция close закрывает файл, связанный с текущим дескриптором, установленным функцией select o.

Следует отметить, что закрывать файлы в программе функцией close o не обязательно. Дело в том, что открытие нового файла с дескриптором, уже связанным с каким-либо файлом, закрывает этот старый файл. Более того, при завершении программы все открытые в ней файлы закрываются. Однако такое неявное закрытие файлов таит в себе потенциальные ошибки из-за невозможности определить, завершилась ли эта операция корректно. Может оказаться, что при записи в файл переполнится диск, или будет разорвана связь с удаленным устройством вывода. Подобные ошибки можно "отловить", если использовать явное закрытие файла и проверять содержимое специальной переменной $!:

```
closet FILEIO ) or die "Ошибка закрытия файла: $!";
```

Существует еще один нюанс, связанный с явным закрытием файлов. При чтении из файла специальная переменная $. (если ее значение не изменено явным образом в программе) хранит номер последней прочитанной записи файла. При явном закрытии файла функцией close o значение этой переменной обнуляется, тогда как при неявном закрытии оно остается равным номеру последней прочитанной записи старого файла и продолжает увеличиваться при операциях чтения из нового файла.

Чтение информации из файла осуществляется операцией o, операндом которой является дескриптор файла. В скалярном контексте при первом выполнении эта операция читает первую запись файла, устанавливая специальную переменную $., отслеживающую количество прочитанных записей, равной 1. Последующие обращения к операции чтения из файла с тем же дескриптором приводят к последовательному чтению следующих записей. В списковом контексте эта операция читает все оставшиеся записи файла и возвращает список, элементами которого являются записи файла. Разделитель записей хранится в специальной переменной $/, и по умолчанию им является символ

новой строки "\n". Perl позволяет задать и другой разделитель записей обычной операцией присваивания переменной $/ нового символа разделителя записей. В примере 7.4 демонстрируются некоторые приемы чтения из файла.

```
#! perl -w
open(Fl, "in.dat") or die "Ошибка открытия файла: $!";
open(F2, "out.dat") or die "Ошибка открытия файла: $!";
$linel = <F1>; # Первая запись файла in.dat $line2 = <F1>; # Вторая запись файла in.dat
@rest =• <F1>; # Оставшиеся записи файла in.dat
$/=":"; I Задание другого разделителя записей файла @f2 = <F2>;
# Печать прочитанных записей файла out.dat for($i=0; $i<=$#f2; $i++) { print "$f2[$i]\n";
}
$/ = "\n"; # Восстановление умалчиваемого разделителя записей
close(Fl) or die $!; close(F2) or die $!;
open(F3, "out.dat") or die "Ошибка открытия файла: $!"; print <F3>; # Печать всего файла
close(F3) or die $!;
```

Несколько комментариев к программе примера 7.4. В переменные $iinel и $iine2 читаются соответственно первая и вторая строка файла in.dat, так как используется умалчиваемый разделитель записей "\n". Элементы массива @rest хранят строки с третьей по последнюю этого же файла: в операторе присваивания операция чтения <FI> выполняется в списковом контексте.

Перед чтением записей файла out.dat устанавливается новый разделитель записей — символ ":". Если файл out.dat, например, содержит только одну строку

```
111: 222: 333: Конец
```

то элементы массива @ *f г* будут содержать следующие значения:

```
$f2[0] = "111:" $f2[l] = "222:" $f2[2] = "333:" $f2[3] = "Конец"
```

Замечание

Если при создании файла out.dat его единственная строка завершена переходом на новую строку (нажата клавиша <Enter>), то $f2[3], будет содержать строку "конец\п".

При достижении конца файла операция о возвращает неопределенное значение, которое трактуется как Ложь. Это обстоятельство обычно используется для организации чтения записей файла в цикле:

```
while($line = <F1>) {
print $line; f Печать очередной строки связанного
# с дескриптором F1 файла } •
```

Запись в файл, открытый в режиме записи или добавления, осуществляется функцией print () с первым параметром, являющимся дескриптором файла:

```
print ДЕСКРИПТОР СПИСОК_ВЫВОДД;
```

Эта операция записывает содержимое элементов списка в том порядке, в котором они определены в вызове функции, и не добавляет в конец списка разделителя записей. Об этом должен позаботиться сам программист:

```
$/=":"; # Разделитель записей
print Fl @recll, $/; # Запись в файл первой записи
print Fl @rec!2, $/; tt Запись в файл второй записи
```

Замечание

Между дескриптором и первым элементом списка вывода *не должно* быть запятой. Если такое случится, то компилятор perl выдаст ошибку:

```
No comma allowed after filehandle
```

Если в функции print не указан дескриптор файла, то по умолчанию вывод осуществляется в стандартный файл вывода с дескриптором STDOUT. Эту установку можно изменить функцией select (). Вызванная без параметров, она возвращает текущий умалчиваемый дескриптор для вывода функциями print () и write (). Если ей передается единственный параметр, то этот параметр должен быть дескриптором файла. В этом случае она также возвращает текущий умалчиваемый дескриптор и меняет его на дескриптор, определенный переданным ей параметром.

```
$oldfilehandle = select(Fl); I Сохранение текущего дескриптора по
# умолчанию и назначение нового F1 print $line;
# Вывод в дескриптор F1 select($oldfilehandle);
# Восстановление старого дескриптора
# по умолчанию print $line;
# Вывод в старый дескриптор
```

Файлы в Perl интерпретируются как неструктурированные потоки байтов. При работе с файлом через дескриптор отслеживается его *текущая позиция.* Операции чтения/записи выполняются с текущей позиции файла. Если, например, была прочитана запись длиной 80 байт, то следующая операция чтения или записи начнется с 81 байта файла. Для определения текущей позиции в файле используется функция tell (), единственным параметром которой может быть дескриптор файла. Она возвращает текущую позицию в связанном с дескриптором файле. Эта же функция без параметра возвращает текущую позицию в файле, для которого была в программе выполнена последняя операция чтения.

Текущая позиция в файле автоматически изменяется в соответствии с выполненными операциями чтения/записи. Ее можно изменить с помощью функции seek о, которой передаются в качестве параметров дескриптор файла, смещение и точка отсчета. Для связанного с дескриптором файла устанавливается новая текущая позиция, смещенная на заданное параметром СМЕЩЕНИЕ число байт относительно точки отсчета:

```
seek ДЕСКРИПТОР, СМЕЩЕНИЕ, TO4KAJDTC4ETA;
```

Параметр ТОЧКА_ОТСЧЕТА может принимать одно из трех значений: о — начало файла, 1 — текущая позиция, 2 — конец файла. Смещение может быть как положительным, так и отрицательным. Обычно оно отрицательно для смещения относительно конца файла и положительно для смещения относительно начала файла. Для задания точки отсчета можно воспользоваться константами SEEK_SET, SEEK_CUR и SEEK_END из модуля ю: :Seekabie, которые соответствуют началу файла, текущей позиции и концу файла. Естественно, необходимо подключить этот модуль к программе с помощью ключевого слова use. Например, следующие операторы устанавливают одинаковые текущие позиции в файлах:

```
use 10::Seekable; seek FILE1, 5, 0; seek FILE2, 5, SEEK_SET;
```

Для перехода в начало или в конец файла следует использовать нулевое смещение относительно соответствующих точек отсчета при обращении к функции seek ():

```
seek FILE1, 0, 0; # Переход в начало файла seek FILE1, 0, 2; § Переход в конец файла
```

Кроме операции чтения записей файла о, Perl предоставляет еще два способа чтения информации из файла: функции getc () и read (). Первая читает один байт из файла, тогда как вторая читает записи фиксированной длины.

Функция getc возвращает символ в текущей позиции файла, дескриптор которого передан ей в качестве параметра, или неопределенное значение в случае достижения конца файла или возникновении ошибки. Если функция вызывается без параметра, то она читает символ из стандартного файла ввода STDIN.

```
getc; t Чтение символа из STDIN
getc Fl; # Чтение символа в текущей позиции файла с дескриптором F1
```

Функции read () передаются три или четыре параметра и ее синтаксис имеет вид:

```
read ДЕСКРИПТОР, ПЕРЕМЕННАЯ, ДЛИНА [,СМЕЩЕНИЕ] ;
```

Она читает количество байтов, определенное значением параметра ДЛИНА, в скалярную переменную, определяемую параметром ПЕРЕМЕННАЯ, из файла с дескриптором, заданным первым параметром ДЕСКРИПТОР. Возвращаемое значение — действительное количество прочитанных байтов, о при попытке чтения в позиции конца файла и неопределенное значение в случае возникновения ошибки. Параметр

СМЕЩЕНИЕ определяет количество сохраняемых байтов из содержимого переменной ПЕРЕМЕННАЯ, т. е. запись прочитанных из файла данных будет добавлена к содержимому переменной после байта, определяемого значением параметра СМЕЩЕНИЕ. Отрицательное значение смещения -п (п — целое число) означает, что из содержимого переменной ПЕРЕМЕННАЯ отбрасываются последние п байтов и к оставшейся строке добавляется запись, прочитанная из файла. Пример 7.5 демонстрирует чтение записей фиксированной длины в предположении, что файл in.dat содержит три строки данных:

```
One Two Three
#! perl -w
open(Fl, "in.dat") or die "Ошибка открытия файла: $!";
$string = "1234567890";
read Fl, $string, 6; I Чтение шести байт в переменную без смещения
print $string,"\n"; # $string = "OneXnTw"
read Fl, $string, 6, length($string);
print $string,"\n"; # $string = "One\nTwo\nThre"
```

Функция length о возвращает количество символов (байтов) в строковых данных, хранящихся в скалярной переменной, переданной ей в качестве параметра. После выполнения первой операции чтения содержимое переменной $string было уничтожено, так как эта функция read о вызывалась без смещения. Тогда как при втором чтении хранившиеся данные в переменной $string были полностью сохранены.

Операции о, print, read, seek и tell относятся к операциям буферизованного ввода/вывода, т. е. они для повышения скорости выполнения используют буферы. Perl для выполнения операций чтения из файла и записи в файл предлагает также аналоги перечисленных функций, не использующие буферы при выполнении соответствующих операций с содержимым файла.

Функции sysread и syswrite являются не буферизованной заменой операции

```
о И ФУНКЦИИ print, а ФУНКЦИЯ sysseek Заменяет ФУНКЦИИ seek И tell.
```

Функции не буферизованного чтения и записи получают одинаковые параметры, которые соответствуют параметрам функции read:

```
sysread ДЕСКРИПТОР, ПЕРЕМЕННАЯ, ДЛИНА [,СМЕЩЕНИЕ]; syswrite ДЕСКРИПТОР, ПЕРЕМЕННАЯ, ДЛИНА
[.СМЕЩЕНИЕ];
```

Смысл всех параметров аналогичен параметрам функции read (). Возвращаемым значением этих функций является истинное количество прочитанных/записанных байт, о в случае достижения конца файла или undef при возникновении ошибки.

Параметры функции sysseek о полностью соответствуют параметрам функции seek():

```
sysseek ДЕСКРИПТОР, СМЕЩЕНИЕ, ТОЧКА_ОТСЧЕТА;
```

Все, сказанное относительно использования функции seek о, полностью переносится и на ее не буферизованный аналог.

Функциональность буферизованной операции tell () реализуется следующим вызовом функции sysseek:

```
$position = sysseek Fl, 0, 1; # Текущая позиция указателя файла
```

Пример 7.6 демонстрирует использование не буферизованных функций, ввода/вывода для обработки содержимого файла.

```
#! perl -w use Fcntl;
# Открытие файла в режиме чтение/запись sysopen Fl, "in.dat", OJRDWR;
# Чтение блока в 14 байт
$read = sysread Fl, $string, 14;
warn "Прочитано $read байт вместо 14\n" if $read != 14;
# Установка текущей позиции (на 15 байт). $position = sysseek Fl, 0, 1; die "Ошибка
позиционирования: $!\n" unless defined $position;
# Запись строки в текущей позиции $string = "Новое значение"; $written = syswrite Fl, $string,
length($string);
die "Ошибка записи: $!\n" if $written != length($string); # Закрытие файла
close Fl or die $!;
```

При работе с не буферизованными функциями ввода/вывода следует всегда проверять завершение операции чтения, записи или позиционирования. Стандартная система ввода/вывода, через которую реализуется буферизованный ввод/вывод, сама проверяет и отвечает за завершение указанных операций, если процесс был прерван на середине записи. При не буферизованном вводе/выводе об этом должен позаботиться программист.

Совет

При работе с одним и тем же файлом не следует смешивать вызовы буферизованных и не буферизованных функций ввода/вывода. Подобная практика может приводить к непредсказуемым коллизиям.

Операции с файлами

Перед изучением функций, выполняющих действия с целыми файлами, мы напомним читателю основные положения, связанные с организацией файловой системы UNIX и процедур доступа к файлам. Функции Perl разрабатывались для работы именно с этой файловой системой, хотя в определенной степени многое из того, о чем пойдет речь, применимо и к файловым системам других платформ.

Работа пользователя в UNIX начинается с процедуры регистрации в системе, во время которой он вводит свое регистрационное имя и пароль. Регистрационное имя назначается администратором системы и хранится в специальном учетном файле. Пароль задает сам пользователь.

Регистрационное имя легко запоминается пользователем, но для системы удобнее вести учет пользователей, идентифицируя их не по символическим регистрационным именам, а по числовым идентификаторам. Поэтому каждому пользователю системы UNIX помимо мнемонического регистрационного имени присваивается также числовой идентификатор пользователя (uid – User IDentifier) и идентификатор группы (gid – Group IDentifier), к которой он относится. Значения uid и gid приписываются процессу, в котором выполняется командный интерпретатор shell, запускаемый при входе пользователя в систему. Эти же идентификаторы передаются и любому другому процессу, запускаемому пользователем во время его сеанса работы в UNIX.

Файловая система UNIX представляет собой дерево, промежуточные вершины которого соответствуют каталогам, а листья файлам или пустым каталогам. Каждый файл идентифицируется своим уникальным полным именем, которое включает в себя полный путь (pathname) от корня файловой системы через промежуточные вершины (каталоги) непосредственно к файлу. Корневой каталог имеет предопределенное имя, представляемое символом "/". Этот же символ используется и для разделения имен каталогов в цепочке полного имени файла, например /bin/prog.exe.

Каждый файл в файловой системе UNIX характеризуется значительно большим объемом информации, чем, например, файл в файловой системе FAT. Эта информация включает, в частности, данные о владельце файла, группе, к которой принадлежит владелец файла, о том, кто имеет право на чтение файла, запись в файл, на выполнение файла и т. д. Эта информация позволяет задавать разные права доступа к файлу для следующих категорий пользователей: владелец файла, члены группы владельца, прочие пользователи. Вся существенная информация о файле хранится в специальной структуре данных, называемой индексным дескриптором (mode). Индексные дескрипторы размещаются в специальной области диска, формируемой при его форматировании в системе UNIX.

При запуске процесса с ним связываются два идентификатора пользователя: действительный (real) и эффективный (effective) и два аналогичных идентификатора группы пользователей. Действительные идентификаторы пользователя и группы – это постоянные идентификаторы, связываемые со всеми процессами, запускаемыми пользователем. Эффективные идентификаторы – это временные идентификаторы, которые могут устанавливаться для выполнения определенных действий. Например, при изменении пользователем пароля программа passwd автоматически устанавливает эффективные идентификаторы процесса таким образом, чтобы обеспечить права записи в файл паролей.

Как только с процессом связаны соответствующие идентификаторы, для него начинают действовать ограничения доступа к файлам. Процесс может получить доступ к файлу только в случае, если это позволяют хранящиеся при файле ограничения доступа.

Для каждого зарегистрированного пользователя системы создается так называемый "домашний" (home) каталог пользователя, к которому он имеет неограниченный доступ, а также и ко всем каталогам и файлам, содержащимся в нем. Пользователь может создавать, удалять и модифицировать каталоги и файлы из своего домашнего каталога. Потенциально возможен доступ и ко всем другим файлам, однако он может быть ограничен, если пользователь не имеет достаточных привилегий.

Любой пользователь, создавший собственный файл, считается его *владельцем.* Изменить владельца файла из сценария Perl можно функцией chown (). Параметром этой функции является список, первые два элемента которого должны представлять новые .числовые идентификаторы uid и gid. Остальные элементы списка являются именами файлов, для которых изменяется владелец. Эта функция возвращает количество файлов, для которых операция изменения владельца и группы прошла успешно.

```
@list = ( 234, 3, "filel.dat", "file2.dat");
$number = chown(@list);
warn "Изменился владелец не у всех файлов!" if $number != @list-2;
```

Замечание

Изменить владельца файла может только сам владелец или суперпользователь (обычно системный администратор) системы UNIX. В операционных системах с файловой системой отличной от UNIX (DOS, Windows) эта функция отрабатывает, но ее установки не влияют на доступ к файлу.

Функция chmodo изменяет права доступа для файлов, представленных в списке, передаваемом ей в качестве параметра. Первым элементом этого списка должно быть трехзначное восьмеричное число, задающее права доступа для владельца, пользователей из группы, в которую входит владелец, и прочих пользователей. Каждая восьмеричная цифра определяет право на чтение файла, запись в файл и его выполнение (в случае если файл представляет выполняемую программу) для указанных выше групп пользователей. Установленные биты ее двоичного представления отражают соответствующие права доступа к файлу. Например, если установлены все три бита (восьмеричное число 7), то соответствующая группа пользователей обладает всеми перечисленными правами: может читать из файла, записывать в файл и выполнять его. Значение равное 6 определяет право на чтение и запись, 5 позволяет читать из файла, выполнять его, но не позволяет записывать в этот файл и т. д. Обычно не выполняемый файл создается с режимом доступа 0666 – все пользователи могут читать и записывать информацию в файл, выполняемый файл – с режимом 0777. Если владелец файла желает ограничить запись в файл пользователей не его группы, то следует выполнить следующий оператор:

```
chmod 0664, "file.dat";
```

Возвращаемым значением функции chmodo, как и функции chowno, является количество файлов из списка, для которых операция изменения прав доступа завершилась успешно.

Замечание

В операционных системах DOS и Windows имеет значение только установка режимов доступа владельца.

В структуре индексного дескриптора файла существует три поля, в которых хранится время последнего обращения (atime) к файлу, его изменения (mtime) файла и изменения индексного дескриптора Xctime): Функцией utimeO можно изменить время последнего обращения и модификации файла. Ее параметром является список, содержащий имена обрабатываемых файлов, причем первые два элемента списка — числовые значения нового времени последнего доступа и модификации:

```
gfiles = ("filel.dat", "file2.dat");
$now = time;
utime $now, $now, 6files;
```

В этом фрагменте кода время последнего доступа и модификации файлов из списка @files изменяется на текущее время, полученное с помощью функции time.

Отметим, что при выполнении функции utime о изменяется и время последней модификации индексного дескриптора (ctime) — оно устанавливается равным текущему времени. Возвращаемым значением является количе- . ство файлов, для которых операция изменения времени последнего доступа и модификации прошла успешно.

Файловая система UNIX позволяет создавать ссылки на один и тот же файл. Это реализуется простым указанием одного и того же индексного дескриптора для двух элементов каталога. Такие ссылки называются *жесткими* (hard) *ссылками,* и операционная система не различает элемент каталога, созданный при создании файла, и ссылок на этот файл. При обращении к файлу по ссылке и по имени изменяются поля индексного дескриптора. Физически файл уничтожается только тогда, когда уничтожается последняя жесткая ссылка на файл.

В UNIX существует еще один тип ссылок на файл — *символические ссылки.* Эти ссылки отличаются от жестких тем, что они косвенно ссылаются на файл, имя которого хранится в блоке данных символической ссылки.

Жесткие ссылки создаются в Perl функцией linko, а символические — функцией symiinko. Синтаксис этих функций одинаков — их два параметра представляют имя файла, для которого создается ссылка, и новое имя файла-ссылки:

```
link СТАРЫЙ_ФАЙЛ, НОВЫЙ_ФАЙЛ; symlink СТАРЫЙ_ФАЙД, НОВЫЙ_ФАЙЛ;
```

При успешном создании жесткой ссылки функция link о возвращает Истина, иначе Ложь. Создание символической ссылки функцией symiinko сопровождается возвратом ею числа i в случае успешного выполнения операции иов противном случае.

Замечание

В версиях Perl для DOS эти функции не реализованы, и при попытке их вызова интерпретатор выдает фатальную ошибку:

```
The Unsupported function link function is unimplemented at D:\EX2.PL line 2.
The symlink function is unimplemented at D:\EX2.PL line 2.
```

Удалить существующие ссылки на файл можно функцией unlink o. Эта функция удаляет одну ссылку на каждый файл, заданный в списке ее параметров. Если ссылок на файл не существует, то удаляется сам файл. Функция возвращает количество файлов, для которых успешно прошла операция удаления. Вызов функции unlink без списка параметров использует содержимое специальной переменной $_ в качестве списка параметров. Следующий фрагмент кода удаляет все резервные копии файлов текущего каталога:

```
unlink <*.bak>;
```

В структуре индексного дескриптора поле nlink содержит количество жестких ссылок на файл. Его можно использовать совместно с функцией unlink o для удаления всех ссылок на файл. Если ссылок нет, то это поле имеет значение 1 (только имя файла, определенное при его создании, ссылается на индексный дескриптор файла).

Замечание

Каталоги в UNIX являются файлами специального вида. Однако их нельзя удалить функцией unlink, если только вы не суперпользователь или при запуске perl не используется флаг -и. Для удаления каталогов рекомендуется использовать функцию rmdir ().

Две последние операции, связанные с файлами, — это переименование и усечение файла. Функция rename o меняет имя файла, заданного первым параметром, на имя, определяемое вторым параметром этой функции:

```
rename "old.dat", "new.dat";
```

Этот оператор переименует файл old.dat в файл new.dat. Функция переименования файла возвращает i при успешном выполнении этой операции и o в противном случае.

Функция truncated усекает файл до заданной длины. Для задания файла можно использовать как имя файла, так и дескриптор открытого файла:

```
truncate ДЕСКРИПТОР, ДЛИНА; truncate ИМЯ_ФАЙЛА, ДЛИНА;
```

Функция возвращает значение Истина, если длина файла успешно усечена до количества байт, определенных в параметре ДЛИНА, или неопределенное значение undef в противном случае. Под

усечением файла понимается не только уменьшение его длины, но и увеличение. Это означает, что значение второго параметра функции truncate о может быть больше истинной длины файла, что позволяет делать "дыры" в содержимом файла, которые в дальнейшем можно использовать для записи необходимой информации, не уничтожая уже записанную в файл (пример 7.7).

```
#! perl -w
# Создание файла с "дырами" for($i=l;$i<=3;$i++H
open(F, "»out.dat") or die $!; print F "Запись".$i;
close F;
open(F, "»out.dat") or die $!; truncate F, 19*$i;
close F;
} ' tt Запись информации в "дыры" open(F, "+<out.dat") or die $!; for($i=l;$i<=3;$i++){
seek F, 0,1;
read F,$reel,7;
seek F,0,l;
print F "<CONTENTS:".$i.">"; } close F;
```

На каждом шаге первого цикла for примера 7.7 в конец файла out.dat записывается информация длиной 7 байтов, а потом его длина увеличивается на 12 байтов, образуя пустое пространство в файле. Следующий цикл for заносит в эти созданные "дыры" информацию длиной 12 байтов, не затирая хранящуюся в файле информацию. Обратите внимание, что для изменения длины файла функцией truncate приходится закрывать его и снова открывать. Это связано с тем обстоятельством, что функция truncate о добавляет пустое пространство в начало файла, сдвигая в конец его содержимое, если применять ее, не закрывая файл. Можете поэкспериментировать с программой примера 7.7, открыв файл перед выполнением первого цикла for, и закрыв его после завершения цикла. Содержимое файла даст вам наглядное представление о работе функции truncate в этом случае. У нас же после выполнения первого цикла for содержимое файла out.dat выглядит так:

```
Запись! Запись2 Запись3
\
```

По завершении всей программы файл будет содержать следующую строку:

```
ЗаписьKCONTENTS:1>Запись2<СОЫТЕЫТ3:2>Запись3<СОМТЕ№Г3:3>
```

Получение информации о файле

Мы знаем, что в файловой системе UNIX информация о файле хранится в его индексном дескрипторе (inode). *Структура индексного дескриптора* состоит из 13 полей, для которых используются специальные обозначения. Все они перечислены в табл. 7.2.

Таблица 7.2. *Структура индексного дескриптора*

Поле	Описание
dev	Номер устройства в файловой системе
ino	Номер индексного дескриптора
mode	Режим файла (тип и права доступа)
nlink	Количество жестких ссылок на файл (в отсутствии ссылок равно 1)
uid	Числовой идентификатор владельца файла
gid	Числовой идентификатор группы владельца файла
rdev	Идентификатор устройства (только для специальных файлов)
size	Размер файла в байтах
a time	Время последнего обращения к файлу с начала эпохи
mtime	Время последнего изменения файла с начала эпохи
c time	Время изменения индексного дескриптора с начала эпохи
blksize	Предпочтительный размер блока для операций ввода/вывода
blocks	Фактическое количество выделенных блоков для размещения файла

Замечание

Начало эпохи датируется 1 января 1970 года 0 часов 0 минут.

Замечание

Не все перечисленные в табл. 7.2 поля структуры индексного дескриптора поддерживаются всеми файловыми системами.

Для получения значений полей структуры индексного дескриптора файла в Perl предназначена функция stato. Ее единственным параметром может быть либо имя файла, либо дескриптор открытого в программе файла. Она возвращает список из 13 элементов, содержащих значения полей структуры индексного дескриптора файла в том порядке, как они перечислены в табл. 7.2. Типичное использование в программе Perl представлено ниже

```
($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size, $atime,$mtime,$ctime,$blksize,$blocks) =
stat($filename);
```

Присваивание значений полей списку скалярных переменных с идентификаторами, соответствующими названиям полей, способствует лучшей читаемости программы, чем присваивание массиву скаляров:

```
@inode = stat($filename);
```

В последнем случае получить значение соответствующего поля можно только с помощью индекса, что не совсем удобно, так как надо помнить номер нужного поля структуры.

Если при обращении к функции stato не указан параметр, то она возвращает структуру индексного дескриптора файла, чье имя содержится в специальной переменной $_.

Функция получения информации о файле при успешном выполнении в списковом контексте возвращает список значений полей структуры индексного дескриптора файла или пустой список в случае неудачного завершения. В скалярном контексте она возвращает булево значение Истина или Ложь в зависимости от результатов своего выполнения.

Для удобства использования информации о файле функция stato при успешном выполнении кэширует полученные значения полей. Если вызвать эту функцию со специальным дескриптором файла _ (символ подчеркивания), то она возвратит информацию, хранящуюся в кэше от предыдущего ее вызова. Это позволяет проверять различные атрибуты файла без повторного вызова функции stat () или сохранения результатов ее выполнения в переменных программы.

Функцию stato можно использовать для получения структуры индексного дескриптора не только файла, но и жестких ссылок на него, а также каталогов, так как они являются также файлами, блоки данных которых содержат имена файлов каталога и их числовых индексных дескрипторов. Для получения информации о символических ссылках следует использовать функцию is tat o, которая возвращает список значений полей структуры индексного дескриптора самой ссылки, а не файла, на который она ссылается. Эта функция работает аналогично функции stat (), включая использование специального дескриптора _.

> **Замечание**
>
> Если операционная система не поддерживает символические ссылки, то обращение к функции Istat {} заменяется обращением к функции stat ().

Кроме двух этих функций, позволяющих получать информацию о файлах системы, в Perl предусмотрен набор унарных операций, возвращающих значение только одного поля структуры индексного дескриптора. Эти операции в документации называются "операциями -x", так как их названия состоят из дефиса с последующим единственным символом. Все они являются унарными именованными операциями и имеют свой приоритет в сложных выражениях, о котором мы рассказывали в гл. 3. Полный перечень унарных операций проверки атрибутов файлов представлен в табл. 7.3.

Таблица 7.3. *Унарные именованные операции проверки файлов*

Операция	Проверяемый атрибут
-r	Файл может читаться эффективным uid/gid
-W	Записывать в файл может эффективный uid/gid
-x	Файл может выполняться эффективным uid/gid
-o	Владельцем файла является эффективный uid
-R	Файл может читаться действительным uid/gid
-W	Записывать в файл может действительный uid/gid
-X	Файл может выполняться действительный uid/gid
-0	Владельцем файла является действительный uid
-e	Файл существует
-Z	Размер файла равен нулю
-S	Размер файла отличен от нуля (возвращается размер)
-f	Файл является обычным (plain) файлом
-d	Файл является каталогом
-1	Файл является символической ссылкой
-P	Файл является именованным программным каналом (FIFO) или проверяемый дескриптор связан с программным каналом
-S	Файл является сокетом
-b	Файл является специальным блочным файлом
™ C	Файл является специальным символьным файлом

-t	Дескриптор файла связан с терминалом
-и	У файла установлен бит setuid
-g	У файла установлен бит setgid
-k	У файла установлен бит запрета (sticky bit)
-k	У файла установлен бит запрета (sticky bit)
-т	Файл является текстовым файлом.
-b	Файл является двоичным (противоположным текстовому)
-м	Возраст файла в днях на момент выполнения программы
-A	То же для времени последнего обращения к файлу
-C	То же для времени последней модификации индексного дескриптора файла

Унарные операции применяются к строке, содержащей имя файла, к выражению, вычисляемым значением которого является имя файла, или к файловому дескриптору Perl. Если параметр операции не задан, то она тестирует файл, чье имя содержится в специальной переменной $_. Каждая операция проверки атрибута файла возвращает 1, если файл обладает соответствующим атрибутом, пустую строку "" в противном случае и неопределенное значение undef, если указанный в параметре файл не существует.

Несколько слов об алгоритме определения текстовых и двоичных файлов (операции -т и -в). Эти операции анализируют содержимое первого блока файла на наличие "странных" символов – необычных управляющих последовательностей или байтов с установленными старшими битами. Если обнаружено достаточно большое количество подобных символов (больше 30%), то файл считается двоичным, иначе текстовым. Любой файл с пустым первым блоком рассматривается как двоичный.

Если эти операции применяются к файловым дескрипторам Perl, то проверяется содержимое буфера ввода/вывода, а не первого блока файла. Обе эти операции, примененные к файловым дескрипторам, возвращают булево значение Истина, если ввязанный с дескриптором файл пуст или установлен на конец файла.

При выполнении унарных именованных операций проверки файла на самом деле неявно вызывается функция stato, причем результаты ее вычисления кэшируются, что позволяет использовать специальный файловый дескриптор _ для ускорения множественных проверок файла:

```
if( -s("filename") && -T J {
^ Что-то делаем для текстовых файлов не нулевого размера.
}
```

Операции с каталогами

Как мы отмечали ранее, в UNIX каталоги являются файлами специального формата, помеченными в структурах своих индексных дескрипторов как каталоги (поле rdev). Содержимым блоков данных каталогов является множество пар, состоящих из объекта, содержащегося в каталоге, и числового значения его индексного дескриптора.

Для работы с каталогами в Perl предусмотрены функции открытия, закрытия и чтения содержимого каталога, синтаксис и семантика которых аналогичны синтаксису и семантики соответствующих операций с файлами:

```
opendir ДЕСКРИПТОР, ИМЯ_КАТАЛОГА; closedir ДЕСКРИПТОР; readdir ДЕСКРИПТОР;
```

Доступ к содержимому каталога осуществляется, как и в случае с файлом, через создаваемый функцией opendir о *дескриптор каталога.* Отметим, что для дескрипторов каталогов в таблице символов Perl создается собственное пространство имен. Это означает, что в программе могут существовать, совершенно не конфликтуя между собой, дескрипторы файла и каталога с одинаковыми именами:

```
open FF, "/usr/out.dat" # Дескриптор файла opendir FF, Vusr" # Дескриптор каталога
```

Замечание

Использование одинаковых имен для дескрипторов файла и каталога может запутать самого пользователя. Однако для perl такой проблемы не существует: интерпретатор всегда точно знает, какой дескриптор следует использовать.

Функция readdir о для открытого каталога в списковом контексте возвращает список имен всех файлов каталога или пустой список, если все имена уже были прочитаны. Эта же функция в скалярном контексте возвращает следующее имя файла каталога или неопределенное значение undef, если были прочитаны все имена файлов.

Функцией rewinddiro текущая позиция в каталоге устанавливается на начало, что позволяет осуществлять повторное чтение имен файлов каталога, не закрывая его. Единственным параметром этой функции является дескриптор открытого каталога.

Программа примера 7.8 проверяет, являются все файлы каталога двоичными (содержимое вложенных каталогов не проверяется).

```
#! perl -w
opendir FDIR, "/usr/prog"; while( $name = readdir FDIR) { next if -d $name; # Каталог
print("$name: двоичный\n") if -B $name; tt Двоичный файл } closedir FDIR;
```

Функция readdir o возвращает относительное имя файла. Для получения полного имени файла следует создать его в программе самостоятельно. Например, добавить имя проверяемого каталога в примере 7.8:

```
print("/usr/prog/$name: двоичныйХп") if -B $name; # Двоичный файл
```

Для создания нового каталога следует воспользоваться функцией mkdiro, параметрами которой являются имя каталога и режим доступа (восьмеричное число):

```
mkdir ИМЯ_КАТАЛОГА, РЕЖИМ;
```

Если задается не полное имя каталога, то он создается в текущем каталоге, устанавливаемом функцией chdirf). Возвращаемым значением, функции создания нового каталога mkdir o является Истина, если каталог создан, и Ложь, если произошла какая-то ошибка. В последнем случае в специальной переменной $! хранится объяснение не выполнения операции создания каталога.

Совет

Для каталогов рекомендуется задавать режим доступа равным 0777.

Удалить каталог можно функцией rmdirf) с параметром, содержащим строку с именем каталога. Если параметр не задан, то используется специальная переменная $_. Как и функция создания каталога, эта функция возвращает значение Истина в случае успешного удаления каталога и Ложь в противном случае, записывая в переменную $! объяснение возникшей ошибки.

Замечание

Функция rmdir () удаляет только пустой" каталог. Если он содержит другие пустые каталоги, их надо удалить ранее.

* * *

В этой главе мы узнали, как получить доступ к содержимому файлов и каталогов через соответствующие дескрипторы. Научились читать и записывать информацию в файлы, создавать и удалять каталоги. Познакомились с большим набором унарных именованных операций для получения информации об атрибутах файлов из полей структуры индексного дескриптора.

Вопросы для самоконтроля

1. Как можно получить доступ к файлу из программы Perl?
2. Перечислите операции, позволяющие читать содержимое файла и записывать в него информацию:
3. Какие существуют режимы открытия файла и чем они отличаются?
4. Что такое режим доступа файла и как его можно изменить в программе Perl?
5. Что такое дескриптор каталога и зачем он нужен?
6. Как можно получить имена всех файлов определённого каталога?

Упражнения

1. Найдите ошибки в программе:

```
#1 perl -w
$var = (stat "filel.dat")[7];
open FILE1 ">filel.dat";
print FILE1, "Длина файла: " . $var . "\n";
```

2. Напишите программу, которая удаляет каталог, имя которого передается через командную строку. Если сценарий запущен без параметров, то предусмотрите отображение сообщения о правильном синтаксисе его вызова. (Указание: сначала следует удалить все файлы из нижележащих каталогов, если таковые имеются.)

3. Напишите программу копирования одного файла в другой. Предусмотрите ввод имен файлов как через командную строку, так и с экрана монитора.

4. Напишите программу копирования содержимого одного каталога в другой каталог. Предусмотрите ввод имен каталогов как через командную строку, так и с экрана монитора.

5. Напишите программу чтения строки текстового файла с заданным номером. Предусмотрите случаи, когда номер заданной строки превосходит количество строк в файле. Если номер строки отрицательный, то следует прочитать все строки, начиная со строки с номером, равным абсолютному значению введенного отрицательного значения.