附錄A

C Shell——程式語言結構

Cshell是在加州柏克萊大學，由William Joy，vi的作者，所發展出來的。Korn shell和bash均沿用了大部分C shell的特色。Linux則提供了另一個更優越的C shell，稱為tcsh。我們在第十七章中討論過C shell與環境設定相關的特色；在本附錄中我們討論它的程式語言結構。

計　算

C shell可以處理整數運算。變數值的設定可以用set或@：

```
% set x=5
% @ y = 10                                    @後面有一個空白
% @ sum=$x + $y
% @ product = $x * $y
% @ quotient = $y/$x                          除號/兩邊需要各空一格
@: Badly formed number
% @ quotient = $y / $x
```

數值的遞增可以用以下幾種方式：

```
@ x = $x + 1
@ x++
@ x ++
```

符號@後面一定要有一個空白，而等號=兩邊則不需要。運算符號（例如+, -, *, / 以及 % 符號）兩邊也必須各留一個空白。變數值可以用指令echo來檢視。

陣　列

C shell具有許多變數：有些是C shell獨具，有些則是其他shell也有的。您已經在第十七章中學會如何設定或計算這些變數，不過您可能也注意到有一個變數(path)的設定有點不同：

```
set path = (/bin /usr/bin /usr/local/bin /usr/dt/bin .)
```

上述的清單實際上是個陣列變數。上例中**path**一般被稱為一個變數，不過實際上也可以想成是具有五個元素的陣列。第一個元素可以這樣存取 **$path[1]**，第二個則是 **$path[2]**，依此類推。這個陣列包含的元素總數可以用參數 **$#** 後面加上陣列名稱來表示。以下稍微看幾個例子：

```
% echo $path
/bin /usr/bin /usr/local/bin /usr/dt/bin .
% echo $path[4]
/usr/dt/bin
% echo $#path
5
```

變數可以利用**set**敘述來設進陣列變數中，而shift也可以操作陣列：

```
% set numb = ( 9876 2345 6213 )          如同在別的shell中的set 9876 2345 6213
% echo $numb[1]
9876
% echo $#numb                            整個清單儲存在 $numb[*] 中
3
% shift numb                             使用陣列變數
% echo $numb[1]
2345
```

執行指令手稿

在預設的情況下，以C shell所寫的指令手稿會由Bourne shell來執行（Linux使用bash來執行tcsh手稿）。有兩種方式可以避免這種情況發生，利用指令csh來執行，或者在每個C shell手稿的第一行明確指定所用的命令解譯器：

```
csh script_name
#!/bin/csh
```

大部分C shell都支援較為方便的第二種形式，但如果您的版本不支援，那麼就只好使用csh 指令本身了。

if敘述式

if 敘述式的形式在這裡也有點不同。首先，關鍵字then一定要處在與if的同一行才行。其次，敘述式的結尾字是endif 而非 fi。最後，條件判斷式必須包含在成對的小括號裡面：

```
% cat filesize.csh
#!/bin/csh
# Program: filesize.csh - Converts file blocks to size in bytes
```

```
if ( $#argv != 2 ) then            # Condition tested with != instead of -ne
    echo Two parameters required
else
    @ size = $1 * 512
    echo Size of $2 is $size bytes
endif
% filesize.csh 124 tulec04
Size of tulec04 is 63488 bytes
```

數值的比較是經由類似C語法的運算子來達成 >, == 以及 != 等等，而非像Bourne shell所用的運算符號（-gt, -lt等等）。**$#argv**（而非 **$#**） 被設為傳入引數的個數。每個傳入引數可以被個別存取，利用 **$argv[1]**, **$argv[2]**，依此類推。為了維持和其他shell的相容性，C shell也允許您使用 **$1**、**$2**來存取引數。

您可以使用單行的條件判斷，而不需搭配關鍵字else：

```
if ( $#argv == 2 ) @ size = $1 * 512 ; echo Size of $2 is $size bytes
```

當條件判斷式本身是一UNIX指令時，必須用大括號括住，而非小括號：

```
if { grep “director” emp.lst } then
```

最後要記得，C shell沒有test語法或者等效的 [ ] 表示法。不過，您仍然可以測試某些檔案的屬性。利用if (-r foo) 來測試檔案**foo**是否可讀，if (! -d bar) 來測試檔案**bar**是否非目錄，等等。完整的檔案測試列表，請參考附錄D。

switch敘述式

switch敘述式（類似case）可以比對多個不同的樣式，分別加以處理，所使用的關鍵字為endsw , case 和breaksw。以下的例子為三個選單的判斷：

```
% cat menu.csh
#!/bin/csh
set choice = $argv[1]              # Option supplied as argument to script
switch ($choice)                   # Like case $choice
case 1:
     ls -l
     breaksw                       # Stops further matching
case 2:
     ps -f
     breaksw
case 3:                            # Can also use default: for the last option
     exit
     breaksw
endsw
```

case在這裡有不同的用法，並且末尾要加上冒號。breaksw會停止程式進一步比對下去，直接跳出判斷式之外。如果您忘了加上這個關鍵字，那麼接下來的所有選項都仍會被執行。這在大部分情況下是不應該發生的，所以要記得加上這樣一行以確保程式不會出錯。通常case的最後一個選項用來執行某些動作，而通常是終止程式。在這樣的情況下，最後一個變數值其實並不需要，您可以直接用關鍵字default來取代。

while和foreach迴圈

這裡有兩種迴圈，即while和foreach（而非for）。這兩種迴圈和其他shell的類似結構有三個明顯的不同：

- 條件判斷式必須用小括號括住。
- 關鍵字do沒有被使用到。
- 迴圈的結尾是利用end而非done。

先來看看while迴圈。我們會在提示符號下輸入一些字：

```
% set x = 5
% while ( $x > 3 )                          也可以使用while { true }
?   ps -f                                  C shell的第二提示字串PS2是一個問號？
?   sleep 5
? end
```

foreach迴圈也和在Bourne shell中不同，而且已被perl 模擬出來。關鍵字foreach取代for，並且關鍵字in不需要。在第18.16節的例子可以重新改寫成以下：

```
% foreach file (chap20 chap21 chap22 chap23)
?    cp $file ${file}.bak
?    echo $file copied to $file.bak
? end
```

也可以在小括號內使用檔案清單：

```
foreach item ( `cat clist` )
foreach fname ( *.c )                      目前目錄下的所有C程式
foreach fname ( $* )                       傳入指令手稿的引數
```

repeat敘述式

如果某一個指令想要重複數次，您可以使用repeat敘述式：

```
% repeat 3 date
Mon Jan 17 22:40:52 EST 2000
Mon Jan 17 22:40:52 EST 2000
Mon Jan 17 22:40:52 EST 2000
```

goto敘述式

您也可以使用goto敘述式。雖然這個敘述式現今已經很少人用了，但如果小心使用的話它仍然是個很快速的解決方法。考慮下面的手稿：

```
% cat gotoexamp.csh
#!/bin/csh
if ( $#argv == 0 ) then                                    # No arguments entered
   goto endblock
else
   grep $1 emp.lst
   exit
endif
endblock:
echo "You have not keyed in an argument"
```

exit敘述式可以確保在指令grep執行完之後中止程式執行，而不會繼續執行echo指令。如果您不寫這一行，最末尾的錯誤訊息行就一定都會被執行，不論是否有提供引數。

onintr敘述式

onintr敘述式（類似其他shell的trap）用來指定當中斷訊號發生時的處置。通常它會被放在程式一開頭：

```
% cat onintr.csh
#!/bin/csh
onintr cleanup
cut -c1-10 index > $$
cut -c21- index > $$.1
paste $$ $$.1 > pastelist
rm $$ $$.1
exit                                  # Required to stop intrusion into cleanup

cleanup:
rm $$ $$.1
echo "Program interrupted"
```

如同goto敘述式，onintr敘述式後面也是接一標籤名。當中斷發生時程式就會跳到標籤所在位置繼續執行。您可能想要忽略中斷信號繼續執行，這樣的情況您可以使用onintr後面加個 -：

```
onintr -
```

C shell在功能上已經被更優越的程式Korn shell和bash所取代。如果您想要找一個改良版的C shell，那麼使用Linux所提供的tcsh 吧。