



## 第三十五章

# 你不能稱它為編輯

### 35.01 為何不？

有許多特殊的編輯動作與格式化動作很常用，因此我們將它儲存在 script 中。這類例子有：

- `fmt` (35.02) 與其相關的 `script` (35.03)：將沒對齊的行，重新格式化為整齊的段落。
- `recomment` (35.04)：用來重新格式化在程式和指令檔裡的註解。
- `behead` (35.05)：用來移除 Mail/New s 標題的 script。
- `center` (35.08)：將檔案裡的行置中。

此外，有許多程式提供修改檔案的方法，但本身不是編輯器：

- `split` (35.09) 與 `csplit` (35.10)：將大檔案分割成數個小檔。
- `tr` (35.11) 用來替換某些字元，你甚至可以用八進位值表示一些無法印出的字元。

- `dd` (35.06, 35.12, 35.13) 讓你在檔案中進行不同資料轉換。
- `cut` (35.14) 與 `col rm` (35.15) 切掉檔案欄位，`paste` (35.18) 則是貼回。

本章將會討論這些內容。

- TOR

## 35.02 用 `fmt` 整理文字

`fold` (43.08) 的一個致命缺點，是會將文字內容隨意的斷開，即使恰好斷在某個單字中間。其實 `fold` 是個相當原始的工具，功能僅限於讓印表機可以印出很長的一行。

而 `fmt` 功能較佳，因為它會考慮類似段落之類的語言架構。`fmt` 可以 `warp` 連續的行，而不單是將長行分開。它假設段落以空白行結束。

使用 `vi` (30.37) 編輯時，可以用 `fmt` 來整理郵件訊息或檔案內容（Emacs 有內建的行整理程式。）在 `shell` 程式，或螢幕上太長或太短的行，`fmt` 都很有用處。



---

在某些 UNIX 系統，`fmt` 是磁碟初始化（磁碟格式化）命令。所以不要隨意執行此命令！請檢查線上說明，確定你系統安裝的 `fmt` 到底是幹嘛用的。

---

`fmt` 有些不同的版本。一般而言，程式會假設下面諸項：

- 段落之間有空白行。
- 如果行已有縮排，就會維持縮排的樣子。
- 維持每行輸出約 70 個字元。有些版本還可自行設定長度。例如，`fmt - 132`（或 `fmt - l 132`）會重設檔案格式，使得每行長度不超過 132 字元。

- 從檔案或標準輸入讀取；格式化後的行會寫到標準輸出上。



fmt

隨書光碟提供免費的版本的 GNU `fmt`，當然，還有許多其他免費的 `fmt` 可供選擇。某些版本的 `fmt` 有其它選項可以處理不同結構的資料。選項 `-p` (35.04) 可以重新格式化程式的原始碼。（如果你的 `fmt` 沒有選項 `-p`，`sed` 裡的 `recomment` (35.04) 有類似功能。）選項 `-s` 在空白處分割長行，但不會將短的行合併成長行。

或者，你可以用 `sed` 與 `nroff` 來自己製造 (35.03) 簡單的版本，但是會比較慢。如果想要具備精心設計的功能（用到 `nroff` 與 `tbl` 碼），還可以做到自動格式化文字表格，排行清單，與其它許多功能。

- JP, TOR

### 35.03 `fmt` 的替代品

一旦學會 `fmt` (35.02)，就知道它的好處。不幸的是，有些 UNIX 版本沒有 `fmt`。上冊的隨書光碟提供了 GNU `fmt`。但如果要用 `sed` (34.24) 與 `nroff` (43.13) 來模擬，也不是難事。使用這兩個工具的 `macros` (43.15)，能得到較複雜且具彈性的格式。（如果你在用 `tbl` (43.15) 表格（註），可能還需要 `col` 或 `colcrt` (43.18) 來清除 `nroff` 輸出。）



fmt.sh

下面是 `script` 的內容：

```
#!/bin/sh
sed 'li\
.l1 72\
.na\
.hy 0\
.pl 1' $* | nroff
```

---

註 結合 `tbl`、`nroff` 和 `col`，只需幾個步驟就能作出 ASCII 表格。它們可以用於電子郵件或列印，而且不需要特殊的瀏覽工具。但很少人知道 `tbl` 的存在，這點我深感遺憾。它讓你製作表格時，不用考慮文字在欄位的排列位置與換行方式。而且，如果要有美觀的輸出，將同一個 `tbl` 檔案輸入 `groff` 即可 (43.16)。 - JP

它看起來會如此複雜的原因，是因為你需要改變 `nroff` 的許多預設動作。例如，它假設每頁長度是 11 吋 (66 行)，並會在檔案末加上空白行。最快速的取巧方法，就是在要重設格式的文字內容的最上端，自行加上 `nroff` 的格式 `request .pl 1` (設定每頁長度為一行)。 `norff` 也會嘗試對每行進行調整，所以你得要用 `.na` 來取消這個動作。此外，你還得要用 `.hy 0` 來取消使用連字號的動作；如果要保持與 `fmt` 的一致性，你還得把行長度設為 72，以取代 `nroff` 的預設值 - 65。這些雜七雜八的命令，都要安插在由 `sed li` 命令輸入的第一行之前。

較好的 `script` 會採用 `-nn` 選項來設定行長度，並將之轉換成 `.ll` 的 `nroff request`。

- T O R

## 35.04 `recomment` : 重整程式註解區

在一般程式語言的原始碼中，註解通常以特殊字元開頭，如下：

```
#      井字號常用於 Shell script
//     雙斜線常用於 C/C++ 程式語言
;      組合語言常用分號當註解識別符號
...

```

如果要在註解行加入更多的內容，有一點要注意，因為行會愈來愈長，可能把內容推到下一行來，這時需要再加個註解字元。

程式 `fmt` (35.02) 會整理 (`neaten`) 文字檔裡的行，但標準版的 `fmt` 不會幫助你整理程式註解行：它會混淆註解字元 (如果你的 `fmt` 有 `-p` 選項，就可處理此問題)，`recomment` 程式可用來改進 `fmt` 在這方面的不足，它可以處理 `shell script`、`awk`、`C` 的多行註解。



`recomment`

把你要整理的程式檔餵給 `recomment` 當作其標準輸入，接著它會查看第一行，找出註解字元 (註解字元由 `$cchars` 變數定義，通常是 `SPACE`、`TAB`、`#` 或 `*`)，然後 `recomment` 移除每行的註解字元，將剩餘的文字內容丟給 `fmt` 去處理，然後再使用 `sed` (34.24) 來加入註解字元。

我經常在 vi 裡搭配 filter-through (30.22) 的命令來使用 recomment，如下：

```
!)recomment          對下一空白行重新排格式
5!!recomment         對目前的行與下四行重新排格式
```

recomment 會讓 fmt 選擇註解行的寬度 (通常是 72 字元)，要使用別的寬度，你可以：

- 在命令行給寬度，就像：

```
recomment -50
```

- 設環境變數 CBLKWID，指定註解內容的最大寬度 (以字元為單位計算)。例如，在 C shell 裡使用：

```
% setenv CBLKWID 50
```

雖然 recomment 不是很完美，但總比沒有好。下面是達成此功能的部份 script。前兩個命令會先取得註解字元，然後計算其長度，接下來的三個命令移除註解字元，並整理剩餘的註解區，然後在經過處理的所有行的開頭加入註解字元：

```
expr 45.29
# GET COMMENT CHARACTERS USED ON FIRST LINE; STORE IN $ comment:
comment=`sed -n \"/^\\([${cchars})*\\.*/\\1/p\" $temp`
# GET NUMBER OF CHARACTERS IN COMMENT CHARACTER STRING:
cwidth=`expr "$comment" : \".*\"`

#RE-FORMAT THE COMMENT BLOCK. IF $widopt SET, USE IT:
colrm 1 $cwidth < $temp | # STRIP OFF COMMENT LEADER FROM LINES
fmt $widopt | # RE-FORMAT THE TEXT, AND
sed "s/^/$comment/" # PUT THE COMMENT CHARACTERS BACK
```

如果你的系統沒有 colrm (35.15)，把倒數第三行程式碼用 cut (35.14) 改寫如下：

```
cut -c`expr $cwidth +1`- < $temp | #STRIP OFF COMMENT LEADER
```

它利用 cut -c4- 來達到 colrm 1 3 的效果。

某些版本的 `fmt` 有 `-p` 選項可以做相同的事。但是與 `recomment` 不一樣的是，你要用 `fmt -p` 告訴註解的前置字元是什麼。例如，下面是 C 程式的開端。前置字元是 `*`：

```
% cat prog.c
/*
 * This file, load.cc, reads an input
 * data file
 * Each input line is added to a new node
 * of type struct Node.
 */
% fmt -p '*' prog.c
/*
 * This file ,load.cc, reads an input data file. Each input line is
 * added to a new node of type struct Node.
 */

-JP
```

## 35.05 用 `behead` 移開 Mail/News 標題

當你儲存或送出 Mail 或 News 訊息時 (1.33) 時，有時會想要去移除標題行 (Subject:、Received: ... 等等)，這個小 script 會處理一個或多個檔案，並寫到標準輸出，下面是一些範例：

- 對於已經儲存的訊息，在 Shell 提示符號下輸入下列指令：

```
% behead msg* | mail -s "Did you see these?" fredf
```

- 利用管線 (pipe) 儲存文章並移除標題：

```
What now? [ynq] s- | behead > filename
```

下面的 script，是轉移自 Arthur David Olson 的程式而來：

```
behead
```

```
#!/bin/sh
case $# in
0) exec sed '1,/^$/d' ;;
```

```
*) for i
    do sed '1,/^$/d' "$i"
    done
    ;;
esac
```

這個 script 假設 Mail 的標題行與信件內容間有一空白行，因此程式刪除信件的開始直到空白行。

- JP

## 35.06 用 dd 屠殺低階檔案

dd 實在很好用。當你想刪除檔案的前 100 個字元，下面命令會幫你完成；當然，你可以利用管線餵資料給 dd，或使用 if= (input file) 選項，直接給它一個檔案：

```
% dd bs=100 skip=1
```

或者你也可以：

```
% dd bs=1 skip=100
```

dd 通常以 512 -byte 為區塊大小來讀寫資料，用 ibs= 選項可以改變輸入區塊的大小，obs= 選項可以改變輸出區塊的大小，若要同時設定兩種區塊大小，請使用 bs=；用 skip= 可以設定從檔案的開始要跳過多少區塊。

為什麼你會想這樣做？在 22.17 有一個關於加密檔案的範例，在 20.06 說明透過網路使用磁帶機時，使用 dd 的時機；在 35.12 說明轉換 ASCII 與 EBCDIC 檔案；而 35.13 說明 dd 更進一步的用法。

-TOR

## 35.07 offset : 文字縮排

不知道你的印表機的起印位置是否太靠近左邊界？如果只有你一個人用印表機，你當然可以調整紙張位置，但是，印表機通常是共用的資源，這表示如果你調整紙張位置，可能就會有人向你抱怨他精心調整過的格式被你破壞了。所以，你可能會想到乾脆縮排文字，看的順眼，印的舒服。下面的程式可以幫你，它讀取檔案或標準輸入，然後寫到標準輸出，預設值是向右縮排 5 個空白字元。例如，用下列指令可以把 graph 檔案送到 lp 印表機，並縮排 12 個空白：

```
% offset -12 graph | lp
```

還有更簡單的方法來達成（例如，用 `awk` (33.11)）。這程式用到了 Bourne shell 的 `case` 敘述，或許你還有更好的主意：



offset

```
#!/bin/sh

# GET INDENTATION (IF ANY) AND CHECK FOR BOGUS NUMBERS:
case "$1" in
-[0-9]|-[0-9][0-9]) indent="$1"; shift ;;
-*) echo "`basename $0`: '$1' isn't -number or is >99." 1>&2; exit 1 ;;
esac

# SET DEFAULT:
case "$indent" in
"") indent=-5 ;;
esac

# BUILD THE SPACES FOR sed
# FIRST case DOES MULTIPLES OF 10; SECOND case DOES SINGLE SPACES:
s="          " # TEN SPACES
case "$indent" in
-?) ;; # LESS THAN 10; SKIP IT
-1?) pad="$s" ;;
-2?) pad="$s$s" ;;
-3?) pad="$s$s$s" ;;
```

```

-4?) pad="$s$$$s$$$s" ;;
-5?) pad="$s$$$s$$$s$$$s" ;;
-6?) pad="%s$$$s$$$s$$$s" ;;
-7?) pad="$s$$$s$$$s$$$s$$$s" ;;
-8?) pad="$s$$$s$$$s$$$s$$$s$$$s" ;;
-9?) pad="$s$$$s$$$s$$$s$$$s$$$s$$$s" ;;
*) echo "`basename $0`:Help! \${indent} is \${indent}!?!" 1>&2; exit 1;;
esac

case "${indent}" in
-0|-?0) ;; # SKIP IT; IT'S AMULTIPLE OF 10
-0|-?1) pad="$pad " ;;
-0|-?2) pad="$pad  " ;;
-0|-?3) pad="$pad   " ;;
-0|-?4) pad="$pad    " ;;
-0|-?5) pad="$pad     " ;;
-0|-?6) pad="$pad      " ;;
-0|-?7) pad="$pad       " ;;
-0|-?8) pad="$pad        " ;;
-0|-?9) pad="$pad         " ;;
*) echo "`basename $0`:Help! \${indent} is \${indent}!?!" 1>&2; exit 1;;
esac

#MIGHT ADD expand FIRST TO TAKE CASE OF TABS:
sed "s/^\$pad/" $*

```

首先，程式設定縮排值 (indent)，像是 -12 或 -5。然後建立 pad 這個 shell 變數，使其有足夠的空間來縮排文字 (如果 indent 是 16，pad 就是 16 個空白字元)。用一個 case 檢查 \$indent 的十位數部份。下一個 case 完成其餘的部份。最後利用 sed (34.24) 在每行的一開頭加入所需的空白，如果你的行內容本身有定位字元 (TAB)，把這程式的最後一行改成下列這個樣子：

```
expand $* | sed "s/^\$pad/"
```

·JP

## 35.08 置中行文

下面是 Greg Ubben 設計的 awk script，程式會將行置中，如果你的系統了解 `#!/(44.04、45.03)`，這程式會可以不經過 shell 而直接傳給 awk，否則，就把它放在 Bourne shell 的 script 吧！(44.14)：

```
center
```

```
#!/usr/bin/awk -f
{
    printf "%*" int(40+length($0)/2) "s\n", $0
}
```

對每個輸入的行，這程式會用建立具有指定寬度（像 `%widths`）的 `printf` 命令，這寬度值剛好夠讓該行可以置中。

用 vi 編輯時，你可以使用 `filter-through` (30.22) 來置中行，或直接在命令列使用 `center`。例如：

```
% center afile > afile.centered
% sort party_list | center | lp

-JP
```

## 35.09 用 split 在固定位置分離檔案

大部份的 UNIX 提供 `split` 工具程式，讓你肢解大型檔案成為數個小檔案，因為有些編輯器無法處理太大的檔案，或是郵件系統拒絕處理過大的郵件；例如，你有一個很大的檔案，你想用電子郵件寄給別人：

```
% ls -l bigfile
-r--r--r-- 1 jik      139070 Oct 15 21:02 bigfile
```

在這個檔執行 `split`，會將檔案拆成各個不到 1000 行的小檔：

```
% ls -l
total 283
-r--r--r-- 1 jik      139070 Oct 15 21:02 bigfile
-rw-rw-r-- 1 jik      46444 Oct 15 21:04 xaa
```

```

-rw-rw-r-- 1 jik      51619 Oct 15 21:04 xab
-rw-rw-r-- 1 jik      41007 Oct 15 21:04 xac
wc 29.06  % wc -l x*
          1000 xaa
          1000 xab
          932  xac
          2932 total

```

注意，檔名的設定是在字母“x”後加上“aa”、“ab”與“ac”等等。你可以改變 split 的動作。例如，你想要每個小檔不超過 1500 行而不是 1000 行：

```

% rm x??
% split -1500 bigfile
% ls -l
total 288
-r--r--r-- 1 jik      139070 Oct 15 21:02 bigfile
-rw-rw-r-- 1 jik      74016 Oct 15 21:06 xaa
-rw-rw-r-- 1 jik      65054 Oct 15 21:06 xab

```

你也可以改變其命名慣例，例如，用別的名稱來取代前置的“x”：

```

% rm x??
% split -1500 bigfile bigfile.split.
% ls -l
total 288
-r--r--r-- 1 jik      139070 Oct 15 21:02 bigfile
-rw-rw-r-- 1 jik      74016 Oct 15 21:07 bigfile.split.aa
-rw-rw-r-- 1 jik      65054 Oct 15 21:07 bigfile.split.ab

```

前述的方法都是可行的，但不同 UNIX 下的 split 有不同功能，split 有四種基本的變化：

1. 第一類 split 只知道如將文字檔分成小於 n 行的多個檔案。
2. 第二類 split 也稱為 bsplit，僅知道如將非文字檔分成 n 個字元檔案。你可以在上冊的隨書光碟片裡取得 bsplit。



bsplit

3. 第三類 `split` 可以自動辨識其所處理的檔案到底是文字檔還是非文字檔，然後會將文字檔分成小於 `n` 行的多個檔案，或將非文字檔分成 `n` 個字元檔案。
4. 第四類 `split` 可以處理文字檔或非文字檔，但是它不會自動辨識其作業對象。當你希望用這類 `split` 肢解非文字檔時，要特別註明。

請先參考 `manpage`，確定你用的是什麼類型的 `split`，這樣才能使用正確的語法格式。

第三類 `split` 有個問題，因為它會試著在辨識文字檔與非文字檔，但不保證一定不會搞錯。有時會把文字檔當非文字檔來用，或把非文字檔當文字檔來用，這會產生難以預期的結果。因此，如果你用的是第三類 `split`，不妨換個版本試試看。

第一類與第二類 `split` 從來不會有問題。但是如果你的環境只有其中一個的話，此時就會不適用了。如果你想拆開一個非文字檔，但只有應付文字檔的 `split`；或要拆開一個文字檔，但只有應付非文字檔的 `split`，這時就必須去找另一個版本。

第四類 `split` 最可靠，也最穩定，每個系統都該必備。這類的程式很多，甚至也有適用於 BSD UNIX 的免費版本。如果你有安裝 Perl (37.01)，用 Perl 寫 `split` 程式是相當容易的，當然，用 C 也行，但是如果你要用 C 寫一個可以跨平台的 `split`，你要把程式碼轉移到其它平台，並重新編譯成可用的執行檔。

如果你不想用 `split` 拆非文字檔，可以考慮用標準的 UNIX 工具 `dd` (35.06)。例如，假設 `bigfile` 是一個非文字檔，而你想要把它拆成每 20000 位元組一個檔案，你可以這樣做：

```

$ ls -l bigfile
-r--r--r-- 1 jik      139070 Oct 23 08:58 bigfile
for 44.16 $ for i in 1 2 3 4 5 6 7
> 9.13    > do
          > dd of=x$i bs=20000 count =1 2>/dev/null
done 45.23 > done < bigfile

```

```

$ ls -l
total 279
-r--r--r-- 1 jik      139070 Oct 23 08:58 bigfile
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x1
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x2
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x3
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x4
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x5
-rw-rw-r-- 1 jik      20000 Oct 23 09:00 x6
-rw-rw-r-- 1 jik      19070 Oct 23 09:00 x7

-JIK

```

## 35.10 依內容拆開檔案：csplit



csplit

和 `split` (35.09) 相同，`csplit` 讓你將檔案分成較小的個檔，但 `csplit` 可讓各小檔大小不同，用 `csplit` 你必須給定位置（行號或搜尋樣式）來分成各小檔。`csplit` 來自 System V，但也有許多免費的版本。

我們先看搜尋樣式的方法。假設你有個檔案，開頭是大綱標題，其後有三個段落，你想將它們拆開成不同的檔，你可以輸入下列指令：

```

% csplit outline /I./ /II./ /III./
28          各個小檔的字元數
415         .
372         .
554         .
% ls
outline
xx00      大綱標題 ...
xx01      段落 I
xx02      段落 II
xx03      段落 III

```

這個命令產生四個新檔案（大綱依舊一樣）。`csplit` 顯示每個檔案的字元數。第一個檔案 `xx00` 包括原先檔案的開始到第一個段落前，`xx01` 是段落 I，這也是為什麼要用 `xx00` 命名的原因。（即使大綱是的一開頭就是 I，`xx01` 仍然是段落 I，但 `xx00` 將會變成空檔案）

如果你不想要儲存樣式前面的內容，可以使用 `%` 符號當成樣式分隔字元：

```
% csplit outline %I.% /II./ /III./
415
372
554
% ls
outline
xx00      段落 I
xx01      段落 II
xx02      段落 III
```

樣式之前的內容將不會變成小檔案，新產生的檔由大綱開始。

我們還可以進一步修改，可使用 `-s` 選項可以避免顯示每個檔案的字元數，`-f` 選項可用來描述檔名，而不是之前的 `xx`：

```
% csplit -s -f part. outline /I./ /II./ /III./
% ls
outline
part.00
part.01
part.02
part.03
```

這裡有個小問題，在搜尋樣式中，句點（.）是萬用字元（2610）。因此像 `/I./` 的樣式可能會比對到類似 `Introduction` 這種字。所以我們需要在句點前加上反斜線；但是反斜線在 `shell` 及樣式中皆有其它意義，所以，事實上要用引號（8.14）把該樣式包起來，或在之前多加上一條反斜線（變成兩條反斜線）。沒錯，這的確很麻煩，但是如果你不記得這些，事情會更麻煩。所以我們的命令會變成下列模樣：

```
% csplit -s -f part. outline "/I\./" /II./ /III./
```

如果檔案中某個樣式會重複出現，你就可以使用搜尋樣式的方法來拆檔案。例如，你有一個檔案，內容是告訴你如何整治烤雞的 50 種方法，但是你想要把每種方法都另存成一個小檔案，仔細觀察內容後，你發現每種方法的一開頭其標題皆為 WAY #1、WAY #2.. 的形式；要拆這個檔案，有個技巧，就是使用 `csplit` 的重複參數（repeat argument）：

```
% csplit -s -f cook.fifty_ways /^WAY/ "{49}"
```

這會在第一個 WAY 分開檔案，在大括號裡的數字告訴 `csplit` 要重複拆解 49 次，請注意，我們用了一個 `^` 來比對行的開始，而且 C shell 需要在大括號（9.05）外用引號把它包裝起來。這個命令會產生 50 個檔案：

```
% ls cook.*
cook.00
cook.01
...
cook.48
cook.49
```

你往往需要重複地拆開檔案，但不知道會產生多少個小檔，可是你想要確定會拆成多少個檔案；在這種情形下，你會猜想稍微超估一下大略值（最大是 99）應該是很合理的事，但不幸的是，如果你真的這樣做，`csplit` 會發生“out of range”（超過範圍）而結束掉。此外，當 `csplit` 遇到錯誤，它會移除它所產生的任何檔然後結束（如果你問我的話，我會告訴你這是一個 bug），這就是為什麼要用 `-k` 選項的原因，它讓你在發生錯誤時，不會刪掉已產生的小檔案。

`csplit` 允許你用行數或搜尋樣式來拆檔案。例如，要在出現“Sincerely”字樣後的第五行開始拆檔案：

```
% csplit -s -f letter.all_letters /Sincerely/+5
```

這種狀況其實常常有，如果你的檔案裡有一系列的商業書信，每封信的內容不同，但是每次出現“Sincerely”的後的第五行，就是下一封信。另一個例子是來自 AT&T 的 UNIX User's Reference Manual：

```
% csplit -s -k -f routine.prog.c '%main(%' '/^}/+1' '{99}'
```

這狀況是這樣的，檔案 prog.c 包括了 C 的所有常式，而我們想要把各個常式分開放在不同的檔案裡 ( routine.00、routine.01 等等)。第一個樣式使用 %，因為我們想要忽略在 main 之前的所有內容。而下一個參數的意義是說：尋找行的一開始就是大括號的右半部 ( } )，然後在下一行拆開 ( 假設是另一個常式的開始 )，然後重複這個動作 99 次。記得使用 -k 選項保留新產生的檔案。

除了樣式之外，csplit 也接受以行號當參數：

```
% csplit stuff 50 373 955
```

這會在指定行號拆檔案；在這個例子裡，新檔案 xx00 有第 1 到 49 行 ( 總共 49 行 )，xx01 有第 50 到 372 行 ( 總共 323 行 )，而 xx02 有 373 到 954 行 ( 總共 582 行 )，xx03 有 stuff 檔其餘的內容。

如果使用“重複”參數，csplit 與 split 功能相同：

```
% csplit top_ten_list 10 "{18}"
```

這會將 top\_ten\_list 檔案每 10 行一段，拆成 19 段 ( 註 )。

-DG

## 35.11 tr 的妙用

tr 命令可用來做字元轉換，它讀取標準輸入 ( 13.01 )，並刪除或取代某個字元。

tr 最常用的用法，是把第一個字串的每個字元，換成為第二個字串的相對應字元。( 如果是一連串‘連號’的 ASCII ( 51.03 ) 的字元，就可以在頭尾中間加一個連字號來表示。 )

例如，下面的命令：

```
< 13.01      $ tr 'A-Z' 'a-z' < file          Berkeley版
```

---

註 其實並不盡然，第一個檔案僅包括九行(1-9)，其餘的包括十行，因此最好用 split -10 top\_ten\_list。

會將檔案裡所有大寫字元轉換到對應的小寫字元，並將結果顯示在標準輸出。

在 System V，你必須用方括號標示一連串的字元，例如用 [a-z] 取代簡單的 a-z。但因為方括號對 shell 而言有其它意義，所以你必須用引號把字串保護起來。

如果你不確定你所使用的版本，這裡有個測試方法。Berkeley 版本會把 [] 轉換成 A，因為 [] 不被當作是範圍運算元。

```
$ echo '[]' | tr '[a-z]' A
AA
Berkeley版
$ echo '[]' | tr '[a-z]' A
[]
System V版
```

當你轉換某個範圍到另一範圍，且範圍內有相同的字元數，此時可以不用擔心這兩個版本的差異。例如，下面的命令在兩種版本都適用：

```
$ tr '[A-Z]' '[a-z]' < file 兩種版本都可
```

Berkeley 的 tr 會轉換第一字串的 [ 換成為第二字串的 [，對 ] 而言也是一樣。System V 使用 [] 為範圍運算子 (range operator)，不管是 BSD 或是 System V，這指令在這兩個版本都能正常運作，但是這技巧只當兩個範圍內有相同字元數才會成功。

System V 版的 tr 有個不錯的特性：就是 [a\*n] 語法，其中的 n 是數字，意味著字串包含 n 個連續的 a 字元，如果你不給定 n 值，或給了 0 值，它會被當成無限大的數。當你不知道第一個字串裡有多少個字元時，這是很有用的。

如同 30.22 所描述的，這類轉換用在 vi 內解譯字串是很有用的。你甚至也可以用它來刪除某的特殊字元，用 -d 選項可以刪除輸入字串裡的一個或多個所指定的字元（被指定的字元要放在引號裡，保護它們遠離 shell），例如，下面命令將檔案裡所有的標點符號刪除。這同時也是讓你複習使用 shell quoting (8.14) 的好時機：

```
$ tr -d ",.!?;:'\"" < file
```

`tr` 的 `-s` (squeeze) 選項，會以移除由第二個參數所指定的連續相同字元。例如，下列命令：

```
$ tr -s " " " " < file
```

將會把 `file` 內的連續空白字元換成單一空白，並印到標準輸出。

我們也發現 `tr` 可以轉換其它系統的文件到 UNIX 上，如同 1.05 所述，`tr` 可已用來轉換 Mac 文字檔裡的 CR 成為 UNIX 的 newline (換行) 字元。此外，`tr` 也允許你用八進位的數值來描述字元，只要在數字之前加個 `\` 就行，例如：

```
$ tr '015' '012' < file.mac > file.unix
```

就可以把完成把 Mac 的文件檔案格式轉成 UNIX 的格式。

而下列命令：

```
$ tr -d '015' < pc.file
```

會刪除 PC 檔案裡當成換行字元的 CR+LF 中的 CR 換掉。(也可用來刪除用 `script` (51.05) 所產生的檔案的多餘 CR 字元。)

在 29.10 使用 `tr` 將句子拆成單字。

`-T OR ,JP`

## 35.12 在 ASCII 與 EBCDIC 間轉換



dd

有一次我處理 EBCDIC (譯註：IBM 特有的字元編碼方式) 磁帶時，我發現 `dd` 的神奇之處，它可以處理非 UNIX 系統所產生的磁帶 (上冊的隨書光碟包含了 `dd` 的 GNU 版本)。

你應該要多了解外來磁帶的區塊大小 (20.06)，一旦弄熟了磁帶結構，就知道如何處理特殊狀況。

例如要在磁帶設備 /dev/rmt0 讀取 EBCDIC 磁帶，然後轉換到 ASCII，並把輸出放在 was\_ibm 檔案裡：

```
% dd if=/dev/rmt0 of=was_ibm ibs=800 cbs=80 conv=ascii
```

dd 會讀取標準輸入並寫到標準輸出，但如果想要描述檔案或設備名稱，你可以使用非標準的 if= 與 of= 選項來各別描述輸入與輸出檔。

如果想要用另一種方法，請參考下列命令：

```
% dd if=was_unix of=/dev/rmt0 obs=800 cbs=80 conv=ebcdic
```

還有一個 conv=ibm 選項可以使用不同的 ASCII 到 EBCDIC 轉換表，根據 dd 的參考文件：“ASCII/EBCDIC 轉換表是取自 CACM 於 1968 年 11 月發表的 256 字元標準，而 ibm 轉換法是一個不完整的標準，只適用於 IBM 本身的列印慣例，很不幸的，這裡並不存在一個一體適用的標準。”

注意事項：

- 因為磁帶機不一定支援標準的 UNIX 磁帶區塊大小，所以你需要具備能讀取 raw device(2003)(原生設備)的能力。
- 必須知道外來磁帶的區塊大小，這樣才能使用 dd。
- 如果外來磁帶上有許多檔案，你必須使用磁帶設備名稱，並允許“結束時不倒帶(2003)”，以便讀取第一個檔案之後的資料。

最後要叮嚀的一點，就是除非另外指明，否則 dd 的所有關於大小的選項皆是用位元組來計算。你可以使用 k 來代表乘以 1024，b 來代表乘以 512(一個區塊的大小)，或 w 代表乘以 4(一個字)，你也可以用 x 來代表任意兩個數的乘法。

-TOR

## 35.13 利用 dd 做其它的轉換

除了在 ASCII 與 EBCDIC 間的轉換 (35.12) 外，你可使用 dd 來轉換：

- 固定長度到可變長度的轉換 (conv=unblock)，或反向的動作 (conv=block)。
- 轉換大寫字到小寫字 (conv=lcase)，或反向的動作 (conv=ucase)。
- 每個位元組對的順序交換 (conv=swab)。

當使用 block 與 unblock 時，必須用 cbs= 選項描述轉換暫存區的大小 (當轉換 ASCII 與 EBCDIC 時也是一樣)，對於 ascii 與 unblock 轉換，在輸出前，結尾的空白會被去掉，而在每個暫存區末會加一個新行字元。對於 ebcdic、ibm 與 block，會用空白字元填補所指定的暫存區剩餘空間。

-TOR

## 35.14 用 cut 移除欄位或欄區 (譯註)



cut + paste

System V 提供 cut 命令，讓你從一個或多個檔案裡選擇欄位 (columns) 或欄區 (field)。在上冊的隨書光碟中，我們也提供免費版本，以供你的系統所需。

你可以用 -c 選項表示依欄位切割 (cut by column)，或用 -f 表示依欄區切割 (cut by field)。(如果你沒有使用 -d 設定欄區分隔字元，欄區之間就以 tab 分隔。如果要以空白字元或其它特殊字元當成分隔字元的話，使用引號 (8.14))。

要刪除的欄位或欄區必須緊接在選項之後，中間不能有空白。在兩個不同數值間使用逗號，或用破折號來描述範圍 (如 1-10、15-20 或 50-)。

---

譯註 欄位表示表格的 column，而欄區則表示檔案中每行的 field。

cut 非常容易上手。下面是一些範例：

- 找出誰有登入，但僅列出登入名稱：

```
who 51.04          % who | cut -d" " -f1
```

- 從 /etc/passwd/ (36.03) 裡擷取使用者名稱與真實名字：

```
% cut -d: -f1,5 /etc/passwd
```

- 移除 file 的第四個欄位，並把此內容放回同一個檔案的第一個欄位：

```
paste 35.18      % cut -c4 file | paste - file
```

```
-TOR, DG
```

## 35.15 用 colrm 移除欄位

BSD 版的 cut (35.14) 是 colrm。

colrm 只能移除欄位位置，而不能用在欄區。你必須做的就只有給定開始欄位位置，並選擇性地給予結束位置。colrm 從標準輸入讀取本文。(colrm 無法直接讀取檔案，用 < 或 | (13.01) 來重導向輸入)。

如果只有給定開始欄位位置，則從給定位置開始的欄位到行末都會被刪除。若給定兩個欄位位置，從位置開始到結束之間的欄位都會被刪除。

下面命令會將 BSD ls -l 的輸出結果 (22.02)，透過 colrm 的運算之後，印出存取權限 (欄位 1 到 10) 與檔名 (欄位 45 到行末，包括檔名前的空白)：

```
% ls -l | colrm 11 44
drwxr-xr-x manages
-rw-r--r-- misc.z
-rwxr-xr-x myhead
```

下面命令會移除遠端機器名稱，這個名稱是在 `who` (51.04) 輸出結果的第 33 個欄位：

```
% who | colrm 33
-JP, TOR
```

## 35.16 用 `cols` 自動產生欄位



`cols`

如果有些程式的輸出跑到了螢幕的左邊邊界，或是需要一個以上的螢幕畫面來顯示，你可以使用 `pager` (25.03) 來過濾程式的輸出。如果行的長度很短，你可以將它重新設定為多個欄位的格式。命令 `pr` 可以產生欄位 (35.17)。但若要輸入內容在某欄位中由上而下排列，這就不大簡單了。如果你想要儘量多的欄位來填滿你的螢幕，哇... 太難了吧 – 在算出每個欄位的寬度前，你必須找到最長的資料。

有些 UNIX 提供將資料轉換為欄位的程式 – 但大部分都不提供。`cols` 就是這類用途。它會讀取你的文字內容，找出最寬的，然後用 `pr` 選項來產生儘量多的欄位填滿你的螢幕。`cols script` 也有 7 個其它名稱 – `links` (18.03) 叫做 `c2`、`c3`、`c4`、`c5`、`c6`、`c7` 與 `c8` – 它會產生 2、3、4、5、6、7 與 8 個欄位的輸出。如果你用任一名稱來呼叫這個 `script`，它會用欄位的數目來填滿你的螢幕。

例如要以欄位的格式列出拼錯的字：

```
% spell somefile | cols
word1 word2 word3 word4 word5 word6 word7 word8
word9 word10 word11 word12 word13 word14 word15 word16
```

像上面的例子，`cols` 預設是以橫向排序輸入內容，這是最快的方法。如果你需要縱向輸出每個欄位，可使用選項 `-d`。然後，`script` 會計算欄位長度並以縱向順序輸出：

```
% spell somefile | cols -d
word1 word3 word5 word7 word9 word11 word13 word15
word2 word4 word6 word8 word10 word12 word14 word16
```

script 讀取你指定的檔案；否則從標準輸入讀取。如果設定了 COLUMNS 環境變數，它會依此算出螢幕寬度；否則呼叫 tcap (41.10) 來讀取你的 termcap。(在 terminfo 系統，會用 tput (41.10) 而不是 tcap)。如果使用可調整寬度的視窗系統，script 會檢查 stty size 或 stty -g (42.04) 的輸出。

關於程式設計的細節：欄位的數目 nc 是來自程式的名稱 (c2 等) – 或者，如果你呼叫 cols，script 會使用 awk (33.11) 來找出最長的輸入行，並計算欄位數目。(case 敘述 (44.05) 測試 \$0 而加以決定)。expr (45.28) 會進行其它的計算。若沒有選項 -d，產生欄位的命令 pr 是相當簡單：

```
pr -$nc -t -w$width -ll $temp
```

\$temp 檔保留輸入內容。有使用 -d 的話，命令列較為複雜。它使用 wc -l (29.06) 來計算輸入行的數目，然後 expr 用來除以行的數目，再加上 1：

```
pr -$nc -t -w$width -l `expr\(\`wc -l < $temp\` / $nc \`) + 1` $temp
```

跳脫反引號 (\`)(45.31) 意味著 wc -l < \$temp 會先被執行。expr 的命令行會被 wc 所取代。在 -l 完成 pr 頁長度選項後，expr 的結果將被計算。如果你不想將命令擠成這樣，你可以將命令 wc 與 expr 移動到其它行，然後用 shell 變數來傳遞數值。

你可以從光碟或從線上檔案卷 (52.07) 來安裝這個 script。如果從線上檔案卷安裝，用 tar 來安裝 cols 與它的七個其它連結：

```
% tar xvf archive.tar cols c2 c3 c4 c5 c6 c7 c8
x cols, 2160 bytes, 5 tape blocks
c2 linked to cols
c3 linked to cols
...
```

-JP

## 35.17 用 pr 產生欄位裡的字

命令 `pr` (43.07) 可以整齊的印出檔案 – 印出上下對齊、檔名、日期與頁數。它也能以欄位的格式列印內容：每個欄位一個檔案，或每個檔案多個欄位。

選項 `-t` 會移去每頁的上端和底部的標題和邊界處。當你想要將資料“貼入”欄位而中間沒有中斷時，這點相當有用的。

### 每個欄位一個檔案：`-m`

選項 `-m` 會讀取命令列中的所有檔案，然後在各自的欄位中列印出來，如下：

```
% pr -m -t file1 file2 file3
The lines      The lines      The lines
of file1      of file2      of file3
are here      are here      are here
...           ...           ...
```

`pr` 使用 TAB 字元來分隔欄位。如果你覺得不好，你可以使用 `expand` (41.04) 來過濾 `pr` 的輸出。許多 `pr` 版本有選項 `-sX`，讓你將欄位分隔字元設定為單一字元 `X`。

### 每個檔案多個欄位：`-number`

這個選項會依照設定的欄位數目印出檔案。例如，`-3` 選項會以三個欄位印出檔案。這個檔案會被一行一行地讀取，直到第一個欄位填滿為止（預設是 56 行）。接著第二個欄位填滿。然後第三個欄位填滿。如果檔案還有內容沒有讀完，就從第二頁的第一個欄位開始 – 重複這些動作：

```
% pr -3 file1

Nov  1 19:44 1992  file1  Page1
```

```

Line1 here      Line57 here     Line115 here
Line2 here      Line58 here     Line116 here
Line3 here      Line59 here     Line117 here
...             ...             ...

```

這些欄位是不平衡的 – 如果在第一個欄位填滿之前讀完檔案，就不會用到其它欄位。你可以用 `-l` (頁長度選項) 來改變這種情況；看下面一節。

### 用 `-l` 來要求行穿過欄位

你想要排列資料使之跨越欄位，讓前三行越過每行的頂端印出，接著的三行是在每行的第二列印出，依此類推，就像下列情況嗎？

```

% pr -l1 -t -3 file1
Line1 here      Line2 here      Line3 here
Line4 here      Line5 here      Line6 here
Line7 here      Line8 here      Line9 here
...             ...             ...

```

使用 `-l1` (頁長度為一行) 與 `-t` (沒有標題) 選項。每“頁”會被三行填滿 (或是你所設定的行數)。你必須使用 `-t`；否則 `pr` 會忽略任何頁長度而不保留空間給標題。使用 `-t` 你的欄位就不會有標題。

如果你也想要標題，用另一個 `pr` 來過濾 `pr` 的輸出：

```

% pr -l1 -t -3 file1

Nov  1 19:44 1992  file1  Page1

Line1 here      Line2 here      Line3 here
Line4 here      Line5 here      Line6 here
Line7 here      Line8 here      Line9 here
...             ...             ...

```

`-h file1` 會把檔名放入標題。

請參考 `paste` (35.18) 與 `cols` (35.16)。當然，像 `awk` (33.11) 與 `perl` (37.01) 之類的語言也可使文字內容轉換為欄位。

-JP

## 35.18 在欄位下張貼



cut + paste

你曾經想過要把兩個（甚至三個）檔案緊緊地貼在一起？如果有 System V 的 `paste` 程式，就可以達到這個目的。

例如要從檔案 `x`、`y`、`z` 來建立一個三個欄位的檔案：

```
$ paste x y z > file
```

要讓 `paste` 讀取標準輸入，就要使用 `-` 選項，然後再對你想要的每個欄位重複按 `-`。例如，要讓老舊的 System V `ls`（它用單一欄位來列出檔案）使用四個欄位來列出檔案：

```
$ ls | paste - - - -
```

當使用 `cut` 時，“標準輸入”選項也很方便。你可以從某行的某個位置剪下文字資料，然後再貼回另一個地方。

已合併的不同資料流，預設是用 `tab` 加以分開，但你可使用 `-d` 選項來改變。不像 `cut` 的 `-d` 選項，你不需給定單一字元；你也可以給定一個字串。

字串中的字元可以是任何正規字元，或是下列的脫離字元：

```
\n  換行字元（新行）
\t  tab
\\  反斜線
\0  空字串
```

必要時使用引號 (8.14)，使字元不受 shell 約束。

選項 `-s` 可將同一檔案中連續的幾行合併在一起。例如，要將兩行合併成一行：

```
$ paste -s -d"\t\n" list
-TOR, DG
```

## 35.19 用 `join` 合併行

如果你用過資料庫，或許就知道 UNIX 的 `join` 命令的作用；請參考線上使用手冊。如果沒用過，你可能有用過 `join`：它用來結合兩個欄位格式的檔案。`join` 會搜尋檔案裡的特定欄位；當它找到和另一個欄位匹配的欄位時，它會在那個欄位處“將這幾行連在一起”。這點很容易用範例來說明。

我想對 MH 郵件系統下的幾千個郵件訊息進行摘要。使用 MH 很容易：它的命令 `scan` 會以我所需要的格式，給予每個郵件訊息中我所需的所有資訊。但是，我也必須使用 `wc -l` (29.06) 來計算每個訊息的行數。在結束時我產生兩個檔案，分別是 `scan` 和 `wc` 的輸出。在兩行中有個欄位是訊息數；我用 `sort` (36.01) 將此欄位加以排序。然後，使用 `awk '{print $1 " "$2}'` 來處理 `wc` 在類別的輸出，使之成為以逗點分隔的欄位。最後，使用 `join` 在訊息數的欄位處將兩行合併起來。

下面是我要 `scan` 輸出的檔案。欄位（訊息數、電子郵件、註解、名稱與傳送訊息）是用逗號來分隔：

```
0001,andrewe@isc.uci.edu,,Anfy Ernbaum,19901219
0002,bc3170x@cornell.bitnet,,Zoe Doan,19910104
0003,zcode!postman@uunet.uu.net,,Head Honcho,19910105
...
```

下面是 `wc` 與 `awk` 所產生的檔案，其欄位分別是訊息數和行數：

```
0001,11
0002,5
0003,187
...
```

然後，命令 `join` 會依據第一個欄位的內容，合併這兩個檔案（`-t` 告訴 `join`：欄位是用逗號分隔）：

```
% join -t scanfile wcfile
```

輸出檔如下：

```
0001,andrewe@isc.uci.edu,,Anfy Ernbaum,19901219,11
0002,bc3170x@cornell.bitnet,,Zoe Doan,19910104,5
0003,zcode!postman@uunet.uu.net,,Head Honcho,19910105,187
...
```

當然，`join` 的功能比這個例子要強多了。參考你的線上使用手冊。

-JP

## 35.20 快速參考：`uniq`

`uniq` 用來移除已排序檔案裡的相同的相鄰行，並將每行（移除相同行之後）傳送到標準輸出或第二個檔案（如果在命令列中有指定的話）。

---

### 注意：

```
% uniq file1 file2
```

將不會把 `file1` 與 `file2` 移除相同行後的各行，列印到標準輸出。它會用 `file1` 移除相同行後的各行，取代 `file2` 的內容。

---

`uniq` 常用來作過濾程式（1.30）。參考 `comm`（28.12）、`sort`（36.01）與 `sort -u`（36.06）。

**選項**

- c 每行印出一次，並在每行開頭顯示此行重複的次數。
- d 重複行印出一次，而唯一的行則不印出。
- u 只印出唯一的行（重複行都不會印出）。
- n 忽略行的前 n 個欄位。欄位以空白或 tab 來分隔。
- +n 忽略欄位的前 n 個字元。

只能選用 -c、-d 與 -u 其中一個選項。

**範例**

移除檔案 list 的重複行之後，輸出到檔案 list.new：

```
uniq list list.new
```

顯示出現一次以上的名字：

```
sort names | uniq -d
```

顯示出現次數恰好三次的行：

```
grep 27.01 sort names | uniq -c | grep " 3 "
```

- DG 摘自 UNIX in a Nutshell (SVR 4/Solaris)

## 35.21 使用 IFS 分隔字串

Bourne shell 有個 IFS（內部欄位分隔字元）shell 變數是幹嘛的？它預設有三個字元：SPACE、TAB 與 NEWLINE。這是 shell 分析命令列的地方。那又怎樣？

如果你有一行文字，想要把它分成各個欄位，IFS 變數就可派上用場。將欄位分隔字元暫時放入 IFS，使用 shell 的 set (44.19) 命令將欄位儲存在命令行參數中，然後還原舊的 IFS。

例如，下面的 shell script 是從 `stty -g` (42.04) 得來的目前終端機設定：

```
2506:5:bf:8a3b:3:1c:8:15:4:0:0:0:11:13:1a:19:12:f:17:16:0:0
```

shell 使用反引號 (9.16) 來分析由 `stty` 傳回的行。它將 `x` 儲存在 `$1`。如果基於某種原因 `stty` 失敗了，這種技巧會停止錯誤 – 在沒有 `x` 的情況下，若 `stty` 沒有產生標準輸出，shell 的 `set` 命令將會印出所有的 shell 變數。2506 儲存在 `$2`，5 儲存在 `$3`，依此類推。最早的 Bourne shell 僅能處理九個參數 (`$1` 到 `$9`)；如果你的輸入行有超過 9 個欄位，這就不是個好方法。但是，若這個 script 使用 Korn shell 則不會有此限制。

```
#!/bin/ksh
oldifs="$IFS"
#Change IFS to a colon:
IFS=:
#Put x in $1 stty -g output in $2 thru ${23}:
setx `stty -g`
IFS="$oldifs"
# Window size is in 16th field (bot counting the first "x"):
exho "Your window has ${17} rows."
```

由於不需要子行程來分析 `stty` 的輸出，上述方法的執行速度比外部命令 `cut` (35.14) 或 `awk` (33.11) 還快。

有些地方 IFS 不能使用，因為在 IFS 分隔前，shell 會在空白處分隔命令行。它不會分隔變數取代或命令取代 (9.16) 的結果。下例是分析取自 `/etc/passwd` 的行的三種方法：

```
% cat splitter
#~/bin/sh
IFS=:
line='larry:Vk9skS323kd4q:985:100:Larry Smith:/u/larry:/bin/tcsh'
set x $line
echo "case 1: \${6} is '\${6}'"
set x `grep larry /etc/passwd`
echo "case 2: \${6} is '\${6}'"
```

```

line='larry:Vk9skS323kd4q:985:100:Larry Smith:/u/larry:/bin/tcsh
echo "case 3: \$6 is '$6'"
% ./splitter
case 1: $6 is 'Larry Smith'
case 2: %6 is 'Larry Smith'
case 3: %6 is 'Larry

```

方法 1 使用變數取代，方法 2 使用命令取代；第六個欄位包含空白。方法 3 藉由命令列上的冒號，將第六個欄位分成：`$6` 是 Larry 與 `$7` 是 Smith。另一個會遇到的問題是，如果有任何的欄位是空的話（如 `larry::985:100:etc..`）— shell 會“吃掉”空的欄位，而 `$6` 會包含 `/u/larry/`。使用 `sed` 的 escaped parentheses (34.10) 來進行這類搜尋和分析工作，會解決最後兩個問題。

-JP

## 35.22 對齊欄位

撰寫本書時，我決定將所有的章節列成一個表，表中包含每個章節的行數、字元、描述、狀態碼與標題。經過 `wc -l -C` (29.06)、`cut` (35.14)、`sort` (36.01) 與 `join` (35.19) 的處理之後，我得到一個看來像下面的檔案：

```

% cat messfile
2850 2095 51441 ~BB A sed tutorial
3120 868 21259 +BB mail - lots of basics
6480 732 31034 + How to find sources - JIK's periodic posting
..900 lines
5630 14 453 +JP Running Commands on Directory Stacks
1600 12 420 !JP With find, Don't Forget -print
0495 9 399 + Make 'xargs -i' use more than one filename

```

哇... 還是很難看懂。各個欄位需要對齊。`awk` (33.11) script 可以幫助你：

```
% cat cleanfile
2850 2095 51441 ~BB A sed tutorial
3120 868 21259 +BB mail - lots of basics
6480 732 31034 + How to find sources - JIK's periodic posting
    ..900 lines
5630 14 453 +JP Running Commands on Directory Stacks
1600 12 420 !JP With find, Don't Forget -print
0495 9 399 + Make 'xargs -i' use more than one filename
```

下面是我使用的 script，以及輸入的命令：

```
% cat neatcols
{
printf ("%4s %4s %6s %-4s\n", \
    $1, $2, $3, $4, substr($0, index($0,$5))
}

% awk -f neatcols messfile > cleanfile
```

你可以使用這個 script 來對齊各個欄位。如果你不知道 awk，下面是個簡介：

- printf 第一行介於雙引號 (") 之間的命令，說明欄位的寬度與對齊方式。例如，第一個欄位應該向右對齊 4 個字元 (%4s)。第四個欄位應該向左調整 4 個字元 (%-4s)。第五個欄位可以不用調整 (%s)。我使用字串 (%s) 而不用十進位 (%d)，因此 awk 會刪除欄位最前面的 0。
- 第二行將輸入的資料欄區排列到輸出行上。在這裡，輸入和輸出採用相同的順序，但可以這個順序可以改變。前面的四個欄區取得前四個欄位 (\$1、\$2、\$3、\$4) 的資料。

第五個欄位是彙總；它意指取得其餘的所有資料。substr (\$0,index (\$0,\$5)) 表示“找到第五個輸入欄位，把它及之後的所有資料都印出來”。

-JP

## 35.23 旋轉文字

有時你會遇到這種狀況：“看起來每個都不錯，但就是不知道要選哪一個”。尤其當你在跳蚤市場逛街時，更容易發生這種情形；在瀏覽公共軟體時，也會遇到類似的狀況。

這引領我們看看 `rot` 的作用。基本上 `rot` 只是旋轉文字行與列。例如，下面的第一欄是輸入檔，而其餘的三欄分別是用 `rot` 旋轉一次、二次、三次的結果：

```

rot
$ cat file      $ rot file      $ rot file | $ rot file |
                rot                rot | rot
abcde          54321                5                e
1              a                4                d
2              b                3                c
3              c                2                b
4              d                1                a
5              e                edcba           12345

```

現在，將它與 `rot` 和 `tail -r` (25.19) 結合使用的情況相比較：

```

$ cat file      $ rot file      $ rot file | $ tail -r file |
                tail -r                rot
abcde          54321                e                12345
1              a                d                a
2              b                c                b
3              c                b                c
4              d                a                d
5              e                54321           e

```

`rot` 會旋轉文字內容 90 度。`tail -r` 則是將文字“倒置”(最後一行變成第一行... 以此類推)。

`rot` 也可以旋轉 `banner` (43.11) 的輸出。最後希望你對 `rot` 的功能有點概念！

```
-JP, LM
```

