Emacs

使 用 手 冊

中央研究院計算中心
ASPAC計劃
E-Mail : aspac@beta.wsl.sinica.edu.tw
技術報告:94016

84 年 3 月 17 日 version:2.1

ASPAC 計畫版權聲明

因一般的 PoorMan LaTeX中無法使用中文當成章節名稱,
所以本文中暫以英文來表示章節名稱.

Contents

Chapter 1

Introduction

本文的主旨以討論 EMACS 的入門為主，其內容包括 EMACS 發展的概述， EMACS 整合 環境的簡介以及 EMACS 的基石－ editor 的詳談。本文在結構上分成八章，第一章討論有關 EMACS 的源起、發展歷史以及 EMACS 的特性簡介。第二章討論 EMACS 的 on-line help。第三章則簡單介紹 EMACS 的整合環境。接下來就是本文的重點，探討 EMACS 的本質。所以，第三章之後的各章節則詳細探討 EMACS 為 editor 的這一個主題。

現在就來討論與 EMACS 發展有關的機構組織以及 EMACS 所扮演的角色特質。

1.1 What is GNU?

GNU 是 Richard Stallman 於 1975 年，在 MIT 所成立 的 Free Software Foundation (FSF) 中所執行的一項計劃。GNU 的本意代表 “Gnu's Not Unix”；雖然如此，它卻是一個與 UNIX 完全相容的軟體系統。二者最大不同在於，GNU 是一個 *free* 的 軟體， UNIX 卻是一個要付費的軟體系統。 GNU 之所以與 UNIX 完全相容，是因為 UNIX 的使用者很多，為了讓 UNIX 的使用者在使用 GNU 的時候不會有疏離的感覺，所以 GNU 儘量與UNIX 相容，它的相容只是看起來與用起來像 UNIX 而已， GNU 其實改進了所多 UNIX 的缺點，使它能盡善盡美。

Stallman 倡導 FSF 的宗旨，是要建立一個軟體 *free* 的王國。他所謂的 *free* 並不是意謂著便宜或免費，因為在 FSF 下的有些軟體還是必需付些工本費。他所謂的 *free* 是指使用上的自由。

要瞭解 Stallman 所言 *free* 的真意，就必須知道一般軟體的使用情形。大部 份的商業軟體在使用上都或多或少有些限制。例如，必須付費才可使用軟體程式，若運氣不好，同一程式在不同的電腦上使用，可能還要另行付費。有些軟體程式的使用也有一定的期限，期限一到就需另繳費用，否則使用權就會被撤銷；甚至有些軟體是以使用時間來計算費用的。當然，這些商業軟體是絕對不予許使用者將這些軟體任意給他人使用。除非給予軟體公司相當的代價，使用者是絕對不可能拿到軟體的原始程式的。

上述的種種都還是付錢就可以解決的，有些問題是錢也乏術的。因為，當商業程式銷售失敗或遠景不被看好時，使用者所購置的軟體就可能被開發者遺棄。此時，若想維持此軟體的正常運作

，就必須自求多福了。

Stallman 所領導的 GNU 計劃就是要脫離商業軟體的種種枷鎖，這也是 GNU 計劃所謂 “*free*” 的真諦。GNU 計劃下的任何軟體，只要使用者能找到軟體的來源，任何人都可以自由的使用它。獲得 GNU 軟體的來源很多，如 public archives、anonymous FTP、UUNET 等， 甚至任何一個裝有 GNU 軟體的地方都可以取得。GNU 計劃下的軟體，不只提供軟體的使用權，也提供軟體的原始程式，任何人都可以根據需要來修改程式，也可以盡己之力來找出程式的錯誤，使隸屬於 GNU 的軟體在大家的努力下能盡善盡美。 GNU 計劃下的軟體，是可不需付費而享有使用權。GNU 對使用者唯一的要求就是，當使用者對於 GNU 計劃下的軟體做了進一步的修改時，仍必須維持 GNU 的精神，就是對於修改過的軟體仍然必須將其無條件的奉獻出來，任何人都不可將修改過的 GNU 軟體當成商品來買賣。 所以 Stallman 一直強調 GNU 計劃下的軟體是 *free* 的，而且永遠會是 *free* 的。

GNU 計劃的最終目標是要完成一個全新的作業系統。目前完成的有 EMACS text editor、debugger、yacc-compatible parser generator 以及 linker 等等；也完成了將盡三、 四十個公用程式（utilities）；而 shell 也已經接近完成的階段。目前只要等 kernel 和 compiler 完成，就可以在 GNU 的系統上發展程式。

未來完成的 GNU 系統將可以執行 UNIX 上的所有程式，但它卻與 UNIX 系統不完全一致。GNU 系統將改進 UNIX 系統的缺失，使它使用起來更方便。例如，未來的系統將會有較長的檔名，檔名會有版次等等；新的系統會使用 C 與 LISP 做為系統程式語言。

1.2 What is GNU Emacs?

GNU EMACS 是 GNU 計劃下的第一個產品， EMACS 為 Editor MACros 的縮寫。 Richard Stallman 於 1975 年在 MIT 首次撰寫 EMACS editor。目前 GNU EMACS 已發展到 19.28 版，本文所討論的 GNU EMACS 也以 19.28 版為主。 GNU EMACS 秉持著 GNU 的精神，它依然是 *free* 的。任何人都可以 “anonymous ftp” 的方式至 prep.ai.mit.edu 這個 site ，取得 GNU EMACS 的原始碼。 GNU EMACS 不是 public domain 的產物，它有版權也有使用上的限制，那就是任何得到它的人都不可以將其視為私有的財產。修改過的 GNU EMACS 也不可以作為商品來牟利，金錢交易的行為是絕對被禁止。

EMACS 有多種版本可適合不同的工作平台（platfrom）。適合 UNIX 環境的有 GNU EMACS、Unipress EMACS 以及 CCA EMACS ，其中以 GNU EMACS 最常被使用，本文也只介紹 GNU EMACS。適合個人電腦的 EMACS 有 FreEMACS、MicroEMACS 以及 Epsilon。當然，也有使用在 VMS 環境的版本。

GNU EMACS 是由 C 與 LISP 語言寫成，任何人都可依據需要將個人所發展的函式 (function) 加入 GNU EMACS 上。當然，新發展的軟體是不可以從事商業買賣，只能將它無條件的奉獻出來。新發展的函式可以直接在 EMACS 中使用，不需重新編譯（complie） 整個 EMACS ，而且新增的函式也不會破壞 EMACS 原有的結構。就因為有此特性，EMACS 的函式 可以與日俱增。願與他人共享成果的使用者，可以透過電子郵件或電子公佈欄，將函式的原始碼公諸於世。公佈的函式，最後會經由 FSF 的審查，以決定是否要加入新版的 EMACS 中。FSF 也鼓勵使用者將所發現的錯誤，透過相同的管道，提供給 FSF 作為改進之用。GNU EMACS 就是在如此的運作

下，靠大家共同的努力來提昇品質，以達產品的穩定性。

1.3 GNU Emacs Characteristics

GNU EMACS 與其說它是一個編輯器（editor），倒不如說它是一個以編輯器為主幹的環境軟體。一般的軟體都是將編輯器視為一個附屬功能，只有 EMACS 以編輯器為基石，在其上發展其它的功能。以收發電子信件的軟體為例，電子信件軟體是以收發信件為環境的基石。任何電子信件的軟體都只有在使用編輯器時，才呼叫相關的編輯器。 EMACS 的出發點就與眾不同， EMACS 一切以編輯器為主，任何的功能都是建基在編輯器之上。

所以，進入 EMACS，就等於是進入了一個編輯環境，這個編輯環境提使用者許多功能，讓使用者如置身在一個全功能的作業系統中。 EMACS 自行發展了一個 “bourne-shell-like” 的 shell，除了 EMACS 自己的 shell 外， EMACS 還可以讓使用者自行選擇所使用的 shell；EMACS 可以讀送 e-mail。 EMACS 可以透過 ftp 來編輯遠方 host 上的檔案，而不需要簽入（login） 檔案所在的 host；EMACS 也可做 telnet 與 relogin 的動作。 EMACS 也可以讀 news； EMACS 也提供了年曆（calendar），可以讓使用者查閱日期，也可以將重要的事情在年曆上標示出來；EMACS 又提供了 “Diary” 的功能，當特定的日期與時間到來時，會在螢幕上將 “Diary” 上的事情顯示；EMACS 也有撰寫文章大綱的功能。 EMACS 更提供多種程式的編譯功能，讓使用者可以在 EMACS 中一邊編輯程式一邊編譯程式； EMACS 更有自己的 debugger，使程式的除錯、編輯與編譯在 EMACS 中同時完成。

所以 EMACS 所提供的不只是一個編輯器而已，它所提供的是一個整合的工作環境，而這個環境是建立在編輯的基石上。它希望使用者進入了 EMACS 以後，可以在 EMACS 的環境中完成所有的工作，不需要離開 EMACS，要離開 EMACS 就是要離開電腦的時候（logout）。

EMACS 除了是一個整合的工作環境外，它還具有以下的特性：

- **display editor**
- **real-time editor**
- **advanced editor**
- **Self-documenting**
- **Customizable**
- **Extensible**
- **support X window enviornment**

EMACS 是一個 display 編輯器，因為每一個被編輯的文字都會被顯示在螢幕上。

EMACS 是一個 real-time 的編輯器，因為當新的文字被鍵入時，螢幕會在非常短的時間內被更新。

EMACS 的編輯功能不只具備一般編輯器所有的功能，它還提供如下的功能：

- 文字的填充（filling of text）。
- 程式自動內縮功能。
- 可以同時閱讀一個以上的檔案。
- 對於字元（characters）、字（words）、列（lines）、句子（sentences）、段落（paragraphs）、頁（pages）以及各種程式中的 expression 和註解（comments）都有其自有的處理方法。

EMACS 是一個 Self-documenting 的軟體，因為在任何時候都可以 “**Ctrl-h**” 指令 來得立即的幫助。因為， 透過 “**Ctrl-h**” 可以得知每一個 EMACS 的指令。

EMACS 是可以 Customizable 的。使用者可視個人的需求，來改變 EMACS 指令的定義。GNU EMACS 的 Customization 的設定非常簡單，使用者可以很方便的使用它。

EMACS 是 Extensible 的。GNU EMACS 是由 LISP 語言所寫成的函式共同組成的， 函式與函式間的互動關係（dependency）不強。就因為 EMACS 是由函式所組合而成的，所以可以隨時將函式作增減而不會破壞 EMACS 既有的結構。

EMACS 也針對 X 的視窗環境，提供自己的選單（menus）和滑鼠按紐（mouse bottons）功能。 EMACS 對於 text-only 的終端機也提供許多與 X 環境相當的服務品質，例如在文字模式的終端機，可以同時開啟數個的檔案，檔案可以互相切換，當使用 shell 模式 來執行 shell 指令的同時也可以編輯檔案。但本文的只討論 text-only 的 EMACS 使用法。

以上的種種就是有關 EMACS 的特色。至於如何使用這些功能就下列章節所要討論的重點。

Chapter 2

Emacs on-line help

EMACS 是一個整合的工作環境，初次使用 EMACS 或不熟悉 EMACS 的人，常會不知所措。 所以，EMACS 提供了非常便捷且功能強大的線上輔助說明（on-line help），來幫助使用 EMACS 。

2.1 Emacs Basics

在未討論 EMACS 的線上輔助說明時，先談談如何啟動 EMACS。

啟動 EMACS 的方法很簡單，只要在螢幕的提示下鍵入 “`emacs`” 即可；離開 EMACS， 只要鍵入“`Ctrl-x Ctrl-c`” 即可離開 EMACS 。

鍵入 “`Ctrl-x Ctrl-c`” 的方法是，先按住鍵盤上的 Ctrl 鍵不放再按下英文字的 “x” 即可。鍵入 “`Ctrl-c`” 的方法一樣，先按住鍵盤上的 Ctrl 鍵不放再按下英文字的 “c” 即可。當然進入與離開 EMACS 的方法還有多種，這裏只是介紹一種方法，讓使用者可以很快的使用 EMACS 的線上輔助說明，至於其它的方法會在以下各章節中陸續提及。

除了進入與離開 EMACS 的方法必須知道外，還有一個指令非常用那就是 “**Ctrl-g**”。 “**Ctrl-g**” 可用來取消鍵入的指令，如果不想執行所鍵入的指令，可以隨時將其取消。

EMACS 鍵入指令的方法有二種：

- Ctrl 鍵
- Meta 鍵

所有 EMACS 的指令都可以 Meta 鍵表示出來，鍵盤上若無 Meta 鍵，則可以 ESC 鍵來取而代之。常用的 EMACS 指令通常會有一個 Hotkey 與之連結。 Hotkey 的構成，通常是以 Ctrl 為開頭的型態出現。例如 “**Ctrl-x Ctrl-c**” 這一個 Hotkey，則代表了 EMACS 指令的 “**save-buffers-kill-emacs**” 。若要使用 Meta 鍵來表達與 **Ctrl-x Ctrl-c** 相同的效果 ，則必須鍵入 “`ESC-x save-buffers-kill-emacs`”。

使用 Meta 鍵，可以利用 EMACS *completion* 的功能。因為 *completion* 可以讓使用者鍵入最少的字，就可使系統喚起所欲執行的命令。以下就介紹 EMACS 的 *completion*。

EMACS *completion* 的意思是，只要鍵入字串的部份，EMACS 會將其餘的部份自動填入其應在的位置。如果所給予的字串不足以決定其餘的部份， EMACS 會將所有可能的結果都列出來，以供使用者來決定所需要的字串名稱。 *completion* 所適用的範圍如下：

- 指令
- 特定目錄下的檔案
- 緩衝區
- EMACS 的變數

至於何謂特定目錄下的檔案、緩衝區以及 EMACS 的變數，會在以後的章節中陸續談及。此處討論 *completion* ，只是為了使用 *completion* 於 EMACS 的線上輔助說明。

使用 EMACS 的 *completion* 有三種方法：

- **TAB**
  儘可能將其餘的字串填滿。
- **SPACE**
  將 puncutation 字元之前的字填滿，填充的字不會超過一個字以上。
- ?
  將所有可能的 completions 選擇都列出來。

使用 *completion* 的做法是將部份字串鍵入後，再按下 `TAB`、 `SPACE` 或 `?` 即可。例如，鍵入 “`M-x au TAB`” ，則螢幕的最下方會出現 “`M-x auto-`”。再鍵入 `TAB` 則螢幕會在另一個視窗出現：

```
Possible completions are:
auto-fill-mode                     auto-lower-mode
auto-raise-mode                    auto-save-mode
```

若鍵入 “`M-x au SPACE`”，則螢幕的最下方也會出現 “`M-x auto-`”。 若鍵入 `SPACE` ，則螢幕的另一個視窗也會出現如下的命令：

```
Possible completions are:
auto-fill-mode                     auto-lower-mode
auto-raise-mode                    auto-save-mode
```

這似乎意味著 TAB 與 SPACE 的功能一樣，其實不然，二者的差異可從下一個例子看出。鍵入 “`M-x auto-f TAB`”，可得 “`M-x auto-fill-mode`”；但鍵入 “`M-x auto-f SPACE`”，只能得到 “`M-x auto-fill-`”，欲得到 “`M-x auto-fill-mode`”，則必須再鍵入一次 `SPACE`。這就是前面所說的 “SPACE” 一次只填一個 “puncutation” 之前的一個字的意思；而 TAB 則是儘可能的將所有可以判斷出來的字串呈現出來，其顯示字串的長度並不以一個 “puncutation” 為限。鍵入 `?` 的作用，是在 EMACS 的另一個視窗上顯示所有可能的字串，此時使用者可根據視窗上的訊息鍵入適當的命令。例如鍵入 “`M-x au ?`”，螢幕上出現另一個視窗顯示如下的資訊：

```
Possible completions are:
auto-fill-mode                     auto-lower-mode
auto-raise-mode                    auto-save-mode
```

鍵入 “`M-x au ?`” 的地方，則不會執行 *completion* 的動作，這是 ? 與 TAB 、SPACE 最大不同的地方。

若視窗的內容太多無法一次窮盡，此時就必需捲動視窗。捲動視窗可以用 “**Ctrl-v**” 與 “**Meta-v**” 二個指令來使視窗做上下的移動。

2.2 Emacs Tutorial

想快速瞭解 EMACS 的人，可以參考 EMACS 的自學教材（tutorial）。使用 EMACS 自學教材的方法很簡單，只要鍵入 “`Ctrl-h t`” 即可進入 EMACS 的自學的狀態了。

EMACS 的自學教材可分成以下幾部份：

1. 介紹 EMACS 指令的鍵入方法，即介紹 Ctrl 與 Meta 鍵 。
2. viewing screenfuls
3. basic cursor control
4. Ctrl-g 的用法
5. EMACS 的 window 與 multiple windows
6. inserting and deleting
7. undo
8. EMACS 檔案的處理
9. EMACS 的 buffers
10. extending the command set
11. 簡介 EMACS 的 mode line 與 echo area

12. searching
13. recursive editing levels
14. getting more help
15. leaving EMACS
16. 有關 EMACS 的版權問題

EMACS 的自學教材是以編輯功能的介紹為主。雖然 EMACS 的功能不只如此， 但編輯是 EMACS 的最基礎的功能，要瞭解 EMACS 當然要從它的基本著手。所以 EMACS 的自學教材也以編輯的介紹為主，至於 EMACS 其它的工件環境，是無法從 此自學教材中得知。所以，本文會在第三章簡介 EMACS 的工作環境。

2.3 Ctrl-h Prefix

EMACS 除了自學教材可供參考外，它還提供了其它的線上輔助說明功能，讓使用者可以隨時查閱需要的相關訊息。

EMACS 的線上輔助說明都是以 "**Ctrl-h**" 為開端，其種類有以下二種：

- **Ctrl-h**
- **Ctrl-h Ctrl-h**

鍵入 `Ctrl-h`，螢幕的最下端會出現如下的訊息：

```
C-h (Type ? for further options) -
```

此時的 "Ctrl-h" 只是用做前置字（prefix key），它是用來等候使用者輸入其它的指令。若輸入 "?"，則螢幕的下方會出現所有可使用的選擇，使用者可根據需要來選擇合適的選項。

若鍵入兩次的 "`Ctrl-h`"，螢幕下方會出現：

```
type one of the options listed or Space to scroll
```

此時 EMACS 會另外開啟一個視窗，將所有與求助的選項都列出來，且會做一簡要的說明，要捲動此視窗則鍵入 `Space`。此新開啟的視窗共有二十一個選項，包括：

```
a b c f C-f i k C-k l m n p s t v w C-c C-d C-n C-p C-w.
```

使用這二十一個選擇項的方法非常簡單，只要在 “**Ctrl-h**” 之後輸入任一個選擇就可以了。例如，要選擇 “a” ，則執行 “`Ctrl-h a`”即可。

現在將 EMACS 常用的線上輔助說明一一作解釋。EMACS 常用的線上輔助說明有：

- **Ctrl-h c**
- **Ctrl-h k**
- **Ctrl-h w**
- **Ctrl-h a**
- **Ctrl-h v**
- **Ctrl-h i**

“**Ctrl-h c**” 與 “**Ctrl-h k**” 的功能相似， 二者都是在尋求與 Hotkey 有關的訊息。二者唯一的差別，就在於對指令的解釋詳細與否而已。“**Ctrl-h c**” 是簡述與 Hotkey 連結的命令，而 “**Ctrl-h k**”則詳述連結 Hotkey 的命令。二者都有一個前題的預設，那就是都是先知道 Hotkey 為何，而想進一步知道此 Hotkey 所使用命令的名稱。今舉 “**Ctrl-x Ctrl-c**” 的例子來說明二者的差別。

鍵入 “`Ctrl-h c RET`” [1] 則螢幕下方會出現

```
Describe key briefly: -
```

在 “:” 的後面鍵入 “`Ctrl-x Ctrl-c`”，則在原先出現

```
Describe key briefly: -
```

的地方則出現

```
C-x C-c runs the command save-buffers-kill-emacs''
```

鍵入 “`Ctrl-h k RET`”，則螢幕下方出現

```
Describe key: -
```

[1]RET，亦即鍵盤上的 Enter 鍵。任何一個指令 輸入完畢時，必需緊跟著一個 Enter。此作用是用以告知系統，指令輸入 已經結束，可以開始執行相關的動作了。

在 “:” 的後面鍵入 “`Ctrl-x Ctrl-c`”，則 EMACS 會另以一個視窗顯示如下的訊息：

```
save-buffers-kill-emacs:
Offer to save each buffer, then kill this emacs process.
With prefix arg, silently save all file-visiting buffers, then kill.
```

此訊息的第一行是 Hotkey 所連結命令的全名，以後的行數則是對此命令的詳細說明。

“**Ctrl-h w**” 的用法與 “**Ctrl-h c**” 和 “**Ctrl-h k**” 正好相反。 “**Ctrl-h w**” 是在知道 EMACS 的命令而欲知是否有相對應的 Hotkey 時所使用的。例如鍵入 “`Ctrl-h w`”，螢幕的下方會出現

```
where is command:
```

在 “:” 之後鍵入 “`save-buffers-kill-emacs`”，則在原處會出現

```
save-buffers-kill-emacs is on C-x C-c
```

所以想知道命令是否有相對應的 Hotkey，可以此方法查知。

“**Ctrl-h w**” 可以使用 EMACS 的 *completion*，但其最大的不便處，就是必需給予指令的第一個字元。如 “save-buffers-kill-emacs”，必需先給予以 s 開頭的子字串，才能逐步使用 EMACS 的 *completion*。若不鍵入 s 開頭的子串，而鍵入 s 之後的任何字串，則無法找到相對應的指令，所以使用 “**Ctrl-h w**” 必需要記著指令的第一個字。除此之外， EMACS 的線上輔助說明還提供了另一種幫助，可讓使用者鍵入任意的子字串，都可以找到相對應的指令，那就是 “**Ctrl-h a**”。

鍵入 “`Ctrl-h a`” 則螢幕的下方會出現

```
command-apropos (regexp):
```

此時只要給予與命令相關的任一子字串或 “*regular expression*” (6.3 節會討論何謂 *regular expression*) 再按下 RET， EMACS 會另開一個視窗，將所有涵蓋此子字串或 *regular expression* 的指令全列出來。此指令與 “**Ctrl-h w**” 最大不同處如下：

1. 使用 “**Ctrl-h a**” 所鍵入的子字串，並不限於指令的第一個字元，而 “**Ctrl-h w**” 則必需以指令的第一個字為起始字。 使用 “**Ctrl-h a**”，可給予指令中任何位置的子字串。
2. “**Ctrl-h a**” 無法使用 *completion*，而 “**Ctrl-h w**” 可使用 *completion*。

“**Ctrl-h a**” 主要的目的是當使用者無法正確的鍵入指令的第一個字元時，可以借此將所有包含使用者記得的部份子字串的指令都列舉出來。

EMACS 除了以上幾個常用的線上輔助說明之外，還有一個非常實用的資料查閱中心，那就是“**Ctrl-h i**”。“**Ctrl-h i**” 執行 “Info program”，它主要是用來瀏灠已建構成樹狀結構的文件檔案。目前所有與 EMACS 有關的文件檔案都可透過 Info 來閱覽，最終所有與 GNU 有關的文件資料，將可以由此而窺得其文件檔案全貌。

使用 info 模式（info mode）的方法很簡單，只要鍵入 “`Ctrl-h i`” 就可查閱所有與 EMACS 相關的文件資料。進入了 info 之後要如何有效的使用它呢？會在下一節詳細討論。

以上所談的，就是較常使用的線上輔助說明種類。若線上輔助說明的訊息，是以另一個視窗顯示出來，此時的訊息又無法一「幕」瞭然。想參考其它部份的資料，就必需捲動視窗。視窗捲動的指令，可參考如下的方法：

- **Ctrl-x 1 (delete-other-windows)** [2]
  保留游標所在的視窗，而將其它的視窗關閉。
- **Ctrl-x o (other-window)**
  可使游標在不同的視窗間切換。換言之，如果游標在工作的視窗，可以此指令將游標移出顯示線上輔助說明訊息的視窗，反之亦然。
- **Ctrl-v (scroll-up)**
  將螢幕向上捲，如此則可看清螢幕下方的訊息。
- **Meta-v (scroll-down)**
  螢幕向下捲，如此可以重複參考已經看過的資料。

2.4 info

前已略述 info 的使用法，現在就更進一步詳述之。在 EMACS 的線上輔助說明功能中， info 的內容可說是最為豐富的。因為，任何線上輔助說明的文件都可在 info 模式中找到。因為，info 就是用來放置整個 EMACS 手冊。

info 對於 EMACS 文件的安排是採取樹狀的結構，所以是以根部（root）為出發點。 info 執行 info program，使用 info program 的方法有二：

- **Ctrl-h i**
- **ESC-x info**

info 對於檔案的編排，是以 Hypertext 的方法來處理所有的相關文件。當鍵入 “`Ctrl-h i`” 或 “`ESC-x info`” 後，會先進入 info 樹狀（tree）結構的最頂端。如下就是進入 info 時的第一個畫面：

[2]本文所有與 Hotkey 相對應的指令都放於括號中。

```
File: dir Node: Top This is the top of the INFO tree
  This (the Directory node) gives a menu of major topics.
  Typing "d" returns here, "q" exits, "?" lists all INFO commands, "h"
  gives a primer for first-timers, "mTexinfo<Return>" visits Texinfo topic,
  etc.
  --- PLEASE ADD DOCUMENTATION TO THIS TREE. (See INFO topic first.) ---

* Menu: The list of major topics begins on the next line.

* Info: (info).         Documentation browsing system.
* Emacs: (emacs).       The extensible self-documenting text editor.
* VIP: (vip).           A VI-emulation for Emacs.
* Forms: (forms).       Emacs package for editing data bases
                        by filling in forms.
* GNUS: (gnus).         The news reader GNUS.
* CL: (cl).             Partial Common Lisp support for Emacs Lisp.
* Gcc: (gcc).
* Cpp: (cpp).
* Makeinfo: (makeinfo).
* Info-stnd: (info-stnd).
* Texi: (texi).
* Hyperbole: (hypb.info).
* Octave: (octave.info).
```

此時，出現在螢幕的第一列是標頭（header），它包含此結點（node）的基本訊息。表頭所提供的訊息，最多可有五件事情：

1. 結點所在的檔案（File）
2. 結點的名稱（node）
3. 此結點的下一個結點（Next）
4. 此結點的上一個結點（Prev）
5. 此結點的上一層結點（up）

此時的畫面，只顯示了二件事情，是因為此畫面為 info 樹狀結構的最上層。

在表頭之下的資訊，是用來告知如何用 info 。它提供了五件事情，現一一說明。

- **h**

  不知如何使用 info 模式者，可在進入 info 模式後，使用 info 的線上輔助說明。欲使用 info 的線上輔助說明，只要在進入 info 模式後，鍵入 “h” 即可。此不只是一個線上輔助說明，還是一個教學指引。跟隨著它的說明，即可明瞭如何使用 info 模式。

- **d**
  “Ctrl-h i” 指令，會先進入 info 樹狀結構的根部。任何情況下，可鍵入 “d” 回到此根部。

- **?**
  想知所有與 info 有關的指令，只要鍵入 “?”，就可以得知所有指令的全貌。

- **q**
  欲離開 info，只要鍵入 “q” ，就可以離開 info 而回到先前的緩衝區。

- **mTexinfo<Return>**
  這一個指令是用來使用 info 的 Menu Item。使用的方法如下：

  1. 鍵入 “m” 。
  2. 鍵入所欲參考的文件名稱，也可使用 completion 的功能來簡化輸入的 工作。
  3. 鍵入 RET。

  何為 Menu Item 會再解釋。

在這五個指令之下的是，此結點可以使用的 Menu Item。使用者可以直接至所要參 考的 Menu 之下，鍵入 “RET”。此時， info 會將此 Menu 的相關文件顯示在螢幕上。

使用 info 除了以上的指令之外，還有幾個必需知道的指令。現介紹如下：

- **n**
  將結點移至下一個與此結點相連的結點。

- **p**
  將結點移至上一個與此結點相連的結點。

- **u**
  將結點移至上一層的結點。

- **m**
  以上結點間移動的指令，必需結點間有相連接的關係。若想做跨越的移動，這幾 個指令是無法做到的。此時，必需透過 Menu 來做誇越結點的文件閱讀。
  使用 Menu 的方式有二：

  – “`mTexinfo<RET>`”
    這種使用 menu 的方法，在進入 info 的第一個畫面就可以看到了。 此方法由三件事共同組成：

    1. `m`
       鍵入 `m` 是使用 menu 的指令。
    2. `Texinfo`
       鍵入 `m` 時，*minibuffer* 會出現

```
menu item:
```

此時，可在其後輸入所欲閱灠的文件名稱。所以，Texinfo 意指 文件的名稱。此時，就可以鍵入畫面上以 “*” 為開頭的名稱。如果是剛進入 info，可使用的 Texinfo 名稱有：

```
Info、 Emacs、 VIP、 Forms、 GNUS、 CL、 Gcc、 Cpp、
Makeinfo、 Info-stnd、 Texi、 Hyperbole、 Octave
```

3. RET

待文件名稱輸入完畢後，要按下 RET，以告知系統文件名稱已經 輸入完成了，可以開始執行的動作了。

– 直接鍵入 “RET”

如果在閱灠某一個結點時，文件中出現以 “* Menu:” 為首的文字，就表示此列以下若有以 “*” 為開頭的列，均為一個可以使用的 Menu Item 。此時，可以直接鍵入 “RET” 參考另一個結點的文件。 如以剛進入 info 的畫面來說，可以直接鍵入 RET 的資料如下所示：

```
* Menu: The list of major topics begins on the next line.

* Info: (info).         Documentation browsing system.
* Emacs: (emacs).       The extensible self-documenting text editor.
* VIP: (vip).           A VI-emulation for Emacs.
* Forms: (forms).       Emacs package for editing data bases
by filling in forms.
* GNUS: (gnus).         The news reader GNUS.
* CL: (cl).             Partial Common Lisp support for Emacs Lisp.
* Gcc: (gcc).
* Cpp: (cpp).
* Makeinfo: (makeinfo).
* Info-stnd: (info-stnd).
* Texi: (texi).
* Hyperbole: (hypb.info).
* Octave: (octave.info).
```

使用 Menu 有些條件，就是所要使用的 Menu，必需在此結點的文件中有明列 出來的才可以使用。要使用其它結點的 Menu，必需先到有要使用的 Menu 的 結點，才可以使用它的 Menu。

Menu 有其一定的結構。所有 Menu 的第一列，都是以 “ Menu:” 為開 端。如下所示：

```
* Menu: The list of major topics begins on the next line.
```

此列之後的所有列，只要以 “*” 為開始的列，就表示一個可以使用的 item。 Menu Item 的表示法如下所述：

```
* Info: (info).        Documentation browsing system.
```

它主要由四個部份主成：

1. *****
   每一個 Menu Item 都是以 "*" 為開頭。
2. **Subtopic:**
   "*" 之後就是 Subtopic 的名稱，再加上一個 ":"。 info 就是根據此 Subtopic 找到所對應的結點名。此例子的 Subtopic 的名稱就是 "`Info`"。 如果要以指令 "`m`" 的方式，使用 Menu Item 所要給予的 Texinfo 的名稱就是此 Subtopic 的名稱。
3. **node name**
   在 Subtopic : 之後就是所使用的結點名。一般為了使用上的方便，會儘可能的使用相同的 Subtopic 和結點的名稱。若二者的名字相同時，在 Subtopic 後的結點名會省略而以兩個 ":"表示之。如 "* Info::"。
4. **node description**
   結點後面的內容是選擇性，它主要是用來概略描述此結點。

- **l**
  如果想要回到前一個（last）所參考過的結點，可以使用指令 **l** 一步一步的往回走。
- **b**
  指令 **b** 可以使游標移至文件的最前端。
- **SPC**
  若文件太長可鍵入空白鍵（**SPC**）來捲動螢幕。
- **Ctrl-g**
  任何時候想要取消所鍵入的指令，可鍵入 `Ctrl-g`

Chapter 3

Emacs as a working enviornment

前面不斷的強調，EMACS 不只提供一個編輯的環境，而提供一個整合的工作環境。所以，在未進入本文的正題 – EMACS 的編輯環境之前，先對 EMACS 的工作環境做一個簡介。

EMACS 所能提供的工作環境如下：

1. EMACS 可執行 Shell 的指令。
2. EMACS 可做為 Directory Editor（Dired）。
3. EMACS 可以編輯、編譯及除錯程式。
4. EMACS 具有編輯其它 host 上檔案的能力。
5. EMACS 可以列印檔案。
6. EMACS 具有年曆、日記的功能（Calendar、Diary）。
7. EMACS 具有讀 Man Page 的能力。
8. EMACS 可以收發電子郵件（Mail、Rmail）。
9. EMACS 可以閱讀網路上的電子佈告欄（GNUS）。
10. EMACS 具有 version control 的功能
11. EMACS 對於檔案的處理，具有 Outline 的能力。
12. EMACS 具有資料庫的處理能力。
13. EMACS 可以提供電子計算機的功能。
14. EMACS 提供了娛樂的環境（game），讓工作者可以暫時放下工作的壓力。

由以上的分析可知，只要進入 EMACS 的工作環境，就可以在其下完成所有的工作。 所以 EMACS 的理想，是離開 EMACS 就是離開電腦工作的時候，因為它的最終目標，就是要完成一個以編輯器為軸心的作業系統。

EMACS 所提供的這些功能，都是先喚起代表此功能的模式（*mode*），EMACS 的模式，分成主要模式（*major mode*) 與次要模式（*minor mode*）。每一次只能使用一個主要模式，而且必須要的。但一個主要模式可以搭配一個以上的次要模式。現在就來簡述 EMACS 工作環境的功能，至於如何使用這些 EMACS 的工作環境，本文就不多做介紹。

3.1 Running Shell Commands from Emacs

在 EMACS 中有兩種執行 shell 指令的方法：一種是進入 *shell command mode*，另一種是進入 *shell mode*。二者都可以執行 shell 的指令，其最大不同之處是，進入 *shell mode* 的狀態，執行 shell 指令的同時，仍可以切換到其它模式處理別的工作，但如果使用 *shell command mode*，就必須等指令執行完後才可以做其它的事。

使用 *shell command mode* 時，使用者在螢幕的最下方輸入欲執行的指令，EMACS 會開啟一個名為 “`*Shell command output*`” 的視窗，將 shell 指令執行的結果顯示在此視窗中。*shell mode* 則是執行一個 subshell，其輸入與輸出都是透過同一個緩衝區，所以輸入與輸出是在同一個地方，它不似 *shell command mode*，指令輸入與結果的顯示在不同的地方。

shell command mode 又可以有兩種模式，一種就是很單純的執行一個 shell 的指令；另一種是對某一特定區域的資料執行 shell 的指令。*shell command mode* 容許執行後的結果，直接輸入到目前所使用的工作區內。有了如此的功能，使用者可以很輕易的將 shell 指令執行的結果，直接放入適當的位置，而不需另外從事剪貼的工作。

要如何使用 EMACS 所提供的 shell 功能呢？以下是最基本的方法，至於高階的用法則請自行參考 GNU EMACS 所提供的 “GNU Emacs Manual”。

1. *shell command mode*
   (a) **ESC-! (shell-command)** 喚起 *shell command mode*。
   (b) **ESC-| (shell-command-on-region)**
   針對某一特定區域[1]的資料，執行 shell command mode 的 shell 指令。
   (c) **Ctrl-u ESC-! 與 Ctrl-u ESC-|**
   在 ESC 前加上 Ctrl-u，可以將 shell 指令執行的結果，輸出到游標所在的位置。
2. *shell mode*
   ESC-x shell 是喚起 *shell mode* 的指令。

3.2 The Directory Editor

Dired 是專門針對目錄來運作的編輯功能。進入 *Dired mode* 後，EMACS 會根據使用者所指定的目錄來列出其下的檔案及次目錄，此時可根據 需要EMACS 對這些檔案及次目錄作些運作。

EMACS 所提供可操作 **Dired** 的種類如下：

[1]特定區域，是指緩衝區的某一範圍 (region)而言，所以此指令只是針對緩衝區的某一部分運作

1. 可閱讀、編輯 **Dired** 所列舉出來的檔案
2. 操作 **Dired** 下的檔案
   (a) 在 **Dired** 可以刪除（delete）檔案
   此功能可以很容易的將EMACS 的備份檔（其檔名以 ~ 結尾）、暫存檔（檔名在兩個 # 中間）或具某一特殊檔名模式的檔案刪除。
   (b) 檔案的拷貝
   (c) 檔名的更新
   (d) 改變檔案的 mode
   (e) 改變 gid、uid
   (f) 檔案的列印
   (g) 檔案的壓縮、解壓縮
   (h) 載入、編繹 EMACS 的 LISP file
   (i) 可產生 hard links 與 symbolic links
   (j) 可將檔名換成大寫或小寫的英文字母
3. 可在 *Dired* 中執行 shell 的指令
4. 可使用 UNIX 的 diff 指令比較檔案間的異同
5. 可隱藏次目錄
6. 可使用 find 的公用程式來尋找檔案

以上所列舉的就是 **Dired** 的功能，有人說它類似 PC 上的 PCTOOLS，讀者是否有相同的感覺呢？

進入 **Dired** 模式的方法很簡單，只要鍵入 “`ESC-x dired`” 即可。此時的 *minibuffer* 會顯示出如下的文字：

```
Dired (directory):
```

“:” 之後是目前所在的目錄，此時可以修改目錄名。確定所要使用的目錄，按下 RET 後，系統會另開啟一個視窗來顯示此目錄下的所有檔案。之後就可以對這些檔案做運作。 Dired 所使用的緩衝區是一個唯讀（read-only）的緩衝區，所以 *mode line* 會出現二個 `%%` 來表示其為唯讀的狀態。如果要緩衝區的唯讀狀態改為可讀，可以鍵入 `Ctrl-x Ctrl-q` 的指令，將緩衝區的狀態改變。運作此緩衝區有其特別的方法，因此對此緩衝區做編輯並無實質上的作用。以下就介紹運作 Dired 的方法：

- 在 Dired 中刪除檔案
  Dired 最基本的指令，就是將要刪除的檔案做上旗標（flag）之後，再將有旗標的檔案刪除。

- **d**
  將游標移至所欲刪除的檔案列，鍵入 `d`。此列的最前方會出現 `D`，這就是刪除的旗標。此時的游標會移至此列的下一列。
- **u**
  若想放棄已定好的旗標，可以鍵入 `u` 使螢幕上的 `D` 消失。
- **x**
  鍵入指令 `d` 只是將要刪除的檔案先做上旗標，並未真正執行刪除的動作。只有鍵入 `x` 才會將所有做上旗標的檔案刪除。執行刪除檔案之前，會先詢問是否真要刪除的意見。此時如果回答 “`yes`” 則執行刪除的動作，若回答 “`no`” 不執行刪除的動作，但旗標依然存在著。

• 在 Dired 中將多個檔案同時做上旗標

- #
  鍵入 `#`，系統會自動將所有的自動儲存的檔案（auto-save file）做上刪除的旗標。
- ~
  鍵入 `~`，系統會自動將所有的備份檔（backup file）做上刪除的旗標。
- **%d** *regexp* **RET**
  將所有適合的 *regular expression* 檔案做上刪除的旗標。
- 所有做上旗標的檔案要執行刪除的動作，都必需執行指令 `x`。

• 在 Dired 中訪問檔案

- **f**（dired-find-file）
  如果想要訪問目前游標所在列的檔案，只要在此列上鍵入 `f` 即可。此時，檔案的內容會顯示在原先顯示 Dired 緩衝區的視窗上。使用此方法訪問檔案，就如同以 **Ctrl-x Ctrl-f** 訪問檔案一樣。
- **o**（dired-find-file-other-window）
  此方法也是用來訪問檔案，但與鍵入 `f` 有些不同之處。鍵入 `o` 後，所訪問的檔案會出現在另一個視窗上而游標也會移至所訪問的視窗，顯示 Dired 緩衝區的視窗並未消失在螢幕上。
- **Ctrl-o**（dired-display-file）
  此方法與鍵入 `o` 雷同，二者不同之處在於鍵入 `Ctrl-o` 後所訪問的檔案會出現在另一個視窗上，但游標不會移至所訪問檔案的視窗，依然留在顯示 Dired 緩衝區的視窗上。
- **v**（dired-view-file）
  此指令僅供流灠檔案之用，因為以此而開啟的檔案是唯讀檔案。

• 將 Dired 的檔案做上標記

- **m**（dired-mark）
  將目前游標所在的檔案做上標記 “`*`”。如果給予數值引數，則 做上標記的檔案數目會依所給予的數目而定。

- * (dired-mark-exectables)
  將所有的可執行檔（executable files) 做上標記 “*，若給予數值引數。則會將所有做上標記的可執行檔的標記取消（unmark）。
- @ (dired-mark-symlinks)
  將所有的 symbolic files 做上標記 “*，若給予數值引數。則會將所有做上標記的 symbolic files 檔的標記取消 （unmark）。
- / (dired-mark-directories)
  將所有為目錄的檔名，但除了 “.” 與 “..” 之外，均做上標記 “* 。若給予數值引數，則會將所有做上標記的目錄名稱的標記取消（unmark）。
- **ESC-DEL** *markchar* (dired-unmark-all-files)
  消除所有以字元（character) （*markchar*) 做為標記的記號。 如果給予數值引號，則在消除每一個記號時，會詢問是否要消除記號。 回答 “y” 則表示要將記號消除，回答 “n” 則表示 不要消除已做好的記號。若此時鍵入 !，則表示消除其餘的記號 不再詢問意見。
- **c** *old new* (dired-change-marks)
  使用此指令，可將原本以 *old* 為標記的記號，換成以 *new* 為標記的記號。
- **%m** *regexp* **RET** (dired-mark-files-regexp)
  可使用 *regular expression*，將具有某一類型的檔案做上標記。

• 在 Dired 中的運作方式，有幾件事情必需注意：

1. 如果給予指令數值引數 *n* 時，此時指令所運作的檔案是從目前游標所在的檔案起往後算 *n* 個檔案（包括游標所在的檔案）。如果給予負數的數值，則往游標所在處之前算 *n* 個檔案（包括游標所在的檔案）。
2. 如果不給予任何的數值引數，則指令的運作範圍會以做了標記的檔案為主。
3. 如果不給予數值引數也不對任何檔案做標記，則指令只對目前游標所 在列的檔案運作。
4. 所有運作 Dired 緩衝區的指令都是大寫的英文字母，所有的指令都是使用 *minibuffer* 來接收所需的訊息。

以下就是運作 Dired 緩衝區的指令：

- **C** *new* **RET** (dired-do-copy)
  拷貝檔案。若有多個檔案同時要拷貝，則引數 *new* 代表檔案所要拷貝到的目錄。若只拷貝一個檔案，可利用此引數 *new* 將檔案從新命名。
- **R** *new* **RET** (dired-do-rename)
  更換檔名。若有多個檔案同時要換檔名，則引數 *new* 代表檔案換名稱後所要放置的目錄。若只有一個檔案，此引數 *new* 代表更換的檔名。當檔名更換完畢， Dired 緩衝區的檔案名稱會自動跟著更換。
- **H** *new* **RET** (dired-do-hardlink)
  將檔案標上 hard links 的標記。引數 *new* 代表 hard links 所要連接的目錄。若只有一個連接時，此引數 *new* 代表連結的名稱。

- **S** *new* **RET** (dired-do-symlink)
  將檔案標上 symbolic links 的標記。引數 *new* 代表 symbolic links 所要連接的目錄。若只有一個連接時，此引數 *new* 代表連結的名稱。
- **M** *midespec* **RET** (dired-do-chmod)
  更改特定檔案的模式（mode，permission bits）。此程式使用 `chmod` 的程式，所以 *modespec* 的引數值適用於 `chmod` 程式適用的引數。
- **G** *newgroup* **RET** (dired-do-chgrp)
  改變特定檔案的團體（group）為新的團體（*newgroup*）。
- **O** *newowner* **RET** (dired-do-chown)
  改變特定檔案的擁有者（owner）為新的擁有者（*owner*）。
- **P** *command* **RET** (dired-do-print)
  列印特定的檔案，可利用 *minibuffer* 輸入列印的指令 *command*。
- **Z** (dired-do-compress)
  壓縮或反壓縮特定的檔案。如果檔案已被壓縮則將其反壓縮，反之則將 檔案壓縮。
- **L** (dired-do-load)
  載入特定的 EMACS Lisp 檔案。
- **B** (dired-do-byte-compile)
  位元編譯（byte compile）特定的 EMACS Lisp 檔案。

• Dired 在編輯遠方檔案與使用 ftp 的用法
使用 EMACS 的遠方編輯或使用 EMACS 來做 ftp 時，若只給予目錄名，系統會進入 Dired 的模式。此時可使用運作 Dired 緩衝區的指令來操作所要的檔案。至於何為遠方的編輯會在下一節中討論。

3.3 Editing Files from Remote Host

EMACS 除了提供了一般編輯器所具有的功能之外，它還提供了一般編輯器所沒有的功能，那就是編輯遠方 host 的檔案。 EMACS 編輯遠方 host 的檔案，是使用了 ftp 的技巧，將所欲編輯的檔案 ftp 到目前的 host 上，待編輯完畢再以相同的技巧，把檔案 ftp 傳回遠方的 host 而已。

往昔要編輯遠方的檔案只有兩種做法，一是簽入（login）到檔案所在的 hsot 去，另一種就是以 ftp 的方法將檔案先傳回目前所在的地方，修改完後再 ftp 回去。

Remote Editing 也可以用到 "anonymous ftp" 上，它可以進入遠方的 目錄下，使用者即可根據需要挑選要 ftp 的檔案。

使用 remote editing 的方法非常簡單，在鍵入 "**Ctrl-x Ctrl-f**" 後，再根據語法給予適當檔名，EMACS 就會處理自行 ftp 遠方 host 上的檔案，其語法如下：

```
Find file:/host:filename
```

host 是指遠方 host 的名稱，*filename* 是指存放在遠方 *host* 的檔案。例如：

```
Fine file:/user1@gate.sinica.edu.tw :.login
```

就是編輯 *host*為 “`gate.sinica.edu.tw`” 的機器，而使用者為 “`user1`”，檔案的名稱為 “`.login`” 的檔案。

3.4 Editing , Compiling and Testing Programs

EMACS 是一個整合的環境，在提供程式編輯的同時，自然會提供一個可供程式執行的環境。以下就要談談 EMACS 可以為程式撰寫者提供那些服務。

EMACS 對於不同的語言提供不同的編輯模式。 EMACS 提供的服務有程式內縮的安排、括號對應的提示、程式註解的安排、游標移動的方式與程式的刪除等等。基本上，EMACS 是提供一個撰寫程式的格式，只是此格式可根據使用者的需要而自行設計。

EMACS 選擇適合的語言模式，是根據所編輯的檔案名稱附名來判斷的。如附名為 .c 的 C 語言程式，EMACS 會自動給予 C 語言模式，而不需使用者自行處理。EMACS 提供的程式語言模式有 LISP、SCHEME、C、C++、FORTRAN、MAKEFILE、AWK、PERL、ICON 與 MUDDLE 等。

編輯好的程式可以直接進入 EMACS 的編譯模式，不需離開 EMACS 到 UNIX 的 shell 下進行編譯的動作。進入 EMACS 的編譯模式很簡單，只要鍵入 “`ESC-x compile`” 即可。 EMACS 預設的編譯指令是 make ，執行 **ESC-x compile** 指令的結果如下所示：

```
compile command: make -k
```

若要使用其它的編譯器，只需在 “`compile command :`” 的後面給予適當的編譯指令即可，此指令與在 UNIX shell 下使用編譯的方法完全相同。

除了編輯、編譯之外，程式撰寫者還需要的功能是 Debugger 的提供。EMACS 也提供了此項的服務。 EMACS 提供了四種 debugger，分別為 gdb、dbx、xdb 與 sdb，使用者可根據需來選擇合適的 debugger。 此處，只將使用 debugger 的指令列舉如下：

- **ESC-x gdb RET** *file* **RET**
- **ESC-x dbx RET** *file* **RET**
- **ESC-x xdb RET** *file* **RET**
- **ESC-x sdb RET** *file* **RET**

3.5 Printing from Emacs

除了以上的功能外， EMACS 還提供了列印的功能。

EMACS 的列印可針對個整緩衝區或某部份的區域列印，其相關的指令如下：

1. **ESC-x print-buffer**
   列印整個緩衝區的內容。EMACS 處理此工作的方法是先使用 shell 的 pr 指令，而後再使用 shell 的 lpr 列印指令。

2. **ESC-x lpr-buffer**
   此指令與上一個指令相似，只是不透過 pr 而直接使用 lpr。

3. **ESC-x print-region**
   與 ESC-x print-buffer 相似，唯一不同之處在於，此指令只列印部份的區域。

4. **ESC-x lpr-region**
   與 ESC-x lpr-buffer 相似，唯一不同之處在於此，指令只列印部份的區域。

3.6 Sending and Reading Mail with Rmail

在 EMACS 眾多的整合功能中，信件收發的功能自然是不可或缺的。EMACS 對電子郵件的設計卻與一般的電子郵件系統背道而馳，它是在以編輯為前提的條件下來提供電子郵件的子系統；換言之，電子郵件為以編輯器為主導的一個子功能。

EMACS 所提供的電子郵件系統分成二個部份，一部分為發送信件（**mail**），另一部為收取信件（**rmail**）；收取信件的同時也能發送信件，它的作法是使用發送信件的功能將信件發送出去。

EMACS 讀取電子郵件，是將作業系統存放電子郵件的檔案拷貝至 EMACS 自己的檔案中，此檔案名為 `RMAIL`， EMACS 在讀取信件時會至此檔 （`RMAIL`）中讀取所要的信件。 EMACS 如此設計檔案的讀取有它的理由，其理由如下：

1. 作業系統存放電子郵件的格式不一，讀取電子郵件的軟體也是變化萬千。EMACS 的 RMAIL 就是要將如此複雜的事情簡單化。

2. RMAIL 記錄了信件所有的相關資料，但作業系統所提供的信件檔案，並未有做如此詳細的記錄。

3. 一般作業系統為了要確保信件讀取時的安全性，必需經由一套繁雜的方法來保障信件讀取的安全以及資料的不流失。EMACS 的 RMAIL 所採取的措失就是既安全又簡單。 RMAIL 的方法是，先把存於系統內的信件讀出後，再拷貝至 RMAIL 的檔案內，待一切就緒才將存在系統內的信件刪掉。如此作的好處是，即使系統當掉只會使信件多做一份拷貝，也不會有流失信件的事件發生。

EMACS 對於發送信件與讀取信件提供了許多有用的操作功能，讀者可自行參考相關手冊，此處只告訴讀者如何進入送信件與讀信件的模式：

- **ESC-x mail** （發送信件）
- **ESC -x rmail** （讀取信件）

3.7 Other Working Environment with Emacs

EMACS 除了提供以上種種的工作環境境，還有其它的環境可以運用，現在讓我們一一道來。

- **The Calendar and the Diary**
  EMACS 的 Calendar 與一般的月曆功能相似 。 `Diary` 更可以適時提醒使用者該注意的事情。 進入 Calendar 的方法如下：

  ESC-x calendar

  `Diary` 的使用可以在進入 calendar 的模式中後，再來設定相關的資料。

- **Reading Man Page from EMACS**
  閱讀 manual page 的方法非常簡單，其用法如下：

  ESC-x manual-entry RET *unix-command-name* **RET**

- **Reading News with GNUS**
  讀送網路新聞為現今交換電腦資訊的重要媒體與管道， GNU EMACS 也提供了此一子系統，稱為 GNUS。
  EMACS 的 GNUS 是將 ".newsrc" 檔的內容顯示出來，它的內容包括所有被訂閱（subscribe）的 newsgroups，以及未被閱讀的文章。在 GNUS 中還可以看到或隱藏未被訂閱的 newsgroups，並可以再訂閱未訂閱的 newsgroup 或取消訂閱某一個newsgroup。
  當然 GNUS 也提供了可遊走於各 newsgroups 間的指令。在 EMACS 中使用 GNUS 這一個子系統只需鍵入如下的指令即可。

  ESC-x gnus RET

- **version control** 管理原始檔案（source files）也是EMACS 所提供了服務項目之一。 功能。
  version control 是一個套裝軟體，它可以記錄一個原始檔案（source file）所有改變的版本（multipe versions），它保留所有改變的記錄且存放於一個檔案中，對於每一版本重複的部份只會保留一分記錄。version control 也會記錄每一版本被創造、誰創造了它等等的相關資料。

目前 EMACS 是透過 VC，來使用作業系統所提供的 RCS 或 SCCS 的version control 軟體。若作業系統提供 RCS， EMACS 會先使用 RCS，若無RCS 則會使用 SCCS。 EMACS 允許使用者自行決定使用 RCS 或 SCCS 的 version control。
透過 EMACS 的 VC，所能使用的 version control 的功能並不多，它只提供最基本的 version control 的功能，但確是最常為人使用的功能。若想使用 SCCS 或 RCS 所提供的所有功能，就必須進入 EMACS 的 shell mode 中了。以下就來簡介 EMACS version control 的功能。
EMACS 的 VC 提供的功能如下：

- 將檔案註冊於 version control 之下。
- 可將註冊的檔案從 version control 的控制中取出與放入。
- 放入 version control 的每一個版本都可以隨時取出。
- 可比較任一版本間的異同。
- 可將一組相關的檔案，置於 version control 之下。
- 可自行設計標頭（version header），此標頭可置於 version control 下的檔 案中。

至於其它的 version control 功能，則必需進入 EMACS 的 shell 中直接使用 RCS 或 SCCS。例如，將數個版本合併、使用 help 協助使用 version control 等的功能，就從 EMACS 所提供的 VC 得到解答。

- **Outline Mode** 以 editor 為基石的 EMACS 當然少不了製作文章大綱的功能。 EMACS 的大綱模式 (outline mode) 可以使文章的部份主體暫時隱藏起來，只呈現文章的大綱部份。如此一來，吾人不需維護一套為本文，另一套為大綱部份的兩套系統了。因為，製作本文的同時就已經隱含了大綱的部份了。

前面已經將 EMACS 可以做的事以及它的特質都做了簡要的介紹。以下就開始討論 EMACS 的基本要素 – 編輯功能。

Chapter 4

About the Emacs

上一章簡介了 EMACS 的整合環境，在討論編輯之前，先介紹 如何啟動 EMACS 與離開 EMACS；EMACS 對於螢幕的安排如何；以及緩衝區與視窗在 EMACS 中角色定位等等的問題。

EMACS 可用在 text-only 的終端機與 X window System 的視窗環境，但本文只針對 text-only 的終端環境來說明任何有關 EMACS 的介紹。

4.1 Starting Emacs

啟動 EMACS 的方法非常簡單，只要在 shell 的提示下鍵入 “emacs” 五個英文字就可以啟動 EMACS 了。例如：

```
$ emacs RET
```

啟動後的 EMACS 做了如下的啟始 (initialize) 動作：

1. 清除目前的螢幕，開始一個全新的 EMACS 螢幕。
2. EMACS 會在這個全新的螢幕，顯示一些與 EMACS 有關的基本訊息。其中包括，目前使用的 EMACS 版本、基本的線上輔助說明訊息以及有關 EMACS 版權的相關資訊等等。
3. 此時若不輸入任何指令， EMACS 會在一段時間之後（約二分鐘）自動將螢幕重新清除成一個空白的螢幕。
4. 若在螢幕自動重新清除之前鍵入指令，EMACS 會根據所給予的指令來做適當的 運作。

啟動 EMACS 的方法，不需要給予任何的檔名，只要輸入 emacs。因為 EMACS 是要建立一個能同時開啟多個檔案的編輯環境；更進一步希望開啟的檔案，能彼此共享一些訊息。 所以，在鍵入 “emacs” 的同時，給予所要編輯的檔名，就變得不實際了。

4.2 Exiting Emacs

知道如何啟動 EMACS 後，接下來就要探討如何離開 EMACS 了。離開 EMACS 的方法有兩種，一種是暫時離開 EMACS (suspending EMACS) ，另一種是永遠離開 EMACS (killing EMACS) 。其使用方法如下（為求統一，以後的各章節都先列出 EMACS 的 Hotkey，其相對應的命令則列舉在小括號內，若無 Hotkey 則直接列出其命令。要使用EMACS 的命令，通常要在每個命令前加上 Meta-x 或 ESC-x ）：

- **Ctrl-z (suspend-emacs)**
  暫時離開 EMACS 回到其上一層的狀態，一般是回到 shell 的狀態。
  若想回到 EMACS 的狀態，只要鍵入 ”`%emacs`”，則可以回到 EMACS 了。
- **Ctrl-x Ctrl-c (save-buffers-kill-emacs)**
  永久離開 EMACS。
  以此方法離開 EMACS，除了重新啟動 EMACS 方外（即在 shell 的提示下鍵入 `emacs`），沒有其它的方法可以再回 EMACS 了 。

暫時離開 EMACS 意思是回到上一層的狀態（parent process），一般是指 shell。使用者可以隨時回到原先所啟動的 EMACS 下，對於所使用的緩衝區、 kill ring 以及 undo history 等相關資訊，仍保持與離開前相同的狀態[1]。以 **Ctrl-z** 指令暫時離開的 EMACS ，可以在 shell 的提示下，以 “`%emacs`” 回到離開前的 EMACS 下。有些系統或 shell 並不提供這種暫時離開的功能，此時只能永遠離開 EMACS 而無法暫時離開 EMACS 了。

要永遠離開 EMACS 則必需鍵入 “`Ctrl-x Ctrl-c`” 或 “`ESC-x save-buffers-kill-emacs`”，EMACS 接收此指令後會展開如下的動作：

1. EMACS 會主動提醒使用者，儲存所有修改過的檔案。
2. 當使用者對需要儲存的檔案做了適當的處理後， EMACS 對於所有仍在執行的 subprocess，也會主動提醒使用者是否要結束它們。因為離開 EMACS 的同時也就是結束這些 subprocess 的時侯。

在永久離開 EMACS 前， EMACS 會再三的提醒使用者有關檔案的儲存與仍在執行的程式等等。因為，一旦永久離開 EMACS 之後，所有未存檔或尚在執行的 subprocess 都會隨之消失。

EMACS 對於所有未儲存的檔案與仍在進行的程序，會利用 *echo area* 一一提醒遺忘它們的使用者。*echo area* 會提示需要儲存的檔案，同時也提供可處理這些檔案的方法。 所以 *echo area* 除了顯示要存檔的檔案名稱外，還會在檔名之後出現如下的訊息：

```
(y, n, !, ., q, C-r or C-h)
```

這些訊息提供，就是要讓使用者對於檔案或程序有適當處理的機會。現在就對這些訊息做一討論。

[1]有關 kill ring 、undo history 等相關 資料會在以後的各章陸續提及

1. **y**
   同意對 *echo area* 所顯示的緩衝區存檔，並徵詢對於其它檔案是否存檔的意見。

2. **n**
   放棄對 *echo area* 所顯示的緩衝區存檔，但徵詢對於其它檔案是否存檔的意見。

3. **!**
   同意對 *echo area* 所顯示的緩衝區存檔，且對其它的緩衝區也一併存檔，不再徵詢其它檔案是否存檔的意見。

4. **.**
   同意對 *echo area* 所顯示的緩衝區存檔，但對其它的緩衝區則不再徵詢是否存檔的意見，直接放棄其它緩衝區的存檔，且離開此存檔的狀態。

5. **q**
   放棄存檔的狀態而不執行任何存檔的動作。

6. **C-r**
   此指令可用來流灠目前所要儲存的檔案內容，當離開此流灠狀態即回復存檔的模式，系統會再度詢問與存檔有關的訊息。

7. **C-h**
   對於以上的選項若有不明白的地方，可以此功能查閱其意思。

4.3 Screen Organization

在 text-only 的終端機啟動 EMACS 時， EMACS 會佔據整個螢幕，此時的螢幕稱為 *frame*。再一次的強調，本文只討論 text-only 的終端機，至於 X Window 的環境則不在討論的行列中。

text-only 的 *frame* 又由數個 *window* 所組成。 啟動 EMACS 時，會產生二個預設的視窗，一個視窗用來輸入一般的文件，在未有文件輸入前先用來展示前面提到的 EMACS 版本、線上輔助說明以及有關版權等訊息；另一個視窗用來輸入指令或是用做訊息的回應，稱為 *minibuffer* 或 *echo area* 。

若終端機提供反白的功能，在反白區域以上的地方是用來輸入文件的視窗；反白區域以下的地方則是 *minibuffer* 或 *echo area*。此反白的長條型則稱為 *mode line* ，它是用來描述輸入文件視窗的一些訊息。現在就來談談組成 EMACS *frame* 的這三個部份。

有關文字視窗的部份，因為還牽涉到緩衝區的問題，現在先略過不談，下一節再行討論。現在先討論與其有關的*mode line* 和位於 *mode line* 下的 *minibuffer* 或 *echo area*。

mode line 出現在每一個文字視窗的最後一列，其描述此視窗的相關資訊。*mode line* 所描述的訊息如下:

```
--ch-Emacs: buf    (major minor)  --pos--------------
```

現在則分別解釋其所代表的意義。

- **Ctrl-h** 代表緩衝區的狀態（何謂緩衝區會在下一節討論）。
  - – 表示緩衝區未被修改過。
  - ** 表示緩衝區已被修改過。
  - %% 表示緩衝區為 read-only 的緩衝區。
  - %* 表示 read-only 的緩衝區被修改過。
- **buf**
  表示此視窗緩衝區的名稱，一般即為所編輯的檔案名稱。
- **major minor**
  此緩衝區所有使用的模式（*mode*）都列舉在此括號內。其中包括一個主要模式 (major mode) 和數個次要模式（*minor mode*）。EMACS 允許一個緩衝區有數個次要模式，但只能有一個主要模式。
- **pos**
  表示文件在視窗顯示的情形。其表示的種類如下：
  - **All**
    如果資料很少可以一「幕」瞭然，則會以 All 來表示。
  - **Top**
    若資料無法一「幕」瞭然，但出現的位置在最前面，則以 Top 來表示。
  - **Bot**
    無法一「幕」瞭然的資料，出現的位置是在最尾端，則以 Bot 來表示。
  - **nn%**
    若資料出現的位置不在第前端也不在最後端，則以百分比來表示資料出現的情形。

介紹完了 *mode line*，現在來談談 *echo area* 與 *minibuffer*。在 *frame* 的最後一列，也就是 *mode line* 的下一列就是 *echo area* 或 *minibuffer* 出現的地方。二者使用同一區位但所代表的意思卻不相同。

Echoing 的意思就是將鍵入的字元在螢幕上回應出來。EMACS 對於只有一個字元的指令並不會把它 Echoing 出來，例如 "**Ctrl-e**"。對於多個字元的指令，只要在鍵入指令的時候給予稍許的停頓，*echo area* 就會把鍵入的指令回應出來。等第一次的回應產生時，再輸入的部份就不需要再給予停頓的時間，其回應會在鍵入的同時立即產生。*echo area* 除了回應鍵入的指令，也會將指令所產生的訊息顯示出來；錯誤訊息的顯示也是利用此區域。

minibuffer 所使用的地方與 *echo area* 相同。它本身也是一個視窗，是用來輸入執行指令所需的引數（argument）。使用 *minibuffer* 的同時也會使用 *echo area*。

minibuffer 輸入引數的地方，是在 *echo area* 回應字串的 "：" 之後。因為 *echo area* 的回應是以 "：" 的出現做為結束。換言之，"："冒號之後就是 *minibuffer* 輸入引數的地方。

例如，要訪問一個檔案，鍵入指令 "`Ctrl-x Ctrl-f`" 時，*echo area* 會出現

```
Find file:
```

“`Find file`” 就是 *echo area* 的回應字，而此回應字串以 “：” 做為結束。 所以 “：” 之後，就是 *minibuffer* 的地盤了，也就是 *minibuffer* 輸入引數的地方。

使用 *minibuffer* 時，游標會自動移至 *minibuffer* 所在處，當游標在 *minibuffer* 時，就表示可以輸入引數了。若游標因為某些原因不出現在 *minibuffer* 的位置，此時可以 “**Ctrl-x o** (**other-window**)” 使游標在視窗間移動，直到游標出現在 *minibuffer* 所在的視窗為止。 若已在 *minibuffer*的狀態，但不想輸入任何引數，此時可以 “**Ctrl-g** (**keyboard-quit**)” 離開 *minibuffer*。輸入 “`Ctrl-g`” 後，游標會移至其它的視窗。

minibuffer 也是一個視窗，所以可以從別的視窗移至此視窗； “**Ctrl-x o**” 的指令就是用來使游標在各個視窗間移動的。一般的 *minibuffer* 都只有一列的高度，但有時一列的高度無法將資料顯示完畢，此時的 *minibuffer* 就需要調整其大小了。至於如何將 *minibuffer* 的視窗做調整，就是下一節所要討論的重點之一了。

4.4 Buffers and Windows

EMACS 的緩衝區與視窗的關係密不可分，緩衝區是用來存放編輯文件的，但視窗卻是用來顯示緩衝區的文件。現在就來談談緩衝區和視窗。

緩衝區 (buffer) 是 EMACS 編輯文件時，暫時存放文件的地方。 這個地方只用來暫時存放文件，要想永久保留這些文件，必需將暫時存放的文件儲存起來，一般是使用硬碟來安置緩衝區的文件。

在 EMACS 中所做的任何事情，都是先暫放於緩衝區內。 EMACS 處理檔案的方式，也是先將檔案從硬碟中取出後，再放於緩衝區內。所以不論是刪減、修改與新增文件，都是在緩衝區內進行，除非將緩衝區內的文件存回硬碟，否則硬碟的內容都不會因緩衝區內容的改變而改變。

文件未存回硬碟而離開 EMACS (kill EMACS)，將永遠消失。但 EMACS 有一個自動儲存文件的功能，稱為 “*auto save*”。每當鍵入一定數量的字元（通常是三百個字元），EMACS 就會自動做儲存的動作；經過一段停置的時間（通常是三十秒）， EMACS 也會做自動儲存的動作。

EMACS 自動儲存的功能並非將文件直接存回該檔案所在的硬碟中，而是將緩衝區的文件存入一個暫存檔內。只有以存檔的指令，例如 “**Ctrl-x Ctrl-s**” 的指令，將緩衝區的文件存回硬碟時，緩衝區內的文件才會存回硬碟中。只有當文件存回硬碟中，EMACS 才會自動清除此暫存檔。若緩衝區的內容一直未存回硬碟，此暫存檔就會一直存在著，直到存回硬碟才會消失。

EMACS 如此安排暫存檔有兩個好處，第一個好處是可以確保編輯的檔案資料不會流失；第二個好處是可預防機器意外關機或當機，檔案不及存回硬碟，所造成的損失。EMACS 命名此暫存檔的方式，是以緩衝區所使用的檔名為依據。在檔名的前後各加上一個 “#”，就是暫存檔的名稱。舉例說明，若所編輯的檔名為 “`emacs.doc`”，其產生的暫存檔即為：

```
#emacs.doc#
```

若所編輯的檔案未存回硬碟時， EMACS 會自動產生一個暫存檔。下次編輯此檔時， EMACS 允予使用者從暫存檔中將流失的資料回復。例如編輯的檔案為 “emacs.doc”，在離開 EMACS 時未存回硬碟 ， EMACS 會自動產生一個 “#emacs.doc#” 的自動儲存檔。當重新啟動 EMACS 且編輯 “emacs.doc” 檔時， EMACS 會提示使用者此檔案已被更改過但未給予適當的儲存。此時，使用者可自行決定是否要從自動儲存的檔案中 (#emacs.doc#) 將 “emacs.doc” 檔中未被儲存的資料找回。

如何從自動儲存檔中將資料找回呢？想要從自動儲存的檔案中，恢復原始檔案中流失的資料，可以使用 “**Meta-x recovery-file**” 的指令。若存放於硬碟中的檔案，有相對應的自動儲存檔時，可經由如下的步驟將資料找回：

1. 鍵入 “Ctrl-x Ctrl-f RET”
   Find file: ~/*filename*

2. 在 Find file: 處輸入所欲編輯的檔案後後， *echo area* 會出現如下的訊息：
   Auto save file is newer: consider M-x recovery-file

3. 鍵入 “Meta-x recovery-file RET”
   此時 *echo area* 會自動出現相對應的自動儲存的檔案名稱，若願意執行恢復的動作，只要直接按下 RET 即可。否則，以 “Ctrl-g” 指令，放棄此命令的執行。

除了暫存檔外， EMACS 對於每一個編輯的檔案，都會在編輯前做一份備份，以防止在編輯的過程中因一時的疏忽而將檔案毀損。備份檔的設計是，當檔案被存回硬碟後，備份檔也不會因此而消失。

EMACS 命名備份檔的方式，是在所要編輯的檔名之後加上 “~” 。例如， “emacs.doc” 的備份檔就為 “emacs.doc~”。

以上的設定是可以改變，因為它們都是變數。下面列出相關的變數，使用者可自行決定其所需
。

- **auto-save-visited-file-name**
  設定自動儲存檔案的種類。可以設為暫存檔也可設為正在使用的原檔案。

- **delete-auto-save-file**
  設定檔案被存回硬碟後，自動儲存的暫存檔是否會自動刪除。

- **auto-save-interval**
  設定自動儲存時的字元數。

- **auto-save-timeout**
  設定自動儲存時的時間。

想知道如何設定變數嗎？在 EMACS 中任何設定變數的方法都是以 “**Meta-x set-variable** (或 **ESC-x set-variable**) ” 的指令來完成變數的設定。變數值的設定，可以只設定真假值或設定數值或是設定字串。

若只是設定變數的肯定或否定值時， EMACS 有一個遵循的規則。EMACS 中以任何 “`non-nil`” 的值來代表肯定，習慣上是以 “`t`” 來表示肯定；而以 “`nil`” 來代表否定。

在設定新的變數值之前，若想知道目前變數的值， EMACS 可以 “**Ctrl-h v**” 來查閱變數的值。現在就舉設定 “auto-save-visited-file-name”，“和 auto-save-interval” 二個變數來說明變數設定的方法。

1. 以 “**Ctrl-h v**” 查閱 auto-save-visited-file-name 的變數。
2. 以 “**Meta-x set-variable**” 來設定變數 。
3. 再以 “**Ctrl-h v**” 來查閱所設定的 auto-save-visited-file-name 變數。

現在來看看設定 auto-save-visited-file-name 這一個變數的實際過程：

1. 鍵 “`Ctrl-h v RET`”
   (a) *echo area* 處會出現 `Describe variable`：
   (b) 在 `Describe variable:` 後鍵入 `auto-save-visited-file-name`
   (c) 螢幕上會另開一個視窗，顯示如下的訊息：

```
auto-save-visited-file-name's value is nil

Documentation:
*Non-nil says auto-save a buffer in the file it is visiting, when
practical.
Normally auto-save files are written under other names.
```

2. 鍵入 “`Meta-x set-variable`”
   (a) *echo area* 處會出現 `Set varaible:`
   (b) 在 `Set variable:` 後鍵入 `auto-save-visited-file-name RET`
   (c) *echo area* 處會出現 `Set auto-save-visisted-file-name to value`：
   (d) 此時可以利用在 “：” 之後的 *minibuffer*，輸入變數的值。此變數的值不是肯定就是否定的。目前的值是 `nil`，要改其值為肯定的可以輸入 `t`。
3. 再以 “`Ctrl-h v`” 來檢視變數設定的情形。

上一個例子是設定肯定與否定值的例子，現在來看看設定變數值為數字的例子。 auto-save-interval 實際執行的過程：

1. 鍵入 “`Ctrl-h v RET`”
   (a) *echo area* 處會出現 `Describe variable`：
   (b) 在 `Describe variable:` 後鍵入 `auto-save-interval`
   (c) 螢幕上會另開一個視窗，顯示如下的訊息：

```
auto-save-interval's value is 300

Documentation:
*Number of keyboard input characters between auto-saves.
Zero means disable autosaving due to number of characters typed.
```

2. 鍵入 “`Meta-x set-variable`”
   (a) *echo area* 處，會出現 `Set varaible :`
   (b) 在 `Set variable:` 後鍵入 `auto-save-interval RET`
   (c) *echo area* 處會出現 `Set auto-save-interval to value`：
   (d) 此時可以利用 “：” 之後的 *minibuffer*，輸入變數的值。此變數的值為數字。目前的值是 300，使用者可根據需要輸入適當的數字。
3. 再以 “`Ctrl-h v`” 來檢視變數設定的情形。

在 EMACS 執行過程中所設定的變數值，只對目前所執行的 EMACS 有用，一旦離開此 EMACS，所有的設定就恢復成原來的預設值。要想永久保留此設定的變數值，就必需將所設定的變數值儲存在檔名為 “`.emacs`” [2]的檔案中。因為啟動 EMACS 時， EMACS 會先執行 `.emacs` 檔，所有存於此檔案的變數會被重新設定一次。

在 `.emacs` 檔中設定變數 “auto-save-visited-file-name” 與 “auto-save-interval” 的方法如下所示：

```
(setq auto-save-visited-file-name t)
(setq auto-save-interval 350)
```

前已述及 EMACS 可以容許多個緩衝區的同時存在，既然如此，自然有其處理每個緩衝區的方法。現在就來看看 EMACS 如何處理緩衝區。

- **Ctrl-x b** *buffer* **RET (switch-to-buffer)**
  此指令用來選擇不同的緩衝區，其預選的緩衝區是目前所使用的緩衝區之外，最近被使用過的緩衝區。此指令可以使用 *completion* 。
  使用此指令， *echo area* 會出現如下的訊息：

[2].emacs 檔為 EMACS 的啟始檔，進入 EMACS 時會先執行此檔內的指令， EMACS 的設定也是根處此檔而來的

```
Switch to buffer: (default filename)
```

若所要選擇的緩衝區不是系統所預設的，可以利用 *minibuffer* 將所要選擇的緩衝區名稱鍵入。

- **Ctrl-x k** *buffername* **RET (kill-buffer)** 此指令是用來刪除 *minibuffer* 所顯示的緩衝區。若只鍵入 RET，則刪除目前的緩衝區，否則，刪除所輸入的緩衝區名稱。同樣的，此指令可以使用 *completion*。

- **Ctrl-x Ctrl-b (list-buffer)** 將目前 EMACS 所使用過的緩衝區顯示出來。
  以下的就是執行 "`Ctrl-x Ctrl-b`" 時，視窗所顯示的資料：

```
 MR Buffer         Size   Mode            File
 -- ------         ----   ----            ----
.*  chap4.tex      17460  LaTeX          /home/usr/hsko/work/chap4.tex
 *% RMAIL          8788   RMAIL          /home/usr/hsko/RMAIL
 *  *Buffer List*  241    Buffer Menu
    *scratch*      0      lisp Interaction
    diary          928    Fundamental    /home/usr/hsko/diary
  % *man ls*       15420  Man
    *Help*         64     Fundamental
```

以上資料的每一欄位各有其所代表的意思，詳述如下：

- 欄位 **MR**，標記緩衝區的狀態，其可能的狀態如下所示：
  - "*"
    表示此緩衝區被修改過。
  - "`.*`"
    "`.`" 表示此緩衝區為目前被選擇的緩衝區， "`.*`" 表示此選用的緩衝區被修改過。
  - "%"
    "%" 表示此緩衝區為 read-only 的緩衝區。
  - "%*"
    表示此 read-only 的緩衝區被修改過。
- 欄位 **Buffer**，顯示所使用的緩衝區名稱。
  - **Buffer** 中的資料若為檔案名稱時，則表示緩衝區所放置的資料為一個檔案。
  - 若 **Buffer** 中的資料前後加上了 "*"，則表示此緩衝區不是任何被訪問的檔案。
- 欄位 **Size**，顯示緩衝區的大小。
- 欄位 **Mode**，顯示緩衝區所使用的主要模式。
- 欄位 **File**，表示所訪問檔案的絕對名稱。若緩衝區的資料不是來自訪問的檔案，亦即欄位 **Buffer** 的名字前後加上 "*" 時，則以空白表示。

- **Meta-x buffer-menu**
  此指令好似 Dired 的功能，應用在緩衝區上。
  此指令可對列出來的緩衝區各別做運作。其運作內容包括，儲存緩衝區、刪除緩衝區、顯示緩衝區以及編輯緩衝區等等。其實運作於 "**Meta-x buffer-menu**" 的指令同樣也可用在 "**Ctrl-x list-buffer**" 上，只是使用 "**Meta-x buffer-menu**" 指令時， echo area 處會顯示出可運用的選項。其可運用的選項內容如下所示：

  ```
  Command: d, s, x, u; f, o, 1, 2, m, v; ~, %; q to quit; ? for help.
  ```

  現舉較常使用的選項說明，至於其餘的選項，使用者可鍵入 "?"，來使用其所提供的線上述助。

  - **d**
    標示所欲刪除的緩衝區。
    在 **MR** 欄位的最前方會出現 `D`。此時並未真正刪除緩衝區，只是將要刪除的緩衝區做上標記，直到下達執行標記的命令時，才會真正將標示 `D` 的緩衝區刪除。此執行的指令為 "**x**"。
  - **s**
    標示所欲儲存的緩衝區。
    在 **MR** 欄位處標示上 `S` 。此時並未真正做儲存的動作，只是在要儲存的緩衝區做上標記，直到下達執行標記的命令時，才會真正將標示 `S` 的緩衝區存檔。
  - **x**
    對做好標記的緩衝區，下達執行的命令。也就是對標示有 `D` 與 `S` 的緩衝區，做執行的動作。
  - **u**
    將設好的標記取消。
  - **f**
    選擇目前游標所在處的緩衝區。此時的視窗會將此緩衝區的內容顯示出來、

談完了 EMACS 的緩衝區，現在來談談與其關係密切的視窗。前已略述，進入 text-only 的 EMACS，即進入一個 *frame*。 一個 *frame* 由數個視窗組成，每一個視窗顯示一個 EMACS 的緩衝區，且一次只顯示一個緩衝區的內容。

EMACS 在任何時候，總有一個視窗為選擇的視窗（*selected window*） 。此視窗所顯示的緩衝區，則稱為目前的緩衝區（*current buffer*） 。 *point* (或稱為游標） 所在的視窗，就是 EMACS 的選擇視窗。EMACS 是透過游標來表示 *point* 所在的位置。所以說，若想要知道目前的選擇視窗，觀察游標所在的位置就可知道。

至於什麼是 *point* 呢？ *point* 就是用來標示目前所使用的視窗或緩衝區所在的位置。EMACS 的每一個視窗，各有其所專屬的點位置 （*point location*） 。每一個緩衝區 也有屬於它自已的點位置。每一個緩衝區或視窗 *point* 的位置，並不會隨著視窗或緩衝區的改變而變動。換言之， *point* 的位置會隨時被記錄下來，當再次訪問其它的視窗或緩衝區時，游標仍會回到離開前的位置。 所

以，任何移動 *point* 的指令，只會對所選擇的視窗產生影響，對於其它視窗的 *point* 是不會有任何影響的。

EMACS 下的每個視窗除了有各自的 *point* 外，也各自有其相對應的 *mode line*。

EMACS 視窗的大小是容許重新調整的。除了大小是可以調整的，一個視窗也可以再分成兩個視窗。其分割的方法，可以做水平或垂直的化分。視窗的操作，除了分割視窗之外，也可以使游標在不同的視窗間移動；當然，將不需要的視窗刪除也是基本的功能。現在就來看看與視窗相關的指令。

- **Split Window and Rearranging Windows**
  - **Ctrl-x 2 (split-window-vertically)**
    將一個視窗分成上下兩個視窗。此時化分出來的兩個視窗，分享著化分前視窗的緩衝區。換言之，此時兩個視窗的緩衝區內容是一樣的。因為共享著同一個緩衝區，所以改變其中一個視窗緩衝區的內容，也會改變另一個視窗緩衝區的內容。分割成兩個視窗的好處之一是，可以編輯一個緩衝區，將另一個緩衝區作為參考的依據。
    * **Ctrl-x ^ (enlarge-window)**
      將目前游標所在的視窗拉長一列。此指令，只有在 *frame* 存在一個以上的視窗時才有效果。若只有一個視窗，此視窗就佔據了整個 *frame*，此時自然就無多餘的空間可以放大了。
    * **Ctrl-u *n* Ctrl-x ^ (enlarge-window nn)**
      與 **Ctrl-x** ^ 指令相似。不同之處在於，此指令可以將目前游標所在的視窗拉長 *n* 列以上。
  - **Ctrl-x 3 (split-window-horizontally)**
    將視窗分成左右兩個視窗。此時的兩個視窗依然擁有相同的緩衝區，所以，改變一個視窗緩衝區的內容，同時也會改變另一個視窗緩衝區的內容。
    * **Ctrl-x } (enlarge-window-horizontally)**
      將目前游標所在的視窗拉寬一行。對 *frame* 存有一個以上的視窗才有效。若只有一個視窗，此視窗已經佔據整個 *frame* 了，即使想放大，恐怕也愛莫能助了。
    * **Ctrl-u *n* Ctrl-x } (enlarge-window-horizontally nn)**
      與 **Ctrl-u** } 指令相似。此指令可以將目前游標所在的視窗拉寬 *n* 行。
- **Using Other Windows and Deleting Windows**
  - **Ctrl-x o (other-window)**
    此指令是用來選擇所欲使用的視窗。注意，此 "o" 是英文字的 "o"，而非數字的 "0"。
  - **Ctrl-x 0 (delete-window)**
    將目前游標所在的視窗刪除。提醒大家注意，此 "0" 是阿拉伯數字的 "0"。
  - **Ctrl-x 1 (delete-other-window)**
    保留目前游標所在的視窗，其餘的視窗全部刪除。

視窗的大小是有限的，但緩衝區的內容卻經常超過視窗所能顯示的範圍。接下來就是要告訴各位，如何在有限的空間中，以窺緩衝區的全貌。

想要在視窗的局限下，洞悉緩衝區的全貌，其最基本動作就是捲動螢幕。所謂螢幕的捲動，就是螢幕上下左右的移動。除了捲動螢幕之外，還需考慮螢幕的清除。在那些情形之下要清除螢幕呢？例如，遠方送來的 message， 在螢幕的顯示久久不退；系統送來的訊息也在螢幕上不會消失。這些情況的發生，都值得將螢幕做清除且重新顯示的動作。以下就來看看這些相關的指令。

- **Ctrl-l (recenter)** （清除螢幕）
  清除螢幕且重新顯示螢幕。
- 使螢幕上下捲動
  - **Ctrl-v (scroll-up)** （向上捲動螢幕一列）
    向上捲動螢幕，且將目前螢幕的最後二列做為捲動後螢幕的前二列。 *point* 出現在螢幕的第一列。
  - **Ctrl-u** *n* **Ctrl-v** （向上捲動螢幕 *n* 列）
    螢幕向上捲動 *n* 列。
    若指定捲動的列數 *n* ，不超過 *point* 在此螢幕上所在的列數，捲動後的 *point* 仍會留在原處不動，否則 *point* 移至螢幕的第一列。
  - **Meta-v (scroll-down)** （向下捲動螢幕一列）
    向下捲動螢幕， 且將目前螢幕的前二列做為捲動後螢幕的後二列， *point* 出現在螢幕的最後一列。
  - **Ctrl-u n Ctrl-v** (向下捲動螢幕 *n* 列)
    螢幕向下捲動 n 列。
    若向下捲動的列數 *n*，不超過 *point* 在此螢幕所在位置以下的列數， 捲動後的 *point* 仍會留在原處不動，否則 *point* 移至螢幕的最後一列。
  - **Meta < (beginning-of-buffer)** （螢幕捲至緩衝區的最前端）
    將螢幕捲至緩衝區的最前端， *point* 也移至第一列。
  - **Meta > (end-of-buffer)** （螢幕捲至緩衝區的最尾端）
    將螢幕捲至緩衝區的最尾端， *point* 也移至最後一列。
  - **Ctrl-ESC-v (scroll-other-window)**
    前面六個捲動螢幕的指令，都是針對游標所在位置的視窗而言，若要捲動其它 的視窗，則必需使用 “**Ctrl-ESC-v**”。
    “**Ctrl-ESC-v**” 指令是用來捲動游標所在位置下一個視窗的螢幕。所以，如果開啟了兩個視窗，可以使用此指令來參考非游標所在位置的緩衝區資料。此指令方便之處在於，可以省略移動游標的步驟，就可以捲動其它的視窗。
    如果今開啟了 A 與 B 二個視窗，游標在視窗 A 處，以 “**Ctrl-ESC-v**” 可以捲動視窗 B。如果開啟二個以上的視窗， “**Ctrl-ESC-v**” 指令所捲動的視窗，就必需看那一個視窗最接近游標所在的視窗了。
- 使螢幕左右捲動
  - **Ctrl-x < (scroll-left)**
    螢幕向左捲動。
  - **Ctrl-x > (scroll-right)**
    螢幕向右捲動。

- 上下移動游標
  - **Ctrl-p** 或 $\uparrow$ **(previous-line)**
    Ctrl-p 可以使游標向上移動一列。若有設定功能鍵，則可以用 $\uparrow$ 來移動游標。
  - **Ctrl-u** n **Ctrl-p**
    使游標向上移動 n 列。
  - **Ctrl-n** 或 $\downarrow$ **(next-line)**
    Ctrl-n 可以使游標向下移動一列。若有設定功能鍵，則可以用 $\downarrow$ 來移動游標。
  - **Ctrl-u** n **Ctrl-n**
    使游標向下移動 n 列。

Chapter 5

Basic Emacs Editing Commands

前面的四個章節已經把 EMACS 的環境與架構做了扼要的介紹，現在開始討論 EMACS 做為編輯器（editor）所能提供的服務。

5.1 Loading/Saving File

使用編輯器最基本的需求，就是要能載入檔案以便編輯。現在就先來看看 EMACS 是如何處理檔案的載入。

EMACS 載入檔案的方法很簡單，只要在鍵入 "`Ctrl-x Ctrl-f`" 之後，再利用 *minibuffer* 輸入所要編輯的檔名即可。檔名的輸入可以與 *completion* 相互搭配。

鍵入 "`Ctrl-x Ctrl-f`" 命令之後，在 *mininbuffer* 處輸入已存在硬碟的檔案， EMACS 會執行如下的過程：

1. 產生一個新的緩衝區。
2. 將所欲編輯檔案的內容，拷貝至緩衝區內。
3. 將緩衝區的內容顯示出來，以便編輯。

這整個事件的過程，在 EMACS 的編輯系統中稱為「訪問檔案」（*visiting file*）。

雖然載入檔案的方法很簡單，但仍有幾件事情值得討論的，現說明如下：

- 鍵入 "`Ctrl-x Ctrl-f`" 後， *echo area* 會顯示目前緩衝區的目錄，此時若編輯檔案的目錄與 *echo area* 所顯示的目錄相同，可逕在其後輸入檔名即可。
- 若檔案所在的目錄與目前緩衝區的目錄不同，輸入檔名的方法有如下幾種：
  - 檔案在同一個機器上的做法：

  * 無視 *echo area* 所顯示的目錄，由使用者重新輸入檔名。鍵入新檔名的方法，是在 *echo area* 顯示目錄的最後方鍵入 “/” 。
    “/” 代表， 忽略“/” 之前面的路徑，新的路徑從 “/” 之後開始有效。新的路徑必須鍵入絕對路徑名 (absolute file name) 。現舉一實例說明。
    鍵入 “`Ctrl-x Ctrl-f`” *echo area* 顯示的目錄如下：

    ```
    Find file: ~/work/emacs/emacs.tex
    ```

    所要編輯的檔案在

    ```
    ~/work/doc/text.tex
    ```

    此時輸入正確檔名的做法如下所示：

    ```
    Find file:~/work/emacs/emacs.tex//~/work/doc/text.tex
    ```

  * 以 EMACS 提供的刪除功能，將不必要的字刪除之後，再輸入正確的資料。
    除非顯示的目錄與所欲編輯的目錄相差無幾，使用刪除的方法才有意義，否則不如放棄所顯示目錄，重新開始新的生涯才是正途。至於如何刪字，會在 5.2.5 一節中說明，此處就不贅言。

- 所欲編輯的檔案，不在目前所在的機器上的做法，如下所示：
  前已述及 EMACS 可以 FTP 的方式來編輯遠方的檔案， 但 EMACS 是如何以 FTP 的方法來編輯遠方旳檔案呢？
  想要編輯遠方的檔案，只要給予正確的語法， EMACS 就會根據其語法來決定是否要使用 FTP 來編輯此檔案了。
  其語法很簡單，只有 `/host:filename` 而已。如何來使用其語法呢？由其語法可知，其語法是由四組元素所組成的字串，包括二組子字串 (host 和 filename) 以及二個符號 (/ 和 :) 。在 *minibuffer* 處輸入 / 後，緊接給予檔案所在的機器名稱 (host) ，在其後立即給予 “:” ，在 “:” 後則輸入所欲編輯檔案，整個訪問檔案的過程就完成。注意此四組資料間，不可以留有任何的的空白。現舉一實例子來說明之。

  ```
  Find file:~/work/emacs.tex//hsko@gate.sinica.edu.tw:~/work/text.tex
  ```

• 鍵入 “`Ctrl-x Ctrl-f`” 當 *echo area* 顯示出目前緩衝區的目錄後，只給予 RET 而別無它物時， EMACS 以目前緩衝區所使用的檔案為預設的檔案。

• 鍵入 “`Ctrl-x Ctrl-f`” 後，突然改變心意，想知此目錄以外其它目錄的檔名，除了使用 “**Ctrl-x d**” (使用目錄的編輯指令－dired) 指令外，還可以利用現有的目錄名稱來得到想要的訊息。其作法是直接修改 *echo area* 所顯示的目錄，直到所要的目錄出現後，鍵入 RET，此時會另開啟一個視窗來顯示出此目錄下的所有檔名。若想操作這些檔案，其操作方式與運作 Dired 的方式相同。若對 Dired 的印象已經模糊的人，請參閱 3.2 節 。

以上是介紹檔名的輸入方式。在編輯資料的同時，是否可以輕易查得目前所在的目錄？"**Meta-x pwd**" 指令可以滿足這種需求。鍵入 " `Meta-x pwd`" 後， EMACS 會假借 *echo area* 將目前所在的目錄顯示出來。

以上談的都是 EMACS 輸入檔案的方式，但 EMACS 又是如何為其緩衝區命名的呢？其實使用者並不需要為緩衝區命名，因為 EMACS 會自動給予緩衝區合適的名稱。EMACS 為緩衝區的命名，可以從 *mode line* 上得知。

EMACS 命名緩衝區的方式是根據所鍵入的檔案名稱而來，它捨棄了所有的目錄名稱只保留編輯檔案的檔名。所以若全名為 "`/user/work/emacs.tex`" 的檔案，其緩衝區的名稱則為 "`emacs.tex`"。

除了將檔案從硬碟直接載入緩衝區外，在編輯檔案的同時，有時需要參考其它的檔案，甚至需要引進其它的檔案到目前所使用的緩衝區內。 EMACS 的 "**Ctrl-x i**" 指令，就是讓使用者能隨時引進其它的檔案到目前的緩衝區內。 "**Ctrl-x i**" 的 "**i**" 是 insert 的意思，相信瞭解意思後，對於該指令應有較深刻的印象，否則怎有知已知彼，百戰百勝之說呢？

以下是 EMACS 有關檔案載入指令的整理：

- **Ctrl-x Ctrl-f (find-file)**
- **Ctrl-x i (insert-file)**
- **Meta-x pwd**

文件編輯完後，最重要的事情就是要能將其保留下來。以下就來談談文件儲存的方法。檔案儲存不外乎將緩衝區的內容以原名或易名存回磁碟；此儲存方式可以選擇一次存一個檔案或一次存數個檔案；最後的考量是存完檔案後是否要直接離開 EMACS。 下面就來看看 EMACS 所提供的相關指令。

- 存檔但不離開 EMACS
  - **Ctrl-x Ctrl-s (save-buffer)**
    將目前緩衝區的內容，存回磁碟中。存回的檔案名稱與緩衝區的名稱相同。此指令只對目前所使用的緩衝區做存檔的動作。若緩衝區的內容未有任何的變動則 *echo area* 會顯示如下的文字：

    ```
    (No changes need to be saved)
    ```

  - **Ctrl-x Ctrl-w (write-file)**
    此指令與上一個指令相似，不同之處在於可以使用與緩衝區不同的檔案名稱存檔。。換言之，可另行指定存回磁碟的檔名。
    鍵入 "`Ctrl-x Ctrl-w`" 後， *echo area* 會顯示出目前緩衝區所在的目錄，此時，使用者可根據需要輸入檔名。若不輸入任何檔名只鍵入 RET ，系統仍會將緩衝區內的資料存回原先訪問的檔案中。此指令也只對目前所使用的緩衝區做存檔的動作。

– **Ctrl-x s (save-some-buffers)**
此指令可用來儲存所有被修改過的緩衝區。使用此指令時， *echo area* 除了顯示檔案名稱外，還會在檔名之後出現

```
(y, n, !, ., q, C-r or C-h)
```

這些訊息是提供給使用者做參考的。現在就告訴使用者這些訊息所代表的意義。

1. **y**
同意對 *echo area* 所顯示的緩衝區存檔，進一步徵詢其它檔案是否存檔的意見。
2. **n**
放棄對 *echo area* 所顯示的緩衝區存檔，但徵詢其它檔案是否存檔的意見。
3. **!**
同意對 *echo area* 所顯示的緩衝區存檔且一併對其它的緩衝區存檔，此時不再一一徵詢其它檔案是否存檔的意見。
4. **.**
同意對 *echo area* 所顯示的緩衝區存檔，但放棄對其它未存檔的緩衝區存檔，且直接離開此存檔的狀態。
5. **q**
離開存檔的狀態而不執行任何存檔的動作。
6. **C-r**
可以此指令流灠目前所要儲存的檔案內容，當離開此流灠狀態即回復存檔的模式，系統會再度詢問與存檔有關的訊息。
7. **C-h**
對於以上的選項若有不明白的地方，可以此功能查閱其意思。

- 存檔後直接離開 EMACS
Ctrl-x Ctrl-c (save-buffers-kill-emacs)
此指令容許用者在決定是否將緩衝區的內容存檔後，立即離開 EMACS。此指令實際上是先執行 “save-some-buufers” 的動作再離開 EMACS。

討論完了載入與儲存檔案的功能之後，相信已經迫不急待想要知道如何編輯一份心目中想要的文件。現在就是介紹如何編輯檔案的時候了。

5.2 Emacs Basic Editing Command

編輯器做些什麼事呢？現在先談談編輯器的基本功能，至於編輯器的進階功能則在下一章討論。此處所談的基本或進階的功能，所指的都是 EMACS 可以提供的功能，所區分的基本和進階也只是為了討論上的方便而已。

所謂的編輯器就是用來編輯文件的器具。一張紙與一支筆，就可以滿足文件的編輯，這也是最簡單且最原始的編輯器。將紙筆產生文件的動作以電腦來代步，就是電子編輯器（以下簡稱編輯器）主要的功能，也是本文所要討論的編輯器。

文件的編輯不外乎文件的鍵入、游標的移動、文件的搬移與刪除以及文件資料的找尋與取代等等。現將 EMACS 可提供的基本編輯功能簡列如下，其詳細的用法會在以下的各節中一一敘及。

EMACS 所能提供的基本編輯功能如下：

- 字元輸入與顯示在螢幕上的方式
  - 字元的顯示方式，可使用插入法（insert）或覆蓋（overwrite）的方法。
  - 除了一般的字元（ASCII Character）外，還可以顯示特殊字元（Special Character）以及任何八進位超過 200 的字元。
  - 文件在螢幕上所能顯示的範圍。
- *point*（*cursor*） 的移動方式（move point）
  - *point* 能左右移動一個或數個字元（character）
  - *point* 能左右移動一個或數個字（word）
  - *point* 能移至一列的開頭或結尾
  - *point* 能上下移動一列或數列（line）
  - *point* 的設定，使游標能上下移動至指定的欄位
  - *point* 的移動能以頁為單位（page）
  - *point* 能至螢幕的前端或尾端
  - *point* 能移至緩衝區的前端或尾端
- 顯示緩衝區大小與 *point* 所在的位置
- 刪除螢幕上所顯示的文件
  - 向左或向右刪除一個字元（character）
  - 向左或向右刪除一個字（word）
  - 刪除游標所在位置以後的所有文件
  - 刪除一個區塊的文件（region）
  - 刪除的文件可以再使用（yank）
- 文件的搬移與拷貝（move and copy）

編輯器具備以上的功能，就可以編輯出想要的文件。現在就來看看如何在 EMACS 所提供的編輯環境，來編輯出想要的文件。

5.2.1 Inserting and Displaying Text

EMACS 允許輸入文件時，將資料直接輸入在 *point*所在位置的正前方，此方法稱為 *insert mode*；或將輸入的資料以覆蓋的方式取代 *point*所在位置的字元，此方法則稱為 *overwrite mode*。現舉一實例來說明 *insert mode* 與 *overwrite mode* 的異同。

- 字串 `food`， *point* 所在位置為 `d` 處，此時的狀態為 *insert mode*。
  在 *point* 所在的 `d` 處鍵入 `t` ，原字串變為 `footd`。

- 字串 `food`， *point* 所在位置為 `d` 處，此時的狀態為 *overwrite mode*。
  在 *point* 所在的 `d` 處鍵入 `t` ，原字串變為 `foot`。

EMACS 對輸入模式的預設值是 *insert mode*，若想將模式轉換成 *overwrite mode*，指令 “**Meta-x overwrite-mode RET**” 可滿足此一需求。若想恢復 *insert mode*，只需再使用一次 “**Meta-x overwrite-mode RET**”就可以了 。當輸入模式轉為 *overwrite mode* 時，螢幕下方的 *mode line* 會顯示 “Ovwrt” 的訊息，用以提示目前是使用 overwrite 的模式。指令 “**Meta-x overwrite-mode**” 是用來轉換 *insert mode* 與 *overwrite mode* 。在此前提下，原為 *insert mode* ，經轉換則為 *overwrite mode*。反之，若原為 *overwrite mode* 則轉換成 *insert mode*。

EMACS 的 *overwrite mode* 只針對從鍵盤輸入的文件有效，若文件不是從鍵盤輸入，而是以別的方式產生的，則一律失去 overwrite 的效用。例如，拷貝而來的文件或以 “**Ctrl-x i**” 得來的文件， EMACS 一律使用 *insert mode*。

除了 insert 與 overwrite 的顯示方法外， EMACS 還允許使用者輸入一些從鍵盤上無法輸入的字，那就是一些控制碼和八進位超過 200 的字元。要輸入這些特殊的文字時，只要在這些字的前方加上 “**Ctrl-q**” 即可。例如，要輸入分頁碼 (formfeed ，ASCII Ctrl-L， octal code 014) ，則輸入 “`Ctrl-q Ctrl-l`” 即可。此時螢幕會出現 ^L 的符號。

當輸入文件的長度，超過 EMACS 視窗寬度所能顯示的範圍，EMACS 對此情形的處理如下所示：

- 若文件太長需要換列時，在換列處鍵入 RET ，其後的文字會自動轉到下一列且以第一個欄位為新列的起始點。若其後沒有文件而鍵入 RET ，游標會會停在下一列的第一個欄位。

- 不理會文件是否會超過視窗的寬度，也就是不鍵入 RET 而繼續輸入文字。 EMACS 會自動在視窗的最後加上 “\” ，而將其餘的文字移至下一列；若下一列還是無法顯示出所有的文字，會在此列的最後再加上一個 “\” ，而繼續將多餘的文字移至下一列。 EMACS 就是不斷重複如此的動作，直到所有的文字都能完全顯示出來為止。
  \ 所代表的意思與鍵入 RET 並不相同。鍵入 RET 表示重新使用一個新列；不鍵入 RET 而令 EMACS 自動加入 \ 所產生的文件，仍代表著同一列，只是這一列太長， EMACS 無法以其視窗的寬度來一次窮盡，必須分為數次來表示。

- 不鍵入 RET ，也不使 EMACS 自動產生 \，而使超過螢幕寬度的部份暫時隱藏起來。
  EMACS 處理這種情形，是在視窗的最後加上一個 “$” 。“$” 表示其後的內容在視窗上暫時看不到，但仍安在緩衝區內。
  EMACS 的基本預設是自動加入 “\”。要使多餘的文字隱藏起來，必須設定 “truncate-lines” 變數的值為正值。變數設定的方法請參考 4.4 節。
  4.4 節曾談過一個視窗可以分成左右二個小視窗，此視窗可以做水平的捲動，此情形下的視窗在處理太長的列時，就是將 truncate-line 變數的值設成正值，使超過寬度的文件隱藏起來。

前面所談的都是「文字」的插入方式，但如何插入一個「非文字」的空白列呢？在編輯的過程中，若想在某列之前加入一個新列，只需將游標移至此列的最前端，隨後再按下 RET 即可。此時 EMACS 會在游標所在處的前一列，加入一空白列。 EMACS 為何要將空白列加在游標之前而不是游標之後呢？因為將空白列加在游標之前，有一個最大好處，就是可以很輕易的在緩衝區的最前端加入一個空白列。 此時所鍵入的 “RET” ，代表著 newline 。若不鍵入 RET， 也可以使用 EMACS 所提供的 Hotkey ，也就是 **Ctrl-j** 來獲得新的一列。

5.2.2 Moving the Point

文字的鍵入及顯示是編輯過程不可或缺的。除此之外，移動游標到適當的位置，也是編輯過程不可缺的功能。現在就來看看在 EMACS 下如何移動 *point* 或可稱為如何移動游標。移動 *point* 也就是移動游標，因為 *point* 是透過游標來顯示的。所以在本文會將游標與 *point* 交互使用。

游標移動不外乎以固定的單位，將其做上下左右的移動。此固定的單位可能是字元（character）、字（word）、列（line）或頁（page）。現在就來看看如何將游標以這些單位來移動。

- 左右移動一或數個「字元」（character）
  - **Ctrl-f (forward-char)**
    游標往前（右）移動一個字元。
  - **Ctrl-u** *n* **Ctrl-f （Ctrl-u** *n* **Meta-x forward-char)**
    游標往前（右）移動 *n* 個字元。
  - **Ctrl-b (backward-char)**
    游標往回（左）移動一個字元。
  - **Ctrl-u** *n* **Ctrl-b （Ctrl-u** *n* **Meta-x backward-char)**
    游標往回（左）移動 *n* 個字元。
- 游標左右移動一或數個「字」（word）
  - **Meta-f (forward-word)**
    游標往前（右）移動一個字。
  - **Ctrl-u** *n* **Meta-f （Ctrl-u** *n* **Meta-x forward-word)**
    游標往前（右）移動 n 個字。
  - **Meta-b (backward-word)**
    游標往回（左）移動一個字。
  - **Ctrl-u** *n* **Meta-b （Ctrl-u** *n* **Meta-x backward-word)**
    游標往回（左）移動 *n* 個字。
- 游標移至一列的「最前端」或「最尾端」
  - **Ctrl-a (beginning-of-line)**
    游標移至一列的最前端。
  - **Ctrl-e (end-of-line)**
    游標移至一列的最尾端。

- 游標上下移動「一列」或「數列」 (line)
  - **Ctrl-n (next-line)**
    游標向下移動一列。
    下移一列的游標其所在的水平位置，與移動前的水平位置相同。若游標在文件的最後一列時， Ctrl-n 會自創新列再移到此新列的第一個欄位。
  - **Ctrl-u *n* Ctrl-n (Ctrl-u *n* Meta-x next-line)**
    游標向下移動 *n* 列。
  - **Ctrl-p (previous-line)**
    游標向上移動一列。
    上移一列的游標其所在的水平位置，與移動前的水平位置相同。若游標已在文件的第一列時，再使用 “**Ctrl-p**” 時 *echo area* 會顯示如下的訊息：

    ```
    Beginning of buffer
    ```

  - **Ctrl-u *n* Ctrl-p (Ctrl-u *n* Meta-x previous-line)**
    游標向上移動 *n* 列。
- 上下捲動緩衝區
  - **Ctrl-v (scroll-up)**
    向上捲動緩衝區，且將目前視窗所顯示的最後二列，做為捲動後的前二列， point 出現在視窗上的第一列。 EMACS 如此做只是讓使用者有一個承先啟後的感覺罷了。
  - **Ctrl-u *n* Ctrl-v**
    緩衝區向上捲動 *n* 列。
    數字 *n* 若小於目前 *point* 在此視窗所在位置的列數，捲動後的 *point* 位置，仍留在捲動前的同一列上；否則 *point* 移至視窗的第一列。
  - **Meta-v (scroll-down)**
    向下捲動緩衝區，且將目前視窗上的前二列做為捲動後的後二列，此時的 *point* 出現在螢幕的最後一列。
  - **Ctrl-u *n* Ctrl-v**
    緩衝區向下捲動 *n* 列， 若數字 *n* 小於 *point* 在此視窗以下的列數，捲動後的 *point* 位置，仍留在捲動前的同一列處。否則 *point* 移至視窗的最後一列。
- 左右捲動緩衝區
  - **Ctrl-x < (scroll-left)**
    緩衝區向左捲動。
  - **Ctrl-x > (scroll-right)**
    緩衝區向右捲動。
- 游標移至緩衝區的前端尾端
  - **Meta < (beginning-of-buffer)**
    將游標移至緩衝區的最前端， *point* 也移至視窗的第一列。

  - **Meta > (end-of-buffer)**
    將游標移至緩衝區的最尾端， *point* 也移至視窗的最後一列。

- 頁數的移動
  還記得討論 "**Ctrl-q**" 時，提過分頁的控制碼。有了分頁的設定，自然會有以頁數做為移動游標的單位了。

  - **Ctrl-x [(backward-page)**
    將 *point* 移至上一頁分頁指標（^L）之後且緊鄰 ^L。若 *point* 已緊鄰在 ^L 之後了，則會略過此 ^L 到上一個 ^L 之後。
  - **Ctrl-x] (forward-page)**
    將 *point* 移至下一頁分頁指標（^L）之後且緊鄰 ^L。若 *point* 已緊鄰在 ^L 之後了，則會略過此 ^L 到下一個 ^L 之後。
  - **Ctrl-x Ctrl-p (mark-page)**
    以頁為單位做上記號之後，再配合其它的指令對此頁做處理。此指令類似 *mark region*（5.4 節會討論其意思）。例如
    Ctrl-x Ctrl-p Ctrl-w
    是將游標所在的那一頁刪掉。此指令可以配合 *numeric arguments* (5.3 節會討論其意思）使用。

除了以移動游標的方法來游走緩衝區外， EMACS 還提供一個可以直接將游標移到指定的行列。所要到達的目的地，不論是以字元數或列數為移動的單位，其起算點都是以緩衝區的第一列第一個欄位為起算的標準。以下就是此方法的介紹。

- **Meta-x goto-char RET**
  鍵入此指令後再按下 RET ， *echo area* 會出現 "`Goto char:`"，在其後輸入想要游標移動的總字元數即可。再一次的提醒各位，字元的計算是以緩衝區的第一列的第一個字元為起算點。例如，在 "`Goto char:`" 給予 100 的數字，則游標會從緩衝區的第一字元移動到第 100 個字元。

- **Meta-x goto-line RET**
  鍵入此指令再按下 RET 後， *echo area* 會出現 "`Goto line:`"，在其後輸入想要游標移動的列數即可。再一次的提醒各位，列數的計算是以緩衝區的第一列為起算列。例如，在 "`Goto line:`" 給予 100 的數字，則游標會從緩衝區的第一列開始往下移動 10 列。

5.2.3 Numeric Arguments

在上一小節 *point* 移動一文中不斷的看到 **Ctrl-u** n ，它是什麼呢？它是 EMACS 的數值引數（numeric argument），但數值引數又是什麼的呢？

數值引數適用於所有 EMACS 的指令。它的用途可分成以下數類：

- 對某一指令做重複執行的動作

- 用法： **Ctrl-u** *n command* 或 **Meta-***n command*
- 說明：
  - *n* 代表重複的次數； *command* 代表要重複執行的指令。
  - 若鍵盤提供 Meta 鍵（個人電腦可以使用 ESC 鍵），則 Meta 鍵是使用數值引數最方便的方法。
  - *n* 若為負值的數字，則表示以反方向運行指令。
- 例子：
  - *n* 為正值
    `Ctrl-u 5 Ctrl-f` 或 `Meta-5 Ctrl-f`
    表示游標往前（右）移動 5 個字元。
  - *n* 負值
    `Ctrl-u -5 Ctrl-f` 或 `Meta-5 Ctrl-f`
    表示游標往回（左）移動 5 個字元。
- 重複動作四次
- 用法： **Ctrl-u** *char*
- 說明：
  - Ctrl-u 後面所接的不是數字而是字元，它代表著一個 Ctrl-u 重複指令的動作為一個四的倍數。換言之，一個 Ctrl-u 執行四次，二個 Ctrl-u 則執行 16 次，以此類推。
- 例子：
  - `Ctrl-u Ctrl-f`
    表示游標往前移動 4 個字元。
  - `Ctrl-u Ctrl-u Ctrl-f`
    表示游標往前移動 16 個字元。

- 利用引數變數自動鍵入相同的「字元」
  - 用法： **Ctrl-u** *n char* 或 **Meta-***n char*
  - 說明：省略 *n* 與否，代表不同的意義
    - Ctrl-u 後緊接著數字與字元，表示重複字元 *n* 次。
    - Ctrl-u 後面不接數字而緊接字元，表示重複字元四次。
  - 例子：
    - `Ctrl-u 10 r` 或 `Meta-10 r` 則螢幕上出現 10 個 r。
    - `Ctrl-u r` 或 `Meta-4 r` 則螢幕上出現 4 個 r。
- 使用引數變數自動產生相同的「數字」的方法
  - 用法： **Ctrl-u** *n* **Ctrl-u** *n* 或 **Meta-***n* **Ctrl-u** *n*
  - 說明： Ctrl-u 後面有二個 *n*，第一個 *n* 表重複的次數，第二個 *n* 表重複的數字。第二個 *n* 之前一定要再給予一次 Ctrl-u，若不給第二個 Ctrl-u，系統會以為重複 *nn* 次。
  - 例子： `Ctrl-u 5 Ctrl-u 4` 或 `Meta-5 Ctrl-u 4`
    螢幕上出現 5 個 4。
    `Ctrl-u 5 4`，則系統以為重複 54 次。

數值引數的用途很廣，且適用任何一個 EMACS 的指令，當需要重複執行某一指令的動作時別忘了它。有人說電腦最強大的功能之一，就是能不厭其煩的執行重複的事情。以下有關 EMACS 指令的介紹，不再特別強調 numeric arguments，但並不表示它就此而消失了，因為數值引數適合所有的指令，所以沒有必要每次都重複強調。 只需記得，需要時可隨時使用 numeric arguments。

5.2.4 Point Position Infromation

前面談過 EMACS 移動 *point* 的方法，現在介紹探知 point 位置的方法。 *point* 所在的欄位、列數與頁數等訊息，可從整個或部份的緩衝區而得知。現在就開始討論 Point 位置。

- **Meta-x what-page**
  告知 *point* 所在的頁數與列數。若緩衝區沒有以分頁（^L）符號分頁，則 *point* 所在的頁數永遠為第一頁。 若 *point* 在第一頁的第 200 列，其顯示的訊息如下：

  ```
  Page 1, line 200
  ```

- **Ctrl-x l (count-lines-page)**
  "**Meta-x what-page**" 用來告知 *point* 所在位置的頁數與列數，但無法從中得知此頁的總列數。若想知道某一頁的總列數，必需靠 **Ctrl-x l** 來得知。
  此指令除了得知某頁的總列數外，同時還知道 *point* 所在位置之前與之後的尚有的列數。使用此指令時， *echo area* 會出現如下的訊息：

  ```
  Page has 23 lines (20 + 4)
  ```

  echo area 出現的第一個數字為總列數，括號內的二個數字分別代表 *point* 所在處之前與之後的列數。括號內的兩個數是以 *point* 為分界點而得來的。所以當 *point* 不出現在某一列的第一個欄位時，括號內二個列數的總和會比總列數多出一列，因為 *point* 所在的列被重複計算了兩次。此指令在決定如何分頁時可以幫上大忙，因為可以輕易得知 *point* 前後的列數。

- **Meta-x what-line**
  告知 *point* 在緩衝區的列數。若此時 *point* 在第 200 列，則 *echo area* 會出現如下的訊息：

  ```
  Line 200
  ```

- **Ctrl-x =**
  指出游標所在欄位的字元資料。這些資料包括字元的八進位碼、字元所在的位置佔整個緩衝區的比例（此比例以字元為基本單位）以及字元所在的欄位。例如以此指令來得知此 "a" 字元的資料， *echo area* 會出現如下的資訊：

  ```
  Char: a (0141)  point=23905 of 38784 (62%)  column 19
  ```

臨時想知某一字元的八進位，也不妨試試此法。

- **Meta-=**
  用來得知某特定區域（*region*）的總列數與總字元數。至於如何設定區域會在 5.3 節討論。以此指令得知的結果如下所示：

  ```
  Region has 200 lines, 2000 characters
  ```

- **Meta-x line-number-mode**
  以上所討論的指令，只在使用指令時才會顯示想要的訊息， 訊息的出現是無法長存的。 基於此，EMACS 提供了一個可使訊息永久存在 *mode line* 的方法，那就是使用 "**Meta-x line-number-mode**"。此指令可以在 *mode line* 上顯示 *point* 所在的列數，直到離開此狀態或離開此視窗才會消失。若想使每次進入EMACS 都能顯示列數，最好的方法是在 "`.emacs`" 檔加上如下的敘述：

  ```
  (setq line-number-mode t)
  ```

  如此一來，只要進入 EMACS ，*mode line* 會自動將列數顯現出來。至於自動顯示欄位的方法，目前的 EMACS 尚未提供此服務。

5.2.5 Deleting and Killing Text

修改文件不外乎將原有的文件刪除後，再加入新的內容；或將現有的資料做重新的排列組合。現先討論文件的刪除部份，文件的重組就留待 5.4 節再進行討論。

EMACS 中的刪除有兩種形式，一種是指文件的 *killing* ，另一種則指文件的 *deletion*。在 EMACS 中所謂的 *killing* 是指將文件從目前的緩衝區移到一個稱為 *kill-ring* [1]的地方去。文件在緩衝區中是消失了，但卻儲存在 *kill-ring* 這一個變數中。 EMACS 可以有許多的緩衝區，但卻只有一個 *kill-ring* 的儲存變數。也就是說， 多個緩衝區彼此共享一個 *kill-ring*, 而且也只有一個 *kill-ring*。

EMACS 所設計共享的 *killing-ring* 的用意是讓被遺棄的文件可以找回，而且各緩衝區彼此也可借由 *killing-ring* 來建立一個互通的管道。所謂的互通就是把甲緩衝區的東西給乙，反之，也可把乙緩衝區的東西給甲。此模式在 EMACS 中就是透過 *killing-ring* 來完成。因此，想從甲緩衝區中得到某些文件給乙緩衝區，只要將甲緩衝區的文件放入 *kill-ring* 中，乙緩衝區就可以至此共享的 *kill-ring* 中將文件取出。如此一來，就可以共享資源了。

另一種模式的刪除，在 EMACS 中稱為 *deletion*。此種刪除，並不將刪除後的資料放入 *kill-ring* 中，而是將刪除的文件遺棄。此種情形的刪除是無法失物復得的。

[1] kill-ring 為一個變數

以 *deletion* 刪除的資料，可以鍵入 `Ctrl-x u (undo)` 來找回。使用一次 “**Ctrl-x u**，恢復前一個指令的景象；使用二次 **Ctrl-x u**，則恢復前二個指令的容貌，如此周而復始的使用 **Ctrl-x u** ，可恢復更改前的全貌。至於 “**Ctrl-x u**” 更詳細的說明會在 5.5 節進一步討論。

前已說過，不放入 *kill-ring* 的指令稱為 *deletion* 的指令。 EMACS 刪除字元、空白字元以及空白列的指令都不放入 *kill-ring* 中的。具體說來就是， **Ctrl-d**、**DEL**、 **Meta-**、**Meta-SPC** 與 **Ctrl-x Ctrl-o** 等指令。 現就為各位說明這些指令。

文件的修改不是重組資料就是刪減與新增資料，新增資料已論述過了，而重組資料即將在 5.4 節登場，現在就開始談文件的刪減。刪除文件也不外乎刪除一個字元、一個字、一行或一個區塊；刪除的方向可以選擇左右刪除的方式。現在就來看看如何以 EMACS 所提供的指令，來做刪除的工作。此處將刪減的指令分成 *deletion* 和 *killing* 而大類。

- 屬於 *deletion* 的指令集
  - 刪除「字元」（character）的方法
    * **Ctrl-d (delete-char)**
      刪除 *point* 所在位置的字元。
    * **DEL (delete-backward-char)**
      刪除 *point* 之前的字元。
      此指令與 **Ctrl-d** 為最基本的刪除指令，只要耐心夠，任何的刪除都可以此二個指令完成。
  - 刪除 **spaces** 和 **tabs** 的方法
    * **Meta-\\ (delete-horizontal-space)**
      輸入資料時，常會不自覺的輸入無意的空白（space） 和 tab 。當合併上下列而為一列時，也常會出現 space 或 tab 從中作梗。為此 EMACS 提供了 **Meta-** 的指令，讓使用者可以很容易將不必要的 space 和 tab 刪除。當然其它的刪除指令，也可以將不需要的 space 和 tab 刪除，此指令只是更方便刪除 space 和 tab 而已。
      Meta- 可刪除 point 前後所有的 space 和 tab 。
      例如：`abc def        g`，此時的游標在 `f` 與 `g` 之間。鍵入了 `Meta-\` 其結果如下所示：

      ```
      abc defg
      ```

    * **Meta-SPC (just-one-space)**
      刪除 *point* 前後的 space 和 tab 時，若希望留下一個 space 或 tab 做為彼此的分隔，就必須使用 “**Meta-SPC**” 來完成了。
      例如：`abc def        g`，此時的游標在 `f` 與 `g` 之間。鍵入了 “**Meta-SPC** 其結果如下所示：

      ```
      abc def g
      ```

- 屬於 *killing* 的指令
  - 刪除「字」（word）的方法

* **Meta-d (kill-word)**

  此指令往前（右）刪除 *point* 所在位置的字。
  其所刪除字的範圍端賴 *point* 所在的位置而有不同。若 *point* 在一個字的第一個字元，則會刪除此字；若 *point* 在此字的其它位置，則刪除 *point* 所在位置及其之後的所有字元，包括 *point* 所在位置的字元。
  例（一）： `This is a test.` 游標若在 `test` 這個字的 `t` 處，使用此指令的結果如下：

  ```
  This is a .
  ```

  例（二）： `This is a test.` 游標若在 `test` 這個字的 `e` 處，使用此指令的結果如下：

  ```
  This is a t.
  ```

  若想刪除一個以上的字，可以使用 numeric argument 。

* **Meta-DEL (backward-kill-word)**

  此指令往回刪除 *point* 所在位置的字。
  其所刪除字的範圍端賴 *point* 所在的位置而有不同。若 *point* 在一個字的第一個字元，則會刪除此字之前的字；若 *point* 在此字的其它位置，則刪除 *point* 之前的所有字元，但不包括 *point* 所在位置的字元。
  例（一）： `This is a test.` 游標若在 `test` 這個字的 `t` 處，使用此指令的結果如下：

  ```
  This is test.
  ```

  例（二）： `This is a test.` 游標若在 `test` 這個字的 `e` 處，使用此指令的結果如下：

  ```
  This is a est.
  ```

  同樣的，回想一下 numeric argument 吧。

– 刪除列的方法

* **Ctrl-k (kill-line)**

  刪除列的指令。所刪除列的範圍，以 *point* 所在的位置為準則， 以 *point* 所在位置為起始點，以此列的結束為終點。
  被刪除掉的文件會在視窗上留下一空白，若想將空白列也一併刪除，必需再使用一次 **Ctrl-k**。此時的 **Ctrl-k** 是用來刪除換列指令的控制碼。

* **Ctrl-x Ctrl-o (delete-blank-lines)**

  編輯的過程中會不經意的加入許多空白列，此種情形常發生在緩衝區的最尾端。每按下一個 RET ，就輸入了一個新列，但在緩衝區的的尾端按下 RET ，此時所加入的新列並不易為人查覺。此指令可以將 *point* 所在位置「前後」的空白列刪除只

留下一列。當然並非一定要以此指令來刪除空白， **Ctrl-k** 自然也可以用來刪除空白列，也可以將空白列以刪除區塊的方式刪除之，提供此指令只是方便滿足使用者的需求而已。

- 刪除區塊的方法
  - **Ctrl-w (kill-region)**
    EMACS 允許刪除某一特定的區塊。要刪除區塊，必需先標示區塊，此標示的動作稱為 *mark*。如何標示區塊，就是下一節的主題。

EMACS 對於被 *killing* 而刪除的資料，都是放在 *kill-ring* 中。 *kill-ring* 其實只是一個變數而已，所有 *killing* 刪除掉的文件，就是此變數的值。其已述及查閱變數值的方法，現不厭其煩的再論述一次：

1. 鍵入 “`Ctrl-h v`” 後， *echo area* 處會出現：

   ```
   Describe variable:
   ```

2. 在 `Describe variable:`後輸入變數的名稱， EMACS 會另開一個視窗來顯示此變數的值。此時所要鍵入的變數名為 “`kill-ring`”

此節討論 EMACS 編輯的基本指令，有了這些指令之後，編輯的工作就不是難事了。。接下來各節則討論其它的編輯功能。

5.3 Yanking

將 *kill ring* 的內容取出的動作，稱為 *yank*。 *yank* 除了可 *yank* 最新 *killing* 的資料，也可 *yank* 早先 *killing* 的內容。現在就以二種不同的 *yank* 做為討論的對象。

將最新 *killing* 的文件從 *kill-ring* 中取出的方法很簡單，只要使用 “**Ctrl-y**” 即可。 但在 *yank* 時，一定要確保在 *killing-ring* 中存有被刪除的資料。

想要 *yank* 最新 *killing* 之前的文件，就較為複雜了。所謂的較為複雜，只是多了一個移動指向 *kill-ring* 變數值的程序而已。因為 *yank* 指令所要 *yank* 的內容，全視指標指向 *kill-ring* 的位置而定。此指標一般都是指向最新放入的文件。若想 *yank* 其它的內容，就必需先移動指標了。

移動 *kill-ring* 指標的方法是使用 “**Meta-y**”，但在使用 “**Meta-y**” 之前，一定要先使用 “**Ctrl-y**”。 換言之， “**Meta-y**” 的使用，一定要緊跟在 “**Ctrl-y**” 之後。

“**Meta-y**” 的運作是將指向 **kill-ring** 入口的指標，向前移動，再將 “**Ctrl-y**” *yank* 出來的資料 ，以此時指標指向的文件取而代之。這就是為何使用 “**Meta-y**” 指令之前，一定要先執行 “**Ctrl-y**”。 以 “**Meta-y**” 來移動指標，並不會影響 *kill-ring* 變數值的內容次序。

5.4 Marking Text

前面有些指令是專門運作區塊的，如 “**Ctrl-x Ctrl-p**” 與 “**Ctrl-w**” 等等。但什麼是區塊呢？EMACS 所言的區塊，是指從標記（*mark*）處到 *point* 所在位置間的範圍，此範圍就是區塊（*region*）了。換言之，區塊的範圍是指，標記所在位置之後（包括標記所在位置本身）到 *point* 之前（不包括 *point* 所在的位置）的所有文件。知道區塊的定義後，接下來就介紹如何定義區塊了。

既然區塊是指從標記處到 *point* 的範圍，自然設定區塊也意味著設定標記和 *point*。設定 *point* 的方法很簡單，只要移動 *point* 至目的地即可。此時的 *point* 代表著區塊範圍的終點。至於設定標記的方法也不難，只要在想要設定標記的地方，輸入以下的任一個指令即可。

- **Ctrl-@ (set-mark-command)**
- **Ctrl-SPC (set-mark-command)**

所以設定區塊的步驟如下所示：

1. 設定區塊的始位置，也就是所謂的標記（*mark*）。
   可使用 “**Ctrl-@**” 或 “**Ctrl-SPC**” 任一個指令來設定標記。
2. 設定區塊的終結位置，也就是移動游標至區塊的尾端。

在標記與 *point* 之間的範圍就是所謂的區塊。

為何會提供二個設定標記的 Hotkey 呢？ Hotkey 是用來連結 EMACS 的命令， EMACS 會將常用的命令給予一個 Hotkey 與之連結。此連結的 Hotkey，常會受所使用的終端機而不同。因此有的終端機可以使用 “**Ctrl-@**”，但有的終端機則必需使用 “**Ctrl-SPC**”。更甚者，有的終端機卻二者都無法使用，例如筆者以個人電腦模擬成的終端機，卻必需鍵入 “**Ctrl-2**” 才能達到設定標記的效果。 EMACS 可以允許使用者重新設定所使用的 Hotkey，但這已經超過本文討論的範圍。如果無法使用以上任何一個 Hotkey，就使用命令 “**ESC-x set-mark-command**”。使用 EMACS 的指令（*command*）永遠可以達到目的的。

EMACS 的 *point*，是透過視窗上的游標來顯示的。 text-only 的視窗只有一個游標，所無法同時表示標記及 *point*。若想觀測區塊的來龍去脈，可以 “**Ctrl-x Ctrl-x**” (**exchange-point-and-mark**) 將標記與 *point* 做交換，從交換的過程可以觀察標記與 *point* 的位置。“**Ctrl-x Ctrl-x**” 所能做的事，只互換標記與 *point* 而已。

那些指令是使用在區塊上呢？以下指令就是適用在區塊上的。有些是曾經提過的，有些是將要談及的，有些可只能只是列舉出來並不會在本文中被討論。

- **Ctrl-w (kill-region)**

- **Ctrl-x r s (copy-to-register)**

- **Ctrl-x Ctrl-p (mark-page)**
- **Ctrl-x Ctrl-l (downcase-region)**
- **Ctrl-x Ctrl-u (upcase-region)**
- **Meta-x fill-region**
- **Meta-x print-region**

接下來討論與編輯有切身關係的搬移與拷貝。

5.5 Moving Text and Copying Text

變換文件在緩衝區出現的位置稱為搬移（*Move*），促使文件在緩衝區的其它地方重複出現稱為拷貝（*Copy*）。

前已論及放於 *kill-ring* 的文件，可在需要時拿出來使用，此動作稱為 *yank*。使用 *yank* 的方法很簡單，只要鍵入 "`Ctrl-x y`"即可。使用此方法可將某特定區塊的文件做「搬移」的動作，只是所需的步驟比較煩瑣而已。首先將要搬移的文件重複以 "**Ctrl-x k**" 的指令，將其放入 *kill-ring* 中，再將 *point* 移至文件欲搬移的位置，以 "**Ctrl-y**" 將其 *yank* 出來。

「拷貝」一個區塊的方法與「搬移」一個區塊的方法累同，唯一不同是要執行二次 *yank*。第一次使用 *yank*，是將刪除的原文件再放回被刪除的位置，第二次使用 *yank* 是做拷貝的動作，也就是將 *kill-ring* 的文件放入要拷貝的地方。

以上的方法當然可以用來做搬移和拷貝之用，但如要搬移和拷貝的文件有數十甚至數百數千列，豈不要重複 "**Ctrl-k**" 的動作數十甚至數百數千次。因為 "**Ctrl-k**" 基本上是刪除列的指令，當然刪除列的指令，可以配合 numeric argument。但在使用數值引數之前還必需確定所欲搬移或拷貝的確實列數，如此一來豈不使事情愈來愈複雜呢？所以，要搬移與拷貝文件最好的方法，就是使用標記與區塊。下面所談的是針對區塊的搬移與拷貝的方法。

不論搬移或拷貝，只要觸及區塊的使用，首要之事就是先把區塊標示出。區塊定好之後，要搬移就以 "**Ctrl-w**"，將整個區塊刪除之後，再以 "**Ctrl-y**" 將其 *yank* 出來。要拷貝則以 "**Meta-w**" 將區塊的內容拷貝一份放在 *kill-ring* 中，爾後再以 "**Ctrl-y**" 將其 *yank* 出來。現說明搬移與拷貝的實際操作過程。

- 不設定區塊，以刪列的方式，將文件一列列先刪除後，再 *yank* 出來。
  此法可搬移與拷貝 *kill-ring* 內的文件。
  1. 重複執行 "**Ctrl-k**" (刪除文件，可使用 numeric argument)

2. **Ctrl-y**（將放置在 *kill ring* 中的文件， *yank* 至緩衝區內。）
   - 若做搬移的動作，只需使用一次 *yank*。
   - 若做拷貝的動作，則要使用二次的 *yank*。

- 以設定區塊的方式，「搬移」區塊的文字。
  1. **Ctrl-@**（設標記，也就是設定區塊的起始值）
  2. 移動游標以便設定 *point* 的位置（設定區塊的終點）
  3. **Ctrl-w**（刪除所標示的區塊）
  4. 移動游標至要搬移的位置（確定文件搬移處）
  5. **Ctrl-y**（將放置在 *kill ring* 中的文件， *yank* 至緩衝區內。）
- 以設定區塊的方式，「拷貝」區塊的文字。
  1. **Ctrl-@**（設標記 (mark)）
  2. 移動游標以設定 *point* 的位置（設定區塊的終點）
  3. **Meta-w**（不刪除標示區塊的內容，將此內容拷貝至 *kill-ring* 中）
  4. 移動游標至要拷貝的位置（確定文件拷貝處）
  5. **Ctrl-y**（將放置在 *kill-ring* 中的文件， *yank* 至緩衝區內。）

為了更清礎搬移與拷貝後的真實結果，現舉實例來說明。

```
實例：現有二段文件，第一段文字是做搬移與拷貝用的，第二段文字是用來接收搬移%
      後的文字。

      第一段：標記設在第一列的 A 處，而 point 設在 B 的地方。
      1111111111A22222222222
      3333333333333333333333
      4444444444444444444444
      55555B5555555555555555

      第二段： point 在 Y 處
      00000Y0000000000000000
      0000000000000000000000
      0000000000000000000000

結果一、將第一段的文件搬移至第二段，搬移後的游標仍在 Y 處。
      1111111111B5555555555555555

      00000A22222222222
      3333333333333333333333
      4444444444444444444444
```

```
55555Y0000000000000000
0000000000000000000000
0000000000000000000000
```

結果二、將第一段的文件拷貝至第二段，搬移後的游標仍在 Y 處。

```
1111111111A22222222222
3333333333333333333333
4444444444444444444444
55555B5555555555555555

00000A22222222222
3333333333333333333333
4444444444444444444444
55555Y0000000000000000
0000000000000000000000
0000000000000000000000
```

執行搬移與拷貝的動作時， EMACS 所採取的一律是 *insert-mode*。此時，即使設定為覆蓋（*overwrite-mode*），覆蓋的效果也會暫時失效。

區塊的範圍以標記（mark）為起始值，以 *point* 所在位置之前的字元做結束。所以，從以上的實例可以很清礎的看到，搬移或拷貝之後的文件會包括標記所在的字元，但不會包含 *point* 所在的字元。搬移或拷貝的文件會出現在 *point* 之前，原先所有的文件會向後移動，此時的 *point* 仍在移動 前的字元處。

區塊是以標記為開始而以 *point* 為終點所構成的。標記與 *point* 所在的列，可能包含所在列的一部分，其它區塊所包含的列都是完整的一列。若搬移或拷貝的範圍，為文件中某一個長方形的區域，那該如何來設定其範圍呢？設定好的範圍又該如何來操作呢？ EMACS 對於這種形狀的區域有其它的處理方式，稱為長方形的區塊 （*rectangle region*） 。

設定長方形（*Rectangle*）的方法與設定區塊 （*Region*）的方法是一樣的，只是在理解上有所不同。長方形設定的方法，也是設定二個標記，此兩個標記位於長方形相對的兩個直角上。所以，設定了左上角，另一個標記一定要設在右下角；反之，如果設定了右上角，另一個標記就一定要在左下角了。 。此所設好的長方形， EMACS 稱為 *rectangle region*。

長方形與區塊的設定方法是一樣，所以單從設定標記與 *point* 的方式，是無法辨識二者的差別的。要知是使用區塊或長方形，只有等到使用運作於此範圍的指令時才會知道。

若是以刪除長方形的指令將長方形的區塊刪除時，此時刪除掉的長方形文件並不放在 一般的 *kill-ring* 中，而是放於別的地方。因為，刪除長方形的運作方式與刪除區塊的方式不同，所以將長方形刪除後的內容，與放置區塊刪除後的內容分隔，以便管理。

長方形運作的種類與區塊運作的種類纍同，可分為二大類，一類為刪除與插入，另一類則專門用來處理空白。

刪除長方形可以採取摒棄的方式 （**Ctrl-x r d**） 或將其儲存於某處（**Ctrl-x r k**），以利事

後的 *yank*（**Ctrl-x r y**）。刪除掉的長方形資料的保存，只能保留最新刪除的資料，所以只有最新刪除掉的長方形資料，才可以被 *yank* 出來。

長方形區塊的運作種類，除了能將資料做刪除與搬移外，還可以在文件的某處加入固定區域的空白。因為從事編輯時，常需要在某一區域加入一些空白（**Ctrl-x r o**）。例如在文件加上空白做為邊緣（margin）；製作圖表時留些空白；或將某一區域的文件以空白取代（**Meta-x clear-rectangle**）等等。長方形的運作與區塊的運作最大不同處在於，無法直接做拷貝的動作，若要拷貝必需先刪除後再執行二次的 *yank*。現在就說明長方形區域的用法。

- 長方形區域的 *yank* 方法
  1. **Ctrl-@**（設定長方形區域的第一個對角）
  2. 移動 *point* 至第二個對角處（設定長方區域的第二個對角）
  3. **Ctrl-x r k (kill-rectangle)**
     刪除設定好的長方形，以便 *yank* 使用。
  4. 移動 *point* 到要 *yank* 的地方。
  5. **Ctrl-x r y (yank-rectangle)**
     將刪除的長方形，從儲存處取出。
- 永遠刪除長方形的內容
  1. **Ctrl-@**（設定長方形區域的第一個對角）
  2. 移動 *point* 至第二個對角處（設定長方區域的第二個對角）
  3. **Ctrl-x r d (delete-rectangle)**
     刪除設定好的長方形區域。此刪除不會儲存在某一定地方。換言之，一旦摒棄此區域，就無法將其喚回，除非使用 **undo** 的指令。
- 在長方形區域內插入空白，原文件向前（右）移動。
  1. **Ctrl-@**（設定長方形區域的第一個對角）
  2. 移動 *point*（設定長方區域第二個對角）
  3. **Ctrl-x r o (open-rectangle)**
     填入空白在設定好的長方形區域內。此時區塊內的文字會自動往右移動。使用 *overwrite mode*，原有的文件也不會被加入的空白覆蓋。
- 將設定的長方形區域以空白覆蓋
  1. **Ctrl-@**（設定長方形區域的第一個對角）
  2. 移動 *point* 至第二個對角處（設定長方區域第二個對角）
  3. **Meta-x clear-rectangle**
     將此長方形區塊內的文件以空白取代。
- 在設定的長方形區域內填充某一類型的字串。
  此方法常可用在程式的撰寫或測試上。例如，測試 “`.emacs`” 程式時，常因測試的過程中，需要將設定做增減的工夫，使用此方法可以隨時將測試條件做增減。

1. **Ctrl-@**（設定要填充字串的起點）
2. 移動 *point* 至第二個對角處（設定要填充字串的終點）
3. 鍵入 `Meta-x string-rectangle RET`，此時 *echo area* 會出現

```
String rectangle:
```

利用 *minibuffer* 鍵入要填充的字串，即可將字串填入長方形區內。值得注意的是，此時長方形的寬度由字串的寬度來決定，**Ctrl-@** 與 *point* 只用來決定長方形的長度。

5.6 Undoing Change

在 EMACS 中想要將已經做過的動作放棄，以恢復舊觀。EMACS 將其稱為 **undo**，以下就介紹如何 **undo**：

- **Ctrl-x u** (undo)
- **Ctrl-_** (undo)

以上兩個 Hotkey 都是用來執行 **undo** 的指令。提供二個 Hotkey，是因為有些鍵盤並無明顯使用 "**Ctrl-_**" 的方法，為了彌補無法以一個字元達到 **undo** 效果的鍵盤，故另行提供 "**Ctrl-x u**" 給無法使用 "**Ctrl-_**" 的使用者。

EMACS 所提供的 **undo**，可以連續恢復最近使用過的指令。**undo** 的順序是最新使用過的指令最先被 **undo**，第二次使用 **undo** 則恢復第二新的指令，任何指令的輸入（除了 **undo** 本身之外）都會使指令輸入的順序重整，這也同時影響 **undo** 的順序。

使用 **undo** 有一個限制，就是 **undo** 只能 **undo** 對緩衝區內容造成改變的指令。對於只是改變游標動作的指令，是無法以 **undo** 來恢復舊觀。若所有修改過內容的指令，都以 **undo** 恢復原狀後，再一次使用 **undo** 的指令，*echo area* 會出現如下的訊息：

```
no furter undo information
```

當使用了 **undo** 之後，還想要在 **redo** 這個已被 **undo** 的動作時，有一個技巧可以達成如此的效果。

1. 首先鍵入一個不會改變緩衝區內容的指令（如游標移動的指令），使原來存放指令的順序因新指令的加入而改變。
2. 再使用一次 **undo** 的指令，就可以達到 **redo** 的效果了。

現舉一實例來說明此 redo 的技巧：

- 視窗上現有的資料為：

```
This is a test.
```

- 如下為所執行的一連串指令，括號內表示所用過的指令：

1. 執行四次 DEL，視窗顯示如下的訊息：

   ```
   This is a
   ```

   所使用的指令集如下：
   DEL DEL DEL DEL

2. 將游標至字元 "i" 處，視窗顯示的訊息並無改變：

   ```
   This is a
   ```

   所使用的指令集如下：
   DEL DEL DEL DEL

3. 鍵入 Meta-d，視窗顯示如下的訊息：

   ```
   This  a
   ```

   所使用的指令集如下：
   DEL DEL DEL DEL Meta-d

4. 鍵入 Ctrl-x u，視窗顯示如下的訊：

   ```
   This is a
   ```

   此時恢復最新被使用過指令 "Meta-d" 前的狀況（Ctrl-x u不為恢復的對象），也就是步驟二的情形。 此時使用的過指令集如下：
   DEL DEL DEL DEL Meta-d Ctrl-x u

5. 鍵入 Ctrl-f，視窗仍出現與先前相同的訊息：

   ```
   This is a
   ```

所使用的指令集則增加如下：
DEL DEL DEL DEL Meta-d Ctrl-x u Ctrl-f

6. 此時可以 `Ctrl-x u` 恢復第一次 undo 前的狀況，即步驟三的狀況。視窗出現：

```
This  a
```

因為記錄指令歷史的指標，此時己在第二個 `Ctrl-x u` 處，但 `Ctrl-x u Ctrl-f` 均不在 undo 的行列中，所以此時可以再恢復一次 `Meta-d` 前的狀況。
所使用的指令集如下：
DEL DEL DEL DEL Meta-d Ctrl-x u Ctrl-f Ctrl-x u

一般而言，每一個使用過的編輯指令都有一個與之相對應的 undo 記錄。每一個 undo 的記錄，都只對目前的緩衝區有效。有的指令需要一個以上的 undo 記錄來完成 undo 的動作；有的指令會先匯集一群 undo 的記錄，當使用 undo 時，會將此匯集的結果一次展現出來。例如，單一字元運作的指令，若每次使用 undo ，只恢復一個字元則非常的不經濟，所以遇到此種情形，以集合體的方式處理是比較合理的作法。

Chapter 6

Advanced Emacs Editing Commands

前面的章節談論 EMACS 的基本用法，現在討論 EMACS 進階的用法。現在先談 EMACS 的搜尋 *search* 與字串 *string* 的取代功能。

6.1 Searching Text

搜尋特定的字串，並非 EMACS 所特有的功能，相信大多數的編輯器都具有如此的功能。但 EMACS 所採取的搜尋方法是，每鍵入一個字元就展開搜尋，EMACS 稱此種方式的搜尋為 *Incremental Search* 。當然， EMACS 也提供非 *Incremental Search* ，稱為 *Nonincremental Search* 。

EMACS 對於所要搜尋的字串，仍是利用 *minibuffer* 來輸入所欲搜尋的字串。 此時，輸入 *minibuffer* 的搜尋字串，若全由小寫的英文字母 (lower case) 組成，則 EMACS 在展開搜尋的行動時，不論字母是否有大小寫的差別，會將所有與 *minibuffer* 具有相同英文字母的字串都找尋出來。例如在 *minibuffer* 處輸入 `abc`， EMACS 會找尋 `abc`、 `Abc`、`aBc`、 `abC`、`ABc`、`aBC`、`ABC` 等字串。所以在 *minibuffer* 處輸入小寫的英文字母，就表示所要找尋的字串包括大寫的字母在內。

如何直接找尋上述例子的 `ABC`，而不需經過 `abc`、 `Abc`、`aBc`、 `abC`、 `ABc`、`ABC` 等字串呢？欲達如此的效果，必需在 *minibuffer* 處，給予大寫字母 (upper case) 的字串。例如，在 *minibuffer* 處，給予 `ABC` 的字串，此時，緩衝區的內容若為 `abc`、 `Abc`、`aBc`、 `abC`、`ABc`、`aBC`、`ABC` ，則會直接搜尋 `ABC` 。 所以在 *minibuffer* 處輸入大寫的英文字串時， EMACS 所找尋字串的大小寫，就會與 *minibuffer* 字串的大小寫完全一樣。 此種對大小寫極度敏感的作法，稱為 *case sensitive*。EMACS 對於大小寫出現的位置也有差別，現在就先來討論 *case sensitive* 的問題。

- *minibuffer* 的字母都是小寫，則可能找到的字串，包含所有大小寫的字串。
  原始文件：`abc`、 `Abc`、`aBc`、 `abC`、`ABc`、`aBC`、`ABC`
  輸入 *minibuffer* 的字串： `abc`
  search string：`abc`、 `Abc`、`aBc`、 `abC`、`ABc`、`aBC`、`ABC`

- *minibuffer* 中的字母，若有任一個以上的字母是大寫，則可能找到的字串，就如同 *minibuffer* 所示的一樣。
  原始文件：`abc`、 `Abc`、`aBc`、 `abC`、`ABc`、`aBC`、`ABC`
  - 輸入 minibuffer 的字串： `Abc`
    search string：`Abc`
  - 輸入 minibuffer 的字串： `aBc`
    search string：`aBc`
  - 輸入 minibuffer 的字串： `ABc`
    search string：`ABc`
  - 輸入 minibuffer 的字串： `ABC`
    search string：`ABC`

如果希望 *minibuffer* 輸入什麼，緩衝區就找到什麼時，例如，在 *minibuffer* 中輸入 `abc`，所要找尋的字串就是 `abc`。此時就必需修改 EMACS “case-fold-search” 變數的值為 “`nil`”。因為 EMACS 對此變數的預設值是 `t`，它的意思就是使搜尋成為 *case sensitive*。若將此變數改成非 *case sensitive* 時，在 *minibuffer* 輸入 `abc`，就只會找尋 `abc`。

知道了 EMACS 對大小寫的處理程序後，現在就來談談 *incremental search* 和 *noincremental search* 。鍵入第一個字母至 *minibuffer* 時，搜尋的序幕就展開，是為 *incremental search* 。

使用 *incremental search* 時，當 *minibuffer* 收到第一個搜尋字母時，搜尋行動就從游標所在位置向下開始搜索。此時的游標，會從原先游標所在位置移至其下第一個出現此字母的地方；當 *minibuffer* 出現兩個字元時，游標也移至其下出現此兩個字元的地方。當然，這些都必需要以緩衝區中有這些文字為前題，若找不到任何合適的文字時， *echo area* 會出現 “`Failing I-search:`” 的警示語。

在 EMACS 中執行 *incremental search* 的方法有兩種，一種是往前的搜尋（*forward search*），另一種是回頭的搜尋 （*backward search*）。如下，就是二種搜尋所使用的指令：

- **Ctrl-s (isearch-forward)**
  游標所在處向前（右）搜尋。
- **Ctrl-r (isearch-backward)**
  游標所在處往回（左）搜尋。

當使用 **Ctrl-s** 或 **Ctrl-r** 指令時， EMACS 的 *echo area* 會出現 “`I-search:`”或 “`I-serach backward:`”。當出現這些提示字時，就表示其後是要輸入搜尋字串。只要鍵入第一個字元，搜尋的行動就立即展開了。 雖然如此，仍有幾個有關搜尋的關鍵字必需先行討論，它們是 “RET”、 “DEL” 與 “**Ctrl-g**”。

- RET
  當 *minibuffer* 收到 RET 時，游標會停留在最後一個滿足搜尋條件之處， 且結束搜尋的動作。 這種做法的好處是，若所找的資料就是所要的，且又是必需修改的資料，游標停留在該

處，即可立即處理。因為，有些離開搜尋狀態的方法，會使游標回到第一次執行搜尋的位置，這就是下面要談的 “**Ctrl-g**” 的處理原則。

- **Ctrl-g**
  “**Ctrl-g**” 的用法有二種，其一、是離開搜尋，承如上面所言；其二、就是當所鍵入的搜尋字串，無法找到完全符合的對應字串時，可利用 “**Ctrl-g**” 將 *minibuffer* 中所能找到的子字串留下，而將不能找到的資料除去。
  所以，若已經找到所要的字串而想離開搜尋時，鍵入 “**Ctrl-g**” 除了可以離開搜尋外，還可以將游標帶回原先的出發點。根據 *minibuffer* 處所給予的資料，完全無誤的在緩衝區中找到相對應的文字後，以 “**Ctrl-g**” 可將游標歸回原位。但若所給予的資料在緩衝區中無法找到完全對應的字，此時，若想離開搜尋的狀態，就必需用「兩次」的 “**Ctrl-g**” 才可完全脫離搜尋的狀態。此時，鍵入的第一個 “**Ctrl-g**” 會將 *minibuffer* 所找到的字串留下，而將無法兌現的字串刪除。此時再鍵入一次 “**Ctrl-g**”，則會完全離開搜尋的狀態，而游標也會自動移回展開搜尋時的位置。

- **DEL**
  想要修改 *minibuffer* 內的字串時，請用 **DEL (delete-backward-char)** 。還記得它嗎？它是用來刪除字元的，此刪除的動作是往回的刪除。每執行一次的**DEL**，就是一個新字串的誕生， 也就是一個新搜尋的展開。值得注意的是，此新展開的搜尋與原先的搜尋方向相反，這個設計是合理的。 因為，只要 *minibuffer* 輸入任何一個字，搜尋就己展開了，此時，若以 **DEL** 來刪除錯誤的輸入，則表示先前所找到的字串也不符需要，所以只能重新來過才是可行之路。這也就是為何每刪掉一個字元，搜尋的行動會回走的理由了。

要使用搜尋的指令，只需鍵入 “`Ctrl-s`”。此時 ， *minibuffer* 會出現 “`I-search:`”。若鍵入 `Ctrl-r` 後，則出現 “`I-search backward:`”。

使用搜尋的指令，會有若干情形出現：

- 首次搜尋，即覓得所要的字串。
  使用者此時可根據所需來做適當的處理。例如，離開搜尋，修改搜尋到的資料等等。

- 搜尋不利，必需一而再，再而三的努力，才能找到所要的字串。
  此時只要重複使用 “**Ctrl-s**”，游標就會移至下一個符合的字串。所以，只要所找尋的字串不變．就不需要再給予搜尋的字串。重複使用 “**Ctrl-s**” 的結果，仍無法找到所要的字串，*echo area* 會回應出 “`Failing I-search:`” 的訊息。

- 若重複使用 “**Ctrl-s**” 的結果，已使 *echo area* 出現 “`Failing I-search`” 後，則表示緩衝區已到了極限。若想再查看已閱瀏過的資料，只要再使用一次 “**Ctrl-s**”，就會迫使搜尋從原來的啟始點再進行一次的搜尋。

- 搜尋的過程中，若想回頭再查閱已看過的資料時，只要交替使用 “**Ctrl-s**” 與 “**Ctrl-r**”，就可以了。

EMACS 可以將搜尋過的字串，再拿出來重複使用。因為， EMACS 將使用過的搜尋字串，都放在一個名為 *search ring* 的變數中。它與先前討論過的 *kill-ring* 類似，都為變數。既為變數

，其值就可以增減與參閱的。至於如何查閱變數的值，請再行參考 4.4 節 （EMACS buffer and windows） 。

再使用搜尋過的字串，只要將 "*serach-ring*" 此一變數的值取出即可。取出其值的方法有兩種，一種是往前（右）的取出，另一種是往回（左）的擷取，其分界點是以最新使用過的搜尋字串為分野。因為 "*search-ring*" 為一個 ring，所以擷取的方式，不是採順時鍾的方向，就是採反時鍾的方向。今舉一實例說明之。

以下是以 "**Ctrl-h v**" 所得到有關 "*search-ring*" 的資料。

```
search-ring's value is ("kill" "Ctrl" "tex" "Ctrl-h" "text" "search")

Documentation:
List of search string sequences.
```

此時 "*search-ring*" 的變數值是：

```
kill、 Ctrl、 tex、 Ctrl-h、 text 、 search
```

最新的搜尋字串則位於最前端，此時為 "kill" 。往前（右）所得的字串則為 "Ctrl"，往回（左）所得的字串則為 "search"。今以此例，將使用 "*search-ring*" 得取使用過的搜尋資料的步驟說明如下：

- 首先鍵入 `Ctrl-s` 或 `Ctrl-r`
  - **ESC-p**
    當 *echo area* 出現 `I-search:` 後，再鍵入 "`ESC-p`"，則可得到 "`Ctrl`" 的字串。若想繼續往前尋找使用過的字串，只要重複鍵入 "`ESC-p`" 即可。
  - **ESC-n**
    當 *echo area* 出現 `I-search:` 後，再鍵入 `ESC-n`，則可得到 "search" 的字串。若想繼續往回尋找使用過的字串，只要重複鍵入 "`ESC-n`" 即可。

除了 *search-ring* 的資料可再使用外，將拷貝技巧運用在搜尋上，也是資料再使用的另一項運用。

拷貝在搜尋上的運用，就是將緩衝區的資料拷貝至 *echo area* 出現 `I-search:` 後的 *minibuffer* 處。經此拷貝的過程，任何大小的字串，都可使它輕易的出現在 *minibuffer* 處。

以下就是使用的方法：

- 鍵入 `Ctrl-y`
  若要將游標所在位置到列尾的所有資料，放入 *minibuffer* 處， "**Ctrl-y**" 可完成此一任務。
- 鍵入 `Ctrl-w`
  若只想拷貝游標所在位置之後的字（word），使用 "**Ctrl-w**" 是一個很好的選擇。 "**Ctrl-w**" 是拷貝緩衝區上的一個字（word）到 *minibuffer* 。其拷貝的範圍是以游標所在地為準則

，若游標位於一字的第一個字元時，則拷貝整個的字到 *minibuffer* 處，若游標所在位置不在字首，則將游標之後的次字串拷貝至 *minibuffer* 處。

- 鍵入 `ESC-y`
  向 *kill-ring* 借資料也是可行的方法之一。使用 "**Ctrl-s**" 或 "**Ctrl-r**" 指令後，可使用 "**ESC-y**" 將原先放於 *kill-ring* 的資料拷貝於 *minibuffer* 處。可惜的是，此方法只能將最新被刪除的資料來出用。

到此為止，所談的搜尋都是 *incremental search* 的搜尋，現在來談談 *nonincremental search* 。*nonincremental search* 是一般編輯器處理搜尋最常用的方法，所以又將其稱為傳統的用法。

nonincremental search 的使用，必需從 *incremental search* 開始。當以 "**Ctrl-s**" 或 "**Ctrl-r**" 啟動 *incremental search* 後，待 *echo area* 出現 "`I-search:`" 或 "`I-search backward:`" 後，只鍵入 RET 而不給予任何其它的字串，此時就啟動了 *nonincremental search*。當然 *echo area* 出現的訊息會不一樣，此時 *echo area* 出現的訊息 "`Search:`" 或 "`Search backward:`"。所以要使用 *incremental search* 或 *nonincremental search* ，完全取決於鍵入 *minibuffer* 的內容而定。若為只鍵入 RET ，就是選擇 *nonincremental serach*。

在啟動了 *nonincremental search* 之後，在 "`Search:`" 或 "`Search backward:`"之後鍵入 "`Ctrl-w`"，則表示要執行字的搜尋（Words Search）。此功能可以用來搜尋一組的字，只要在 *minibuffer* 中將所要搜尋字群以一個空白隔開，就表示所要搜尋的是一個字群了。以下將 *nonincremental serach* 的相關指令整理如下：

- *nonincremental* 字串（string）的搜尋
  - **Ctrl-r RET** *string* **RET**
  - **Ctrl-s RET** *string* **RET**
- *nonincremental* 字（word）的搜尋
  - **Ctrl-r RET Ctrl-w** *words* **RET**
  - **Ctrl-s RET Ctrl-w** *words* **RET**

還有一種搜尋不論是 *incremental search* 或 *nonincremental search* 都有的，那就是 *regular expression*，簡稱為 *Regexp*。所謂的 *regular expression* 就是以最少的字元組合來表示最大可能的巨集。現先不討論如何來表達 *regular expression* ，只討論如何使用 *regular expression* 的搜尋。因為有關 *regular expression* 會有專節（6.3 節）來討論。

- *Regexp* 的 *incremental search*
  - **ESC-Ctrl-s (isearch-forward-regexp)**
    使用 *Regexp* 的搜尋方法：
    先按下 "ESC" 之後再將其放掉，隨後再按下 "Ctrl-s"， *echo area* 處就會出現
    `Regexp I-search:`
    這時就可以輸入 *Regexp* 的表示字串了。
    當然還有若干事與 *Regexp* 的搜尋有關，討論如下：

    * 若要繼續以 *Regexp* 的方法搜尋，只需繼續鍵入 “`Ctrl-s`” 就可找到下一筆相關的資料。
    * *Regexp* 的搜尋，也有其獨立的 *search-ring*。這也意謂著可以再使用 *Regexp* 的 *search ring* 。其使用法與前所討論的方法一樣，就是以 “**ESC-p**” 與 “**ESC-n**” 來再使用 *Regexp* 的 *search-ring*。
  - **ESC-Ctrl-r (isearch-backward-regexp)**
    往回（左）的 *Regexp* 搜尋，若想繼續的往回找尋想要的字串，只需鍵入 “**Ctrl-r**” 即可。當然，也可以 “**ESC-p**” 與 “**ESC-n**” 來再使用其 *search-ring*。

- *nonincremental search*
  以下是以 *Regexp* 來執行 *nonincremental search* 的方法。
  - **ESC-x re-search-forward**
  - **ESC-x re-search-backward**

搜尋緩衝區的內容，除了要參考其內容外，有很大的機率是希望能將找到的內容以它種內容取代。取代（*Replacement*） 的方法，就是下一節討論的主題。

6.2 Replacing Text

從事編輯工作的時候，常會為了某種需求而將某一共同的用語以另外一種語辭替換，此情形最常發生在撰寫程式的時候。當然，英文字串由大寫改為小寫或小寫改為大寫也是經常可見的。編輯文件在處理這些事時，就如如下的若干問題產生。

1. 所有要修改的文件，都能如願以償的得到適當的修改，不會有漏網之魚發生。
2. 修改過文件的一致性，是必要的條件。

因為有這些問題的考量，所以使用 *replacement* 來完成如是的工作，就成為最佳的解決之道。現在就來談談 EMACS 如何處理 *replacement* 的問題。

EMACS 處理 *replacement* 的方法有一氣呵成的取代，與選擇性的取代二種。所給予的被取代文字也有二種，一、被取代的字串完全與所給予的字串一樣；二、以 *Regexp* 來做為取代的依據。試分述之。

- 二種的取代方法
  - 一氣呵成的取代
    所謂一氣呵成的取代，是一鼓作氣將游標之後所有符合條件的字串，全部以新字串取而代之。此作法不會一一徵求是否要取而代之的意見，而是自動、全盤且無條件的取代。此作法稱為 *Unconditional Replacement*。
  - 選擇性的取代
    選擇性的取代，是會先徵求取代的意見，只有在取得同意權時，才會採取取代的行動。這種取代的行為稱為 *query replacement* 。

- 二種被取代字串的表示法
  - 被取代的字串完全與所給予的字串一樣。
    此方法所要取代的文字，與 *minibuffer* 中所給予的文字一樣。所以，其可能符合條件的選擇最多也只有一種。
  - *Regexp* 來做為取代的依據。
    此方法就是以 *Regexp* 來表示所要找尋的字串集。前已論及，所謂的*Regexp* 就是以最少的字元組來表達最多的巨集。此時的取代，就不是單一字串的取代而是某一集合中的所有字串的取代。所以，以此所得的取代字串就有多重的選擇。如何表達正確的 *Regexp* 會在下一節中詳細討論。

以下就是使用取代（*replacement*）的方法。

- **Unconditional Replace**
  進行取代時不事先徵求意見，逕行將緩衝區中游標所在位置（包括游標所在位置本身）之後，所有符合的字串都以新字串取而代之。
  - **ESC-x replace-string RET** *string* **RET** *newstring* **RET**
    此指令是將緩衝區中，所有出現 *string* 的字串以 *newstring* 取代。其詳細的步驟如下：
    1. 鍵入 “`ESC-x replace-string`”，當按下 RET 時， *echo area* 會出現

       `Replace string:`

       此時可利用 “`Replace string:`” 後的 *minibuffer*，將所要被取代的字串輸入，按下 RET 則表示已完成輸入的工作 。
    2. 按下 RET 時， *echo area* 會出現如下訊息：

       `Replace string` *string* `with:`

       此時可利用 *minibuffer* 給予所要取代的新字串。 鍵入 RET 時，游標所在位置之後的所有 *string* 都會轉換成 *newstring*。
    3. 當轉換完成後，*echo area* 會出現 “`done`”。此時就已大功告成了。
  - **ESC-x replace-regexp RET** *Regexp* **RET** *newstring* **RET**
    此方法與上一個方法雷同，不同之處在於所要取代的資料不是某一個特定的字串，而是某一巨集的字串組。從指令的表示法，也可以看出所給予的被取代字串是 “regexp” 而非 “string”。其詳細的執行步驟與字串的取代雷同，只是在 *echo area* 將所有的 *string* 換成 *Regexp*罷了。
- **Query Replace**
  Query Replace 顧名思義就是在取代時會徵詢取代的意見，使用者可根據需要來取決是否要進行取代。以下就介紹字串和 *Regexp* 的取代。
  - **ESC-x query-replace RET** *string* **RET** *newstring* **RET**
    使用 *query replace* 的方法及步驟與 *unconditional replace* 的方法雷同，不同處只在於所

引用的指令有異，以及多增了詢問的選擇。
其詳細步驟如下（斜體字表示使用者所輸入的資料，粗體字表示系統自行根據輸入資料的回應）：

1. `ESC-x query-replace RET`
2. `Query replace:` *Regexp* `RET`
3. `Query replace` **regexp** `with:` *newstring* `RET`
4. `Query replacing` *Regexp* `with` *newstring*`:(? for help)`
   此步驟是 query replace 與 unconditional replace 最大不同之處。因為所有徵詢的工作都是從此展開。
   不知如何使用徵詢的使用者可鍵入 "?" 來得到線上的求助。以下就是鍵入 "?" 系統給予的資訊。

```
Query replacing regexp with string.

Type Space or 'y' to replace one match, Delete or 'n' to skip to next,
RET or 'q' to exit, Period to replace one match and exit,
Comma to replace but not move point immediately,
C-r to enter recursive edit (M-C-c to get out again),
C-w to delete match and recursive edit,
C-l to clear the screen, redisplay, and offer same replacement again,
! to replace all remaining matches with no more questions,
^ to move point back to previous match.
```

使用 *query replace* 時可有多種選擇項。現只介紹常用的幾個選項：

(a) **Space 或 y**
當決定以新的字串取代原來的字串時，以 **Space** 或 **y** 來表示。
執行 *query replace* 時，游標會移到第一個合適的字串處，此時若決定將其取代，則鍵入 "Space" 或 "y"。當取代完成後，游標會自動移至下一個合適的字串處。當然也可以放棄所找到的字串，這就是下一個要討論的選項了。

(b) **Delete 或 n**
放棄字串的取代，使游標移至下一個目的地，是 **Delete** 或 **n** 所做的事情。

(c) **.** (Period)
若已找到合適的字串，而想終止所有進一步的取代行為時，鍵入 "." 可使目前游標所在處的字串以新的字串取代，並且在取代後立即離開 *query replace* 的狀態。

(d) **!**
"!" 可使 *query replace* 恢復為 *unconditional replace*。
因此，若想放棄詢問的權利而恢復 *unconditional replace* 的狀態，鍵入 "!" 就

可將游標所位置及其以後所有合適的字串，都以新字串取而代之。

(e) **RET** 或 “**q**”
若想就此離開 *query-replace* 而不再做進一步的取代動作，只需按下 `RET` 或 “`q`” 即可。

- **ESC-x query-replace-regexp RET** *Regexp* **RET** *newstring* **RET**
  此方法與 “**ESC-x query-replace RET** *string* **RET** *newstring* **RET**”相似，但此時所取代的不是特定的字串，而是某一字串的巨集。

字串大小寫（*case sensitive*) 的問題與取代也有很大 的相關性。因為大小寫的問題，對於 *unconditional replace* 與 *query replace* 均適用。所以，只舉 *query-replace* 為例說明，至於 *unconditional replace* 就如法泡製。
當啟動 “**ESC-x query-replace RET** *string* **RET** *newstring* **RET**”
時，*string* 與 *newstring* 的大小寫，關係不同字串的取代。其規則如下所述：

- 當 *string* 與 *newstring* 都以小寫的形式出現時，取代工作的進行，就有三種情形：
  - 緩衝區原始字串的第一個字母是以小寫為開端時，不論此字串是否有其它的大寫字母，經取代後全轉換成小寫。
  - 若原始字串的每個字母都是大寫時，經取代後也維持大寫的形式。
  - 若原始字串以大寫為開端，不論此字串是否還有其它的字母為大寫，只要不是全為大寫的情形，取代後只有字串的第一個字母為大寫，其餘一律為小寫。

  以下舉一實例，供參考：
  string ：abc ，*newstring*： `xyz`
  原始文件為： `abc Abc aBc abC ABc AbC aBC ABC`
  經過取代為： `xyz Xyz xyz xyz Xyz Xyz xyz XYZ`

- 當 *newstring* 部份為大寫，而 *string* 依然維持小寫的情形：
  - 若原始字串的第一個字母為大寫時，被取代後的第一個字母仍維持大寫的形式。其餘原始字串的大小寫，就視 *newstring* 的大小寫而定，與原始字串本身的大小寫無關。
  - *newstring* 的字母若以大寫出現，則被取代的原始字串也會在相對應的位置以大寫的形式出現。
  - 若原始字串全為大寫時，取代後仍維持大寫的形式。

  以下為若干實例，供參考：

  - *string* ：abc ，*newstring*： `Xyz`
    原始文件為： `abc Abc aBc abC ABc AbC aBC ABC`
    經過取代為： `Xyz Xyz Xyz Xyz Xyz Xyz Xyz XYZ`

  - *string* ：abc ，*newstring*： `xYz`
    原始文件為： `abc Abc aBc abC ABc AbC aBC ABC`
    經過取代為： `xYz XYz xYz xYz XYz XYz xYz XYZ`

- *string* ：abc ，*newstring*： xyZ
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xyZ XyZ xyZ xyZ XyZ XyZ xyZ XYZ

- *string* ：abc ，*newstring*： XYz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： XYz XYz XYz XYz XYz XYz XYz XYZ

- *string* ：abc，*newstring*： xYZ
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xYZ XYZ xYZ xYZ XYZ XYZ xYZ XYZ

- *string* ：abc ，*newstring*： XYZ
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： XYZ XYZ XYZ XYZ XYZ XYZ XYZ XYZ

• 只要 *string* 有大寫的字母出現時，取代後字串的大小寫，就完全依照 *newstring* 的大小寫。
  換言之，*newstring* 為大寫的地方依舊為大寫，為小寫的地方依然為小寫。
  以下為若干實例，供參考：

- *string* ：Abc ，*newstring*： xyz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xyz xyz xyz xyz xyz xyz xyz xyz

- *string* ：aBc ，*newstring*： xyz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xyz xyz xyz xyz xyz xyz xyz xyz

- *string* ：ABc ，*newstring*： xyz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xyz xyz xyz xyz xyz xyz xyz xyz

- *string* ：ABC ，*newstring*： xyz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： xyz xyz xyz xyz xyz xyz xyz xyz

- *string* ：aBC ，*newstring*： Xyz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC
  經過取代為： Xyz Xyz Xyz Xyz Xyz Xyz Xyz Xyz

- *string* ：Abc ，*newstring*： xYz
  原始文件為： abc Abc aBc abC ABc AbC aBC ABC

經過取代為： `xYz xYz xYz xYz xYz xYz xYz xYz`

- *string* ：`aBc` ，*newstring*： `XYz`
  原始文件為： `abc Abc aBc abC ABc AbC aBC ABC`
  經過取代為：`XYz XYz XYz XYz XYz XYz XYz XYz`

- *string* ：`ABC` ，*newstring*： `XYZ`
  原始文件為： `abc Abc aBc abC ABc AbC aBC ABC`
  經過取代為： `XYZ XYZ XYZ XYZ XYZ XYZ XYZ XYZ`

討論至此，取代的部份應該可以告一段落了。接下來，就是要將一直未正式討論的 *Regular Expression* 做一詳盡的說明。

6.3 Regular Expression

所謂的 *Regular Expression* (以下簡稱 *Regexp*) 是用來表達一連續字元的組合，或是用來描述字樣 (pattern) 的一種方法，它在概念上不同於字串。

字串是指由一連續字元 (character) 所形成的字元組，一個字元組就只代表一個特定的字串，而 *Regexp* 通常所代表的是一組具有共通特性的字串集，此共通特性稱為 *pattern*。*Regexp* 的精神，就是希望以最少的字元，來表達最大的巨集。所以 *Regexp* 所能表達的字串數應不只於一個，而是一個字串的集合。這也是字串與 *Regexp* 最大不同處。當然，也可將字串視為 *Regexp* 的一個特例。

本文一直沒有討論撰寫程式的問題，但不可否認的，程式撰寫使用到 *Regexp* 的機會多不勝數。舉凡變數、常數的重新命名，同類字串的搜尋，數個程式合併後的整理工作等等，若能靈活使用 *Regexp* ，則可達事半功倍之效。這是一篇討論 EMACS 使用入門的文章，並不是討論程式語言的文章，所以只是將 *Regexp* 的用法告知。

撰寫程式常有的經驗，就是要將數個函式合併後，必需將使用這些函式的地方，以合併後的新名稱取代。例如 `read_i.c`、`read_c.c` 與 `read_f` 等三個函式合併成 `read.c` 時，所有使用這三個函式的地方，都必需改成 `read.c`。此時，若以傳統的修改方法，恐會漏一掛萬，若採取代法，則必需執行三次的取代，才可以大功告成。但使用 *Regexp* ，則可以一舉成功。既方便又快速。

又則，編輯文件或程式時，註解的功夫是不可或缺的。不同的文件與語言，各有其不同的註解表示法。例如 LISP 語言是以 “;” 表示註解， LaTex 則以 “%” 為註解。有時為了將註解分類，會以不同數目的註解符號來做區分。此時遇到的問題是，當重新分類註解時，相對應的註解符號也要調整。以 LISP 語言為例，重新分類後，可能將擁有一個 “;” 及二個 “;” 的註解符號，以三個 “;” 註解符號取而代之，這時若使用 *Regexp* ，問題就可迎刃而解。

程式檔名的轉換也可以 *Regexp* 來輕鬆完成。因為不同的程式各有其不同的副名，例如 C 語言的副名為 “`.c`”， C++ 語言的副名為 “`.C`”，而 FORTRAN 語言則為 “`.f`” 等等。此時，若想將 FORTRAN 語言所發展軟體，轉換成 C 語言，則必需將所有 “`.c`” 的檔名做適當的調整。

例如，將所有“filename.f”的檔案，改為“filename.c”的檔名。此時，若使用 *Regexp*，可能在極短的時間就可將所有的檔名轉換成功。*Regexp* 的運用非常的廣泛，這裏所舉的例子只是鳳毛麟角。以下就開始簡介 *Regexp* 的使用法，至於更詳細的介紹，請參照另一篇 94019 的技術報告。

Regexp 並非是 EMACS 的專利，而是 UNIX 系統下的產物。UNIX 系統下的許多工具程式都使用 *Regexp*，雖然彼此在表達 *Regexp* 的方法上，略有差別，但在概念上卻是一致的。UNIX 系統上使用 *Regexp* 的工具，除了EMACS 外，尚有 ed 、ex 、vi 、sed 、awk 、grep 、egrep 等等。這裏所介紹的 *Regexp*，在概念上與其它地方是共通的，但在實際的運作上可能會有所出入。

Regexp 是由字元所組成，此字元分為一般字元與特殊字元兩種。一般字元所組成的 *Regexp*，是最簡單的 *Regexp* 的表示法。因為它所要表達的字串與 *Regexp* 完全一模一樣。由特殊字元所組成的 *Regexp* 就較為複雜了。因為不同的特殊字元，各有特殊的代表意義。EMACS 中 *Regexp* 的特殊字元有 $、 ^ 、 .、 * 、 + 、 ? 、 [、] 及 \ 等九個。例如，“a”是一般的字元，它也只代表“a”，別無其它。但是“a.”、“a*”、“a?”與“a+”等所代表的意義，除了“a”外，尚有其它的意涵。特殊字元之外的字元，都是一般字元，但有些特殊字元是由一般字元加上 \ 而形成的。

現在來談談幾個簡單的 *Regexp* 的表示法。想進一步多瞭解 *Regexp*，仍請參考編號為 94019 的技術報告。以下就開始討論，代表 *Regexp* 的符號有九個。為了討論上的方便，將其分成若干類：

- 不在乎所出現的字母為何，可以『.』來表示。
  所以『.』代表除了 *newline* 之外的任何一個字元。
  例如“a.b”表示任何一個由三個字元所組成的字串，此三個字元必需符合第一個字元是“a”，第三個字元是“b”，但中間的字元只要不是 *newline*，任何字元都可以。所以“a.b”可以為 aab、abb、acb、axb、a1b、a2b 等等，甚至可以是一些特殊字元，只要這些特殊字元在此時沒有特別的意義。例如“a^b”。

- 以特殊字元，來表示重複出現的一般字元。這些特殊字元稱為 postfix character。其表示符號有『*』、『+』與『?』等三種。

  - 『*』
    任何字元之後加上『*』，可表示字元重複出現的次數，從零次到無限多次。因此“ab*”，可表示，a、ab、abb、abbb、abbb 以及 ab...b 到無限個 b。

  - 『+』
    任何字元之後加上『+』，可表示字元重複的出現次數，從一次到無限多次。因此“ab+”，可表示為 ab、abb、abbb、abbb 以及 ab...b 到無限個 b。

  - 『?』
    任何字元之後加上『?』，可表示字元重複的出現次數，不是零次就是只有一次。因此“ab?”不是表示為“a”，就是表示為“ab”。

- 若字元為屬於某一特定集合中的元素時，可以字集 (character set) 來表示。此時的代表符號是 [...]。中括號內就是放置字集的地方。如下討論字集的若干變化。

  - [...]

最簡單的字集表示法，是將所有符合的字元放於在括號內。例如， [a@!d13] 。此時，符合的字元就只有 a、@、!、1 和 3 而已。若所表示的字集，具有 一定範圍的連續性，為了方便表達，可以 [a-g] 來代表 [abcdefg] 。 這一類的表示法，將會有更詳細的討論。

– [...] 與其它的特殊字元的配合。

  * [...] 與 「*」 的配合
    c[ab]d 代表 “cad” 或 “cbd”。
    c[ab]*d 代表頭尾為 “c” 與 “d” 的字串 ，頭尾間的組合，則由 [ab]* 來決定。[ab]* 可表示為 cd、cad、cbd、caabd、cabbaar 等形式。

  * [...] 與 + 的配合
    c[ab]+d 表示頭尾為 “c” 與 “d” 的字串 ，頭尾間的組合，則由 [ab]+ 來決定。[ab]+ 可表示為 cad、cbd、caabd、cabbaad 等形式。

  * [...] 與 ? 的配合
    c[ab]?d 表示頭尾為 “c” 與 “d” 的字串 ，頭尾間的組合，則由 [ab]? 來決定。[ab]? 可表示為 cd、cad、cbd 等形式。

– 特殊字元，為字集內容時的解釋

任何 *Regexp* 的特殊字元，出現在中括號內都可將其視為一般的字元，除了二個符號之外。這二個例外的符號，也要看其出現的位置而做不同的解釋。這兩個符號是 “^” ，“-”。

1. “^”
   “^” 若出現在括號的第一個位置時，表示「以下皆非」的意思。所以， [a-zA-Z] 表示所有英文字母的集合，而 [^a-zA-Z] 則表示除了英文字母之外的所有集合。
2. “-”
   簡化範圍性的字集的表示法。其用法是將字集的起始點放於 “-” 的右邊，而終點放於 “-” 的右邊。
   例如，[abcdefg] 可以 [a-g] 來表示，[a-z] 則表示二十六個字母的集合。[0-9] 則表示數字的集合，要表示二位數字的集合，則可以 [0-9][0-9] 來表示。

• 指定 *Regexp* 出現的地方為列首或列尾。

  – “^”，指定 *Regexp* 出現在列首的符號。
  在 *Regexp* 前加上 “^” 的符號，即表示此 *Regexp* 必需出現在列首。例如， “^ The” ，則找尋每列以 “The” 為首的字串。

  – “$”， 指定 *Regexp* 出現在列尾的符號。
  指定 *Regexp* 必需出現在列尾時，只要在 *Regexp* 後加上一個 “$” 即可。例如， xxxx+$ 會將列尾以 “x” 結束的字串找出。

• “\” 的用法
\ 在 *Regexp* 中有二種涵意：一、使特殊字元變為普通字元，二、使普通字元轉為特殊字元。

  – 特殊字前加上 \ ，此特殊字元就不具特殊意義，只是一個普通字元而已。例如，列首要以 ^ 為開端，則以\^ 來表示，此時的 ^ 則為普通字元。

– 將如下的普通字元

|、 (、)、 d、 '、 `、 b、 B、 <、 >、 w、 W、 sc、 Sc

之前加上 \ 之後，則此普通字元就已特殊化了。現在只討論幾個常用的字元（|、(、)、d、<、>）。

* \|，（表示選擇的用法）
  欲在二個 *Regexp* 中擇一而用，可以 \| ，將二個 *Regexp* 放於其左右來表示。
  例如， `the\|it` 為二選一的 *Regexp* 的表示法。可能的符合字串為大小寫穿插其間的 the 或 it。例如，The、tHe、thE、THe、tHE、THE、iT、IT 等。為何會有不同的大小寫表示法，請再一次注意 EMACS 對於 *case sensitive* 的處理方式。若不清礎者，請參照 6.1 節。
  以 \| 所表示的 *Regexp* 有一個特色，那是在找尋合適的字串時，會找出與 *Regexp* 從開始至結尾都符合的字串。所以，若想找出 read 或 get 其後立即接上 file 的字串，就必需以 "`readfile\|getfile` 來表示。乍看之下，似乎沒有任何的疑問。事實上，這種表示法也沒有不對，可是各位是否還記得 *Regexp* 的真諦，就是要以最少的字元來表示最大的字集，在這個例子中的重複性很高，似乎不太能符合 *Regexp* 的精神。下面就是改進的方法。
* 利用 \(...\) 將 *Regexp* 的範圍規範出來。
  利用此方法，上例就可以 "`\(read\|get\)file`" 來找尋 readfile 或 getfile。這樣是不是簡捷多了嗎？
  \(...\) 還可以配合 *、+、? 等特殊字元使用。 *、+、? 等特殊字元在處理字元的重複性時，只適用在特殊字元之前的一個字元，所以若想重複一組字元時，就必需以 \(...\) 將其組合起來。例如， "`ba\(na\)*`，可以表示 bana、banana 與 bana.......na 等無數個 na 的組合。

– 除了以上所述的表示法外，若干普通字元加上 \ 還可以有特別的用法，以下討論常用的幾個用法。

1. *d*
   在 *Regexp* 的表示法中，可以只擷取部份的符合 egexp 。其做法是將要保留的 *Regexp* 暫存在緩衝區內之後，再將其拿出來使用。例如，要將所有副名為 .c 的檔名，換成為 .f 的檔名，其做法如下：

   ```
   ESC-x replace-regexp RET \(file[0-9]\)\.c RET \1\.f
   ```

   如此，會將所有名為 file1.c file2.c file3.c ...... file9.c 的檔名，改為 file1.f file2.f file3.f .... file9.f。
2. \<，尋找一字的開頭
   \>，尋找一字的結尾
   例如，`\<b[a-z]g\>` 會將 beg、 big、 bag 等字串找出。

Chapter 7

More About Emacs

這章所談的內容較為紛雜，但都環繞在加強 EMACS 的編輯功能為核心。所談的內容包括 *register*、*bookmark*、錯字的自動偵測以及如何在 EMACS 中使用繪圖功能。現在就開始討論 *register* 和 *bookmark*。

7.1 Registers and Bookmarks

EMACS 的 *register* 是一個可將文件與游標位置 (*point loaction*) 暫時存放其間的地方。

register 有它的缺失，那就是一旦離開目前所使用的 EMACS，所有存於 *register* 中的資料，也會隨著消失。若想再使用這些放於 *register* 中的文件或游標位置時，必需重新將資料存入 *register* 中。所以 *register* 只適合暫時性的儲存，若想永久使用文件或游標位置，必需仰賴 *bookmark* 的幫助了。

bookmark 與 *register* 相似，它們都用來儲存物件，但在相似中又有不一樣的地方。二者不同的地方如下：

- 命名上的不同
  bookmark 的名稱可由「一個以上」的字元 (character) 組成，但 *register* 的名稱只能由「一個」字元來命名

- 資料保存的時間不同
  存於 *bookmark* 的資料具有永恆性，它可以在離開 EMACS 後還存在。但存於 *register* 中的資料，在離開 EMACS 後就無效了。

現在就分別介紹 *register* 與 *bookmark* 二者的用法。首先討論 *register* 的用法。*register* 可存放的內容有，游標的位置、一般文件與長方形文件的內容、檔案的名稱以及本文未曾討論的視窗資料等等。現在就一一來介紹這些內容如何存放，以及存放後如何將其取出用，與視窗有關的 *register* 在此依然不予討論。

- 儲存與移動至特定 point 位置的方法
  - **Ctrl-x r SPC** *r* **(point-to-register)**
    將游標目前所在的位置，存於 *register* 中。存於 *register* 的步驟如下：
    1. 將游標移至所要儲存游標位置的地方。
    2. 鍵入 "`Ctrl-x r`" 以及 "SPACE" (空白鍵) 後， *echo area* 會出現

       ```
       Point to register:
       ```

    3. 此時可輸入任意一個字元 (character) ，做為此 *register* 的名稱。以後要使游標移至此位置就要靠此字元。 *register* 的名稱，只能由一個字元組成。
  - **Ctrl-x r j** *r* **(jump-to-register)**
    根據 *register r* 所儲存的位置，將游標移至 *r* 所設定的位置。使用此指令前，必需確定 *register r*，已設定的妥當了。 以下就是設定游標位置以及使用游標位置的步驟：
    1. 先以 "`Ctrl-x r SPC` *r*"，將游標的位置儲存於 *register r* 內。
    2. 鍵入 "`Ctrl-x r j`"， *echo area* 會出現 "`Jump to register:`" 的訊息。
    3. 在 "`Jump to register:`" 之後，輸入已設定妥善的 *register r* 的名稱。
       此時游標所在的緩衝區若與 *register r* 所設定的緩衝區不一樣時，游標會自動移至另一個緩衝區，視窗的內容也會換成新的緩衝區內容。所以，使用 *register* 所存放的游標位置，是可以跨越不同的緩衝區來做移動的。
- *register* 還可用來存放經常被訪問到的檔案。
  有人或許會不解，為何耗費如此的功夫，只為了從事 "**Ctrl-x Ctrl-f**" 指令可以做到的事。使用 "**Ctrl-x Ctrl-f**" 有一個不便之處，就是若所欲訪問的檔案與 EMACS 的預設檔案位置不一樣時，則必需告之完整的路徑名。所以，此時若使用 *register* 來儲存檔名，就只需要給予 *register* 的名稱即可。至於，冗長的全稱就交給 *register* 去處理了。以下就是以 *register* 存檔案的方法：

  1. 使用 *register* 來儲存檔名與儲存游標位置，最大的不同點在於，以 *register* 來儲存檔名必需借助 "`.emacs`" 檔。
     因為以 *register* 來儲存檔案名稱，必需借由 "`.emacs`" 來設定。
  2. 以 *register* 來儲存檔案名稱的實際執行步驟如下：
     (a) 設定 "`.emacs`" 的方法
     在 "`.emacs`" 檔中以 "`set-register`" 函式 ，將 *register* 的名稱以及其所儲存的檔名設定清礎。以下的例子就是將 "`chap7.tex`" 的檔名，放於名為 "a" 的 *register* 時 ， "`.emacs`" 檔的設定方法 。

     ```
     (set-register ?a '(file . "\~/report/emacs/basic/chap7.tex"))
     ```

     (b) 當檔案名稱的設定已在 "`.emacs`" 檔中完成後，必需離開 EMACS ，再重新進入 EMACS 一次。因為只有重新執行 EMACS ， 修改過的 "`.emacs`" 檔才有機會被重新執行。重新執行過的 EMACS ，就可以
     "`Ctrl-x r j` *r*"
     將檔案從 *register r* 中拿出來使用。以下就是使用 *register r* 的方法：

i. 鍵入 “`Ctrl-x r j RET`”， *echo area* 會出現 “`Jump to register:`”。

ii. 待 “`Jump to register:`” 的訊息出現時，即可輸入已設定好 *register* 名稱。以上例為例，若輸入 “a”，則視窗會出現 `chap7.tex` 的文件內容。

- *register* 中儲存文件的方法

  儲存於 *register* 的文件可以有二種：一種為線性的文件 (linear text)，另一種為長方形的文件 (rectangle text)。所以有此區分，原於使用 *register* 儲存文件，需先將所要儲存的文件做上標記，再根據標記放於 *register* 中。線性文件與長方形文件在標記上是一樣的，要區分二者的差異只能仰賴使用時的指令了。想進一步瞭解長方形區域，請參考 5.4 節 “Move Text and Copy Text” 有關長方形部份。以下就討論二者的用法。

  – 線性文件的 *register* 設定方法如下：

  1. 將要放入 *register* 的線性文件做好標記。設定標記的詳細方法，請參考 5.3 節。現略做題示，使用 **Ctrl-@** 或 **Ctrl-SPC** 做為線性文件的起始點，以游標所在的位置為終點。
  2. 使用指令 “**Ctrl-x r s** *r*”，將所設定好的線性文件放入 *register r* 中。
  3. 欲使用 *register r* 中的資料時，可以指令 “**Ctrl-x r i** *r*” 將存放於 *r* 中的資料取出使用。

  以上所設定的 *register* 有一特性，就是離開 EMACS 後所有的設定也成為歷史。想將 *register* 內容保留的方法，是將設定寫在 “.emacs” 中，如下是以 *register* “a” 儲存 *register* 文件，以及以 *register* “.” 儲存 “。” 資料的設定法：

```
(set-register ?a "register")
(set-register ?. "。")
```

  – 長方形文件的 *register* 表示法：

  1. 將要放入 *register* 的長方形文件做好標記。設定標記的方法以及所謂長方形文件的定義，請參考第五章第三節及第四節。
  2. 使用指令 “**Ctrl-x r r** *r*”，將所設定好的線性文件放入 *register r* 中。
  3. 使用 *register r* 中的資料時，可以指令 “**Ctrl-x r i** *r*” 將存放於 *r* 中的資料取出使用。

  放於 *register* 中的文件，不論是線性文件或長方形文件，將其拿出使用的指令都是一樣的。不一樣處在於放入 *register* 時的差異。會有差異是不難理解的，因為二者在處理標記的過程是一樣，只有靠存入 *register* 時，以不同的指令來加以區別。

談完了 *register*，接下來介紹 *bookmark*。 *bookmark* 的一些基本概念，已在前面略做介紹了，現在就進一步詳述它。 *bookmark* 與 *register* 在記錄游標位置的功能是一樣的，但在對 *register* 的命名方面二者就顯得有差異了。 *bookmark* 的命名可以由一個以上的字元組成。換言之，它可以有一個較長的名字，但 *register* 就只能由一個字元來命名。而且以 *bookmark* 設定的資料，可永久儲存起來，即使離開 EMACS， *bookmark* 的資料也不會就此消失。所以想要永久保存的資料，只有借重 *bookmark* 了。以下就是 *bookmark* 的使用方法。

1. 以 “**Ctrl-x r m** *bookmark* ” 指令，將游標所在的位置做上記號（mark），此記號就稱為 *bookmark*。
   當第一次使用 “**Ctrl-x r m**” 指令時， *echo area* 會出現 “`Set bookmark (visited-filename) :`” 的訊息。此時，若不輸入任何字元只鍵入 RET，系統會以所訪問的檔名做為 *bookmark* 的名稱。若輸入其它的字串，則 *bookmark* 就以此字串命名之。

2. 以 “**Ctrl-x r b** *bookmark* **RET**” 指令，來移動游標的位置。
   鍵入 “`Ctrl-x r b`” 時，*echo area* 會出現 “`Jump to bookmark (bookmark-name) :`”，括號內的名稱為系統的預設值。若所給予的預設值不是心中所愛，可以在 ‘:’ 之後輸入想要的 *bookmark* 名稱。

資料以 *bookmark* 的方式儲存時，使用者若不另行指定儲存的檔名，系統會將有的訊息存入一個名為 “`~/.emacs-bkmrks`” 的檔案中。

接下來，討論一個使 EMACS 的編輯功能更強化的議題 － Fixing Typos。

7.2 Fixing Typos

EMACS 所提供的這項功能，並不是內建於 EMACS 的。 EMACS 只是提供使用 UNIX 拼字工具的介面，使用者可以借由此一介面，使用 UNIX 的拼字工具。 UNIX 環境，較常使用的拼字檢查工具程式（spelling checker program）有 **spell** 與 **ispell**。二者又以 **ispell** 較為方便好用。雖然如此，並不是所有的 UNIX 系統都提供有 **ispell** 與 **spell** 等工具程式。使用者可以 “**which**”、“**find**” 等指令，在 shell 下查閱 UNIX 系統所提供的工具為何。不清礎如何使用 **which** 或 **find** 等指令的使用者，請查看 man page 。

EMACS 提供的拼字檢查介面，是針對 **ispell** 的使用法為主。透過 EMACS 的介面，使用者可使用 UNIX 系統所安裝的 **ispell**。若系統沒有安裝 **ispell**，可以 ftp[1] 到適當的地方得取。

ispell 除了會自動偵察文件的拼字錯誤外，還提供可能修改的訊息，供使用者參考。**ispell** 的除錯範圍，可以偵察一個字，也可以偵察一個區域或整個的緩衝區。

使用 **ispell** 時，程式會根據所檢查的內容做出適當的回應。如果檢查無誤， *echo area* 會出現無誤的訊息。如果所檢查的字有誤， **ispell** 會另開一個視窗，將所有可能的勘誤組合顯示出來，以利使用者做評估。若在顯示的視窗找到合適的取代文字，可直接鍵入文字的編號來完成勘誤的程序。

ispell 使用了兩種不同的字典，一種是由 **ispell** 所提供的標準字典 （standard dictionary）。另一種是使用者自已建構的私有字典（private dictionary）。標準字典是系統所提供的；使用者字典是在編輯文件時，因需要而隨時加入的。當標準字典無法找到的字，使用者可隨時將新增的字加入私有字典中。系統所使用的標準字典，可依使用者的需要來指定。當然，若不指定標準字典，系統會使用預定的字典。若想改變預定的字典，可使用如下的指令來設定：

```
ESC-x ispell-change-dictionary
```

[1] Ispell is available via anonymous ftp from ftp.cs.ucla.edu in the directory /pub/ispell.

若想查閱所使用的標準字典為何，可查閱變數 “ispell-dictionary” 來得到相關訊息。如下是筆者以 “**Ctrl-h v**” 指令查閱此變數所得的結果。若已淡忘變數的查詢，請參考 4.4 節。

```
ispell-dictionary's value is nil

Documentation:
If non-nil, a dictionary to use instead of the default one.
This is passed to the ispell process using the "-d" switch and is
used as key in ispell-dictionary-alist (which see).

You should set this variable before your first call to ispell (e.g. in
your .emacs), or use the M-x ispell-change-dictionary command to
change it, as changing this variable only takes effect in a newly
started ispell process.
```

現在就討論 **ispell** 的使用方法。

- 一般文件使用 **ispell** 檢查錯誤所使用的指令，可依所檢查的範圍分成如下三種：
  - 檢查「字」的 **ispell** 指令如下：
    ESC-$
  - 檢查「區域」（region）的 **ispell** 指令如下：
    ESC-x ispell-region
    使用區域的檢查以前，要先將區域的範圍界定清礎，再使用此指令來做區域的檢查。
  - 檢查「緩衝區」（buffer）的 **ispell** 指令如下：
    ESC-x ispell-buffer
    檢查整個緩衝區內的文字內容。
- 信件中使用 **ispell** 檢查錯誤的指令
  ESC-x ispell-message
- 以 **ispell** 檢查文字內容時，*echo area* 會有回應的訊息出現。若有錯誤發生時，會另開一個視窗，將所有的修改可能顯示出來。如何使用 **ispell** 來修改錯誤，就是現在的話題了。
  - 若檢查的範圍只是一個字時，此時，所檢查的字若正確無誤，*echo area* 會出現

    word `is correct`

  - 若 **ispell** 認為所檢查的英文字有誤時，會另開視窗，將 **ispell** 認為可能的正確字列舉出來。使用者可參照所顯示的字，來做出合適的回應。回應的方式如下：
    - **SPC**
      保留現狀而不更改錯誤，但將游標移至下一個發生錯誤的地方。

  * *digit*
    當有錯誤出現時， **ispell** 會將可能的正確字一一編上號碼，供使用者參考。 若所顯示的字就是所要更改的字，此時只要鍵入編號，緩衝區的錯誤即可獲得修正。
    例如，檢查 *dictionory* 這個字是否的正確時，視窗上會出現“ `(0)dictionary` ”的提示，此時若想修改錯誤的字，只需鍵入提示的數字‘`0`’即可。
  * **r** *new* **RET**
    將目前找到的錯字以新字 *new* 取而代之。文章中其它地方若有相同的錯誤，不會因此而有變動。
  * R *new* **RET**
    除了將目前出錯的地方修正外，同時還進行 “query-replace” 的動作，將所有相同錯誤的地方也一併修正。
  * **a**
    將找出來的錯誤，將錯就錯視為是正確的。此錯誤適合所有目前此 EMACS 下的所有緩衝區，但離開 EMACS 後就恢復原狀，錯還是錯。
  * **A**
    大寫的 “A” 與前一個小寫的 “a”，唯一差別的地方就是，將錯就錯的地方，只適用於目前所使用的這一個緩衝區。
  * **i**
    ispell 所使用的字典有二種，一種是標準字典，另一種是使用者自行訂定的私有字典。將標準字典中所沒有的字加入私有字典中就是靠指令 “`i`” 了。
    ispell 在檢查字的正確與否時，是根據標準字典來判斷對錯。此標準字典對於專有術語的搜集，並不是很周延。為了彌補此缺失，私有字典就應運而生了。使用者可將標準字典中沒有的字，加入私有字典中，使這些原本被視為「錯」的字得以正名。
    當 **ispell** 檢查出「錯字」時，鍵入 “`i`” ，就可將此錯字納入私有字典中，而成為「正確」的字。。從此若再出現此字時， **ispell** 就會將其視為「正確」的字。縱使離開 EMACS 後再進入 EMACS，此字也會是「正確」的字。 若想使新字加入私有字典時， *echo area* 能將加入的新字回應出來，則必需使用 “`m`” 指令，而非 “`i`” 指令。

以上所談的是透過 EMACS 所提供的介面，使用 UNIX 的拼字檢查的工具 (**ispell**) 。如果所使用的系統無法提供此工具軟體時，請以 **ftp** 取得軟體後再自行裝置 **ispell**。否則，就算 EMACS 提供了使用 **ispell** 的介面，也是無濟於事的。

除了以上所談的以軟體來修正錯誤外，還有其它的方法可以修正所發生的錯誤，詳述如下：

- **Transposing Text**
  - **Ctrl-t (transpose-chars)**
    游標前後兩個字元 (character) 互調。
  - **ESC-t (transpose-words)**
    游標前後兩個字 (word) 互調。但兩個中間的標點符號 (punctuation) 則不會移動。例如， `transposing,text` 則互換成 `text,transposing` 而不是 `text transposing,`。

- **Ctrl-x Ctrl-t (transpose-lines)**
  前後兩列互換。將游標所在的列與其上一列互換。

- **Case Conversion**

  - **ESC-l**
    將一個字的字元改成小寫的字元。鍵入指令 `ESC-l` 會將游標所在處之後的所有字元換成小寫的字元，其範圍只限於游標所在處的那一個字。
  - **ESC-u**
    將一個字的字元改成大寫的字元。鍵入指令 `ESC-l` 會將游標所在處之後的所有字元換成大寫的字元，其範圍只限於游標所在處的那一個字。
  - **ESC-c**
    將游標所在處的字元換成大寫的字元（capital），所以如果要使一個字以大寫為開始可以使用此指令。

接下來所要討論的議題是，如何使用 EMACS 的編輯器來繪製簡單的圖形。

7.3 Picture Editing

EMACS 的繪圖模式所能提供的繪圖功能並不多，它只予許以鍵盤上出現的字元，來構思圖形，也就是只能以 ASCII 的字元，來編輯所要的圖形。EMACS 的圖形模式可以與任何其它的模式一起搭配使用。所以，只要有編輯圖形的需要，都可以切換至圖形模式，將所要的圖形編輯出來。

EMACS 的繪圖功能，主要是由八個控制游標移動方向的指令來完成。這八個方向也是 EMACS 構圖的主體，它們依序是，東、南、西、北、東南、西南、西北以及東北。雖然可以設定八種游標移動的方向，但一次只能設定一個方向，若想使游標移動的方向改變，必需重新設定游標移動的方向。設定這八個游標移動方向的指令分別是：

- **東：Ctrl-c > (picture-movement-right)**
- **西：Ctrl-c < (picture-movement-left)**
- **南：Ctrl-c . (picture-movement-down)**
- **北：Ctrl-c ^ (picture-movement-up)**
- **東南：Ctrl-c \ (picture-movement-se)**
- **西南：Ctrl-c / (picture-movement-sw)**
- **東北：Ctrl-c ' (picture-movement-ne)**
- **西北：Ctrl-c ` (picture-movement-nw)**

這八個指令似乎頗為複雜，但如配合圖形的解釋，就可瞭這八個指令的由來。因為，每個指令的表示法與所要表達的方向有著密切關係。

```
             北(rignt)
                C-c ^

                  .
       西北(nw)   .   東北(ne)
         C-c ‘    .    C-c ’
              .   .   .
               .  .  .
                . . .
                 ...
 C-c < .................. C-c >
西(left)         ...        東(right)
                . . .
               .  .  .
              .   .   .
         C-c /    .    C-c \
       西南(sw)   .   東南(se)
                  .
                C-c .
              南(down)
```

從以上的圖示，不知能否看出這八個控制方向的指令與所表達的方向有著密不可分的關連性。例如，往東的方向則以 「>」 來表示，往西的方向則以 「<」，往北的方向則以 「^」 來表示等等。基本上，從使用的指令就可知道所代表的方向為何。當然，其原意並不是要以指令來猜方向，而是用方向來詮釋指令。所以，只要掌握西南方的方向，就知道使用 「/」 來表示。

圖形模式可以在編輯的過程中隨時使用的。使用圖形模式的方法很簡單，只要將其喚起即可。喚起圖形模式的方法是： “**ESC-x edit-picture**”。 當進入圖形模式後，*mode line* 會將使用圖形模式的訊息顯示出來。進入圖形模式後，也可以隨時離開此模式，而回到喚起此模式前的狀態。回到先前模式的方法是 “**Ctrl-c Ctrl-c**”。

進入圖形模式後，就可利用設定游標移動方向的八個指令，來控制輸入字元出現的方向，而繪製出合適的圖樣。當選定了游標的移動方向後， *mode line* 會將游標移動的方向顯示出來。此時，鍵入任何一個鍵盤上的字元，字元會依照所設定的方向出現在螢幕上。

圖形模式所採用的是覆蓋模式，這是有其道理的。繪圖就是要在特定的地方繪上所欲的圖樣，如果使用插入的模式，就會破壞整體的設計。所以，使用覆蓋模式是較合理的安排。

圖形模式除了設定游標移動方向的指令外，還有其它的指令是針對編輯功能而設。例如，字或列的刪減，空白列的增加等等，都是在繪圖時，不可或缺的編輯指令。現在就來討論與繪圖有關的編輯指令。繪圖的編輯指令與一般編輯的指令大同小異，為了避免將一般的編輯指令與繪圖時所使用的編輯指令混淆，現將二者的差異整理列舉如下：

Keystrokes	Text Mode	Picture Mode
RET	加入一行空白列	使游標向下移動，若要加入空白列，則使
		用 “Ctrl-o” 指令。
Ctrl-d	刪除一個字元後，文件會向左移動	將游標所在處的字元以空白取代，但文件
		不會向左移動。若要刪除字元，必需使用
		”Ctrl-c Ctrl-d” 指令。
SPACE	輸入空白且將文件移向右邊	游標向右移動，同時游標所到之處，均將
		文字以空白覆蓋。此時若想輸入空白，則
		必需回到 Text Mode，在 Text Mode
		的狀態輸入空白 。
Ctrl-k	刪除所在列的內容，使用兩次	刪除內容但不會刪除列。欲刪除列只有回
	Ctrl-k 則刪除內容與此空白列。	到 Text Mode。
TAB	輸入 TABs 且將文件向右移動	只以TAB 的距離移動游標，但它所經過之
		處並不會輸入 TABs 。
Ctrl-n	游標移至下一列，但游標所在的欄	游標移至下一列，但欄位與上一列相同。
	位則視情形而定。	
Ctrl-p	游標移至上一列，但游標所在的欄	游標移至上一列，但欄位與下一列相同。
	位則視情形而定。	
Ctrl-f	游標向前移動一個字元。	游標向右移動一個字元。
Ctrl-b	游標往回移動一個字元。	游標向左移動一個字元。

除了以上所談的種種功能之外，圖形模式亦可與長方形編輯相配合。長方形編輯是專用來編輯區塊文件，而圖形模式的編輯，從某個角度來看，就是區塊的組合。所以，長方形編輯的運作，就可用在圖形模式的剪貼上。因此，任何可以使用在長方形編輯的指令，都可用在圖形模式中。不熟悉長方形的編輯者，可參考本文的 5.4 節。

圖形模式除了可使用長方形編輯的指令外，圖形模式本身，也提供了適用於此模式的專門指令。現就一一為各位介紹。

- 可利用的長方形編輯指令，如下：
  - **Ctrl-x r d (delete-rectangle)**
    刪除設定好的長方形區塊內的文件，經刪除後的文件無法再將其 **yank** 出來。除非是執行 **undo** 的指令。
  - **Ctrl-x r k (kill-rectangle)**
    與圖形模式 “**Ctrl-c Ctrl-k**” 的作用相同。
  - **Ctrl-x r y (yank-rectangle)**
    與圖形模式 “**Ctrl-c Ctrl-y**” 的作用相同。
  - **Ctrl-x r o (open-rectangle)**
    在設定好的長方形區塊內，插入空白後使文件向右移動。
  - **Ctrl-x r r** *r* **(copy-rectangle-to-register)**
    將設定好範圍的區塊資料，存入 *register* 中，此時區塊內的資料並不會消失。
  - **Ctrl-x r i** *r* **(insert-register)**
    將存入 *register* 中的資料取出。
  - **ESC-x clear-rectangle**
    將設定好的長方形區塊內的文件以空白取代。

- **ESC-x string-rectangle RET** *string* **RET**
  設定好的長方形區塊插入新的字串（*string*）。此時的區塊只決定加入新字串的長度，寬度則由所給予的字串長度來決定。原來區塊內的文件會向右移動。

- 圖形模式自行開發的編輯指令，如下：
  - **Ctrl-c Ctrl-k (picture-clear-rectangle)**
    長方形指令 "**Ctrl-x r k**" 與此指令具有相同的效果。此指令經常被使用，所以將長方形的指令予以精簡，使其在圖形模式中更易於被使用。
  - **Ctrl-c Ctrl-y (picture-yank-rectangle)**
    長方形指令 "**Ctrl-x r y**" 與此指令具有相同的效果。此指令經常被使用，所以將長方形的指令予以精簡，使其在圖形模式中更易於被使用。
  - **Ctrl-c Ctrl-w r (picture-clear-rectangle-to-register)**
    將區塊內的資料存入 *register* 中。此時區塊內的資料會以空白取代，這是與長方形指令 "**Ctrl-x r r** *r* 最大不同之處。
  - **Ctrl-c Ctrl-x** *r* **(picture-yank-rectangle-from-register)**
    將存入 *register r* 中的資料取出。

長方形指令的使用在圖形模式中非常有用。因為，圖形的構成就是一塊塊的區域。所以善用長方形指令會使圖形的編輯更加靈活。

Chapter 8

Editing Under Chinese Enviorment

以上所談的種種，都較適合英文的編輯環境，現在來討論一個適用於中文編輯的環境。 在 EMACS 下可使用的中文環境有二種，一個就是 EMACS 本身，另一個 是 EMACS 的姐妹品 – Mule 。現在就開始討論這二個中文的編輯環境。

8.1 Emacs under Chinese Enviorment

在 EMACS 的環境下要使用中文，必需先做些設定。 因為，中文內碼是使用 8 位元位元組，所以必需將系統所使用之位元組型式設為 8 位元 ，才可以在中文環境下使用 EMACS 。設定系統使其適合使用中文的環境，必需視所使用的作業系統與所使用的 Shell 的 script file 有關，現簡述如下：

- 若使用 C Shell/TC Shell 時， .cshrc 的設定方式又因所用的 UNIX 作業系統不同可分為如下二種方式，
  - 使用 Sun OS 作業系統的設定

    ```
    setenv LC_CTYPE ISO_8859_1
    ```

  - 使用 HP-UX 作業系統的設定

    ```
    setenv LC_CTYPE american.iso88591
    ```

  - 在螢幕上顯示 8 位元的中文時，設定 8 位元的方式如下：

```
stty cs8 -istrip -parenb
```

- 若使用 Bourne Shell/Korn Shell 時， .profile 的設定方式又因所用的 UNIX 作業系統不同可分為如下二種方式，
  - 使用 Sun Solaris 作業系統的設定

    ```
    LC_CTYPE=ISO_8859_1
    export LC_CTYPE
    ```

  - 使用 HP-UX 作業系統的設定

    ```
    LC_CTYPE=american.iso88591
    export LC_CTYPE
    ```

  - 在螢幕上顯示 8 位元的中文時，設定 8 位元的方式如下：

    ```
    stty cs8 -istrip -parenb
    ```

除了以上針對 Shell 的 script file 的設定外，要在 EMACS 的環境輸入與顯示中文尚必需在 EMACS 的起始檔 “.emacs” 中加入如下的設定:

```
(set-input-mode (car (current-input-mode))
        (nth 1 (current-input-mode))
        0)
(standard-display-8bit 160 255)
```

雖然經由這些設定，可以使 EMACS 在中文的環境下使用，但其處理方式還是以英文的模式來處理中文。換言之，就是將二個 byte 所組成的中文字，依然視為二個 byte 的英文字來處理。這種處理方式，自然有其不便之處。例如，在處理換列時就會遇到一些問題。 因為 EMACS 將中文視為是二個 byte 的字元共同組合而成。所以，當一列只剩下一個 byte 時， EMACS 會將中文切割成二個獨立的 byte 分別處理之。此時中文字的第一個 byte 會在此列的最後一個欄位，而下一列的第一個欄位則出現此字的第二個 byte 。所以，在 EMACS 的環境下中文字很容易被切割，而不知其為何物。

因為 EMACS 處理中文的方式與處理英文無異，所以刪除一個中文字，等於刪除二個 byte 的字元（character）。換言之，一個中文字若使用刪除字元的指令 （**Ctrl-d**）來刪除，必需使用兩次的 **Ctrl-d** 才能將一個中文字刪除。

因為 EMACS 在處理這些七位元之外的字型有其基本上的問題，所以就有 Mule 的誕生。 以下就介紹 Mule 的使用法。

8.2 Mule

Mule 是 "The MULtilingual Enhancement of GNU Emacs" 的簡稱，它是針對非英語系國家的使用者而設計的 EMACS。對中文的使用者有很大的助益。

要使 EMACS 能真正的中文化，最好的方法就是安裝 Mule。 目前最新的 Mule 版本是 Mule 2.0 版（1994 年 8 月 6 日）。

Mule 的設計是以 EMACS 為基礎。至於 EMACS 的啟始檔案 — "`.emacs`"，也就是 Mule 的啟始檔案。換言之，使用 Mule 不需在 "`.emacs`" 檔中做任何的設定。所以 Mule 可將 EMACS 的 "`.emacs`" 檔做為其啟始檔。 若無特殊需求，也可不使用 "`.emacs`" 檔。

想要取得 Mule 的軟體，可以 "anonymous ftp" 至以下幾個地方取得：

```
etlport.etl.go.jp [192.31.197.99]:/pub/mule
ftp.mei.co.jp [132.182.49.2]:/public/free/gnu/emacs/mule
sh.wide.ad.jp [133.4.11.11]:/JAPAN/mule/mule-1.0
ftp.funet.fi [128.214.6.100]:/pub/gnu/emacs/mule
cs.huji.ac.il [132.65.16.10]:/pub/gnu/mule
```

前已述及 Mule 是 "MULtilingual Enhancement to GNU Emacs"。它不只處理七位元（7 bits）的 ASCII 字型（ASCII Characters）以及 ISO Latin-1 的八位元字型（8 bits）。它更能處理日文、中文（GB、Big5）、韓文 （16 bits，ISO2022 標準）、泰文 （TIS620）以及越文（VISCII 及 VSCII）。 目前適合吾人使用的繁體中文系統是以 BIG-5 為主。

Mule 有如此多的語文可供使用，所以在一個文字檔案中，使用者可以混合使用這些不同的語言。使用這些語文，只需借由 Mule 所提供的輸入法，將各種不同的文字輸入即可。如果是在 terminal emulator 下使用 Mule（如 cxterm、exterm 或 kterm）， 可使用此 emulator 所提供的輸入方法。

其實使用 Mule 的方法與使用 EMACS 的方法是一樣的， EMACS 的指令在 Mule 中都可以使用。二者最大的差別，就是 Mule 加入了輸入法的使用。欲在 EMACS 的環境下使用中文，首先要起動中文系統（例如倚天中文系統），而輸入法就是使用中文系統所提供的輸入法了。使用 Mule 時也是先起動中文系統[1]，但中文的輸入法，則由 Mule 來提供。換言之， Mule 有自已的中文輸入法。現在就來討論如何使用 Mule 的輸入法。

Mule 所提供適合中文使用的輸入法有，倉頡、注音、拼音、倚天注音以及標點符號等五種。使用這些輸入法的方式如下：

[1]此處仍是以 text-only 的終端機為討論的重點，若使用中文的 X-window 則不需先行啟動中文系統。

1. **Ctrl-x - Ctrl-k - Shift-m RET**
   連續鍵入 “`Ctrl-x`” “`Ctrl-k`” “`Shift-m`” 三個指令後，按下 RET， *echo area* 會出現

   ```
   Quail Package：
   ```

   此時就可鍵入所要使用的輸入法。

2. 輸入法的名稱如下所示：
   - 倉頡：cj
   - 注音：zozy
   - 拼音：pinyin
   - 倚天注音：etzy
   - 標點符號：punct-b5

   要使用任何一種輸入法，只要在 *echo area* 處的 ”`Quail Package:`” 後，鍵入所欲使用的輸入法的名稱即可。

3. **Ctrl-] (self-insert-command)**
   在任何一種的輸入法的狀態下，都可切換至英文的狀態下。 只要鍵入 “`Ctrl-]`” 就可使中英文互換。

Mule 對使用中文的人來說，可說是中文化的 EMACS ，而 Mule 的誕生，確實給予中文的編輯者很大的方便。Mule 的使用方法，除了輸入法與 EMACS 略有不同之外，其它與 EMACS 一樣。所以，已熟悉 EMACS 的人，使用 Mule 可說是易如反掌。如果只安裝 Mule 而無安裝 EMACS 的系統，想要使用 EMACS ，只要進入 Mule 後一直維持英文的 狀態就與使用 EMACS 一樣了。

EMACS 的入門手冊介紹到此，已可暫告一段落。這並不意謂著 EMACS 的介紹已經窮盡了。其實，有關 EMACS 的話題還有許多，這裏只是環繞在編輯的層面略作解說。

Chapter 9

Conclusion

Emacs 入門簡介到此已近尾聲了，希望讀者能有所穫。不論是從此開始使用 Emacs 或從此不再使用 Emacs，都是一種收獲。撰寫這本手冊除了要介紹如何使用 Emacs 外，更希望讀者能從中得知 Emacs 能解決何種問題。畢竟問題得以解決才是大家關切的重點。

當然 Emacs 能解決的問題，其它的軟體或許也能提供相同的解答。那麼使用 Emacs 的理由是否就變得薄弱了呢？其實，天生我材必有用，重點在於要適材而用。如果只是為了編輯一個簡短的檔案而且只從事一次的編輯，就不一定非鍾情於 Emacs 不可。如果在編檔的同時又想從事編譯的工作，在工作的同時還要兼顧信件的收發，這時候選擇 Emacs ，就是個良策。因為 Emacs 本身就是一個整合的環境，它對於需要整合性的工作，非常的有助益。

許多使用過 Emacs 的人，都覺得 Emacs 的指令太過於複雜而使人卻步。關於這一點筆者從不反駁，但是對於 Emacs 的瞭解不能有因噎而廢食的反應。畢竟，指令只是用來實踐軟體的工具而已，軟體的本質才是要費神研究的重心。所以在使用 Emacs 的時候，不要因它五花八門的指令而怯步，希望讀者能多留意 Emacs 所能解決的問題。只有如此，才能見到它的真諦。

至於 Emacs 能做那些事，在前文中或多或少已經介紹過了，讀者可以自行研判它是否可以解決問題。當然，有些 Emacs 能做的事在文中並未論及，除了限於篇幅外，還因為本文一直環繞在編輯器的層面。所以，對於一些程式的編撰與除錯的問題就略過不提。對於 Emacs 巨集指令的安排，也因為超過「入門」太遠也暫時將其放下。這些未曾提及的主題，並不是意味著它們不重要。而是因為不適合於此時此地出現。

文章結束前再一次的強調 Emacs 是以編輯為基石，而以發展整合性的作業系統為志業的軟體。所以，只要進入了 Emacs 的工作環境，就理應可以在其中完成所有的工作。離開 Emacs 的環境就是離開電腦工作的時候。

最後，如果各位在看完此手冊，覺得心動者，請就開始行動吧。只有勤加使用它，才會越發覺得它的好用。開始時的不順，是必然的現象。開始動手吧。如果仍不知從何開始的使用者，請先進入 Emacs 後，使用它的線上輔助說明吧！任何時候請不要忘記 “Ctrl-h”，只要知道使用 “Ctrl-h” 那使用 Emacs 就有希望了。

Index