Глава 14. Запуск интерпретатора и режим отладки

Учебник по Perl

Содержание [скрыть]

При запуске интерпретатора perl из командной строки можно задать разнообразные режимы его работы. Это достигается передачей ему специальных *опций,* называемых еще *переключателями* или просто *ключами,* включающих или выключающих разные режимы работы интерпретатора. Знание всех возможностей, предоставляемых опциями, позволяет более эффективно использовать интерпретатор для решения возникающих задач. Например, опция -e позволяет задать строку кода Perl непосредственно в командной строке. Эта возможность удобна для быстрого выполнения небольшой задачи, не прибегая к созданию файла с исходным текстом программы. Другие опции могут сэкономить время системного администратора, позволив решить некоторые задачи непосредственным вызовом perl из командной строки без написания кода достаточно большого объема.

Внутренний отладчик, которым снабжен интерпретатор perl, неоценим при отладке сложных сценариев Perl, особенно при разработке собственных модулей. Конечно, по общему признанию апологетов Perl, возможности его отладчика не так изощренны, как в C, C++ или Visual Basic, однако они могут оказаться мощным средством выявления ошибок, не обнаруживаемых самим компилятором даже в режиме отображения предупреждений.

В этой главе мы познакомимся с некоторыми опциями командной строки интерпретатора и их использованием для решения повседневных задач системного администрирования. Узнаем, как можно воспользоваться встроенным отладчиком в системе UNIX, а также изучим необходимый набор команд отладчика для выполнения основных действий в процессе обнаружения ошибок программ Perl.

Опции командной строки

Наиболее общая форма синтаксиса строки запуска интерпретатора perl имеет следующий вид:

```
perl [опции] [–] [файл_программы] [параметры_программы]
```

Опции perl задаются сразу же после имени интерпретатора и представляют собой двухсимвольные последовательности, начинающиеся с дефиса "-":

```
perl -a -p prog.pl filel file2
```

Некоторым из них может потребоваться передать параметры, которые задаются непосредственно после соответствующих опций:

```
perl -w -I/usr/include -0055 prog.pl filel file2
```

Опции без параметров можно группировать, задавая целую группу с одним дефисом. Следующие вызовы perl эквивалентны:

```
perl -wd prog.pl filel file2 perl -w -d prog.pl filel file2
```

Опции с параметрами таким способом задавать нельзя. Можно, однако, одну опцию с параметром добавить в конец группы опций без параметров:

```
perl -wdl/usr/include -0055 prog.pl filel file2
```

Передать опции для установки соответствующего режима работы интерпретатора можно непосредственно и из самой программы Perl. Первая строка сценария со специальным комментарием #! предназначена именно для этой цели:

```
#!/usr/bin/perl -w -d -I/usr/include
```

Теперь настало время узнать, какие опции существуют и как они влияют на режим работы интерпретатора. Мы решили сначала перечислить все возможные опции perl, а потом объяснить использование некоторых из них для выполнения задач системного администрирования UNIX. В табл. 14.1 представлены все опции с их кратким описанием. Советуем читателю внимательно ознакомиться с содержанием этой таблицы, так как для некоторых опций приведенное краткое описание в то же время является и исчерпывающим для понимания существа данной опции.

Таблица 14.1. *Опции командной строки*

Опция	Назначение
-0 [nnn]	Задает разделитель записей файла ввода, устанавливая значение специальной переменной $/. Необязательный параметр nnn — восьмеричный код символа; если отсутствует, принимается равным 0 (код символа \0). Эта опция удобна, если при каждом запускесценария Perl в него необходимо передавать файлы с разными символами завершения записи

-a	Включает режим авторазбиения строки ввода при совместном использовании с опцией -п или -р (строка передается функции split (} , а результат ее выполнения помещается в специальный массив @г). Умалчиваемый разделитель — пробел; опция -F позволяет задать регулярное выражение для иного разделителя
-c	Проверка синтаксиса программы без ее выполнения (блоки BEGIN, END и use выполняются, так как это необходимо для процесса компиляции)
-d[: модуль]	Запускает сценарий в режиме отладки под управлением модуля отладки или трассировки, установленного как Devel: : модуль
-D [число/список]	Устанавливает флаги отладки
-e ' строка кода '	Выполнение строки кода Perl, заданной параметром этой опции, а *не* файла сценария файл_программы, указанного в командной строке вызова интерпретатора. В строке кода можно задать несколько операторов, разделенных символом ';'. Допускается задание нескольких опций -e. (В Windows строка кода задается в двойных кавычках и для использования самих кавычек в коде Perl необходимо применять управляющую последовательность \".)
-F/рег выраж/ <•	Задает регулярное выражение для разделителя при автоматической разбивке строки файла ввода в случае задания опции -a. Например, *-F/-.+/* определяет в качестве разделителя одно или более двоеточий. По умолчанию используется пробел / /. Символы "/" при задании рег выраж не обязательны
-i {расширение]	Задает режим редактирования по месту файла, переданного как параметр в сценарий Perl. Используется совместно с опциями -п или -р. Редактирование выполняется по следующей схеме: файл переименовывается (если задано расширение), открывается файл вывода с именем исходного файла и все операции print ( ) сценария осуществляют вьшод в этот новый файл.

	Параметр расширение используется для задания имени копии файла по следующей схеме: если в нем нет символов "*", то оно добавляется в конец имени исходного файла; каждый символ "*" этого параметра заменяется именем исходного файла
-Tкаталог	Задает каталог поиска сценариев Perl, выполняемых операциями do, requiге и use (сохраняется как элемент специального массива @INC). Также задает каталоги размещения файлов, включаемых в сценарий Perl директивой finclude препроцессора С. Можно использовать несколько опций для задания необходимого количества каталогов поиска
- 1 [nnn]	Включает автоматическую обработку концов строк ввода. Работа в этом режиме преследует две цели: 1. Автоматическое удаление завершающего символа записи, хранящегося в специальной переменной $/, при включенном режиме -п или -р; 2. Присваивание специальной переменной $\, в которой хранится символ завершения записи при выводе операторами print, восьмеричного значения nnn, представляющего код символа завершения; если этот параметр не задан, то переменной $\ присваивается текущее значение $ /
-m[-] мо дуль -M [-] модуль -[гаМ][-] модуль= пар! [, пар2] . .	Выполняет оператор use module О (опция -т) или use module (опция -м) перед выполнением сценария Perl. Если параметр модуль задан с дефисом, то use заменяется на по. Третья форма этих опций позволяет передать параметры в модуль при его загрузке
-п	Заданный сценарий Perl выполняется в цикле while (о) { сценарий Perl } Это позволяет обработать программой Perl в цикле каждую строку всех файлов, имена которых переданы в качестве параметров сценария Perl. Эта опция эмулирует функциональность sed с ключом -п и awk

-P	Аналогичен опции -п, но добавляется печать каждой обрабатываемой строки, как в редакторе sed: while (<>) { сценарий Perl } continue { print or die "-p destination: $!\n"; }
_P	Перед компиляцией сценарий Perl обрабатывается препроцессором C, что позволяет использовать в нем команды препроцессора #define, # include и все команды условной компиляции C (#if, felse и т. д.). В опции -I можно задать каталоги расположения файлов, включаемых командой t include
-s	Включает режим синтаксического анализа строки вызова сценария Perl на наличие после имени файла сценария, но до параметров имен файлов "пользовательских опций" (параметров, начинающихся с дефиса). Такие параметры извлекаются из массива @ARGV и в программе объявляются и определяются переменные с именами введенных опций. Следующий сценарий напечатает "Задана опция optl" тогда и только тогда, когда он будет вызван с опцией -optl: # ! /usr/bin/perl -s if ($optl) { print " Задана опция optl\n"; }
-S	Поиск файла программы Perl файл программы осуществляется с использованием значения переменной среды PATH
-т	Включает режим проверки на безопасность полученных извне данных в операциях с файловой системой. Полезен при реализации CGI-сценариев. Эта опция должна быть первой в списке опций интерпретатора perl, если она применяется к сценарию
-и	Создание дампа ядра после компиляции сценария Perl. Последующее использование программы undump позволяет создать двоичный выполняемый файл сценария
-и	Разрешает выполнение небезопасных операций. При использовании этой опции в UNIX программа Perl выполняется в незащищенном режиме и ей предоставляется полный доступ к файловой системе

-*V*	Отображает номер версии, а также другую важную информацию об интерпретаторе perl
- V [: переменная]	Отображает информацию конфигурации интерпретатора perl (расположение необходимых библиотек, значения переменных среды и т. д.). Если задано имя переменной среды, отображает ее установленное значение
-W	Включает режим отображения предупреждений во время компиляции программы Perl. Рекомендуется *всегда* применять эту опцию
-x [каталог]	Извлекает и выполняет сценарий Perl, расположенный в текстовом файле. Эта возможность удобна, если сценарий прислан по электронной почте. Интерпретатор сканирует файл, пока не найдет строку, начинающуюся с символов "# ! " и содержащую слово "perl". После этого выполняется сценарий до лексемы END . Если задан параметр каталог, то перед выполнением сценария он становится текущим

Как видно из табл. 14.1, интерпретатор perl располагает достаточно большим набором опций, разработанных специально в помощь системному администратору UNIX. Эти опции позволяют решать задачи, даже не прибегая к созданию текстового файла сценария Perl.

Одной из наиболее часто встречающихся задач системного администрирования является задача обработки содержимого большого числа файлов и данных (например, изменение значений некоторых переменных среды в конфигурационных файлах). Можно написать сценарий Perl, в цикле обрабатывающий строки переданных ему файлов, а можно воспользоваться опциями -n или -p, которые организуют цикл по файлам, и задать необходимые действия для каждой строки файла в опции -e. Таким образом, быстрота решения задачи гарантирована!

Итак, мы хотим в конфигурационных файлах config1, config2 и configS, содержащих строки вида ключ = ЗНАЧЕНИЕ, изменить значение ключа "key!" на величину 5. Эта задача решается следующим вызовом perl из командной строки:

```
perl -p -i.bak -e"m/ (\w+)\s*=\s*(.+)/i;
if($1 eq V'KeylV){$_ = \"$1 = 5\n\";};" config1 config2 configS
```

Несколько комментариев к строке вызова интерпретатора perl. В ней использованы опции -p и -i для обработки содержимого конфигурационных файлов по месту (опция -i, причем задано расширение .bak для копий исходных файлов) в неявном цикле с печатью] (опция -p). Код модификации

содержимого файлов задается с помощью о/щии -e. Он достаточно прост. Строка файла, прочитанная в специальнуюЛтеременную Perl $_, проверяется на содержание подстроки вида ключ = ЗНАЧЕНИЕ. В регулярном выражении определены две группы: первая соответствует ключу, а вторая — его значению. Таким способом мы сохраняем имя ключа и его значение в специальных переменных $1 и $2, соответственно. Второй оператор кода, в случае совпадения имени ключа с заданным, заносит в переменную $_ строку с новым значением ключа (key! = 5).

Как выполняется этот сценарий? Создается копия исходного конфигурационного файла configl (файл configl.bak) и новый пустой файл с именем configl. Последовательно в переменную $_ читаются строки из файла-копии, а в переменную $1 заносится имя ключа, если таковой обнаружен оператором т/.../. Если имя ключа равно "Keyi", то в переменной $_ формируется строка задания нового значения ключа. Перед чтением следующей строки файла configl.bak в файл configl записывается содержимое переменной $_. Эта процедура выполняется со всеми заданными конфигурационными файлами. В результате выполнения этой программы формируются конфигурационные файлы, в которых значение ключа Keyi изменено на 5.

Опция -p достаточна полезна для анализа содержимого файлов. Например, следующая команда отобразит на экране содержимое файла prog.pl:

```
perl -p -e"1" prog.pl
```

Замечание

Задание кода программы Perl при работе с интерпретатором perl является необходимым действием. Его можно задать, указав имя файла программы после всех опций, либо непосредственно *в* командной строке в опции -e. При печати содержимого файла мы задали в командной строке программу из одного оператора, который, по существу, ничего не делает.

При работе с интерпретатором из командной строки очень часто используются регулярные выражения для отображения какой-либо части содержимого файла. Следующий вызов perl отображает на экране монитора последний столбец данных из файла test.dat (элементы столбцов в строке файла разделены пробелами):

```
perl -p -e "s/\s*.+\s(.+)\s*/$l\n/;" test.dat
```

Замечание

При использовании опции -p следует помнить, что она печатает содержимое специальной переменной $_. Поэтому информацию, подлежащую отображению, следует помещать именно в эту переменную, как в последнем примере.

Можно было бы привести еще массу полезных примеров вызова интерпретатора perl с различными опциями для решения разнообразных задач, но мы ограничимся приведенными примерами. При написании этой главы мы ставили целью всего лишь показать, на что способен интерпретатор perl. Думайте, размышляйте, и вы сможете решать многие задачи, не прибегая к написанию больших файлов с кодом Perl.

Отладчик Perl

Написав программу и устранив ошибки компиляции, любой программист втайне надеется, что она будет работать именно так, как он и задумывал. Практика, однако, в большинстве случаев совершенно иная: при первом запуске программа работает, но выполняет не совсем, а иногда и совсем не то, что надо; при одних исходных данных она работает идеально, а при других "зависает". И это случается не только с начинающим, но и с опытным программистом. В таких случаях на помощь приходит *отладчик* – специальная программа, которая позволяет программисту, обычно в интерактивном режиме, предпринять определенные действия в случае возникновения необычного поведения программы.

Для вызова встроенного отладчика интерпретатора perl его необходимо запустить с опцией -d (задав ее либо в командной строке, либо в строке специального комментария I! самой программы). Отладчик инициируется, когда начинает выполняться сценарий Perl. При этом отображается версия отладчика, первая выполняемая строка кода программы и его приглашение DB<I> на ввод команд пользователя:

```
Loading DB routines from perl5db.pl version 1.0402 Emacs support available.
Enter h or *h h' for help.
main::(examplel.pl:3): print "\000\001\002\003\004\005\006\007\n"; DB<1>
```

После чего можно вводить команды отладчика для выполнения определенных действий: установить точки останова, определить действия при выполнении определенной строки кода Программы, посмотреть значения переменных, выполнить часть кода сценария и т. д. Эта глава и посвящена краткому описанию команд, задающих необходимые действия при отладке разработанного сценария Perl. Мы здесь не учим стратегии поиска ошибок, а только лишь показываем возможности отладчика при ее реализации.

Замечание

Все, что здесь говорится о встроенном отладчике, относится к пользователям системы UNIX. Тот, кто работает в системах семейства Windows и пользуется интерпретатором perl фирмы ActiveState, может совсем не читать этот раздел, так как интерпретатор этой фирмы снабжен отладчиком, работа с которым осуществляется через графический интерфейс пользователя, а не с помощью командной строки. Хоти ничто не мешает таким пользователям воспользоваться встроенным отладчиком, аналогичным отладчику интерпретатора perl для UNIX, и работать с ним

из командной строки. Для этого следует воспользоваться командным файлом cmddb.bat, расположенным в каталоге \perl520\debugger.

Просмотр текста программы

При загрузке отладчика отображается первая строка кода программы Perl. Но для выполнения определенных действий (установки точки останова, задания определенных действий при выполнении конкретного оператора Perl и т. д.) нам необходимо видеть текст программы. Как это осуществить? В отладчике для подобных действий предусмотрен ряд команд.

Команда l отображает на экране монитора 10 строк текста программы, расположенных непосредственно за последней отображенной строкой. Последовательное выполнение этой команды позволяет быстро пролистать текст программы. Вызов команды l с указанием номера строки отобразит ее содержимое. Несколько последовательных строк программы можно увидеть, задав в команде l через дефис номер начальной и конечной строки. Например, команда 1 1-5 отобразит пять строк программы:

```
DB<5> 1 1-5 1 #! perl -w
2==> open FILE, "books" or die $!; 3: open REPORT, ">report"; 4 5: select REPORT;
```

Обратите внимание, что отладчик отображает не только содержимое строк, но и их номера, причем после номера строки с первым выполняемым оператором следует стрелка, а номер каждой последующей выполняемой строки завершается символом двоеточие ":". В угловых скобках приглашения отладчика ов<5> отображается порядковый номер выполненной команды от начала сеанса отладки.

Команда w отобразит блок, или окно строк – три строки до текущей, текущая строка и шесть после текущей. Текущей в отладчике считается строка программы после последней отображенной. Если команде w передать номер строки, то отобразится окно строк относительно заданной строки текста – три строки до заданной, сама заданная строка и шесть после нее.

Последняя команда отображения текста программы – команда - (дефис), которая отображает 10 строк текста, предшествующих текущей строке.

Выполнение кода

Команда s предназначена для последовательного пошагового выполнения программы: каждый ее вызов выполняет следующую строку кода. Эта строка должна выполняться в программе без изменения потока выполнения операторов. После выполнения очередной строки кода отображается строка, которая должна быть выполнена следующей:

```
DB<1> s main: : (exaitple2.pl:3) : @s = split;
```

При следующем выполнении команды s будет выполнена отображенная строка программы:

```
@s = split;.
```

Если следующая выполняемая строка кода расположена в модуле, код которого доступен (например, вызов подпрограммы), то осуществляется переход в него, соответствующая строка кода выполняется и отладчик приостанавливает выполнение программы до ввода следующей программы. Последующие команды s будут построчно выполнять код подпрограммы, пока не выполнится последняя строка ее кода. Таким образом, команда s позволяет выполнять подпрограмму с заходом в нее.

Команда n работает аналогично команде s, последовательно выполняя строки кода программы, за одним исключением – она выполняет код подпрограммы без захода в него, т. е. код подпрограммы выполняется полностью, и при следующем вызове команды n выполняется оператор строки кода, непосредственно следующей за строкой с вызовом подпрограммы. Таким образом, команда n позволяет "обойти" пошаговое выполнение операторов подпрограммы.

Совет

Если мы осуществляем пошаговое выполнение программы (командами s или n), то нажатие клавиши <Enter> эквивалентно вызову последней команды s или n.

Если мы в процессе отладки попали в тело подпрограммы (после, например, выполнения очередной команды s), то командой r можно немедленно завершить ее выполнение. Выполнение программы приостанавливается на первом после вызова подпрограммы операторе в ожидании очередной команды пользователя.

Пошаговое выполнение программы командами s и n может оказаться утомительным делом при отладке кода большого объема. В отладчике Perl предусмотрена команда c, которая выполняет программу от текущего оператора до первой встретившейся точки останова или до завершения программы, если точки останова на соответствующем участке кода не определены.

Эта же команда позволяет выполнить программу от текущей строки до определенной строки кода программы. Для этого необходимо в вызове команды c указать номер строки, до которой должна выполниться программа, если только не встретится точка останова.

Просмотр значений переменных

В процессе отладки можно посмотреть значение любой переменной программы в любой момент времени. Команда

```
V [пакет [переменная]]
```

отображает значение заданной переменной указанного пакета. Выполненная без параметров, она отображает значения всех переменных программы из всех пакетов.

При работе с этой командой следует иметь в виду, что при задании переменной, значение которой необходимо посмотреть, не надо задавать никакого префикса, а только идентификатор переменной. Отладчик отобразит значения всех переменных указанного пакета с заданным идентификатором.

Например, если в программе определена скалярная переменная $ref, маесив скаляров @ref и хеш-массив %ref, то выполнение команды v main ref приведет к следующему результату:

```
DB<1> V main ref $ref = 24 @ref = ( 1
0 1
1 2 | ) %ref = (
'One' => 1
'Two' => 2 )
```

Команда x аналогична команде v, но она отображает значения переменных текущего пакета. Ее параметром является идентификатор переменной, имя пакета указывать не надо. Вызванная без параметров, она отображает значения всех переменных текущего пакета. Например, команда x ref отобразит значения переменной $ref, массива @ref и хеш-массива %ref текущего пакета, внутри которого приостановлено выполнение программы. По умолчанию программа выполняется в пакете main.

Команда t работает как переключатель, включая режим отображения строк выполняемого кода (режим трассировки) или выключая его:

```
DB<11> t Trace = on
DB<11> c 5
main::(example2.pl:3): @ref = (1,2,3,4); main::(example2.pl:4): %ref = ("One",l, "Two",2);
main::(example2.pl:5): $_ = " qwerty \t\tqwerty";
DB<12> t Trace = off
DB<12> c 7 main::(example2.pi:7): mySub();
DB<13>
```

Обратите внимание, что при включенном режиме трассировки отображаются все строки выполняемого кода, тогда как после его выключения, отображается только следующий, подлежащий выполнению оператор.

Точки останова и действия

В процессе отладки программы возникает необходимость приостановить ее выполнение в определенных подозрительных иестах, посмотреть значения

переменных и предпринять дальнейшие действия по отладке кода. Нам уже известна команда с отладчика, которая непрерывно выполняет код программы до первой встретившейся точки останова, но как задать ее?

Для этих целей служит команда ь (сокращение от английского глагола *break* – прервать) отладчика. Ее параметром является номер строки кода, в которой устанавливается точка останова: отладчик приостановит выполнение программы Perl перед заданной строкой. Если команда ь вызывается без параметра, то точка останова определяется в текущей строке.

Можно определить точку останова в первой строке кода подпрограммы. Для этого команде b необходимо передать в качестве параметра имя подпрограммы. Например, следующая команда

```
DB<11> b mySub
```

устанавливает точку останова в первой строке кода подпрограммы mySub.

Иногда необходимо, чтобы выполнение программы приостанавливалось в некоторой точке программы только при выполнении каких-либо условий (например, равенства заданному числу значения какой-нибудь переменной, или совпадения значений двух других переменных и т. п.). Команда b позволяет задавать подобные условные точки останова. Для этого ей можно передать в качестве второго параметра условие, при истинности которого точка останова будет восприниматься отладчиком как действительная точка останова. Если условие перед выполнением строки кода не будет истинно, то останова программы в этой точке не произойдет. Например, следующая команда

```
DB<1> b 4 $r==l I
```

определяет условную точку останова в строке 4. Отладчик приостановит выполнение программы перед этой строкой по команде c только, если значение переменной $r будет равно 1.

Команда L отображает список всех установленных точек останова, как безусловных, так и условных:

```
DB<1> b 4 $r==1 DB<2> b 6 DB<3> L
Iexample2.pl: . •:.,-. , 4: %ref = ("One",!, "Two",2);
break if ($r==l) 6: @s = split;
break if (1) •/;•;'• : • -
```

Отображаемая информация о точке останова представляет номер строки и код Perl, а также условие, при котором действует точка останова (break if (УСЛОВИЕ)). Для безусловной точки останова условие всегда истинно и равно 1.

Для удаления точки останова достаточно выполнить команду d с параметром, равным номеру строки, в которой определена точка останова. Команда o удаляет все точки останова, определенные в сеансе работы с отладчиком.

Полезно при отладке программы задать действия, которые будут предприняты перед выполнением операторов определенной строки. Например, напечатать значения каких-либо переменных или изменить их при выполнении некоторого цикла. Подобное поведение программы можно реализовать, задав действия командой a. Два ее параметра определяют строку кода и сами действия (обычный оператор Perl) перед началом выполнения операторов заданной строки:

```
a 75 print "*** $ref\n";
```

Можно задать несколько операторов для выполняемых действий, однако следует учитывать, что это может привести к смешению отображаемой на экране монитора информации из самой программы и установленных действий.

Любое действие можно выполнить немедленно, набрав код в строке приглашения отладчика DBO. Подобные действия не изменяют текст программы (операторы действий не записываются в ее файл), но позволяют создавать новые переменные и использовать их в вычислениях. Правда, по завершении сеанса отладки подобная информация пропадает.

Мы познакомили читателя лишь с основными командами отладчика, наиболее важными и полезными, с нашей точки зрения, для процесса поиска ошибок. Их полный набор с краткими описаниями представлен в табл. 14.2. Более подробную информацию можно всегда найти в документации, с которой распространяется Perl, или из различных ресурсов Internet.

Таблица 14.2. *Основные команды отладчика*

Команда	Описание
т	Отображается содержимое стека вызванных подпрограмм
s	Пошаговое выполнение программы (с заходом в подпрограммы)
п	Пошаговое выполнение программы (без захода в подпрограммы)
<Enter>	Повтор последней команды s или п
г	Завершение текущей подпрограммы и возврат из нее
с [строка] с [подпрогр]	Непрерывное выполнение кода программы до первой точки останова или указанной строки, или подпрограммы
1 строка+число	Отображает число плюс одну строку кода, начиная с заданной строки –
1 строка!-строка2	Отображает диапазон строк: от строки с номером строка! до строки с номером строка2
1 строка	Отображает заданную строку
1 подпрогр	Отображает первый блок строк кода подпрограммы

1	Отображает следующий блок из 1 0 строк
-	Отображает предыдущий блок из 10 строк
w [строка]	Отображает блок строк вокруг заданной строки
.	Возврат к выполненной строке
f файл	Переключение на просмотр файла. Файл должен быть загружен
/образец/	/ Поиск строки по образцу; направление вперед от текущей строки. Завершающая косая черта не обязательна
?образец?	Поиск строки по образцу; направление назад от текущей строки. Завершающий символ "?" не обязателен
L	Отображение всех установленных точек останова
S [[!] образец]	Отображение имен подпрограмм, [не] соответствующих образцу
t	Включение/выключение режима трассировки
b [строка] [условие]	Установка точки останова в заданной строке и условия ее действия
b подпрогр [условие]	Установка точки останова в первой строке подпрограммы и условия ее действия
b load файл	Установка точки останова на операторе require файл
b postpone подпрогр [условие]	Установка точки останова в первой строке подпрограммы после ее компилирования
b compile подпрогр	Остановка после компилирования подпрограммы

В этой, можно сказать, завершающей главе мы познакомились с возможностями интерпретатора perl для решения некоторых задач системного администрирования в UNIX. Установка некоторых опций

интерпретатора при его запуске из командной строки меняет режим работы, позволяя практически без написания кода изменять, проверять, копировать и отображать содержимое файлов.

Для удобства и ускорения отладки больших программ в интерпретаторе perl предусмотрен встроенный отладчик. Его команды позволяют приостанавливать выполнение сценария Perl в подозрительных точках, задавать определенные действия при выполнении кода программы, просматривать стек вызова подпрограмм, менять в цикле значения переменных программы и многие другие полезные при поиске ошибок действия. Использование отладчика ускоряет процесс разработки программ Perl.

Вопросы для самоконтроля

1. Какую функцию выполняют опции интерпретатора perl?
2. Какие существуют способы задания опций интерпретатора?
3. Перечислите наиболее полезные опции для выполнения задач, связанных с системным администрированием в UNIX.
4. Зачем нужен отладчик и как его инициировать?
5. Перечислите основные действия, которые позволяет выполнять отладчик в процессе отладки программы Perl.