

## 第 二十 章

# perl 大師級的操作工具

**P**erl是UNIX最近出現的重要的附屬工具之一，也是最好的一個。它是由Larry Wall所開發，常被喻為是在UNIX系統上的「瑞士刀(Swiss Army Officer's Knife)」，可以把一些事情做得很好。它擴展了實用性萃取(Practical Extraction)及報表語言(Report Language)，但它已超過最初的設計目的而更有更深遠的助益。在perl中，Wall已經創造了一個多用途工具，它本身是個程式語言而且是有過濾之器之母。perl不是在所有的系統上有效，但它已是Linux及Solaris 8上的標準。不過，它是自由的免費軟體(free)，且包含了所有UNIX平台上的執行版本(<http://www.perl.com>)。

perl結合了許多UNIX上最強大有用工具的特色，包括shell,grep,tr,sed及awk。事實上，perl能做到這些工具所能做到的任何事。它包含所有的控制結構也許在其他地方看過，它有數以百計的相關於處理檔案、目錄、程序及其他的功能，與UNIX及C極像。perl也認識到目前為止我們所有討論過的正規表示式，即使它是一個大型程式，perl比shell及awk執行起來還要更快。

### 目 標

- 一個簡單的perl程式概要。(20.1)
- 使用chop來移除一行中或變數中的最後一個字元。(20.2)
- 使用跳脫序列及連結(concatenation)的運算符號。(20.3)
- 使用較佳的字串處理功能。(20.4)
- 在命令列及內部手稿中指定迴圈來讀取檔案。(20.5)
- 了解預設變數\$<sub>0</sub>的意義。(20.6)
- 運用列表及陣列，並使用他們的運算符號。(20.8)
- 在一個列表中使用foreach來做迴圈工作。(20.10)
- 使用split及join來分割及合併行。(20.11 和 20.12)
- 用非數字的矩陣索引(non-numeric subscript)來處理關連式矩陣(associative array)。(20.14)
- 用s及tr命令，正規表示式來處理替換。(20.15)

- 使用檔案物件(*filehandles*)來取出檔案及資料流。(20.16)
- 測試檔案的屬性。(20.17)
- 研發可重覆使用的子程式。(20.18)

### ► 進階篇

- 使用perl的功能來開發一個可在瀏覽器上執行的CGI程式。(20.20 和 20.21)



Note

您應該先有一些C的工作知識，或至少是awk，才可能從這本書中了解perl，因為有許多perl功能在這裡已被假設您知道（譯注：您最少應該要看得懂才可以），而且已在awk的那章中解釋過了，它們也可以在許多介紹C語言的教科書上被找到。在您能瞭解本章的一些範例之前，也需要先熟悉正規表示式的用法。如果有必要，在投入本章前請先回頭看一下第十五及第十六章。



Tip

您可以在vi編輯器內執行您的perl手稿。在18.2節的「小技巧」中會有詳細的技術做法提示。

## 20.1 perl基本概念

perl程式執行在特別的直譯式模式(interpretive mode)下；被執行之前，整個手稿在記憶體內被編譯，不像shell 及awk等其他的直譯式語言，手稿本身執行前若有錯誤會被產生出來。

```
$ perl -e 'print ("GNUs Not Unix\n");'
GNUs Not Unix
```

perl在列印上述例子（GNU的縮寫）的表現和過濾器並不相同，而比較像echo。和awk不同的是，列印不是perl的預設動作，因比您必須明確的指定它，像C一樣，所有perl敘述後面要加上分號(;)。

perl用-e的選項能從命令列做很多有用的事情，然而，大部分的perl程式相當大，通常非常的大，且最好放到.pl檔案中。這個例子表示了變數的使用及運算：

```
$ cat sample.pl
#!/usr/bin/perl
print ("Enter your name: ");
$name = <STDIN>;
print ("Enter a temperature in Centigrade: ");
$centigrade=<STDIN>;
$fahrenheit=$centigrade*9/5 + 32;
print "The temperature $name in Fahrenheit is $fahrenheit\n";
```

# 從鍵盤輸入  
# 空白不重要  
# 這裡也是

第一行（翻譯程式行）指定被用來執行手稿的程式；shell在這裡使用perl而不是它自己本身。在您的指令手稿中您已經用過簡單的功能了，因此不需要詳述。確定您在每一個perl程式中的第一行要有這一項敘述。

perl變數需要在其定義前面加上\$符號(\$name = <STDIN>)，就像和計算時一樣

( **temperature** , **\$name**等等 )。<STDIN>是一個檔案物件 ( *filehandle* , 一個檔案的邏輯名稱 ) 表示標準輸入。

注意到最後的print函數沒有使用()的圓括號。通常當它們省略會導致模稜兩可時, perl函數才需要用圓括號。可能要花點時間您才會了解, 但最後您終將會喜歡這種簡化的方式。

讓我們使用和指令手稿的相同執行方式來執行這個程式, 這次, 在鍵入實際的資料前也輸入很多空白:

```
$ sample.pl
Enter your name:          stallman
Enter a temperature in Centigrade:      40.5
The temperature          stallman
in Fahrenheit is 104.9
```

注意perl也讀取我們在**stallman**之前輸入的空白, 但是不會讀數字**40.5**之前的。perl也會忽略位於它的符號及運算子周圍的空白符號並當成敘述句中可變的一種特性。



Note

您也能使用perl *scriptname*的命令來執行perl的手稿。在這例子中, 您就不需要在手稿的第一行提供解譯列。

## 20.2 chop(): 移除最後的字元

為什麼perl顯示的輸出有兩行呢? 因為它包含由按下[Enter]鍵而產生新增一行符號(newline)而當成**\$name**的一部分 ( shell的read敘述不會這樣做 )。因此**\$name**實際上是**stallman\n** ( 忽略前面間隔 )。在很多的例子中, 我們需要移除最後一個字元, 特別是當它是一個新增一行符號時, 這時候就可以用chop()函數。讓我們來看一下這個程式:

```
$ cat name.pl
#!/usr/bin/perl
print ( " Enter your name: " );
$name = <STDIN> ;
chop ( $name ) ;                # 從$name中移除新增一行(newline)字元
if ( $name ne " " ) {
    print ( " $name, have a nice day\n" ) ;
} else {
    print ( " You have not entered your name\n" ) ;
}
```

與C及awk同名的if條件式, 在這裡與它們不同的用法是一定要用大括號{}, 不管要執行的是一個或多個敘述句。chop ( 以後我們講到函數時, 就不再顯示() ) 在移除最後一個字元非常好用, 但在這裡也指定「被切除(chopped)」直到相同的變數名稱中, 這次, 您將發現輸出只有一行:

```
$ name.pl
Enter your name: larry wall
larry wall, have a nice day
```

還有其他使用chop的方式：

```
chop ($name = <STDIN>) ;
$name = chop($name) ;
```

一起讀取及指定  
lname 是被切除最後字元的儲存值

第一行結合了C的風格，用單一行敘述來完成讀取及切除，第二個是以substr函數取出字串最後一個字元的特殊例子。



Tip

每當您從鍵盤或是一個檔案中讀取一行，除非您故意想保留新增一系列字元，否則您應該要記得使用chop函數。您會發現一些程式設計師使用chop來移除新增一系列符號，之後又用printf的敘述把它加回來，這樣做是毫無意義的。

## 20.3 變數及運算子

就像您所看到的，perl的變數沒有類別及不需要初始化。字串及數字可以大到電腦所允許的，您應該記得這些變數的屬性：

- 當字串被用來作數值運算或比較時，perl會立即的把它轉換為數字。
- 如果變數未定義，它會被假設為空字串，而空字串的數字是零。
- 如果字串的第一個字元不是數字，全部的字串等同於數字零。

perl使用類似awk同樣的運算比較符號設定(16.4)，用 ==, !=, >, <, >= 及<=。在字串的比較上，您必須在shell中使用的類似運算符號，包括eq, ne, gt, lt, ge 及le。注意到這些運算符號前不需要加上連字符號（shell則需要）。這裡的比對是依照ASCII的排列序列。

您差不多能使用任何字元當作變數值，包含我們所知道的所有串跳脫符號。另外，perl有一些特別的符號可以改變字串的大小寫。下面的例子會告訴您不同面貌的perl變數：

```
$x = $y = $z = 5 ;
$name = " larry\t\twall\n" ;
$y = " A " ; $y++ ;
$z = " P01 " ; $z++ ;
$today's_date = `date` ;
$name = " steve jobs " ;
$result = "\U$name\E" ;
$result = "\u$name\E" ;
```

多重指定  
兩個定位及新增一系列符號  
\$y變成 B  
\$z變成 P02 了！  
使用命令替換  
\$result是STEVE JOBS  
\$result是Steve jobs

這裡有一些特別的東西，您能預期perl提供這樣的功能將P01增加及傳回的P02。跳脫序列\U及\u分別的轉換所有字串及第一個字元為大寫字母。被影響區域的末端用

\E標註，可預期的\L及\I會轉換字串為小寫字母。

perl也提供了C及awk條件指定的功能，使用?及:符號。依照二月份分別來指定是28或29天：

```
$feb_days = $year % 4 == 0 ? 29 : 28 ;
```

perl也能設定變數比較之後的傳回值，以下敘述是依照比較的結果來設定\$х的值：

```
$x = $y == $z ;
```

如果比較的結果是真，傳回的比較值（在這的\$y == \$z）是非零值。如果\$y及\$z相等，這裡的\$х值就是1；否則就沒有值。注意到這裡的perl脫離了一般UNIX使用零來傳回真值的功能。

### 20.3.1 連結的運算符號.及x

不像在shell中，表示式\$х\$у（或\${x}\${y}）不能解譯為變數的連結。相對的，perl使用.(點)來當連結變數的運算符號：

```
$ perl -e '$x=ford ; $y=".com" ; print ($x . $y . "\n") ;'
ford.com
```

注意到\$у本身含有一個點，因此它必須被放在引號內。為了增加可讀性，最好在點運算符號的兩旁放入空白。

perl使用x運算符號來重覆一個字串，以下敘述是列印40個星號在螢幕上：

```
$ perl -e 'print "*" x 40 ;'
*****
```

被列印的字串不限制單一字元，它甚至可以是個表示式，這運算符號在列印報表上的尺規（rulers）時非常有用。

## 20.4 字串處理函數

perl含有所有可能 anywhere 您會用到字串函數。length及index依照它們一般的規則(16.13)，但substr函數則有相當多的變化。下面的例子會告訴您它們的用法：

```
$x = "abcdijklm" ;
print length($x) ;
print index($x,j) ;
substr($x,4,0) = "efgh" ;
print "$x" ;
$y = substr($x,-3,2) ;
print "$y" ;
```

這是9  
這是5  
\$x含有efgh  
\$x現在是abcdefghijklm  
從右邊取出  
\$y是kl

注意到index及substr第一個字元是位於0（可預期的）。substr在perl中照常能取出字串，但它也能插入一個字串。舉例來說，substr(\$x,4,0)將efgh塞進字串\$x中，不會置換任何的字元；0表示不要置換。substr(\$x,-3,2)從右邊算過來的第三個位置取出二個字元來。注意到您能指定從左邊或右邊的位置。

有四個可改變文件大小寫的函數。uc轉換所有的參數成大寫字母，ucfirst轉換第一個字元為大寫字母：

```
$name = " larry wall " ;
$result = uc($name)
$result = ucfirst($name)
```

\$result是 LARRY WALL  
\$result 是 Larry Wall

lc及lcfirst函數執行和“uc”等相反的動作，除了能轉換外，perl能過濾變數內容就像UNIX的過濾器處理文字一樣，我們稍後將討論到tr及s函數，亦即在perl中用來替換的兩個重要函數。

## 20.5 在命令列中指定檔案名稱

perl提供從檔案中存取資料的許多方法，這裡有兩個讀取dept.lst的方法：

```
perl -e 'print while (<)' dept.lst
perl -e 'print <>' dept.lst
```

簡易的迴圈

<>通常表示一個空的檔案物件(*filehandle*)，即檔案名稱以參數的方式提供給命令。當while能讀入(<>)資料，dept.lst檔案的內容就會被列印出來，但perl也有一個-n的選項隱含著這個迴圈

```
perl -ne 'print' dept.lst
```

-en就不能在這裡運作！

上面的兩種perl形式能與多個檔案合用作為連結，這個格式的優點是讓您在一個命令列中簡單的使用一行的條件句，在這有一個極簡單的gerp命令在工作（用perl來模擬grep）：

```
$ perl -ne 'print if /wood\b/' emp.lst
5423|barry wood      |chairman |admin      |08/30/56|160000
```

這一行條件句使用了正規表示式/wood\b。perl使用了擴大的正規表示式設定（表20.1中），此處\b被用來比對文字的邊界，它排除了輸出woodcock及woodhouse。我們稍後將會看到更多的perl的正規表示式。

上面的perl敘述已被置放到手稿中，這次是隱含著一個簡單的迴圈，因此，我們必須指定解譯程式及加上-n選項：

```
#!/usr/bin/perl -n
print if /wood\b/ ;
```

我們通常需要在迴圈的外部做一些處理工作，像列印標題或是總計，**-n**選項就不允許上述功能，因此，我們必須把while迴圈設定在手稿的內部：

```
#!/usr/bin/perl
printf ( "%30s", "LIST OF EMPLOYEES\n" );
while (<>) {
    print if /wood\b|light.*/ ;
}
print "\nREPORT COMPLETE\n" ;
```

egrep型式的表示式

您所看到的是我們已經做過很多次的事：列印一個標題，之後是細節的動作，其後是表尾的訊息。



Tip

就純粹的過濾器而言，在手稿的開頭使用perl -n作為翻譯程式名稱，就不需要分開while迴圈。如果您要做頭尾的列印，放棄-n選項並在手稿內設定while迴圈。

## 20.6 \$\_：預設的變數

之前的程式沒有指定用print來印什麼；perl自動了解到它就是整行。perl指定從輸讀入的一行到一個特別的變數**\$\_**，通常叫作預設的變數。這是一個非常重要的變數，如果您想寫一縮減的程式碼時，您必須了解到它迷人的特性。

假設您想在每一行前附加行編號，這也是您需要用**\$\_**來明確的指定行，以下手稿的注解中顯示了perl在內部使用**\$\_**的動作：

```
$ cat grep1a.pl
#!/usr/bin/perl
while (<>) {
    chop();
    if (/From:.*\@velvet.com/) {
        $sln++ ;
        print ($sln . " " . $_ . "\n") ;
    }
}
```

# 實際上是(\$\_ = <>)  
# chop(\$\_)  
# if (\$\_ =~ /From:.\*\@velvet ...)

雖然函數已很明確的定義，但**\$\_**在三個位置上扮演了預設值的角色，它通常表示了讀取最後的列或是符合最後的樣式。當變數名稱被省略時，**\$\_**扮演了很多函數。如果您看到函數使用很少的參數時，您應該會感到驚訝。不管您是否會用到它，**\$\_**通常不是讓人一看就懂；在這裡的經驗是您最好的入門體驗。

**\$\_**的出現（隱含或其他）在四種時機上，但是強制要有的只有一種，即在列印行。**<>**，**chop**及樣式符合預設就運作在**\$\_**上。我們在print敘述上使用時，是因為它必須與**\$sln**連結；否則print也可以預設使用**\$\_**。上面程式從velvet.com網域上，找出所有送件者的電子郵件地址：

```
$ grep1a.pl $HOME/mbox
1 From: "Caesar, Julius" <Julius_Caesar@velvet.com>
2 From: "Goddard, John" <John_Goddard@velvet.com>
3 From: "Barnack, Oscar" <Oscar_Barnack@velvet.com>
```



Tip

您可以重新指定`$_`的值。因為很多perl的函數直接用`$_`當預設，您可能常要設定`$_`到您要工作的表示式。這允許您在應用所有perl的主要函數時，表示式中不用特別指定`$_`及變數名稱，這最後會導致精簡的程式碼。

## 20.7 目前的行號(`$.`)及範圍運算符號(`..`)

perl儲存目前的行號在另一個特殊的系統變數`$.`上（`$`之後有一個點）。您可以使用它來表示行位址及在一個檔案中選擇任何行：

```
perl -ne 'print if (1..3)' foo
perl -ne 'print if (8..10)' foo
```

但是perl也有這些命令的捷徑，使用範圍運算符號`..`（2個點）：

```
perl -ne 'print if ($. < 4)' foo
perl -ne 'print if ($. > 7 && $. < 11)' foo
```

像 `head -3`  
像 `sed -n '8,10p'`

從一個檔案選取多個段落，您可以用多個print敘述，或是您可以用複合條件式：

```
if ((1..2) || (13..15)) { print ; }
```

不像`$_`可以再被重新指定，`$.`經常保留了行號。如果所有的輸入行被讀過，您可以使用`$.`來為每一行編號；一個額外的變數像`$slineno`是不需要的。

## 20.8 列表及陣列

列表及陣列是perl的精髓。perl有眾多的函數來處理它們。下面是一個列表的例子：

```
( " Jan ", 123, " How are you ", -34.56, Dec )
```

列表可能被分配到一個陣列或組合後指定給一組變數。這些陣列有兩種形式，包含純量列表及關連式陣列。在這節我們將看純量列表，讓我們將以下的列表指定給`@month`的陣列：

```
@month = ( " Jan ", " Feb ", " Mar " );
```

`$month[0]`是Jan

這是從一個列表來建立三個元素的`@month`陣列，第一個值是`$month[0]`，產生文字Jan。注意，雖然陣列本身用`@`符號來定義，每一個元素的存取是用`$month[n]`。在perl中的陣列指定也是相當的有彈性，您能使用範圍運算符號或者有選擇性的指定值：



```
@x = (1..12) ;
@month[1,3..5,12] = ( " Jan ", " Mar ", " Apr ", " May ", " Dec " ) ;
```

第一個例子指定前十二個整數(1..12)到陣列@x的前十二元素中。第二個例子中，注意到現在\$month[4]的值是Apr及如果事先未定義，則\$month[2]是空的。下面的手稿展示perl陣列的一些功能：

```
$ cat ar_in_ar.pl
#!/usr/bin/perl
@days_between = ( " Wed ", " Thu " ) ;
@days = (Mon, Tue, @days_between, Fri) ;           # 沒有引號沒關係
@days[5,6] = ( " Sat ", " Sun " ) ;
$length = @days ;                                   # @days這裡變成陣列長度
@r_days = reverse @days ;                           # 相反的陣列

print ( " The third day of the week is $days[2]\n" ) ;
print ( " The days of the week are @days\n" ) ;
print ( " The days of the week in reverse are @r_days\n" ) ;
print ( " The number of elements in the array is $length\n" ) ;
print ( " The last subscript of the array is $#days\n" ) ;
```

注意到perl允許第二個陣列(@days\_between)變成另一個陣列@days的一部分。perl也提供一個聰明的方式來選擇您自己的陣列索引(@days[5,6] = ...)。您也可以使用reverse來反轉陣列。

陣列長度可以由\$#days及@days兩個函數來決定。\$#days實際上是儲存陣列的最後陣列索引，您應該能回憶起類似Korn shell和bash中被用來決定字串長度(19.8)的結構。陣列實際的長度被存在@days中當它被擺放在指定敘述的右邊。因為陣列開始從0開始索引，\$length的值比\$#days大1：

```
$ ar_in_ar.pl
The third day of the week is Wed
The days of the week are Mon Tue Wed Thu Fri Sat Sun
The days of the week in reverse are Sun Sat Fri Thu Wed Tue Mon
The number of elements in the array is 7
The last subscript of the array is 6
```

注意到@days的計算依照它被擺放的位置不同而不同。print@days顯示所有的元素，但是當@days被指定到一個變數時(\$length = @days)，它變成了陣列的長度！

讀取一個檔案並指定到陣列中 最簡單的方式是讀取一個檔案來填滿一個陣列。每一行變成陣列的一個元素：

```
@line = <> ;                                     從命令列讀取整個檔案
print @line ;                                     列印整個檔案
```

使用單一敘述來讀取整個檔案(`@line = <>`)，陣列`@line`的每一個元素含有檔案的每一行（含有新增一行的符號）。



Note

當您讀取檔案並指定到陣列中時，每一個元素在其最後的字元上有新增一系列的符號。當`chop`函數應用到陣列時，會從陣列中的每一個元素中移除新增一系列的符號，不只是最後一個元素。



Caution

您可以建立陣列的索引從1開始而非0，方法是在perl程式的開頭中做以下的設定：`$[ = 1;`。雖然，這工作表示了非標準的方法並且可能容易困惑使用者，因為第一個索引習慣是用0。

## 20.9 ARGV[]：命令列參數

perl也使用命令列參數，它被存在系統陣列`@ARGV[]`中；第一個參數是`$ARGV[0]`。注意到命令名稱本身不是存在元素中；它包含在另一個系統變數`$0`中。下列的程式是預期一個年份的字串參數，用來決定是否為閏年：

```
$ cat leap_year.pl
#!/usr/bin/perl
die ( " You have not entered the year\n" ) if ( @ARGV == 0 ) ;
$year = $ARGV[0] ;                               # 第一個參數
$last2digits = substr($year, -2, 2) ;             # 從右邊取出
if ( $last2digits eq " 00" ) {
    $yesorno = ($year % 400 == 0 ? " certainly " : " not " ) ;
}
else {
    $yesorno = ($year % 4 == 0 ? " certainly " : " not " ) ;
}
print ( "$year is " . $yesorno . " a leap year\n" ) ;
```

我們注意到`@ARGV`的值（不是陣列索引）等於陣列的長度。您也可以在這看到使用`substr`函數相當特殊的用法，讓我們來執行這個程式：

```
$ leap_year.pl
You have not entered the year
$ leap_year.pl 2000
2000 is certainly a leap year
$ leap_year.pl 1997
1997 is not a leap year
```

上面的手稿有一個缺點；如果您提供五個數字來做閏年檢查，您必須執行程式五次。它能藉由下面要介紹的`foreach`迴圈來解決。



Note

`die()`簡單的印出它的參數及結束一個手稿。它大部分常使用在檔案的開啟錯誤處理或是放棄使用者錯誤的輸入。

## 20.10 foreach：經由一個列表來執行迴圈

perl提供一個非常有用的foreach結構經由一個列表來執行迴圈。這個結構是借用C shell中非常簡單的語法：

```
foreach $var (@arr) {
    敘述
}
```

這個工作像是shell中的for迴圈一樣，每一個在@arr陣列中的元素被取出及被指定給一個變數\$var。只要有項目在列表中它就會持續的重覆多次。下面的程式使用foreach來計算一些數的根號值：

```
$ cat square_root.pl
#!/usr/bin/perl
print ( "The program you are running is $0\n" );
foreach $number (@ARGV) {
    # @ARGV陣列中的每一個元素為$number
    print ( "The square root of $number is " . sqrt($number) . "\n" );
}
```

在陣列@ARGV的每一個元素被指定給\$number變數，現在您可以提供您想要的數目之參數到手稿中：

```
$ square_root.pl 123 456 25
The program you are running is ./square_root.pl
The square root of 123 is 11.0905365064094
The square root of 456 is 21.3541565040626
The square root of 25 is 5
```

我們事先已注意到\$\_會讓它到處出現並且沒有什麼例外。在上面的例子中，您根本不需要使用\$number。foreach把每一個項目儲存在\$\_中，且sqrt也一樣拿它來工作：

```
foreach (@ARGV) {
    print ( "The square root of $_ is " . sqrt() . "\n" );
}
```

\$\_ 是一個預設變數

foreach不是只有使用到名稱陣列，在UNIX命令產生的列表也一樣可被拿來使用，您可以用命令取替來產生列表：

```
foreach $file (`ls`) {
```

這個迴圈結構會在目前的目錄中取得每一個檔案，並指定他們給變數\$file。我們稍後會在本章中使用到這個功能。



Note

perl也有for的迴圈，下面的結構會將圈起的碼區塊執行三次：for (\$i=0 ; \$i<3 ; \$i++) {

## 20.11 split()：分割到一個列表中

CGI程式設計師使用到perl時需要了解兩個重要的陣列處理函數，包括split及join。split 分割一行或一個表示式到區塊(field)中，這些區塊被指定到變數或者陣列中，這裡有兩種語法：

```
($var1, $var2, $var3..... ) = split(/sep/,stg) ;
@arr = split(/sep/,stg) ;
```

split取出三個參數，但是通常只使用到其中二個：

- 分割的發生在表示式sep上，它能是一個字母字元或是一個能展開多個字元的正規表示式。
- 被分割的字串stg，它是可選擇的，如果未指定，\$\_被當成是預設。

分割結果產生的區塊被指定到變數\$var1, \$var2等等，或是到陣列@arr中。我們現在將用到第一個句型，從/etc/passwd 檔分析及建立一個非特權使用者的電子郵件位址列表：

```
$ cat email_create.pl
#!/usr/bin/perl -n
chop() ;
($uname, $password, $uid, $gid, $gcos, $home, $shell) = split (/:/, $_) ;
print "\t"$gcos\t" <$uname@planets.com>\n" if ( $home =~ /\home// ) ;
```

因為\$\_是split所使用的預設字串，未來我們將捨棄不用。非特權使用者是由比對\$home是否為/home/來驗證。注意到我們使用正規表示式的運算符號=~來比對第六個區塊。地址格式的規定在RFC 822中(13.7.4)，GCOS區塊的兩旁必須用雙引號括起來。”和/兩者都是特別符號而且需要被隱藏。現在，讓我們執行這手稿：

```
$ email_create.pl /etc/passwd
"george kennedy" <george@planets.com>
"henry blofeld" <henry@planets.com>
"enquiry for products" <enquiry@planets.com>
```

當一個分割結果是大量的區塊數時，您該怎麼辦？在那樣的例子中，最好是在陣列中分割（第二個形式）。先前split的敘述能用填滿陣列@profile來取代：

```
@profile = split (/:/);
```

\$\_ 是預設字串

第一個區塊就會進入\$profile[0]中，現在，您現在能使用陣列中的個別元素，像這樣：

```
$uname = $profile[0] ;
$gcos = $profile[4] ;
```

GCOS 是第五個區塊

注意到很可能會使用像這樣的指定式：\$profile = \$profile[3]；同一變數及陣列名稱不會相互衝突。split敘述也能被用在沒有指定式的敘述上：

```
split (/:/);
```

填滿陣列@\_

split得到更進一步的縮減。將填滿perl的內建陣列@\_的所有元素\$\_[0], \$\_[1]等等。您應該習慣這樣的形式，您也將看到它使用在很多的程式中。

perl的簡潔傾向允許您使用split而不需要任何的參數：

```
split();
```

分割\$\_上的空白

這行(\$\_)的空白被分離到陣列@\_中。如果不夠的話，您還可以丟掉()！



Note

當split被使用在一個沒有陣列名稱的敘述時（而不是在指定式“=”之右邊），內建的陣列@\_就會被用到。這陣列的元素是\$\_[0], \$\_[1], \$\_[2]等等。此外，當split被使用且空字串(//)當分隔符號，它儲存一個字串的每一個字元當作個別的元素。

## 20.12 join()：結合一個列表

join的功能剛好和split相反，它結合了所有的陣列元素到一個單一字串中。它使用到分隔符號當成第一個參數。剩下的參數可以是一個陣列名稱或一串變數的列表或要被結合的字串。您如何在每一日期名稱之後提供一個空白：

```
$weekstring = join (" ", @week_array);
$weekstring = join (" ", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun");
print $weekstring;
```

任何一個敘述應該會產生這樣的輸出：

```
Mon Tue Wed Thu Fri Sat Sun
```

結合有一個非常重要的應用，如果您分割一行至區塊中，然後編輯一或兩個區塊，之後您必須用join將它們貼回來，讓我們看看/etc/passwd中的這些行：

```
mdom:x:28:28:mailing list agent:/usr/lib/majordomo:/bin/ksh
yard:x:29:29:YARD database admin:/usr/lib/YARD:/bin/ksh
wwwrun:x:30:65534:daemon user for apache:/tmp:/bin/ksh
fax:x:33:14:facsimile Agent:/var/spool/fax:/bin/ksh
```

要轉換第五個區塊(GCOS)為大寫字母，下面的程式是分割每一行到區塊中，應用uc函數在第五個區塊，然後在原處結合所有的區塊：

```
#!/usr/bin/perl -n
split (/:/);
$_[4] = uc($_[4]);
$_ = join(":", @_);
print if (1..3);
```

# 在陣列@\_array上分割\$\_  
# 第五個區塊轉換無大寫字母  
# 列的重新建立  
# 列印\$\_

當程式用/etc/passwd檔案當參數來執行這程式時，前三行會被顯示：

```
mdom:x:28:28:MAILING LIST AGENT:/usr/lib/majordomo:/bin/ksh
yard:x:29:29:YARD DATABASE ADMIN:/usr/lib/YARD:/bin/ksh
wwwrun:x:30:65534:DAEMON USER FOR APACHE:/tmp:/bin/ksh
```

結合在一個分隔符號上是日常的程式設計中共通的應用，名字及姓中間通常需要空格，月，日，年可能需要用-來區分，稍後您將看到一些應用。

## 20.13 修改陣列的內容

perl有許多的函數用來處理陣列的內容，刪除陣列中開頭及結尾的元素，perl使用了shift和pop函數：

```
@list = (3..5, 9) ;
shift(@list) ;
pop (@list) ;
```

*這是3 4 5 9  
3，變成4 5 9  
移除最後的元素，變成4 5*

unshift及push函數增加元素至陣列中，讓我們把它們應用在先前的例子的最後一個，陣列@list中剩餘的值：

```
unshift(@list, 1..3) ;
push (@list, 9) ;
```

*加上1, 2 和3-1 2 3 4 5  
把9 擺在最後1 2 3 4 5 9*

splice函數能做這四個函數所能做到的任何事，此外，它在陣列的任何位置上用到最多四個參數來增加或減少元素。第二個參數表示從何處開始插入或移除的位移。第三個參數表示有多少參數會被移除，如果它是0，元素必須被加入。第四個參數（如果存在）被指明最新的替換列表：

```
splice (@list, 5, 0, 6..8) ;
splice (@list, 0, 2) ;
```

*加在第六個位置—1 2 3 4 5 6 7 8 9  
從頭開始移除—3 4 5 6 7 8 9*

我們現在將合併我們所知的陣列處理函數來開發一個手稿，這手稿接受一個IP位址（像202.54.9.1）的參數並轉換它為二進位制：

```
$ cat ipadd2binary.pl
#!/usr/bin/perl
split(/\./, $ARGV[0]) ;
print "The IP address in binary is " ;
foreach $number (@_) {
    $original_number = $number ;
    until ($number == 0) {
        $bit = $number % 2 ;
        unshift (@bit_arr, $bit) ;
        $number = int($number / 2) ;
    }
    $binary_number = join ( " ", @bit_arr) ;
    substr($binary_number,0,0) = "0" x (8 - length($binary_number)) ;
    print ( "$binary_number " ) ;
    splice(@bit_arr, 0, $#bit_arr+1) ;
}
print chr(10) ;
```

*# 找出剩餘的位元  
# 在開頭插入位元  
# 結合在空位置！  
# 刪除所有的陣列元素  
# 列印出新增一行符號--像是印出 "\n"*

`split`在這裡分離以點表示的IP位址到四個區塊並儲存它們到陣列`@_`中。您將重新收集那些區塊資料並轉換十進位數到二進位數字，您必須再三的用2來除得商數，然後反轉所有收集到的餘數。`unshift`做這反轉工作。

`substr`在開頭之處填加額外的0，因此每一個八個位元組（octet）的IP位址會顯示八個位元（bit）。`foreach`本身印出相等於每一個八位元組的二進位數。在下次重複開始前，`splice`會清除陣列元素。讓我們執行下列的手稿：

```
$ ipadd2binary.pl 224.67.34.06
The IP address in binary is 11100000 01000011 00100010 00000110
```

您也可以用這個程式來轉換子網路遮罩(23.1)，然後檢查二個主機是否位在同一子網路上。

## 20.14 關連式陣列

perl還有使用其他的陣列型式，即關連式陣列。它交替一連串由逗號分隔的陣列索引和值。舉例來說，關連式陣列`%region`可像這樣被定義：

```
%region = ("N", "North", "S", "South", "E", "East", "W", "West");
```

這種陣列使用`%`在陣列名稱前，此指定敘述建立一個有四個元素的陣列，在陣列定義中陣列索引被放在值之前。陣列的索引，也可以是字串，它必須由`{}`（大括號）圍成而不是`[]`。舉例來說，`$region{"N"}`產生`North`。CGI程式程式設計師們必須精通關連式陣列。

下面的程式使用`%region`陣列來展開區域碼，它也能顯示如何使用兩個關連式陣列的函數`keys`及`values`：

```
$ cat region.pl
#!/usr/bin/perl
%region = ("N", "North", "S", "South", "E", "East", "W", "West");
foreach $letter (@ARGV) {
    print (" The letter $letter stands for $region{$letter}" . "\n");
}
@key_list = keys(%region);           # 下標符號的列表
print (" The subscripts are @key_list\n");
@value_list = values %region;        # 值的列表
print (" The values are @value_list\n");
```

`keys`儲存陣列索引的列表在分開的陣列中（在此是`@key_list`），而`values`儲存另外一個陣列（在此是`@value_list`）中的每一個元素的值（在此我們不使用圓括號）。讓我們藉由提供一連串的單一字元字串來測試這個手稿：

```
$ region.pl S W
The letter S stands for South
The letter W stands for West
The subscripts are S E N W
The values are South East North West
```

在這裡有很重要的含意，您可以從一個關連式陣列中分開地取出索引碼(key)和它們的值。您也可以用與set敘述來表示所有的環境變數一樣的方法來顯示這些值：

```
foreach $key (keys %region) {
    print "$key" . " = " . "$region{$key}\n" ;
}
```

我們找出每一個索引值並依序儲存它們在變數\$**key**中，然後是個簡單的動作，把它使用在%**region**的陣列索引中。元素\$**region**{**\$key**}儲存每一個索引碼的值並顯示如下的輸出：

```
S=South
E=East
N=North
W=West
```

通常，keys會在一個隨機序列中傳回索引字串。為了用字母排序列表，您通常會用keys搭配sort函數，您可以有一個正常和反向的排序：

```
foreach $key (sort(keys %region)) {
    @key_list = reverse sort keys %region ;
```

沒有()沒關係



Note

perl的內建陣列%ENV儲存所有shell的環境變數。舉例來說，\$ENV{ ' PATH ' }含有shell的\$PATH值。您能使用此處討論的技術輕易的存取這些變數。

### 20.14.1 計數出現的次數

在關聯式陣列用來計數某個項目的出現次數是非常有用的。在簡單的資料庫中，您能建立一個顯示每個部門人員數的報表。我們以前曾經試著用awk(16.15.2)和基本的UNIX過濾器(9.18.1)來做類似的練習，我們將用perl來完成這個練習。對於一個程式語言來說，可能要花相當的工夫，但是用一個小的perl程式就能做這樣的工作：



```

$ cat count.pl
#!/usr/bin/perl
while (<>) {
    split (/\\|/);
    $dept = $_[3];
    $deptlist{$dept} += 1;
}
foreach $dept (sort (keys %deptlist)) {
    print (" $dept: $deptlist{$dept}\\n" );
}

```

# | 必需有跳脫符號  
# 部門在第四個區塊  
# 同等於++

這個程式被分成兩個部分。while結構首先過濾出每一行讀取的\$dept的值，然後將在陣列%deptlist中的對應元素之計數增加。在所有的輸入行被讀取之後，foreach結構指定%deptlist中的每一個索引值給\$dept變數。\$deptlist{\$dept}現在含有每一個索引的累計總數。這樣的排序輸出展現出perl強大之處：

```

$ count.pl emp.lst
accounts : 2
admin    : 1
marketing : 4
personnel : 2
production: 2
sales    : 4

```

只用到少數幾行perl的程式碼，您就能列出各個部門人員的分布狀況。

## 20.15 正規表示式及替換

perl提供一個在UNIX系統上能找到的所有可能之正規表示式的偉大超集。（除了POSIX特別指定外）。您已經用過它們做一些樣式比對，perl不但懂得使用egrep及awk的型式，而且也知道只有在grep及sed才有的，它也有一些自己的（表20.1）。我們現在要使用他們來做替換。

### 20.15.1 s和tr函數

在perl中，用s和tr函數可以處理所有的替換，s函數相當於在sed中用s命令，tr能轉換字元相當於UNIX的tr命令動作，但是用稍微不同的語法。以下是我們如何在\$\_上使用它們：

表 20.1 使用perl額外的正規表示式序列

符 號	意 義
\w	符合一個文字字元（類似[a-zA-Z0-9_]）
\W	不符合一個文字字元（類似[^\a-zA-Z0-9_]）
\d	符合一個數字（類似[0-9]）
\D	不符合一個數字（類似[^0-9]）
\s	符合一個空白字元（類似[ ]）
\S	不符合一個空白字元（類似[^ ]）
\b	符合文字範圍
\B	不符合文字範圍

```
$ cat substitute.pl
#!/usr/bin/perl -n
s/\|/:/g ;                               # | 要跳脫符號
tr/a-z/A-Z/ ;
s/#/#~#g ;                               # 在這改變分隔符號為#
s/ +/:/g ;                               # 壓縮分隔符號前的多個間隔
print if (1..3) ;
```

所有在這裡的五個動作被執行在\$\_上，第一個s函數用:來替換|，tr改變每一個字為大寫字母。第二個s函數改變所有的/為~。在這裡我們更改分隔符號從預設的/變成#，替換/的理由是因為它需要跳脫符號而#不用。

像sed一樣，每一個perl敘述可以對剛做過動作的前一行再做另外一個動作，在第一個s函數中，|被替換為:，最後一個s函數現在能存取:，這裡的s使用egrep型式的+中介字元來刪除:左邊的空白。下面是其輸出：

```
$ substitute.pl emp.lst
2233:CHARLES HARRIS:G.M.:SALES:12-12-52: 90000
9876:BILL JOHNSON:DIRECTOR:PRODUCTION:03-12-50:130000
5678:ROBERT DYLAN:D.G.M.:MARKETING:04-19-43: 85000
```

這兩個函數同樣可以在變數上工作。在此狀況下，你必須使用運算子=~ 來執行比對工作以及!~ 做否定的比對：

```
$line =~ s/:-/~/g ;                      $line被重指定
$line =~ tr/a-z/A-Z/ ;                    這裡也是
```

s及tr也接受旗標(flags)。s接受g旗標（顯示在上面的例子）來全部替換，另一個(e)用來表示要被替換的樣式需要使用表示式來計算。tr使用了UNIX之tr指令所有選項當作旗標，包括s擠壓多個重複出現的字元，c做補數運算(complement)及d刪除字元(9.13.1)。

在`substitute.pl`中的最後一個`s`函數，可以被修改成用`\s`代替空白：

```
s/\s+:::/g ;
```

perl提供一些跳脫字元來表示空白、數字和文字的邊界（表 20.1）。您通常能使用這些字元來精簡您的正規表示式：

```
\s    一個空白字元
\d    一個數字
\w    一個文字字元
```

所有這些跳脫字元也有相對應的大寫符號可以否定它們的小寫符號的意義，例如`\D`為非數字的字元。我們已經使用了定位序列`\b`來比對在文字邊界上的一個樣式（20.5）。perl和其他工具所使用的正規表示式完整列表，請查看附錄C。

### 20.15.2 IRE及TRE功能

perl也接受在grep和sed中(15.12) 使用的IRE和TRE，例外是`{}`大括號及`()`圓括號不用被跳脫。舉例來說，您如何找出超過512個字元的這些行：

```
perl -ne 'print if /.{513,}/' foo           {和}之前沒有\
```

讓我們考慮用TRE的`\d`功能的重要應用。IP位址使用四個十進位數字，如果您要更改您的一些設定檔案，讓網路位址從`192.168.x.x`變成`172.16.x.x`，加標籤功能就變得非常有用：

```
s/192.168.(\d+).(\d+)/172.16.\1.\2/g ;      (和)之前沒有\
print ;
```

`(\d+)`代表一組數字，月比sed的`\[0-9][0-9]*\` 更精簡的形式。我們現在有兩組數字用任意字元`(.)`來隔開，這些群組以`\1`和`\2`出現在替換字串中。

如同是額外的好處，perl也使用變數`$1`，`$2`等等來記住群組樣式直到下一個重組的動作被完成，您可以在稍後的程式中再重新使用它們：

```
if (/(\d+)\.(\d+)\.(\d+)\.(\d+)/) {
    if ($1 < 127) {
        print "Host Address is $2.$3.$4\n" ;
    }
    elsif ($1 < 192) {
        print "Host Address is $3.$4\n" ;
    }
    else {
        print "Host Address is $4\n" ;
    }
}
```

這段程式找出含有IP的行找出，並取出四個元件。它應用23.1所討論的規則，決定主機位址的組成。注意到群組樣式擴展到目前樣式的範圍和使用在隨後的敘述中。指定一個IP位址到變數\$\_的開頭處來測試這個程式。



Note

\d表示一個數字，\s驗明一個空白字元，\w是文字字元和\b在文字範圍內符合的樣式。它們的大寫字母類似小寫字母的相反。

### 20.15.3 在同一個地方編輯檔案

perl能自己編輯和重新寫入原來的輸入的檔案，以取代寫入到標準輸出或一個分開檔案，用sed時，您必須轉向到一個暫存檔案中，然後再重新命名成原始的檔案。對於一群的檔案，您應該要使用for迴圈。perl不是這樣，-i選項能在同一個地方(in-place)編輯多個檔案：

```
perl -p -i -e "s/<B>/<STRONG>/g" *.html *.htm
```

在所有的HTML檔案中，改變所有行的<B>為<STRONG>。檔案自己被修改成新輸出。如果在同一個地方編輯似乎是一個大膽的嘗試，您可以在開始動作之前備份以前的檔案：

```
perl -p -i.bak -e "tr/a-z/A-Z/" foo1 foo2 foo3 foo4
```

在把每一個檔案中所有大寫字母轉成為小寫字母之前，它首先備份foo1到foo1.bak，foo2到foo2.bak諸如此類。

## 20.16 檔案處理

到目前為止，我們已經從UNIX命令列明確的輸入檔案名稱。perl也提供低階的檔案處理函數，讓您將資料流的來源與目的固定寫死(hard-code)在手稿中。開啟一個可被讀取的檔案像這樣：

```
open (INFILE, " /home/henry/mbox " );
```

**不要忘了引號！**

INFILE在此處是一個檔案mbox（如果路徑沒有被使用，檔案會被假設位於目前的目錄中）的filehandle（一個速寫的表示法）。之後的perl敘述將用檔案物件而不是檔名來存取檔案。這裡的優點是，如果您改變檔名，您只需修改檔案物件定義一次。

開啟一個檔案來寫入使用類似shell的運算符號>和>>並依照它們常用的意思：

```
open (OUTFILE, ">rep_out.lst" );
open (OUTFILE, ">>rep_out.lst" );
```

perl的檔案物件也能結合管線來使用。對shell的程式設計師而言，這些敘述的意義相當顯著：

```
open (INFILE, "sort emp.lst |" );
open (OUTFILE, "| lp" );
```

從sort的輸出來做輸入  
輸出到列印佇列

下一個手稿將輸入和輸出檔案檔名固定寫死 (hard-code)，它最後也會關閉檔案：

```
$ cat rw.pl
#!/usr/bin/perl
open (FILEIN, "desig.lst") || die ("Cannot open file" );
open (FILEOUT, ">desig_out.lst" );
while (<FILEIN>) {
    print FILEOUT if ($. < 4 );
}
close (FILEIN) ;
close (FILEOUT) ;
```

# 只要檔案還有資料可以讀入  
# 也能使用(1..3)

while(<FILEIN>)敘述從FILEIN檔案物件代表的檔案中一次讀入一行並存入\$\_中。  
每當<FILEIN>敘述被執行時，下一行就被讀取。您能用這方法讀取和列出單一行：

```
$_ = <FILEIN> ;
print ;
```

指定到\$\_  
print預設使用\$\_

注意print使用檔案物件當成可選擇的參數，當您指定它時，輸出被寫到那個檔案中。在手稿rw.pl中，print寫進\$\_時直接會被輸出到desig\_out.lst檔案中，即檔案物件FILEOUT指定的檔案。

雖然您在終止手稿前沒有關閉檔案，perl本身會關閉它們。close敘述也會把指標指向檔案的開頭處，以免萬一您稍後在程式中又決定要重新開啟此檔。當我們執行上述手稿沒有加參數時，這次不會輸出到終端機上，而會到檔案desig\_out.lst中。



Tip

如果許多print敘述必須寫到同一個檔案物件（例如，FILEOUT），您可使用select(FILEOUT)指定此檔案物件為預設；在此狀況下，之後的print敘述就不需要使用FILEOUT參數。

## 20.17 檔案測試

perl有一個精心製作的檔案測試系統，它掩蓋了Bourne shell及某些方面甚至find命令的光彩。下面的敘述是測試一個檔案的大部分共同屬性：

```
$x = "rdbnew.lst" ;
print "File $x is readable\n" if -r $x ;
print "File $x is executable\n" if -x $x ;
print "File $x has non-zero size\n" if -s $x ;
print "File $x exists\n" if -e $x ;
```

```
print "File $x is a text file\n" if -T $x ;
print "File $x is a binary file\n" if -B $y ;
```

perl檔案測試功能還要更進一步；它能非常精確告訴您檔案的修改和存取時間。下面的手稿使用了類似C及awk的printf敘述及其格式符號，它可查出小於2.4小時前所修改過的檔案。

```
$ cat when_last.pl
#!/usr/bin/perl
# Finds out files less than 2.4 hours old
foreach $file (`ls`) {
    chop ($file) ;
    if (($m_age = -M $file) < 0.1) {          # 一天的十分之一，換言之是2.4小時
        printf "File %s was last modified %0.3f days back \n", $file, $m_age ;
    }
}
```

**-M \$file**會傳回**\$file**從最後被修改到現在的時間並小時為單位，這是perl的一般功能（一個借用自C的觀念），讓您可以同時進行測試(< 0.1)和指定(**\$m\_age = ...**)的工作。讓我們來觀察這項輸出：

```
$ when_last.pl
File bf2o.sh was last modified 0.063 days back
File profile.sam was last modified 0.082 days back
File when_last.pl was last modified 0.000 days back
```

似乎最後一個檔案才剛剛被修改過；小數點右邊三個數還不夠用。如果您要知道更精確的時間，您必須增加printf格式的長度。

除了測試檔案的屬性之外，perl能輕易的處理檔案和目錄。它使用chmod, chown, chgrp, chdir（像是cd），mkdir, rmdir, rename（像是mv），link, unlink（像是mv）和umask的同樣方法，像檔案物件，它也能用目錄的檔案物件來開啟目錄。

## 20.18 子程式

perl處理子程式，其類似程序和函數能夠傳回值。子程式被&符號加上子程式名稱來被呼叫。子程式的參數被存放到陣列@\_中。子程式內部的變數可以被宣告為local，如果這些變數不想被呼叫它的程式看到。

很多的應用需要使用者提供使用者名稱和密碼，因為這表示要執行同樣的程式碼兩次，子程式就變成是最理想的候選人。以下的程式使用子程式take\_input()，它接受命令提示字串當參數，驗證輸入的文字字元和傳回被輸入的值：

```
$ cat input.pl
#!/usr/bin/perl
system (" tput clear ") ;                                # 執行UNIX命令
$username = &take_input (" Oracle user-id: " ) ;
```

```

$password = &take_input ( "Oracle password: ", "noecho" );
print "\nThe username and password are $username and $password\n" ;
system ( "sqlplus $username/$password @query.sql >/dev/null" );

sub take_input {
    local ($prompt, $flag) = @_ ;                # @_儲存子程式的參數
    while (1) {                                    # (1)是表示總是為真值
        print ( "$prompt" ) ;
        system( "stty -echo" ) if ( @_ == 2 ) ;    # Echo模式關閉
        chop ($name = <STDIN> ) ;
        system( "stty echo" ) if ( @_ == 2 ) ;    # Echo模式開啟
        last if $name =~ /\w/ ;                  # 跳出迴圈如果$name 有至
    }                                              # 少一個的文字字元
    return $name ;                                # 只有 $name也一樣執行
}

```

子程式的參數被讀進及送到系統陣列`@_`，然後再重新指定給兩個區域變數`$prompt`和`$flag`。子程式只用來檢查有多少參數被輸入(`@_ == 2`)，當您傳送兩個參數給它，UNIX的`stty`命令在密碼輸入期間會顯示空白。

`last`是perl中等同於shell的`break`敘述用來離開一個迴圈（perl也使用`next`來代替`continue`）。當至少有一個輸入的文字字元時，在此的迴圈即可跳出，這也是您能確保密碼不會被顯示出來的方式：

```

$ input.pl
Oracle user-id: !@#$$%^&*                      沒有文字字元
Oracle user-id: scott
Oracle password: *****                       密碼沒有顯示
The username and password are scott and tiger
. . . . . 執行SQL*Plus的query.sql手稿 . . . . .

```

您應該儲存經常使用的子程式在分開的檔案中，指示呼叫程式(calling program)來讀取，含有子程式的檔案可以在程式開頭加上`require`敘述。如果您將`take_input`子程式儲存在檔案`oracle_lib.pl`中，您應該做這兩件事：

- 插入敘述`require "oracle_lib.pl"`；放在呼叫程式中指定perl的直譯器（`#!/usr/bin/perl`）的下一行。
- 在含有一個或多個子程式的檔案最後，放入指令敘述`1`；。perl文件需要每一個“required”檔案在其結尾有一個真值。在perl中任何一個非零值是真值，因此`1`；是傳回真值。我們將在介紹CGI程式的小節中以圖例說明這些功能。



Note

在上面的例子中，您可以使用`$_[0]`來代替`$prompt`和`$_[1]`取代`$flag`。當您指定子程式的參數時也可捨棄不用關鍵字 `local`，萬一您希望此變數在子程式外部也可以被看見。

## 20.19 結論

這是一個寬廣和密集的章節卻只有一個應用程式，然而還有很多可以被討論。perl還有特定為了網路及程序間通訊的函數，但在這沒有用到。它的物件導向(object-oriented)工具和技術已經被忽略。perl套件也包括兩個有用的工具，亦即s2p用來轉換sed程式到perl，及a2p用來轉換awk程式。

perl是Larry Wall期望一個程式設計者要具備的三種美德之實現，即懶惰、性急和傲慢。它真的是一個寶藏的技术，每天都有一些人發現新的功能。UNIX精神存在perl中。想到那些不可或缺和迷人的UNIX功能；他們全部在perl中。perl是讓UNIX引以為傲的工具。

### ► 進階篇

## 20.20 用perl設計CGI程式 概要

當您在網頁瀏覽器(Web browser)填滿一個表單和按下送出鈕，瀏覽器將表單資料(formdata)傳送到網頁伺服器的另一個盡頭。一個網頁伺服器本身不會繼承任何能力來處理這資料，所以它又把資料傳給外部的應用程式。應用程式取得送來的資料和執行一些工作。它能處理資料庫的新增、修改和刪除資料，或是查詢，和送回搜尋的結果。網頁伺服器在這扮演像一個與應用程式溝通的管道，即**共用閘道介面(Common Gateway Interface CGI)**，用來傳送或接收資訊給(gatewaying)應用程式。

一個CGI程式需要做一些表單資料的過濾動作，像是分離它們的變數和數值，轉換經過編碼的字元到ASCII。這程式通常必須產生以所有標籤運作的HTML文件並送出此文件到瀏覽器上。這程式可能被任何語言寫成，包括C, Java, shell或perl。當它需要文字分析(parsing)功能時，perl的過濾能力就沒有其他工具比得上，這就是為什麼CGI程式會選擇perl語言。

剩下的部分，我們將使用perl來做兩件事，即過濾表單資料和產生HTML。我們將展示更多perl的功能，它們可被用來開發一個小型的應用程式，並利用其眾所周知的過濾能力。

### 20.20.1 了解HTML的表單

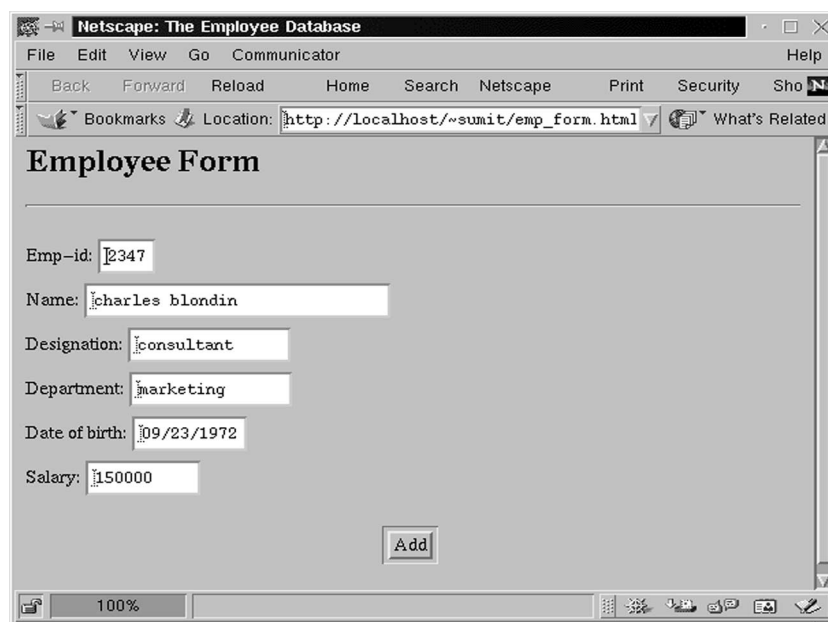
在我們CGI的應用上，我們將使用類似的員工資料庫(15.1)，但這次我們將在網頁瀏覽器上存取它們。在您開始這個探險之前，您必須知道一些HTML(14.11)以瞭解它的標籤的意義。當它送回資料到瀏覽器上時，您應該能讓CGI perl程式來產生正確的標籤。



HTML碼表示在圖20.2用**<form>**標籤指定表單。這個**action**標籤屬性使用一個URL（譯註：資源定址器：Uniform Resource Locator）指到一個在伺服器上的perl程式（emp\_add.pl）。這程式新增一行到資料庫中。接受使用者輸入的表單顯示在圖20.1中；它的程式碼顯示在圖20.2中。

每一個HTML文件含有一些標頭碼（第一到第五行）和一些尾碼（最後兩行）。因為perl必須要在大部分的CGI應用中能產生這些行，我們將建立兩個子程式來用在我們的CGI程式中。

圖 20.1 emp\_form.html：用Netscape來看HTML表單

The image is a screenshot of a Netscape browser window titled "Netscape: The Employee Database". The address bar shows the URL "http://localhost/~sumit/emp\_form.html". The main content area displays a form titled "Employee Form". The form contains six text input fields: "Emp-id:" with the value "2347", "Name:" with "charles blondin", "Designation:" with "consultant", "Department:" with "marketing", "Date of birth:" with "09/23/1972", and "Salary:" with "150000". Below these fields is an "Add" button. The browser's status bar at the bottom shows "100%".

HTML文件本身含有一個簡單的表單，包圍在 **<form>**和**</form>** 標籤間。這裡有六個文字匣，即為了接受員工資料庫的六個區塊的輸入。這些值進入到這些區塊要符合它們對應的變數名稱，包括empid, ename, desig等等。此**<br>**標籤必須要被放在分開每一行的文字匣中。

**<form>**標籤的**action**屬性使用本機(localhost)本身的URL。這個URL指到一個在**/cgi-bin**目錄下的perl程式emp\_add.pl（CGI程式通常被存放的目錄位置）。當標籤Add以及類型是 **submit** 的按鈕由滑鼠鍵點選送出的瞬間，這程式就會被執行。

## 20.20.2 查詢字串

就您所知的(14.10.1)，瀏覽器送出資料到伺服器經由它的需求標頭。要了解表

單資料如何被建構，考慮一個只有三個區塊名稱**empid**、**ename**和**desig**的表單（**<input>**標籤的**name**屬性）。讓我們輸入值**1234**、**henry higgins** 和**actor**到這三個區塊中。在送出的時候，瀏覽器字串和全部的資料以**name=value** 配對的方式一起進入查詢字串中，如下所示：

```
empid=1234&ename=henry+higgins&desig=actor
```

這個單一字串被送到在URL中指定的伺服器，**&**在這扮演每一對**name=value** 的分隔符號。注意到瀏覽會對空白字元編碼成一個**+**，要能用這個資料，perl必須分割這個字串兩次：第一次是取出所有的**name=value**，然後將**name** 及 **value**分開。依照方法不同而有兩個方式可以完成，下面就來跟大家介紹。

圖 20.2 emp\_form.html：一個表單文件的HTML碼

```
<html> <head>                                     ( 出現在標題欄 )
  <title>The Employee Database</title>             Appears on title bar
</head>
<body>                                              ( 以加大粗體字型顯示 )
<h1> Employee Form </h1>                          Appears in a large bold font
<hr>                                              Adds a horizontal rule
<form action="http://localhost/cgi-bin/emp_add.pl" method=get>( 加上水平尺規 )
  Emp-id: <input type="text" name="empid" size=4> <br>
  Name: <input type="text" name="ename" size=30> <br>
  Designation: <input type="text" name="desig" size=15> <br>
  Department: <input type="text" name="dept" size=15> <br>
  Date of birth: <input type="text" name="dtbirth" size=10> <br>
  Salary: <input type="text" name="salary" size=10> <br> <br>
<center>
<input type=submit value="Add">                    The "Add" button is centered
</center>                                           ( 按下 " Add " 按鈕 )
</form>
</body>
</html>
```

### 20.20.3 GET及POST：需求方法

**<form>**標籤顯示另一個屬性，即**method**。這表示資料被傳送到伺服器的方法。

通常，在上一節介紹的查詢字串以兩種方法被傳送：

- GET 這方法附加查詢字串到URL並使用**?**當分隔符號。用這個字串，URL現在看起來像：

```
http://localhost/cgi-bin/emp_add.pl?empid=1234&ename=
henry+higgins&desig=actor
```

伺服器分析到在需求標頭內的`get`敘述句，將`?`之後的資料儲存在它的環境變數`QUERY_STRING`中。這個變數能被使用在任何的CGI程式中。

- `POST` 用這個方法，瀏覽器會在這個字串前放置一個數字用來表示字串內包含的字元數。伺服器儲存這個數字在`CONTENT_LENGTH`變數中。它透過標準輸入提供字串到CGI程式中。`perl`讀取這個資料用`read`函數，讀取正好和`CONTENT_LENGTH`所指定一樣多的字元。

這方法本身以`REQUEST_METHOD`存在伺服器的環境中，我們的範例HTML表單使用了`GET`的方法。`GET`的字串大小被限制在1024個字元。如果您有很多要轉換的資料，那就使用`POST`。然而，查詢字串的結構在這兩個例子中是相同的，因此`emp_add.pl`程式應該能用這兩種方法來處理資料。這個例子無強制的理由來選擇`GET`而不是`POST`。

## 20.21 資料的處理

這個`emp_add.pl` CGI程式（稍後會顯示）必須在`QUERY_STRING`（`GET`方法）或是`STDIN`（`POST`方法）中解析資料，然後必須結合取出的資料成為單一行和新增它到一個類似資料庫的文字檔案中。對於發生的狀況有最佳的瞭解，我們將在瀏覽器的視窗上列出CGI環境變數重要的內容，CGI程式必須能建立需要的HTML來顯示這些訊息。

### 20.21.1 為頭尾建立子程式

因為所有的HTML文件有共通的標頭和尾段，首先架構兩個子程式把它們建立出來，`Htmlheader`列出標頭：

```
sub Htmlheader {
    local ($title, $h1) = @_;
    print << " MARKER ";
    <html>
    <head>
    <title>$title</title>
    </head>
    <body>
    <h1>$h1</h1>
MARKER
}
```

此地文件

變數替換開啟

這個不用縮排！

`Htmlheader`接受兩個參數並傳送到預留位置(placeholders)上的`$title`和`$h1`。當呼叫子程式時，這意味著您有指定標題和第一層標頭的選項。在這裡，我們已使用單一`print`敘述，像此地文件(here document)。在`marker`標籤周圍的雙引號，保證變數取代被開啟（`$title`和`$h1`所需求的）。

還有簡單的結尾子程式：

```
sub Htmlfooter {
print "</body>\n</html>\n";
}
```

放置這兩個子程式(另一個我們不久將會討論)在另外一個檔案web\_lib.pl中。在我們的CGI程式執行時將會需要它們，確定您加上敘述1；在檔案的結尾(20.18)，以便始終傳回一個真值。

### 20.21.2 emp\_add.pl：主要的CGI程式

在我們開始介紹第三個子程式前，讓我們看一下在URL中使用的CGI程式emp\_add.pl。它需要剛剛討論過兩個子程式來加入一行到文字資料庫中：

```
$ cat emp_add.pl
#!/usr/bin/perl
require "web_lib.pl";
open (OUTFILE, ">>/home/sumit/public_html/emp_out.lst");
&Parse(*field);
print "Content-type: text/html\n\n";
&Htmlheader("Testing Query String", "The QUERY_STRING Variable");
print "The query string is $ENV{'QUERY_STRING'}<br>\n";
print "The method of sending data to server is $ENV{'REQUEST_METHOD'}<br>\n";
;
print "THE content length is $ENV{'CONTENT_LENGTH'}<br>\n";
print OUTFILE "$field{'empid'}|$field{'ename'}|$field{'desig'}|$field{'dept'}|
$field{'dtbirth'}|$field{'salary'}<br>\n";
print "A record has been added <a href=\"http://localhost/cgi-bin/emp_query.pl\"
>Click here to see the records</a><br>\n";
&Htmlfooter;
close (OUTFILE);
```

因為這個程式必須產生HTML，它必須明確的詳加說明它的內容形式(Content-Type)，送回資料前，之後會留下一個空白行(\n\n)。HTML標頭會由Htmlheader 子程式印出。結尾會使用Htmlfooter在末端印出。

注意從open敘述知道手稿的輸出寫到檔案emp\_out.lst中。子HTTP程序傳送表單資料到伺服器，它以一般的使用者身份執行(24.14.1)。為了程序能建立這個檔案，目錄public\_html必須是眾人可寫的(用chmod 777)。這需要在剛開始就講清楚，因為當檔案被建立之時，目錄具備的是它舊的權限。

### 20.21.3 Parse子程式

Parse是此地我們需要用到的第三個子程式。Parse讓每一對name=value被當作

是關聯式陣列%field中個別的項目。注意陣列%field藉由前方加上\*以傳址方式被傳入emp\_add.pl手稿中的Parse。

在&Htmlheader的下三個print敘述在瀏覽器的視窗上顯示伺服器的環境變數內容。第四個使用檔案物件OUTFILE來增加一行到資料庫中，使用|當成區塊分隔符號。最後的print的敘述列出完成的訊息和提供一個超連結（用A HREF）到emp\_query.pl程式中。當您點選這個連結，您應該能看到檔案emp\_out.1st中所有的行（包含一個您剛新增的）。

要了解perl如何讓表單值有效的放在關聯式陣列%field中，我們需要仔細研讀Parse子程式：

```
sub Parse {
    local (*in) = @_ ;
    local ($i, $key, $val) ;
    if ($ENV{ ' REQUEST_METHOD ' } eq " GET " ) {
        $in = $ENV{ ' QUERY_STRING ' } ;
    } elsif ($ENV{ ' REQUEST_METHOD ' } eq " POST " ) {
        read(STDIN, $in, $ENV{ ' CONTENT_LENGTH ' }) ;
    }
    @in = split(/&/, $in) ;
    foreach $i (0 .. $#in) {
        $in[$i] =~ s/\+/ /g ;
        ($key, $val) = split(/=/, $in[$i], 2) ;
        $key =~ s/%(..)/pack("c",hex($1))/ge;
        $val =~ s/%(..)/pack("c",hex($1))/ge;
        $in{$key} = $val ;
    }
    return %in ;
}
```

# 區域變數  
# 注意兩個GET  
# ... 和POST  
# 查詢字串放在\$in  
# 分離成一對name=value  
# 解譯一個+成一個空白  
# 在第一個=分割  
# Name和 Value在關聯式陣列中

此處Parse以參考地址傳入一個陣列，這個陣列被拷貝到子程式中的%in。我們使用關聯式陣列%ENV來計算伺服器中三個我們剛討論過的環境變數。查詢字串被指定到變數\$in，不考慮所使用的方法。被POST的資料也會被讀入\$in中，但用read 函數從標準輸入（STDIN是檔案物件）讀入。要讀入多少字元由read的第三個參數決定（內容長度，content length）。

因為查詢字串使用&當作一對name=value的分隔符號，第一個split儲存每一個這樣的pair（一對）在純量陣列@in中。s函數解碼每一個+到一個空白，不論何時有空白在資料中時，編碼（空白到一個+）發生。很多字元被編碼到十六進位的字串中，因為在URL字串中它們有特殊意義。舉例來說，分隔資料區塊的元素的符號/被用來分隔URL字串中的目錄。您能看到在圖20.3中，在查詢字串送到伺服器前它被編碼成%2F。pack函數轉換這些十六進位值，回到它們原始的ASCII字元。注意到s函數如何驗證這些用一個TRE樣式%(..)表示的字元。ge旗標確保pack被解譯為表

示式而不會被當作一般文字。

圖20.3 CGI程式emp\_add.pl的輸出



foreach迴圈從陣列@in中取出每一對name=value。在解碼後，陣列的每一元素再次被分割，這次是在=兩旁的會分別儲存於變數\$key及\$val中。第一個會被設定成陣列索引(subscript)和另外的就是在關聯式陣列%in中的值。這個陣列被傳回到呼叫的程式中，注意到在這個程式中，我們使用in當成一個變數(\$in)，和一個純數列表(@in)及沒有衝突的一個關聯式陣列(%in)。

#### 20.21.4 emp\_query.pl：查詢程式

當您在圖20.1按下表單中的Add按鈕後，圖20.3中在瀏覽視窗上顯示出emp\_add.pl程式的輸出。注意超連結提供顯示資料庫中所有的行，包含一個您剛新增的。這裡點選的動作會執行emp\_query.pl程式和顯示人員列表，以當作是一個表格中的元素（圖20.4）。這裡是程式的列表：

```
$ cat emp_query.pl
#!/usr/bin/perl
require "web_lib.pl" ;
open (OUTFILE, "/home/sumit/public_html/emp_out.lst" ) ;
print "Content-type: text/html\n\n" ;
&Htmlheader( "Retrieving from Database", "Result of Query:" ) ;
print "<table border=1 bordercolor=magenta bgcolor=cyan>" ;
print "<tr><th>Emp-id</th><th>Full Name</th><th>Designation</th>" ;
print "<th>Department</th><th>Date of Birth</th><th>Salary (\$)</th></tr>" ;
while (<OUTFILE>) {
    ($empid, $ename, $desig, $dept, $dtbirth, $salary) = split (/\|/) ;
    print "<tr><td>$empid</td><td>$ename</td><td>$desig</td>" ;
    print "<td>$dept</td><td>$dtbirth</td><td>$salary</td></tr>" ;
}
print "</table>" ;
&Htmlfooter ;
```

這次檔案`emp_out.lst`而被開啟成讀取，表單的表頭被`<tr>`和`<th>`標籤印出。這個程式從`OUTFILE`中取出每一行，將它分割，然後用`<tr>`和`<td>`標籤把它印在表格列中。

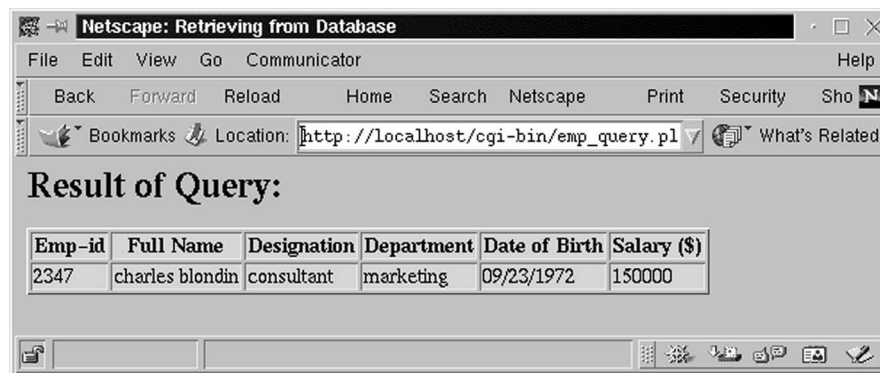
使用perl，所有看起來是如此不可思議地簡單，您不會想要使用任何其他的語言來做文字處理，至少不是在CGI的應用。在網際網路上，perl語言是最廣泛的被使用在CGI程式語言設計。



Note

CGI在網際網路上是一個安全威脅，導致伺服器管理者常常關閉不讓個別使用者處理CGI的運作。萬一您在系統上發現同樣的限制，請聯絡管理者。

圖 20.4 CGI程式`emp_query.pl` 的輸出



## 摘 要

perl是一個grep, tr, sed, awk和shell的超集。它能從命令列上以`-e`選項來被使用，但一個perl程式最好放在檔案中並將所使用的直譯器放於第一行中。所有的perl敘述用分號來做結束。

輸入是從鍵盤讀取並藉由指定檔案物件`<STDIN>`到一個變數中。perl讀入每一個被輸入的資料包含空白和新增一列字元。用`chop` 把一行的最後一個字元移除。

變數需要`$`來做指定和計算，他們不需要宣告類型而且能儲存非常高精密度的一個數字。變數值能包含跳脫序列，像是定位`(\t)`和新增一列`(\n)`，您能將字串的第一個字元`(\u)`或是全部的字串`(\U)`變成大寫。

`.` (點) 被用在字串的連結上。`x`被用來重覆字串。`substr`可以從一個字串的右邊和左邊取出字串，也可插入到一個字串中。`uc`及`ucfirst`轉換全部的參數為大寫字母和各別的第一個字元。

perl使用-n選項來設定一個讀取檔案的隱含迴圈。一個明確的迴圈也可用<>符號來設定，即表示在命令列中指定檔案名稱。while(<>)常被用在一個手稿中讀取一個檔案。

**\$.** 儲存目前的行數，範圍運算符號(..)指定一群行的運算範圍。

**\$\_** 在很多perl函數中被用來當預設值。它儲存最後被讀取的行或是最後符合的樣式。print, chop, split等指令的樣式比對和替換，預設運作在\$\_上。

perl廣泛的使用列表和陣列。當@arr被指定到一個變數時表示陣列的長度。**\$#arr**儲存陣列@arr的最後一個陣列索引。**@ARGV[]**儲存所有的命令列參數，但命令名稱以\$0來表示。

foreach構成迴圈經由一個陣列和指定每一個元素依序到一個變數中。列表也能由命令替換來提供。

split分割一個列表到許多變數或一個陣列中。**@\_**是預設陣列和空白是預設分隔符號。被分割的一行元素能用join黏合在一起。元素能從一個陣列被刪除(shift和pop)或是插入(unshift及push)。splice能在任何的陣列位置上做每一件事。

關聯式陣列使用%符號和替換陣列的陣列索引及一連串逗號分隔的值。這個陣列的陣列索引也能是一個字元字串。keys函數取出這些陣列索引，values過濾出值來。sort可以被用來做取出後的排序。

perl接受所有UNIX命令所使用的正規表示式和一些它本身所擁有的。現有，您能比對一個數字(\d)，一個文字字元(\w)，一個文字的開頭(\b)和一個空白字元(\s)。大寫字母的指令用於否定小寫字母指令的意義。

s和tr函數被使用在替換和字元轉換，類似於sed和tr的方式。運算符號 =~ 和 !~ 被用來在變數上比對正規表示式。檔案用-i選項能在同一個地方被編輯，能選擇性的被備份到另一個副檔名。

IRE和TRE跟之前介紹的工作方式相同，除了()和{}字元之前不需要加\。一組樣式可以使用\$1,\$2等等，在別處重新被產生，直到下一個重組的動作被完成。

perl使用一個檔案物件(filehandle)來存取一個檔案。一個檔案物件也能代表一個管線。print也透過一個檔案物件來寫到一個檔案中。select檔案物件敘述建立print使用預設的檔案物件。

perl的檔案測試能儲存一個檔案的時間(修改和存取)成為許多十進位的數字。

子程式以&來被呼叫，它的參數被儲存在陣列@\_中。子程式能被存在一個外部檔案，但必須有敘述1；在檔案的結尾。呼叫程式「包含有」子程式位於一個外部檔案，可用require敘述來宣告。



## 進階篇

perl是CGI程式設計時會被選擇的語言。一個perl程式能被指明在<form>標籤的action屬性中。GET方法由環境變數QUERY\_STRING傳遞資料到程式中。POST資料是經由標準輸入被傳遞。程式取出成對的name=value，把值(value)跟名稱(name)分開，及產生要被送回瀏覽器上所需的HTML碼。

## 自我測驗

20.1 這個程式有什麼問題？它應該要被列印出什麼？

```
#!/usr/bin/perl
x = 2;
print x ** 32 ;
```

20.2 將一個檔案全部行編號，藉由一個定位鍵(tab)把行號與原來行資料分開。

20.3 列出檔案/etc/passwd中所有GUID為100的行。

20.4 不使用一個迴圈來印出字串“UNIX”二十次。

20.5 您如何將一個檔案中所有字元轉換為大寫字母而不用shell的轉向？

20.6 寫一個程式，接受從鍵盤輸入的一個正整數，然後顯示從1到此正整數之間的整數數字，將它們放在分開的每一行上。

20.7 接受一個從鍵盤輸入的字串，然後印出字串的每一個字元至分開的每一行上。

20.8 要求使用者重複輸入一個數字，當使用者輸入0時，列出到目前為止所有輸入數字的總和。

20.9 接受一個從鍵盤輸入四個數字的年份，然後檢查它是否閏年。（注意：年的最後為00必須被400除盡）

20.10 在下面的程式中，至少找出四個錯誤（行碼顯示在左邊）：

```
1  #/usr/bin/perl
2  print " what is your age ?;
3  $a = <STDIN>
4  chop ($a) ;
5  if ( $a < 18 )
6      print "Not old enough to vote yet  /n " ;
7  } else {
8      print "You are old enough to vote " ;
9  }
```

20.11 寫一個程式提示使用者輸入一個字串和一個數字，依數字來印出字串幾次，每一個字串放在一個分開的行上。

## 練習

---

- 20.1 您如何用perl將一個檔案變做隔一行？
- 20.2 與其用數字編行號，代換成字母A, B, C等等在每一行的開頭處。
- 20.3 您如何以相反順序列印行？
- 20.4 您如何印出在一個檔案中剛好是第一次出現的字串？（字串和檔名是第一和第二個參數）
- 20.5 在一個檔案中，讓每一個文字的第一個字元為大寫。
- 20.6 您如何轉換一個二進位數字（提供的參數）為十進位數字？
- 20.7 找出連續出現三個一樣的字母字元（像aaa或bbb）。
- 20.8 設計一個手稿，它能列出在一個或多個檔案中的所有文字，顯示它們出現的計數（其格式是`word:count`）。
- 20.9 您如何用 `variable=value` 格式，列出一個排序過的所有環境變數列表？
- 20.10 您如何使用 `find` 和 `perl` 來刪除所有更改時間多於一年的檔案？使用這方法有何優點並跟 “`find` 配合 `-exec rm`” 比較有什麼不同？
- 20.11 將目前目錄下的所有perl手稿中，把指定翻譯程式行改成 `#!/usr/local/bin/perl`。
- 20.12 一個HTML的關閉標籤是由/開始。舉例來說，`<b>` 的關閉是`</b>`。轉換所有這些由單一文字組成的標籤（沒有屬性）成為大寫字母。（`<img src= . . . .` 將不會被轉換）
- 20.13 顯示在目前目錄下有多重連結的這些檔案的列表。
- 20.14 參考在15.12.2節在HTML文件中轉換URL的例子，並以perl程式來實現它。這次注意到`<IMG SRC>`標籤也參考到URL。

## 進階篇

---

- 20.15 寫一個CGI手稿，顯示`df` 命令的輸出到您的瀏覽器上。