第19章 Perl 语 言

如果你需要下列问题的一个快速解决方案	请查阅节号
建立一个Perl程序	19.2.1
运行一个Perl程序	19.2.2
把程序输出在下一行显示	19.2.3
接受命令行输入	19.2.4
给变量赋值	19.2.5
使用某个变量的值	19.2.6
测试true（真）或者false（假）	19.2.7
测试文件	19.2.8
使用数学计算	19.2.9
数据值比较	19.2.10
字符串处理	19.2.11
使用数组	19.2.12
使用for语句	19.2.13
使用foreach语句	19.2.14
使用if语句	19.2.15
使用unless语句	19.2.16
使用until语句	19.2.17
使用while语句	19.2.18
把数据写入一个文件	19.2.19
从一个文件中读出数据	19.2.20

19.1 概述

在某些情况下，shell命令脚本程序（在第18章讨论）可以很容易地胜任用户需要完成的任务。但是在另外一些情况下，用户可能就需要一些功能更强的东西。比编写 shell命令脚本程序更高的一个阶段就是学会使用Perl（Practical Extraction and Report Language的首字母缩写，意思是实用抽象总结与报告语言）语言。这种语言是shell命令脚本程序编程语言和C语言的一种混合产物，它综合了这两者的长处。在需要编写运行于 Web上的应用程序的Linux系统管理员中，Perl语言特别受欢迎。

19.1.1 为什么使用Perl语言

根据下面的清单中各个项目对用户吸引力的不同，可以说有使用或者不使用 Perl语言都有许多道理：

- Perl语言的结构类似于编程使用的C语言，它并不是一个完全陌生的新结构。
- 就诸如每行程序的长度、变量名的长度和子程序嵌套深度等方面而言，其语法规则相当松散（可以说是不存在）。
- Perl语言解释器能够检查不安全的数据来源，这对任何系统管理员来说都是极其重要的帮助。

- 如果你要在shell命令脚本程序中使用大量的管道操作，在各种程序之间来回转悠，那么Perl语言就将是一个比较简洁的选择。

19.1.2 Perl语言编程示范

我们下面编写一个示范程序，说明使用 Perl语言编程的过程以及这种语言是如何工作的。需要考虑的因素和好的经验都与编写 shell命令脚本相同或者相似。

1. 程序的功能

这个Perl程序的功能是输入用来建立一个小电话号码本的数据，并把这些数据保存到一个文件中；使用同样的应用程序还可以往这个文件中添加数据。

2. 准备编写程序

在开始编写程序之前，考虑好如何把它的各个部分有机地组合在一起是很重要的。这样可以避免发生因为某些模块无法实现应用要求而重新编写大块程序的情况。就我们这个例子来说，程序需要计划考虑的问题有以下几方面：

1) 这个程序需要接受什么样的数据？它最少应该包括一个姓名和一个电话号码。

2) 姓名应该是什么格式？为了今后的灵活性，把它分断为姓、名两个部分。

3) 电话号码应该是什么格式？包括地区码和7位数字。在屏幕上显示一个示范格式，这样用户就可以知道怎样输入它们了。

4) 怎样通知程序数据输入过程的结束？用户一般不可能在事先就知道他或她有多少项数据要输入。因此使用一个遍历循环，当预定的退出条件满足时，退出程序循环。

5) 应该使用什么样的退出条件？与其强迫用户记住某个退出循环时必须要输入的特定单词，不如在程序中给出一个提示，回答“ y ”表示继续输入数据，回答“ n ”则退出。

6) 要有错误检查，保证用户对查询做出了回答。

7) 我们的这个例子是建立一个电话号码的数据库，因此应该在进行文件处理的时候，允许人们能够随时把新的电话号码记录添加到数据库中。所以，文件的打开和写操作策略应该是“ 如果文件不存在，建立它；如果文件已存在，在其末尾添加数据 ”。

3. 建立程序文件

建立这个程序首先要打开并编辑一个文本文件，就像编写一个命令脚本程序那样：

1) 选择准备在哪里保存和建立这个程序。用来保存命令脚本程序的“ ~/bin ”子目录也是保存Perl程序的好地方。如果用户喜欢把东西分门别类地保存，可以考虑分别建立子目录“ ~/bin/shell ”和子目录“ ~/bin/perl ”。对这个例子来说，我们把文件保存在 ~/bin子目录中。

2) 把路径切换到~/bin子目录。

3) 使用喜欢的文本编辑器程序编辑这个文件——比如说，输入“ vi phonebook ”命令。

相关解决方案	请查阅节号
编写一个命令脚本程序	18.2.1

4. 编写程序文件

现在开始编写程序本身的内容：

1) 因为每个Perl程序本身都是文本文件，所以它的第一行都是一个指向 Perl解释器的声明语句，告诉shell该文件包含着Perl代码。所以phonebook文件的第一行内容就是：

```
# ! /usr/bin/perl
```

2) 在开始主要的程序循环之前，先给用户显示该程序的使用方法，如下所示：

```
print " For each person whose phone number you want to store , \n " ;
print " you will be prompted for the following information : \n " ;
print " First Name , Last Name , and Phone Number . \n " ;
print " \n " ;
```

3) 对查询和计数器变量进行初始化，这样就不会在开始运行之前退出程序循环：

```
$continue = "y" ;
```

4) 打开文件，如下所示：

```
open ( FILE , " >> phonebook.dat " ) ||
        die " Can ' t open phonebook.dat . \n " ;
```

5) 因为程序循环需要不停地继续，直到用户在查询提示符处输入一个字母“ n ”为止，所以我们使用一个while语句来包含它。这个语句如下所示：

```
while ($continue eq "y") {
   #Entering data.
   print "Enter first name: ";
   $first = <STDIN>;
   print "Enter last name: ";
          $last = <STDIN>;
   print "Enter phone number in the format 2135551234: ";
   $phone = <STDIN>;
   #Finished entering. Printing data to file.
          print FILE "$first";
    print FILE "$last";
    print FILE "$phone";
    #Determine if user wants to add more data.
    print "Do you want to add another entry (y/n)? ";
    $continue = <STDIN>;
    #Perl trick to make sure we're only using the first
    #letter from the input, ignoring newlines and carriage
      returns.
    $continue = substr($continue, 0, 1);
    #Test to see if user wants to add more data. Returns
      "False"
    #for any value that is not "y".
    while ($continue eq "") {
       print "\nDo you want to add another entry (y/n)? ";
    }
}
```

6) 关闭文件，如下所示：

```
close (FILE ) ;
```

7) 保存并关闭Perl程序。

8) 使用类似于“ chmod +x filename ”这样的命令把Perl程序转换为一个可执行文件。

注意 在这个程序中明显地没有出错检查处理部分。如果想把这个程序应用到实际中去的话，它还需要检查每一个数据项都有输入的数据，同时这些数据的类型也是正确的。

相关解决方案	请查阅节号
解读文件和子目录清单列表	5.2.6
改变文件和子目录的存取权限	5.2.7

5. 测试程序

如果想测试这个程序，输入它的名字直接运行就可以。在这个例子中，在命令提示符处输入“ phonebook ”就可以了；或者如果它的路径没有包括在 $PATH环境变量的话，请输入“ ./phonebook ”。

相关解决方案	请查阅节号
在path语句中添加新路径	5.2.25

19.2 快速解决方案

19.2.1 建立一个Perl程序

编写一个Perl程序需要经过以下几个基本步骤：

1) 从编辑包含程序的文本文件开始。

2) 添加解释器定义语句，这样Linux操作系统就知道使用哪一种程序来运行用户的代码。

3) 用户编写Perl程序。

4) 保存文件并退出。

5) 使用“ chmod +x filename ”命令改变这个程序的存取权限和所有权限，使它可以由适当的用户帐户执行。

6) 测试这个程序。

7) 如有必要可进行程序纠错，然后返回第 6步。

19.2.2 运行一个Perl程序

运行一个Perl程序有两种方法：

- 使用“ chmod +x filename ”命令把包含这个程序的文本文件转换为可执行的。在命令行输入这个文件的文件名，运行这个程序。
- 保持文本文件原来的非执行属性。把它作为 Perl的一个参数，比如“ perl filename ”。

19.2.3 使程序输出在下一行显示

如果想把Perl程序的运行结果输出到下一行，需要加上控制字符“ \n ”。举例来说，我们来看看下面的代码段：

```
print " Enter your favorite color : \n " ;
$color = <STDIN > ;
print " \n " ;
```

上面代码段的输出结果如下所示：

```
> Enter your favorite color :
> Red
>
```

注意 如果在print语句中使用了单引号，那么换行控制符（\n）将会直接显示为二个字符而不是代码。

19.2.4 接受命令行输入

如果需要在Perl程序中从命令行接受输入，需要使用下面的语句把输入数据赋值给一个变量，变量赋值语句的等号右边是标准输入（Standard Input，也就是键盘）：

```
$variable = < STDIN >
```

19.2.5 给变量赋值

在Perl程序中，请按照下面的格式给变量赋一个值：

```
$variable = value ;
```

如果需要把字符串作为赋值数据的时候，必须把它们放在双引号（" "）中，如下所示：

```
$name = " Janet Smith " ;
```

19.2.6 使用某个变量的值

如果想使用某个变量的值，按照下面的格式输入这个变量：“ $variable ”。

19.2.7 测试true或者false

和命令脚本程序一样，使用操作符进行条件判断将输出一个 0或者1的值。但是在Perl语言中，0值代表假（false），1值代表真（true）。

注意 事实上，所有非零整数都代表真（true）。

19.2.8 测试文件

Perl语言提供了许多可以用来对文件（有时候还可以是子目录）进行测试的标志。如表19-1所示。

表19-1 Perl语言中使用的文件测试标志

标 志	测 试 内 容	1	0
-B	文件是否为二进制？	是	否
-d	它是否为子目录？	是	否
-e	文件是否存在？	是	否
-f	它是否为一个普通文件？	是	否
-r	文件是否可读？	是	否
-s	文件中是否有内容？	是	否
-T	它是否为一个文本文件？	是	否
-W	文件是否可写？	是	否
-X	文件是否可执行？	是	否
-Z	文件是否为空？	是	否

对文件进行测试的语句格式如下所示：

```
-flag "full path to file"
```

举例来说，使用如下所示的语句判断文件 ~/data是否存在：

```
-e "~/data"
```

19.2.9 使用数学计算

每一种计算机语言都有其各自独特的处理数学计算的方法， Perl语言也不例外。

1. 算术运算操作符

在Perl语言中，可以使用多种数值运算符，如表 19-2所示。

表19-2 Perl语言中使用的数学运算符

运 算 符	运 算 结 果
+	两值相加
-	前一个值减去后一个值
*	两值相乘
/	前一个值除以后一个值
**	求幂运算。前一个值是底，后一个值是幂次
%	求余数运算。前一个值除以后一个值所剩的余数

举例来说，我们来看看下面的例子：

```
$value = 9 ;
$power = 4 ;
$valuetopower = $value ** $power ;
```

2. 使用变量进行算术运算

在数学运算语句中使用变量的时候，一定要记住不要使用包含着英文字母的字符串变量。这些字母在计算的时候都被当作数值 0来处理。

19.2.10 数据值比较

每一种计算机语言都有其各自独特的处理数值比较的方法。注意不要把 Perl语言中的比较运算与带有独特参数的脚本程序或其他命令中的搞混了。

1. 比较数字

Perl语言提供了一系列用于算术比较的运算符，如表 19-3所示。

下面给出一个例子：

```
if ( $value1 == $value2 ) {
print " The Values are equal ";
}
```

2. 比较字符串

Perl语言把字符串中的字符当做数字来比较。它先把字符串中的每一个字符转换为对应的ASCII码，然后再按照数字比较的方法比较这些 ASCII数字。Perl语言使用表19-4所示的操作符进行字符串比较操作。

下面给出一个例子：

```
if ( $string1 ne $string2 ) {
print " The strings are identical " ;
}
```

表19-3 Perl语言使用的数学比较运算符

运 算 符	测 试 内 容	1	0
==	两个值是否相等？	是	否
!=	两个值是否相等？	否	是
<	前一个值是否小于后一个值？	是	否
<=	前一个值是否小于或者等于后一个值？	是	否
>	前一个值是否大于后一个值？	是	否
>=	前一个值是否大于或者等于后一个值？	是	否

表19-4 Perl语言使用的字符串比较运算符

运 算 符	测 试 内 容	1	0
eq	两个字符串是否相同？	是	否
ge	前一个字符串是否大于或者等于后一个字符串？	是	否
gt	前一个字符串是否大于后一个字符串？	是	否
le	前一个字符串是否小于或者等于后一个字符串？	是	否
lt	前一个字符串是否小于后一个字符串？	是	否
ne	两个字符串是否相同？	否	是

19.2.11 字符串处理

Perl语言提供了进行两种字符串操作的运算符：重复与合并。

1. 重复一个字符串

使用下面的格式重复一个字符串：

```
$string x numberof-repetitions
```

举例来说，用户打算在程序输出中把各个段落分隔开。如果输出的宽度是 72个字符，用户就可以使用下面的代码段输出一个分隔上下段落的行：

```
$break = "-" ;
print $break x 72 ;
```

这样就可以不必费力地在print语句中准确输入72个短划线了。

2. 合并字符串

合并字符串，也就是把多个字符串合并为一个，需要使用下面的格式：

```
$string1 . $string2
```

举例来说，用户把姓和名分别输入了 $last和$first两个字符变量，现在需要把它们按照姓在前名在后的“ last，first ”格式保存为一个完整的数据。建立这个新变量的语句如下所示：

```
$whole = $last . ", " . $first
```

19.2.12 使用数组

由于Perl语言是一个完备的编程语言，它也具有使用数据数组的能力。这些数组允许用户保存索引表供快速检索之用。举例来说，如果用户有一个 100项的数据表，每一项数据又包括

几个字段（比如商品名称、价格、说明等等），那么数组就是最佳的数据存放方式。

1. 普通数组

普通数组的长度是没有限制的（它只受限于计算机中的内存 RAM容量，而不是受限于Perl语言本身），可以用来保存一批彼此相关的数据。用户可以使用 @符号开始来定义一个普通数组，其格式如下所示：

```
@array = ( item1, item2, item3, item4, . . . )
```

然后，用户就可以根据每个数据保存的位置从普通数组中调用它们。这些保存位置从0开始计算：第一个位置是0、第二个位置是1，以此类推。从普通数组中调用数据的格式如下所示：

```
$array[position]
```

下面的代码段给出了一个使用数组进行会议安排的程序示例：

```
@months = ( "January", "February", "March", "April", "May" ) ;
@week = ( "Monday", "Tuesday", "Wednesday", "Thursday", "Friday" ) ;
@days = ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 ) ;
print " The first meeting is on $week[0] $months[3] $days[8] . " ;
```

这个代码段的输出应该如下所示：

```
The first meeting is on Monday April 7 .
```

2. 在普通数组中添加数据

有两种快速的方法可以在普通数组中添加数据，一个方法是按照下面的格式明确指出把新数据添加到数组中的什么位置：

```
$array[new_slot] = new_value ;
```

如下所示：

```
@array = (1, 2, 3, 4, 5 ) ;
$array[5] = 6 ;
```

另外一个方法是使用一个叫做push的函数。使用push的格式如下所示：

```
push ( @array,new_value) ;
```

还是使用上面的例子：

```
push ( @array, 7 ) ;
```

19.2.13 使用for语句

for循环是遍历语句的一种形式。这个语句会不断循环执行直到某些条件满足为止。 for语句的使用格式如下所示：

```
for ( set start value ; set conditions ; set iteration steps ) {
actions . . . ;
}
```

下面的代码段给出了一个使用 for语句的例子：

```
print "How many entries are there?\n";
$number = <STDIN>;
for ($i = 1; $i < $number; $i = $i + 1) {
    print "Enter item $i: ";
    $item{$I} = <STDIN>;
```

```
        print "\n";
    }
```

19.2.14 使用foreach语句

Perl程序员可以使用的另外一个遍历语句是 foreach语句。每个foreach语句中都会给出一个数组或者数据清单，而该语句将对它所包含的每一项数据执行一遍操作。 foreach语句的格式如下所示：

```
foreach $variable (@array) {
actions . . . ;
}
```

在上面的语句中，变量variable包含着当前遍历执行语句将操作处理的数组内容。

举例来说，我们来看看下面的代码段：

```
@prices = (1, 2, 3, 4, 5);
foreach $total (@prices) {
   print "The price of \$$total with tax is ";
   $total = $total * .07 + $total;
   print "\$$total.\n";
}
```

19.2.15 使用if语句

if语句是Perl语言中的一种条件语句。一般的 if语句格式如下所示：

```
if ( conditions ) {
actions . . . ;
}
```

在一个if语句中用户可以使用多个子句。

注意 请注意，Perl语言中的“ then ”关键字是隐含的，并没有直接写出来。

举例来说，用户可以在程序中使用它来检查某个商品是否需要加上销售税。用户可以把商品的数据按照下面的格式保存在一个普通数组中：

```
@item = ( Item Type , Item Code , Price , Taxable? )
```

某个商品的示例如下所示：

```
@item = ( "Book" , 15301 , 14.30 , 1 )
```

下面的语句检查是否需要加上税并计算出税价合计的总额是多少：

```
if ( $item[3] ) {
$price = $item[2] * .06 + $item[2] ;
}
```

1. if-else语句

if语句的一种格式还包括了else子句。使用了else子句的格式如下所示：

```
if (conditions) {
        actions . . . ;
}
else {
        actions . . . ;
```

```
}
```

我们在这里扩展一下前面介绍的 phonebook电话号码程序的例子，用户可以再添加一段出错检查处理代码：

```
if (-e file ) {
  rm -f file ;
}
else {
  print " File not found \n " ;
}
```

2. if-else-elsif语句

用户可以使用if-else-elsif组合结构对更加复杂的情况进行测试。这个复合子句的格式如下所示：

```
if (conditions) {
  actions . . . ;
}
elseif ( other conditions ) {
  actions . . . ;
}
else {
  actions . . . ;
}
```

举例来说，用户打算在程序中提供用来建立或者删除文件的选项，如下所示：

```
print "Press D to delete the file, or R to rename the file\n";
$input=<STDIN>;
if ($input = "D") {
   rm -f $file;
}
elsif ($input = "R") {
   print "Enter new file name\n";
   $newname = <STDIN>;
   mv $file $newname;
}
else {
   print "Incorrect input\n";
}
```

19.2.16 使用unless语句

Perl语言中的另外一种条件语句是 unless语句。这个语句只有条件为假（ false）的时候才继续执行。

unless语句的格式如下所示：

```
unless ( conditions ) {
  actions . . . ;
}
```

举例来说，如果你不希望让用户们删除某些特定的文件。那么就可以在代码段中包括如下所示的内容：

```
unless ($file = "~/.profile") {
```

```
        print ".profile is necessary for logging in.\n";
        print "Choose another file to delete.\n";
        $file=<STDIN>;
    }
```

19.2.17 使用until语句

until语句是Perl语言提供的另外一种遍历语句。这个语句将会一直执行到条件变为真（ 1）为止。until语句的格式如下所示：

```
until ( conditions ) {
  actions . . . ;
}
```

举例来说，我们来看看下面的代码段：

```
$count = 0;
print "Enter no more than 10 items.\n";
until ($count = 10) {
   $data{$count} = <STDIN>;
   $count = $count + 1;
}
```

19.2.18 使用while语句

在Perl语言的遍历语句中还包括了 while语句。当条件保持为真（1）的时候，这个语句将不断循环执行。while语句的格式如下所示：

```
while ( conditions ) {
  actions . . . ;
}
```

举例来说，使用下面的代码段可以让用户输入大量的数据，却不必关心到底有多少项数据：

```
print "Enter data now. If this is the final item, use the word
END.\n";
$input = " ";
while ($input ne "END") {
   print "Enter term: ";
   $input = <STDIN>;
   print "\n";
}
```

19.2.19 把数据写入一个文件

如果想把数据写入一个文件，必须限考虑几个额外的问题。在写操作开始之前文件必须被打开，操作完成之后还必须再关闭它。在写操作进行的时候，使用的命令与向屏幕上输出数据所使用的命令是一样的，只是这回加上了一个文件句柄。

注意 在Perl语言中，把数据写入一个文件实际上还有许多复杂的因素和细节需要考虑。这里介绍的最基本的知识只是为了让读者能够开始尝试。

使用这个方法打开和关闭文件的方法如下所示：

```
open ( FILE , " >> file.dat " ) ||
  die " Can ’t open file.dat . \n " ;
print FILE "$value\n" ;
close ( FILE ) ;
```

把上面的代码段中open和close语句之间的内容都记下来是很有用的，这样就不会忘记对文件进行操作时需要注意的两个方面。

注意 在文件名前面使用“ >> ”告诉Perl：如果这个文件不存在，就建立一个新文件；如果这个文件已经存在，就从文件尾开始添加数据。如果用户打算建立一个不添加数据的文件，只使用一个“ > ”符号就可以了。

19.2.20 从一个文件中读出数据

当需要从一个文件中读出数据的时候，还要使用与向文件写入数据相同的操作顺序。打开文件，从文件中读出数据，然后再关闭它。请使用下面的格式从一个文件中一次读出一行数据：

```
$value = <FILE> ;
```

这样，整个语句的格式就如下所示：

```
open ( FILE , "file.dat " ) ||
  die " Can ’t open file.dat . \n " ;
$value = <FILE> ;
close ( FILE ) ;
```