第十二章 Sed 與 Awk 處理工具

12-1 串流編輯器 – sed

12-1-1 Sed 運作程序

串流編輯器（Stream edit, sed）如同 vi（或 vim）一樣，都是文件編輯工具，編輯命令也幾乎是相同的。但兩者最大的不同點是，vi 屬於交談式的作業方式，而 sed 是批次處理模式。圖 12-1 為 sed 的文件編輯模式，文件由檔案起頭開始，依序進入 sed 文件編輯器，另外在 sed 命令上設定有若干的編輯命令。當檔案文件依序進入 sed 命令時， sed 則會比較該文件是否有符合編輯條件，如果符合則編輯其內容，如不符合則依序讓文件通過；如此一來，文件就如同串流方式通過 sed 編輯器，並依序輸出編輯後的成果。

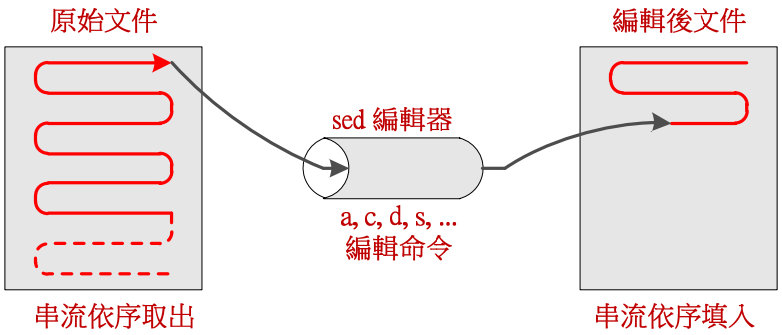

圖 12-1 sed 命令運作模式

12-1-2 sed 命令格式

Sed 如同一般命令，可在提示符號下直接輸入編輯，格式如下：

```
$ sed [option] ‘instruction’ file
```

上述命令中包含了三個主要部分，第一者為選項(option)，它供選擇 sed 的運作模式；第二部分利用兩個單引號包起來的編輯命令（如 ‘instruction’），幾乎所有 vi 環境底下的編輯命令，這裡都可以使用；第三部分為串流輸入給 sed 命令編輯的檔案名稱。如果沒有輸出轉向的話（如 > file_final），則編輯結果將直接顯示於終端機螢幕上。

(A) 【常用選項】

sed 命令的常用選項有：

- -e：當同一命令行有多個編輯命令的話，則利用 –e 表示後面緊接著一個編輯命令。
- -f script：可將編輯命令書寫成一個 script 檔案，並利用 –f 表示之。
- -n：抑制輸入列的自動輸出。

(B) 【編輯命令】

無論 sed 或 vi 的編輯命令，幾乎都與 sed 相容，常用編輯命令有：

- a (append)：位置之後插入文字。如：‘1, 3a \good’，在第一與第三行之後插入 good 文字。
- i(insert)：位置之前插入文字。如：’1, 2i \good’，在第一與第二行之前插入 good 文字。
- c(change)：以文字取代指定範圍。如：’1, 2c \Good’，則表示第一與第二行，由 Good 文字取代。
- d (delete)：刪除範圍。如：’1, 2d’，表示刪除第一與第二行。
- s (substitute)：範圍內尋找字串並置換另一字串。如：’1, 2s/Good/Text/’，在第一與第二行內尋找 Good 字串，並以 Text 字串替換它。
- p (print)：列印範圍。如：’1, 2p’，表示印出第一與第二行。
- q (quit)：離開程式。
- r (read)：讀取檔案。如：’20 r file1’，讀入 file1 檔案內容，並置入於 20 行之前。
- w (write)：寫入檔案。如：’1, 4 w file1’，將第一到第四行內容，寫入到 file1 檔案內。

以下分別說明各編輯命令的使用方法。

12-1-3 sed 替換命令

替換命令的運作模式，首先搜尋文件中某一特定文字（或稱圖樣），再將它替換成另一段文字（或圖樣），命令格式如下：

'[address]s/pattern/replacement/flags'

語法說明如下：

- address：指定命令所處理對應的是哪幾列，如 1, 3 則表示由第 1 到第 3 列，如未指定則由搜尋 pattern 結果而定。
- s：表示替換（substitution）命命。
- pattern：所欲替換的原始圖樣（或文字），可利用正規式（請參閱第四章表 4-2）表示某一共通圖樣的格式。
- replacement：替換後的新文字。
- flags：是用來修飾替換命令，可能出現的有：
  - n：一個數字（1 ~ 512），表示只當第 n 次出現該圖樣時，才進行替換的動作。
  - g：表示對圖樣空間（pattern space）中出現的所有圖樣，都會進行替換的動作；如沒有指定 g，則只會處理第一次出現的圖樣。
  - p：印出圖樣空間的內容。
  - w file_name：將圖樣空間的內容寫入檔案內。

上述各種選項中，如果沒有指定範圍，則表示處理整個檔案內容。

【A. 替換範例一】

將 RedHat 檔案(可到 www.redhat.com 網站複製練習)內 Red Hat 字樣，替換成 RED HAT，操作範例如下：

```
$ cat RedHat
The Fedora Project is a Red Hat sponsored and community-supported
open source project. It is not a supported product of Red Hat, Inc.
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.

$ sed 's/Red Hat/RED HAT/' RedHat
The Fedora Project is a RED HAT sponsored and community-supported
open source project. It is not a supported product of RED HAT, Inc.
The goal? Work with the Linux community to build a complete,
```

```
general purpose operating system exclusively from free software.
```

【B. 替換範例二】

重複上一個範例，但僅搜尋處理第一行，操作範例如下：

```
$ sed '1 s/Red Hat/RAD HAT/' RedHat
The Fedora Project is a RAD HAT sponsored and community-supported
open source project. It is not a supported product of Red Hat, Inc.
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

12-1-4 sed 刪除命令

刪除文件內某一段內容如同 vi 命令的 d 與 dd，是刪除某一行的意思；sed 有兩種方法來尋找所欲刪除的行，一者為直接表示刪除哪幾行的行號或範圍；另一者為搜尋符合條件的內容，如果符合的話則刪除該行的所有內容。命令格式如下：

```
‘[address]d’ 或
‘/pettern/d’
```

其中，address 表示刪除某一行的行號，或某幾行的範圍；pattern 為搜尋的圖樣，可利用正規式表示。

【A. 刪除範例一】

我們利用 RedHat_1 檔案作為以下 sed 刪除命令的輸入範例；第一個範例是刪除檔案內第 2、3 行，操作範例如下：

```
[tsnien@Linux-1 tools]$ cat HatRed_1
The Fedora Project is a Red Hat sponsored and community-supported

open source project. It is not a supported product of Red Hat, Inc.

The goal? Work with the Linux community to build a complete,

general purpose operating system exclusively from free software.

[tsnien@Linux-1 tools]$ sed '2, 4d' HatRed_1
The Fedora Project is a Red Hat sponsored and community-supported
The goal? Work with the Linux community to build a complete,
```

```
general purpose operating system exclusively from free software.
```

【B. 刪除範例二】

刪除 RedHat_1 檔案內的空白行，其刪除命令為 '/^$/d'，其中 ^ 表示行的開頭、$ 為行的結束位置，如果行的起頭與結束之間沒有任何字元的話，則表示該行為空白行。因此，此命令的意思是，只要找到某一空白行，便將該行刪除；操作如下：

```
[tsnien@Linux-1 tools]$ sed '/^$/d' HatRed_1
The Fedora Project is a Red Hat sponsored and community-supported
open source project. It is not a supported product of Red Hat, Inc.
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

【C. 刪除範例三】

刪除 RedHat_1 檔案內空白行與有 Red 字元的行，操作如下：

```
[tsnien@Linux-1 tools]$ sed -e '/Red/d' -e '/^$/d' HatRed_1
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

12-1-5 sed 插入/附加/變更命令

插入、附加與變更命令都是以『行』為執行對象，插入（insert）則表示在搜尋到的行之前一行，插入某一行文字；然而附加（append）則將文字放在所搜尋行的下一行；變更（change）則將所蒐尋的行，以另一行文字取代。命令格式如下：

```
附加：'[line-address]a\text'
插入：'[Line-address]i\text'
變更：'[address]c\text
```

【A. 插入範例一】

將 Big 字元插入 RedHat 檔案內有出現 Red 字元的行之前一行，操作如下：

```
[tsnien@Linux-1 tools]$ sed '/Red/i\Big' RedHat
Big
The Fedora Project is a Red Hat sponsored and community-supported
Big
```

```
open source project. It is not a supported product of Red Hat, Inc.
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

【B. 附加範例一】

重複上述範例，但附加是插入指定行的下一行，操作如下：

```
tsnien@Linux-1 tools]$ sed '/Red/a\Big' RedHat
The Fedora Project is a Red Hat sponsored and community-supported
Big
open source project. It is not a supported product of Red Hat, Inc.
Big
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

【C. 附加範例二】

將 file_1 檔案內容附加到 RedHat 檔案內的第二行後面，操作如下：

```
[tsnien@Linux-1 tools]$ cat file_1
Is this a book?
Yes, this is a book
[tsnien@Linux-1 tools]$ sed '2r file_1' RedHat
The Fedora Project is a Red Hat sponsored and community-supported
open source project. It is not a supported product of Red Hat, Inc.
Is this a book?
Yes, this is a book
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

上述範例是讀入 file_1 檔案並附加在 RedHat 檔案的第二行後面。

【D. 變更範例一】

將 RedHat 檔案內有出現 Hat 的行，以 New-Fedora 字元取代該行，操作範例如下：

```
[tsnien@Linux-1 tools]$ sed '/Hat/c\New-Fedora' RedHat
New-Fedora
New-Fedora
The goal? Work with the Linux community to build a complete,
general purpose operating system exclusively from free software.
```

12-1-6 sed 列印命令

搜尋或指定列印檔案內某些行的內容，命令格式如下：

```
-n '[address]p'
```

【A. 列印範例一】

印出 RedHat 檔案內的第 2、3 行的內容。操作如下：

```
[tsnien@Linux-1 tools]$ sed -n '2, 3p' RedHat
open source project. It is not a supported product of Red Hat, Inc.
The goal? Work with the Linux community to build a complete,
```

【C. 列印範例二】

印出 RedHat 檔案內有出現 Linux 字樣的行，操作如下：

```
[tsnien@Linux-1 tools]$ sed -n '/Linux/p' RedHat
The goal? Work with the Linux community to build a complete,
```

12-2 Awk 資料處理語言

Awk 的名字是取自 Aho、Weinberger 與 Kernighan 等三位設計者的字首，它是 Unix 系統裡非常重要的工具之一；awk 也是由檔案內容上搜尋所欲編輯的內容，因此可稱為『圖樣掃描與處理語言』(Pattern scanning and processing language)。一般 Linux 系統也都有類似 awk 的工具，大多取名為 gawk。雖然不同版本 Unix 的 awk (或 gawk) 或有些微不同，但大致上功能與語法還能夠相容，亦即利用 awk 編寫出來的系統工具，其可攜性都非常高。

命令 awk 是介於 Shell Script 與單獨命令之間的功能，它也有獨立的語法可供編寫程式，可以利用程式編輯的方法來設計命令的功能，除了如同 grep、tr、或 sed 一樣可以搜尋與編輯檔案內容，也可以像處理資料庫系統一樣，針對某些欄位的處理。一般情況下，編輯沒有規則的文件，還是利用 sed 工具較為恰當，至於格式化的檔案文件，利用 awk 就方便許多。再說，一般系統工具的輸出結果，大多有其一定規則的格式化文件，其中每一個欄位多半具有其特殊意思，處理這種文件當以 awk 最為恰當。譬如，執行 ls –l 命令的輸出結果，每一行表示一個檔案或目錄的屬性，其中第一個欄位表示存取權限、第二欄位為連結數量.....等等(如下所示)，如此具有資料庫型態的格式化，就可利用 awk 來萃取所需的資訊。

```
tsnien@Linux-1 tools]$ ls -l
總計 16
```

```
-rw-rw-r--  1 tsnien tsnien  36  2 月  27 10:04 file_1
-rw-rw-r--  1 tsnien tsnien 261  2 月  24 15:09 RedHat
-rw-rw-r--  1 tsnien tsnien 141  2 月  27 14:08 salary
```

12-2-1 Awk 命令格式

Awk 命令格式與 sed 非常相同，但它有兩種使用方法，一者為直接在命令上輸入處理動作 (與 sed 同); 另一者可將許多處理程式整合在一個命令稿上，再引入命令稿執行，其命令格式如下：

```
awk [option] 'statement {action}' input_file
awk –f program_file input_file
```

常用選項有：

- -F fs：指定欄位之間的分隔器，如 '–F ,'，表示欄位之間是以『逗號』(,) 分隔。

其他重點說明如下：

- awk 的命令參數用兩個單引號包起來。
- 處理時導入 input_file，結果顯示在標準輸出。
- 命令敘述 (statement) 可以是單一命令，也可以由多命令組合而成，甚至編寫成檔案。命令敘述多半以搜尋條件為主。
- statement 可利用正規式表示，大部分 sed 或 vi 的命令都可以使用 (不再重複介紹)。
- 當前面敘述的條件成功時，則執行 action 內的命令。

12-2-2 Awk 欄位操作

我們可以發現，處理 Unix 資料檔案都有其一定的格式，如同資料庫的記錄 (record) 一般。Awk 的處理動作就是一筆一筆記錄讀入，再比較判斷命令是否給予處理；同樣的，輸出也是一筆一筆記錄處理後，再送到標準輸出地方。如此一來，針對檔案內每一筆記錄欄位的表示，就顯得格外重要。文件中欄位位元表示方式如下：(如圖 12-2 範例)

- $0 ：為整筆記錄的所有資料。
- $1 ~ $n：為第一個欄位到第 n 個欄位。

✧ NF（Number of Fields）：總共有幾個欄位（NF = 4）。

✧ NR（Number of Records）：總共有幾筆資料（NR = 4）。

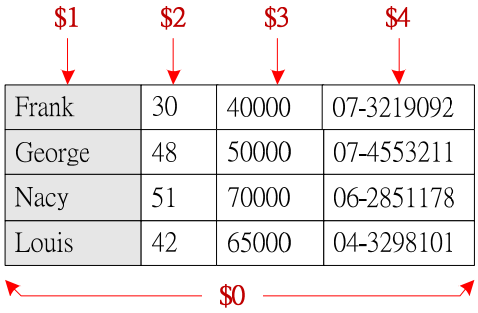

圖 12-2 文件欄位表示方式

以圖 12-2 為例，每一行表示一筆資料（或稱『記錄』），每一筆資料由 4 個欄位表示（許多 Unix/Linux 工具的輸出結果都類似此種型態）。值得注意的是，表示欄位之間的分隔符號號，如果沒有特殊指定的話，多半以一個或多個『空白格』、或『tab 鍵空格』(\t) 表示。接下來，我們用一些簡單範例來說明 awk 欄位的操作方式，相信如此應能讓讀者儘快進入狀況。

【A. awk 欄位範例一】

接下來幾個範例，我們利用 salary 檔案介紹 awk 操作，salary 是某一公司的員工檔案，每一行記錄一位員工的資料，其中包含員工姓名、年齡、底薪、以及電話號碼。我們利用 cat 命令觀察 salary 檔案，接著再利用 awk 讀出每一筆記錄並將它顯示於螢幕上，操作如下：

```
[tsnien@Linux-1 tools]$ cat salary
Frank     30        40000     07-3219092
George    48        50000     07-4553211
Nacy      51        70000     06-2851178
Louis     42        65000     04-3298101
[tsnien@Linux-1 tools]$ awk '{print $0}' salary
Frank     30        40000     07-3219092
George    48        50000     07-4553211
Nacy      51        70000     06-2851178
Louis     42        65000     04-3298101
```

上述 awk 命令 '{print $0}' 中並沒有搜尋條件，表示每一筆記錄都符合條件，並印出該筆記錄所有內容（print $0）。

【B. awk 欄位範例二】

顯示 salary 檔案中第一個欄位(姓名)與第三個欄位(底薪)，欄位之間以 tab 鍵(\t)分隔，操作如下：

```
[tsnien@Linux-1 tools]$ awk '{print $1, "\t", $3}' salary
Frank       40000
George      50000
Nacy         70000
Louis       65000
```

【C. awk 欄位範例三】

印出 salary 檔案內，員工電話於高雄地區（07）的姓名及電號，操作如下：

```
[tsnien@Linux-1 tools]$ awk '/07-/ {print $1, "\t", $4}' salary
Frank       07-3219092
George      07-4553211
```

其中 /07-/ 為搜尋敘述（statement），找出具有 07- 字串的記錄（即是『行』），才經過後面動作（action）的處理。

【D. awk 欄位範例四】

員工中，年齡如超過 45 歲者，底薪增加 5000 元，並印出處理後員工的薪資及姓名，其中欄位之間用分號（；）分隔，操作如下

```
[tsnien@Linux-1 tools]$ awk '$2 > 45 {print $1, ";", $3 + 5000}' salary
George ; 55000
Nacy ; 75000
```

12-3 Awk 搜尋敘述與動作

圖 12-3 為 awk 處理程式的概念圖，其指令格式為 'statement {action}'，其中包含搜尋條件敘述與處理動作。開始執行時，awk 依序由輸入檔案每次讀取一行（一筆記錄），經由搜尋敘述檢蒐是否符合條件，如果不符合則放棄該行，並繼續搜尋下一行；如果符合條件的話，則經由指定『動作』處理後，再選擇是否輸出到其他檔案或螢幕。以下將介紹 awk 有哪些搜尋敘述與處理動作。

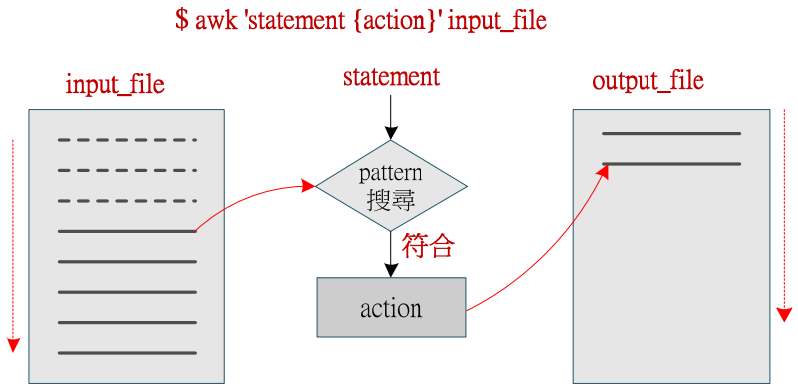

圖 12-3 Awk 的處理程式

12-3-1 Awk 搜尋敘述

Awk 搜尋方法比 sed 活用許多，sed 是以串流方式經過檢蒐器，而 awk 可以前後搜尋該行多次，因此 awk 的搜尋敘述會複雜許多，但也不會偏離以下搜尋敘述太多（每一版本的搜尋敘述並非完全相同）：

- **字串搜尋**：搜尋某一符合字串的敘述，可以使用明文表示，也可用正規式表示某一圖樣空間（pattern space）的文字。
- **比較符號**：比較兩個元素的敘述，可以大於（>）、大於等於（>=）、小於（<）、小於等於（<=）、相等（==）、不相等（!=）等等。比較元素可以是某一指定變數或該筆記錄的欄位內容，比較內容也可以是字串或數值。
- **結合運算**：超過兩個以上的比較敘述，可利用『且』（and，&&）、『或』（or，||）或『否』（not，!）將其結合成一個條件敘述。
- **指定範圍**：可以指定某一範圍的敘述，搜尋兩個指定的邊界之間的文字或數字。
- **BEGIN 與 END 敘述組合**：BEGIN 是指定 awk 處理程式之前所需執行的某些敘述；END 是 awk 處理後，再繼續處理的一些後續敘述。

接下來，我們用一些簡單範例來驗證以上的搜尋敘述。

【A. Awk 字串搜尋範例】

由 salary 檔案（前一小節的範例）中搜尋，是否有 Nacy 字樣的記錄，如有則印出該記錄內容，操作範例如下：

```
[tsnien@Linux-1 tools]$ awk '/Nacy/ { print }' salary
Nacy        51          70000       06-2851178
```

【B. Awk 比較符號範例】

列印出年齡小於 50 歲員工的姓名、年齡與薪資（由 salary 檔案），操作如下：

```
[tsnien@Linux-1 tools]$ awk '$2 <50 {print $1, "\t", $2, "\t", $3}' salary
Frank       30          40000
George      48          50000
Louis       42          65000
```

【C. Awk 結合運算範例】

列印出年齡小於 50 歲而且薪資高於或等於 50000 元員工的姓名與電話。

```
$ awk '($2 <50) && ($3 >= 50000) {print $1, "\t", $4}' salary
George      07-4553211
Louis       04-3298101
```

【D. Awk 指定範圍範例】

列印出薪資介於 50000 與 70000 員工的資料。

```
[tsnien@Linux-1 tools]$ awk '/50000/, /70000/ {print}' salary
George    48          50000       07-4553211
Nacy        51          70000       06-2851178
```

其實，上述範例並非僅比較薪資欄位，awk 每讀取一筆資料，都會比較所有欄位，如果任一欄位內容符合條件的話，則印出該筆資料的內容；由此可見，採用指定範圍搜尋，必須先了解記錄內各欄位資料是否有相似的地方，如有的話，則必須指明比較哪一個欄位的內容。

【E. Awk BEGIN/END 敘述範例】

列印出年齡大於 40 歲員工的姓名，搜尋之前先印出一行 “******”，搜尋完成之後在印出一行 “=====”，範例如下：

```
$ awk 'BEGIN {print "****"} $2 >40 {print $1} END {print "====="}' salary
****
George
Nacy
Louis
```

12-3-2 Awk 指定動作

基本上，awk 是以『行』(記錄) 為單位做搜尋處理，符合搜尋條件之後，指定動作也是針對『行』處理。但並非完全如此，記錄之間或許需要做整合處理的工作，如此一來 awk 的指定動作就顯然會複雜許多。本節先來介紹 awk 的基本指定動作，但當單一命令行無法得到滿意的結果時，也許需要將這些指定動作編寫成命令稿方式，這部份保留到下一節再說明。

【A. 自定與內建變數】

如同其他程式語言一樣，除了保留字集特殊符號外，都可作為變數名稱。自定變數不用宣告就可直接使用，如 price=30。變數內容可以是數值或文字字串，而且可以經由運算(數學或比較運算) 得到另一個變數的結果。Awk 也保留了一些內建變數，如下表 12-4 所示。

表 12-4 Awk 內建變數

變數名稱	意義
FS	輸入欄位分隔字元
OFS	輸出欄位分隔字元
NF	記錄（行）的欄位數量
NR	目前記錄（行）的筆數
FILENAME	輸入檔案名稱
FNR	目前檔案的記錄數目
ARGC	命令列參數的數目
ARGV	命令列參數陣列

另外，每筆資料中有用特殊變數名稱來表示欄位位置，譬如，$1 為第一個欄位、$2 是第二個欄位.....等等，並且以 $0 做為該筆資料的所有欄位資料。

【B. 數值運算與函數】

早期 awk 是在 Unix 系統上發展出來的，一般 C 語言所具有的數值運算式與函數，awk 大致上都有，如下所示：

- **基本運算式：**包含有指定（=）、加（+）、減（-）、乘（*）、除（/）、取餘數（%）、指數（^）等運算式。
- **指定運算式：**包含有：+=、-=、*=、/=、%=、以及 ^= 等運算式。
- **遞增與遞減運算：**遞增運算式為 ++；遞減為 --。
- **內定函數：**如 sin()、cos()、以及 rand() 等等數學函數。

【C. 陣列】

Awk 並不使用宣告或定義方式來產生陣列，而是為每一個陣列元素獨立設定某一數值，再自動將這些元素連結而成。另一方面，awk 陣列是屬於關聯陣列（associative array），其索引值是使用字串型態，而非數值型態。操作方式如下：

- **指定陣列：**產生並指定某一數值給一個陣列，如 stack[1] = $2，表示將第二個欄位指定到 stack 陣列的第一個元素中。
- **使用陣列：**將陣列中內容取出，如 stack[1]++，表示 stack 陣列第一個元素遞增。
- **字元索引陣列：**利用字元索引陣列，如 count[“display”] = 5。

【D. 自定函數】

Awk 允許使用者自定函數，語法如下：

```
function function_name(list of parameter) { action_list}
```

其中，function_name 為自定函數的名稱，傳遞參數不用宣告變數型態，又利用左右大括號（{.....}）表示函數實體範圍。自定一個遞迴函數（fact()）的範例如下：

```
function fact(k) {
      if (k <= 1)
           return 1
      else
            Return k * fact(k-1)
}
```

呼叫自定函數的語法如下：(以 fact() 函數為例)

```
value = fact(5)
```

【E. 內建函數】

Awk 的內定函數，可區分為以下兩大類：

- **內建算數函數**：如同一般程式語言，有 cos(x)（餘弦函數，x 為弧度數值）、exp(x)（x 的指數值）、int(x)（最接近 x 的整數值）、log(x)（x 的自然對數值）、sin(x)（x 的正弦值）、sqrt(x)（x 的平方根值）、atan2(x, y)（x/y 的反正切值）、rand()（0 <r <1 之間的亂數）、以及 srand(x)（x 為 rand() 的亂數種子）等。
- **內建字串函數**：畢竟處理字串（或圖樣）才是 awk 最得手的工作，字串函數的功能也與其他語言相差甚多，我們將較常用的內建字串函數歸納成表 18-2。

表 18-2 Awk 內建字串函數

Awk 函數	功能說明
gsub(r, s, t)	在字串 t 中，所有（global）與正規式 r 相似的地方，都將它取代成 s；如沒有指定 t，則內定值為 $0。
index(s, t)	回傳字串 t 在字串 s 首次出現的位置，如沒有則回傳 0。
length(s)	回傳字串 s 的長度，如沒有指定，則內定值為 $0。
match(s, r)	回傳在字串 s 中，與 s 正規式符合字串的起始位置；沒有相符則回傳 0。
split(s, a, sep)	以 sep 為欄位分隔記號，將字串 s 分割並存入陣列 a 之中；如無指定 sep，則以系統 FS 取代。
sub(r, s, t)	在字串 t 中，第一個與正規式 r 相似的地方，都將它取代（substitute）成 s；如沒有指定 t，則內定值為 $0。
substr(s, p, n)	回傳一個子字串，在字串 s 中自位置 p 開始，最大長度為 n 的子字串。
tolower(s)	將所有字串 s 中所字母改變成小寫字母。
toupper(s)	將字串 s 字母改變成大寫字母。

12-4 Awk 命令稿

當一行命令敘述無法完成任務時，則可考慮將 awk 的搜尋與動作編寫成一個命令稿，再以批次執行方式處理，也許較容易達成目的。Awk 命令稿執行方式如下：

```
$ awk –f script_file input_file
```

其中 script_file 為檢索 input_file 檔案的命令稿，大部分 awk 搜尋與動作敘述都可被編寫入。檢索處理後的結果將會顯示在螢幕上，也可將它轉向輸出到其他檔案上（如 > output_file）。

12-4-1 awk 命令稿格式

某一公司的員工資料檔案（employee）包含有員工姓名、電話、時薪與工作時數，我們可利用此檔案來簡略說明 awk 命令稿的編寫方式。

【A. Awk 範例一】

編寫一個命令稿（list.awk，檔案名稱與型態可以任意命名），其功能是列印出 employee 檔案內的所有資料；處理資料之前先印出一行各欄位的名稱（BEGIN），再一筆接一筆連續印出內容；命令稿的內容及執行結果如下：

```
[tsnien@Linux-1 tools]$ cat list.awk
BEGIN {
      print "姓名", "\t", "電話", "\t\t", "時薪", "\t", "時數"
}
{ print $1, "\t", $2, "\t", $3, "\t", $4 }
[tsnien@Linux-1 tools]$ awk -f list.awk employee
姓名        電話                  時薪       時數
Frank       07-234567           412.0      25
George      04-384123           217.0      18
Nacy        06-672314           516.0      45
Louis       07-384675           311.0      32
Eva         04-243890           358.5      38
Tank        06-631289           482.5      42
```

我們可以看出基本命令稿的語法幾乎與命令行語法相同，唯一不同的是不必用兩個單引號（'...'）包起來。此程式的執行步驟如下：

(1) 步驟 1：還未讀取輸入檔案之前，首先執行 BEGIN 敘述，也就是印出姓名、電話、時薪與時數等字串。

(2) 步驟 2：讀取檔案內一行資料（即是一筆資料），並依照欄位位置將其內容填入 $1、$2、$3 與 $4 變數內。

(3) 步驟 3： awk 敘述中沒有搜尋條件，表示每一筆資料都必須依照後面指定動作執行，亦即印出各變數內容（print ...）。

(4) 步驟 4：如果檔案內還有下一筆資料，則回到步驟 2 繼續執行，否則結束讀取檔案，並執行 END 敘述內容（本範例沒有 END）。

【B. Awk 範例二】

搜尋並印出電話在高雄的員工資料，命令稿（find.awk）與執行結果如下：

```
[tsnien@Linux-1 tools]$ cat find.awk
/07-/ {print}
[tsnien@Linux-1 tools]$ awk -f find.awk employee
Frank     07-234567          412.0     25
Louis     07-384675          311.0     32
```

【C. Awk 範例三】

印出工作時數超過 30 小時員工的姓名、時薪與工作時數，命令稿（time.awk）與執行結果如下：

```
[tsnien@Linux-1 tools]$ cat time.awk
$4 >= 30 {print $1, "\t", $3, "\t", $4}
[tsnien@Linux-1 tools]$ awk -f time.awk employee
Nacy        516.0     45
Louis       311.0     32
Eva         358.5     38
Tank        482.5     42
```

本範例執行動作如下：當 awk 讀入每一資料後，首先判斷 $4（第四欄位，即是工作時數）是否超過 30，如果是則印出欄位 1、3、4 的內容；接著再讀取下一筆，直到檔案內全部讀完為止。

【D. Awk 範例四】

計算出本月每一員工的應領薪資，以及全部薪資總額，命令稿（total.awk）及操作範例如下：

```
[tsnien@Linux-1 tools]$ cat total.awk
BEGIN {print "Name", "\t", "Payment"}
{
  payment = $3 * $4
  print $1, "\t", payment
  total = total + payment
}
END {print "Total payment = ", total}
[tsnien@Linux-1 tools]$ awk -f total.awk employee
Name          Payment
Frank         10300
George        3906
Nacy          23220
Louis         9952
Eva           13623
Tank          20265
Total payment =     81266
```

簡略說明 total.awk 的運作方式，還未讀取輸入檔案(employee)之前，先印出一行'Name'與'Payment'字樣(BEGIN 敘述)。接著按照輸入檔案內資料的排列次序，每次讀入一筆記錄，並計算個人薪資 (payment = $3 * $4)，以及印出員工姓名及個人薪資，再累積計算總共薪資 (total = total + payment)。輸入檔案內資料讀取完畢，最後印出所有薪資的總合 (END 敘述)。

【E. Awk 範例五】

列出時薪超過 350 元的員工姓名與應領薪資，以及平均薪資多少，命令稿(hi-pay.awk) 及執行結果如下：

```
[tsnien@Linux-1 tools]$ cat hi-pay.awk
BEGIN {print "Name", "\t", "Payment"}
$3 >= 350 {
     payment = $3 * $4
     total = total + payment
     number = number +1
     print $1, "\t", payment
}
END {
       average = total / number
       print "High payments = ", number
       print "Average payment = ", average
}
```

```
[tsnien@Linux-1 tools]$ awk -f hi-pay.awk employee
Name         Payment
Frank        10300
Nacy         23220
Eva          13623
Tank         20265
High payments =    4
Average payment =    16852
```

【F. Awk 範例六】

印出時薪低於 400 元而且應領薪資低於 10000 元的員工姓名與應領薪資，並印出人數與其平均薪資多少。命令稿（lowpay.awk）及操作範例如下：

```
[tsnien@Linux-1 tools]$ cat lowpay.awk
BEGIN {
   printf("Name\t Payment\n");
}
($3 < 400) {
     payment = $3 * $4;
     if (payment < 10000) {
          total = total + payment;
          number = number +1;
          printf("%s\t %d\n", $1, payment);
     }
}
END {
     printf("Low payments = %d\n", number);
     printf("Average payment = %.2f\n", total / number);
}
[tsnien@Linux-1 tools]$ awk -f lowpay.awk employee
Name         Payment
George       3906
Louis        9952
Low payments = 2
Average payment = 6929.00
```

由上述 lowpay.awk 範例可以看出，awk 動作敘述(action)幾乎與 C 語言的語法相同；不僅如此，其他控制敘述、函數宣告的語法，也幾乎相同，這對我們學習 awk 語法有相當的幫助。值得注意的是，awk 搜尋敘述包含許多文件處理的特殊語法(如正規式圖樣搜尋)，因此保有自己獨特的語法，但也大多與其他過濾器工具（如 grep 或 sed）相似。

12-4-2 Awk 控制敘述

Awk 控制敘述的語法幾乎與 C 語言相同，這對我們在學習上顯得格外容易，這裡用幾個範例來說明各種控制敘述的使用方法。以下各個範例使用一個輸入檔案（course 檔案），檔案中每一筆資料（每一行）表示學生的姓名、英文、數學、程式設計、作業系統與資料結構的成績（由左到右依序排列），內容如下所示：

```
[tsnien@Linux-1 tools]$ cat course
Frank      60 70 90 80 73
George     50 40 60 70 72
Nacy       90 73 56 75 82
Louis      90 67 89 73 56
Eva        40 56 72 45 51
Tank       72 80 69 90 92
```

【A. if 控制敘述】

if 條件敘述的語法如下：

```
if (expression)
      action_1
[else
      action_2]
```

如果 expression 條件判斷成立，則執行 action_1 敘述；否則執行 action_2 敘述，其中 else 敘述選項（[...]），依照需求決定是否加入。我們利用一個範例（if.awk）來說明 if 敘述的使用方法，該程式可計算並輸出 course 檔案中每一個學生的平均分數，並標示是否及格（平均超過 70 分及格），命令稿（if.awk）與執行結果如下：

```
[tsnien@Linux-1 tools]$ cat if.awk
{
      sum = $2 + $3 +$4 + $5 +$6;
      average = sum / 5;
      if (average > 70)
            grade = "Pass";
      else
            grade = "Fail";
      printf("%s course = %d and    %s\n", $1, average, grade);
}
[tsnien@Linux-1 tools]$ awk -f if.awk course
Frank course = 74 and    Pass
```

```
George course = 58 and    Fail
Nacy course = 75 and    Pass
Louis course = 75 and    Pass
Eva course = 52 and    Fail
Tank course = 80 and    Pass
```

在 if.awk 命令稿內並沒有搜尋條件，表示每一筆資料都必須處理。Awk 依序讀取每一筆資料，並計算其總合（sum）及平均分數（average），接著再判斷是否有超過 70 分。執行當中，awk 會依序讀完所有資料，因此無需指定多少筆記錄。

【B. while 迴圈敘述】

while 迴圈語法如下：

```
while (condition)
    action
```

條件 condition 成立則執行 action 部分。以下範例是計算並印出每一學生的學其平均分數，命令稿（while.awk）與操作結果如下：

```
[tsnien@Linux-1 tools]$ cat while.awk
{
        sum = 0;
        count = 2;
        while (count <= NF) {
                sum = sum + $count;
                count++
        }
        average = sum / (NF-1);
        printf("%s average    = %d\n", $1, average);
}
[tsnien@Linux-1 tools]$ awk -f while.awk course
Frank average    = 74
George average    = 58
Nacy average    = 75
Louis average    = 75
Eva average    = 52
Tank average    = 80
```

動作敘述（action）部分如果超過一個敘述以上，則必須利用左右大括號包起來，這方面與 C 語言相同。

【C. do 迴圈敘述】

do 迴圈敘述的語法如下：

```
do
    action
while (condition)
```

如同 while 迴圈的範例，如下：

```
[tsnien@Linux-1 tools]$ cat do.awk
{
    sum = 0;
    count = 2;
    do {
        sum = sum + $count;
        count++
    }while(count <= NF);
    average = sum / (NF-1);
    printf("%s average    = %d\n", $1, average);
}
[tsnien@Linux-1 tools]$ awk -f do.awk course
Frank average    = 74
George average    = 58
Nacy average    = 75
Louis average    = 75
Eva average    = 52
Tank average    = 80
```

【D. for 迴圈敘述】

for 迴圈敘述的語法與 C 語言很類似，格式如下：

```
for (set_counter ; test_counter ; calculate_counter)
    action
```

其中 set_counter 為設定計數器的初始值；test_counter 為設定一個測試條件，條件成立則執行迴圈內 action 敘述，執行後再依照 calculate_counter 方式計算計數器的內容，大多是遞增或遞減方式。每次執行完 for 迴圈內 action 敘述後，會再重新測試 test_counter，如果不成立的話，則離開並結束 for 迴圈執行。另外，action 如果超過一個敘述的話，則必須利用左右大括號包起來。範例如下：

```
[tsnien@Linux-1 tools]$ cat for.awk
# awk for-loop example
{
```

```
        sum = 0;
        for (count=2; count <= NF; count++)
                sum = sum + $count;
        average = sum / (NF-1);
        printf("%s average    = %d\n", $1, average);
}
[tsnien@Linux-1 tools]$ awk -f for.awk course
Frank average    = 74
George average    = 58
Nacy average    = 75
Louis average    = 75
Eva average    = 52
Tank average    = 80
```

【E. break 與 continue 敘述】

我們可以利用 break 或 continue 敘述來改變 for、while 或 do 迴圈的正常運作，它們的語法大多與 C 語言相似，這裡就不再贅言了。

【F. 變數傳遞】

執行 awk 命令稿可區分為 BEGIN、action 與 END 三個程式區塊，其變數是可以相互引用的。以下範例是計算出 course 檔案內，該班及各科成績的平均值。首先在 BEGIN 區塊內設定變數，接著在 action 區塊內累計各科成績，最後再由 END 區塊內計算出各科成績的平均值（NR 為讀入記錄的筆數）。其命令稿（var.awk）與操作結果如下：

```
[tsnien@Linux-1 tools]$ cat var.awk
BEGIN {
        eng=0; math=0; prog=0; os=0; data=0;
}
{
        eng = eng + $2;
        math = math + $3;
        prog = prog + $4;
        os = os + $5;
        data = data +$6;
}
END    {
    printf("英文    數學      程式設計    作業系統    資料結構\n");
    printf("%d\t%d\t%d\t%d\t%d\n", eng/NR, math/NR, prog/NR,
                        os/NR, data/NR);
```

```
}
[tsnien@Linux-1 tools]$ awk -f var.awk course
英文    數學    程式設計    作業系統    資料結構
67      64      72      72      71
```