目录

Introduction

本文的主旨以讨论 EMACS 的入门为主，其内容包括 EMACS 发展的概述， EMACS 整合环境的简介以及 EMACS 的基石 － editor 的详谈。本文在结构上分成八章，第一章讨论有关 EMACS 的源起、发展历史以及 EMACS 的特性简介。 第二章讨论 EMACS 的 on-line help。第三章则简单介绍 EMACS 的整合环境。接下来就是本文的重点，探讨 EMACS 的本质。 所以，第三章之後的各章节则详细探讨 EMACS 为 editor 的这一个主题。

现在就来讨论与 EMACS 发展有关的机构组织以及 EMACS 所扮演 的角色特质。

GNU 是什麽

GNU 是 Richard Stallman 於 1975 年，在 MIT 所成立的 Free Software Foundation (FSF）中所执行的一项计划。GNU 的本意代表 ``Gnu's Not Unix''；虽然如此，它却是一个与 UNIX 完全相容的软体系统。二者最大不同在於，GNU 是一个 free 的 软体， UNIX 却是一个要付费的软体系统。 GNU 之所以与 UNIX 完全相容，是因为 UNIX 的使用者很多，为了让 UNIX 的使用者在使用 GNU 的时候不会有疏离的感觉，所以 GNU 尽量与UNIX 相容，它的相容只是看起来与用起来像 UNIX 而已， GNU 其实改进了所多 UNIX 的缺点，使它能尽善尽美。

Stallman 倡导 FSF 的宗旨，是要建立一个软体 free 的王国。他所谓的 free 并不是意谓著便宜或免费，因为在 FSF 下的有些软体还是必需付些工本费。他所谓的 free 是指使用上的自由。

要了解 Stallman 所言 free 的真意，就必须知道一般软体的使用情形。大部份的商业软体在使用上都或多或少有些限制。例如，必须付费才可使用软体程式，若运气不好，同一程式在不同的电脑上使用，可能还要另行付费。有些软体程式的使用也有一定的期限，期限一到就需另缴费用，否则使用权就会被撤销；甚至有些软体是以使用时间来计算费用的。当然，这些商业软体是绝对不予许使用者将这些软体任意给他人使用。除非给予软体公司相当的代价，使用者是绝对不可能拿到软体的原始程式的。

上述的种种都还是付钱就可以解决的，有些问题是钱也乏术的。因为，当商业程式销售失败或远景不被看好时，使用者所购置的软体就可能被开发者遗弃。此时，若想维持此软体的正常运作，就必须自求多 了。 Stallman 所领导的 GNU 计划就是要脱离商业软体的种种枷锁，这也是 GNU 计划所谓 ``free" 的真谛。GNU 计划下的任何软体，只要使用者能找到软体的来源，任何人都可以自由的使用它。获得 GNU 软体的来源很多， 如 public archives、anonymous FTP、UUNET 等，甚至任何一个装有 GNU 软体的地方都可以取得。GNU 计划下的软体，不只提供软体的使用权，也提供软体的原始程式，任何人都可以根据需要来修改程式，也可以尽己之力来找出程式的错误，使隶属於 GNU 的软体在大家的努力下能尽善尽美。 GNU 计划下的软体，是可不需付费而享有使用权。GNU 对使用者唯一的要求就是，当使用者对於 GNU 计划下的软体做了进一步的修改时，仍必须维持 GNU 的精神， 就是对於修改过的软体仍然必须将其无条件的奉献出来，任何人都不可将修改过的 GNU 软体当成商品来买卖。所以 Stallman 一直强调 GNU 计划下的软体是 free 的，而且 永远会是 free 的。GNU 计划的最终目标是要完成一个全新的作业系统。目前完成的有 EMACS text editor、debugger、yacc-compatible parser generator 以及 linker 等等；也完成了将尽三、四十个公用程式（utilities）；而 shell 也已经接近完成的阶段。目前只要等 kernel 和 compiler 完成， 就可以在 GNU 的系统上发展程式。

未来完成的 GNU 系统将可以执行 UNIX 上的所有程式， 但它却与 UNIX 系统不完全一致。 GNU 系统将改进 UNIX 系统的缺失，使它使用起来更方便。例如，未来的系统将会有较长的档名，档名会有版次等等；新的系统会使用 C 与 LISP 做为系统程式语言。

GNU Emacs 是什麽

GNU EMACS 是 GNU 计划下的第一个产品， EMACS 为 Editor MACros 的缩写。 Richard Stallman 於 1975 年在 MIT 首次 撰写 EMACS editor。目前 GNU EMACS 已发展到 19.28 版，本文所讨论的 GNU EMACS 也以 19.28 版为主。 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 这个 site ，取得 GNU EMACS 的原始码。 GNU EMACS 不是 public domain 的产物， 它有版权也有使用上的限制，那就是任何得到它的人都不可以将其视为私有的财产。修改过的 GNU EMACS 也不可以作为商品来牟利，金钱交易的行为是绝对被禁止。

EMACS 有多种版本可适合不同的工作平台（platfrom）。适合 UNIX 环境的有 GNU EMACS、Unipress EMACS 以及 CCA EMACS， 其中以 GNU EMACS 最常被使用，本文也只介绍 GNU EMACS。 适合个人电脑的 EMACS 有 FreEMACS、MicroEMACS 以及 Epsilon。当然，也有使用在 VMS 环境的版本。

GNU EMACS 是由 C 与 LISP 语言写成，任何人都可依据需要将个人所发展的函式（function）加入 GNU EMACS 上。当然，新发展的软体是不可以从事商业买卖，只能将它无条件的奉献出来。新发展的函式可以直接在 EMACS 中使用，不需重新编译（complie）整个 EMACS ，而且新增的函式也不会破坏 EMACS 原有的结构。就因为有此特性，EMACS 的函式可以 与日俱增。愿与他人共享成果的使用者，可以透过电子邮件或电子公布栏，将函式的原始码公诸於世。公布的函式，最後会经由 FSF 的审查，以决定是否要加入新版的 EMACS 中。FSF 也鼓励使用者将所发现的错误，透过相同的管道，提供给 FSF 作为改进 之用。GNU EMACS 就是在如此的运作下，靠大家共同的努力来提升品质，以达产品的稳定性。

GNU EMACS 的特质

GNU EMACS 与其说它是一个编辑器（editor），倒不如说它是一个以编辑器为主干的 环境软体。一般的软体都是将编辑器视为一个附属功能，只有 EMACS 以编辑器为基石，在其上发展其它的功能。以收发电子信件的软体为例，电子信件软体是以收发信件为环境的基石。任何电子信件的软体都只有在使用编辑器时，才呼叫相关的编辑器。 EMACS 的出发点就与众不同， EMACS 一切以编辑器为主，任何的功能都是建基在编辑器之上。

所以，进入 EMACS ，就等於是进入了一个编辑环境，这个编辑环境提使用者许多功能，让使用者如置身在一个全功能的作业系统中。 EMACS 自行发展了一个 ``bourne-shell-like'' 的 shell， 除了 EMACS 自己的 shell 外， EMACS 还可以让使用者自行选择所使用的shell；EMACS 可以读送 e-mail。 EMACS 可以透过 ftp 来编辑远方 host 上的档案，而不需要签入（login）档案所在的 host； EMACS 也可做 telnet 与 relogin 的动作。EMACS 也可以读 news； EMACS 也提供了年历（calendar），可以让使用者查阅日期，也可以将重要的事情在年历上标示出来；EMACS 又提供了 ``Diary'' 的功能，当特定的日期与时间到来时，会在萤幕上将 ``Diary'' 上 的事情显示;EMACS 也有撰写文章大纲的功能。 EMACS 更提供多种程式的编译功能，让使用者可以在 EMACS 中一边编辑程式一边编译程式； EMACS 更有自己的 debugger，使程式的除错、 编辑与编译在 EMACS 中同时完成。

所以 EMACS 所提供的不只是一个编辑器而已，它所提供的是一个整合的工作环境，而这个环境是建立在编辑的基石上。它希望使用者进入了 EMACS 以後，可以在 EMACS 的环境中完成 所有的工作，不需要离开 EMACS，要离开 EMACS 就是要离开电脑的 时候（logout）。 EMACS 除了是一个整合的工作环境外，它还具有以下的特性：

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

EMACS 是一个 display 编辑器，因为每一个被编辑的文字都会被显示在萤幕上。

EMACS 是一个 real-time 的编辑器，因为当新的文字被键入时，萤幕会在非常短的时间内被更新。

EMACS 的编辑功能不只具备一般编辑器所有的功能，它还提供如下的功能：

- 文字的填充（filling of text）。
- 程式自动内缩功能。
- 可以同时阅读一个以上的档案。
- 对於字元（characters）、字（words）、列（lines）、句子 （sentences）、段落（paragraphs）、页（pages）以及各种 程式中的 expression 和注解（comments）都有其自有的处理方法。

EMACS 是一个 Self-documenting的软体，因为在任何时候都 可以 `` Ctrl-h" 指令来得立即的帮助。因为，透过 `` Ctrl-h" 可以得知每一个 EMACS 的指令。

EMACS 是可以 Customizable 的。使用者可视个人的需求，来改变 EMACS 指令的定义。GNU EMACS 的 Customization 的设定非常简单，使用者可以很方便的使用它。 EMACS 是 Extensible 的。GNU EMACS 是由 LISP 语言 所写成的函式共同组成的， 函式与函式间的互动关系（dependency）不强。就因为 EMACS 是由函式所组合而成的，所以可以随时将函式作增减而不会破坏 EMACS 既有的结构。 EMACS 也针对 X 的视窗环境，提供自己的选单（menus）和滑鼠按纽（mouse bottons）功能。 EMACS 对於 text-only 的终端机也提供许多与 X 环境相当的服务品质，例如在文字模式的终端机，可以同时开启数个的档案，档案可以互相切换，当使用 shell 模式 来执行 shell 指令的同时也可以编辑档案。但本文的只讨论 text-only 的 EMACS 使用法。 以上的种种就是有关 EMACS 的特色。至於如何使用这些功能就下列章节所要讨论的重点。 回主画面

Emacs 的线上辅助说明

EMACS 是一个整合的工作环境，初次使用 EMACS 或不熟悉 EMACS 的人，常会不知所措。所以，EMACS 提供了非常便捷且功能强大的线上辅助说明（on-line help），来帮助使用 EMACS 。

Emacs 的基本知识

在未讨论 EMACS 的线上辅助说明时，先谈谈如何启动 EMACS。 启动EMACS 的方法很简单，只要在萤幕的提示下键入 ``emacs'' 即可；离开 EMACS，只要键入``Ctrl-x Ctrl-c''即可离开 EMACS 。 键入 ``Ctrl-x Ctrl-c'' 的方法是，先按住键盘上的 Ctrl 键不放再按下英文字的 ``x'' 即可。键入 ``Ctrl-c'' 的方法一样，先按住键盘上的 Ctrl 键不放再按下英文字的 ``c'' 即可。当然进入与离开 EMACS 的方法还有多种，这里只是介绍一种方法，让使用者可以很快的使用 EMACS 的线上辅助说明，至於其它的方法会在以下各章节中陆续提及。

除了进入与离开 EMACS 的方法必须知道外，还有一个指令非常用那就是 ``Ctrl-g"。 ``Ctrl-g" 可用来取消键入的指令，如果不想执行所键入的指令，可以随时将其取消。

EMACS 键入指令的方法有二种：

- Ctrl 键
- Meta 键

所有 EMACS 的指令都可以 Meta 键表示出来，键盘上若无 Meta 键，则可以 ESC 键来取而代之。常用的 EMACS 指令通常会有一个 Hotkey 与之连结。 Hotkey 的构成，通常是以 Ctrl 为开头的型态出现。例如 ``Ctrl-x Ctrl-c" 这一个 Hotkey，则代表了 EMACS 指令的 ``save-buffers-kill-emacs" 。若要使用 Meta 键来表达与 Ctrl-x Ctrl-c 相同的效果 ，则必须键入

``ESC-x save-buffers-kill-emacs"。

使用 Meta 键，可以利用 EMACS *completion* 的功能。因为 *completion* 可以让使用者键入最少的字，就可使系统唤起所欲执行的命令。以下就介绍 EMACS的 *completion*。

EMACS *completion*的意思是，只要键入字串的部份， EMACS 会将其余的部份自动填入其应在的位置。如果所给予的字串不足以决定其余的部份，EMACS 会将所有可能的结果都列出来，以供使用者来决定所需要的字串名称。 *completion* 所适用的范围如下：

- 指令
- 特定目录下的档案
- 缓冲区
- EMACS 的变数

至於何谓特定目录下的档案、缓冲区以及 EMACS 的变数，会在以後的章节中陆续谈及。此处讨论*completion* ，只是为了使用 *completion* 於 EMACS 的线上辅助说明。

使用 EMACS 的 *completion* 有三种方法：

- TAB 尽可能将其余的字串填满。
- SPACE 将 puncutation 字元之前的字填满，填充的字不会超过一个字以上。
- ? 将所有可能的 *completion*s 选择都列出来。

使用*completion* 的做法是将部份字串键入後，再按下 TAB、SPACE 或 ? 即可。例如，键入 ``M-x au TAB" ，则萤幕的最下方会出现 ``键入 TAB 则萤幕会在另一个视窗出现：

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

若键入 ``M-x au SPACE"，则萤幕的最下方也会出现 ``M-x auto-"。若键入 SPACE ，则萤幕的另一个视窗也会出现如下的命令：

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

这似乎意味著 TAB 与 SPACE 的功能一样，其实不然，二者的差异可从下一个例子看出。键入 ``M-x auto-f TAB"，可得 ``M-x auto-fill-mode"；但键入 ``M-x auto-f SPACE"，只能得到 ``M-x auto-fill-" ，欲得到 ``M-x auto-fill-mode"，则必须再键入一次 SPACE。这就是前面所说的 ``SPACE" 一次只填一个 ``puncutation" 之前的一个字的意思；而 TAB 则是尽可能的将所有可以判断出来的字串呈现出来，其显示字串的长度并不以一个 ``puncutation" 为限。键入 ? 的作用，是在 EMACS 的另一个视窗上显示所有可能的字串，此时使用者可根据视窗上的讯息键入适当的命令。例如键入 ``M-x au ?" ，萤幕上出现另一个视窗显示如下的资讯：

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

键入 ``M-x au ?" 的地方，则不会执行 *completion* 的动作，这是 ? 与 TAB 、SPACE 最大不同的地方。

若视窗的内容太多无法一次穷尽，此时就必需卷动视窗。卷动视窗可以用 ``Ctrl-v" 与 ``Meta-v" 二个指令来使视窗做上下的移动。

Emacs 的自学教材

想快速了解 EMACS 的人，可以参考 EMACS 的自学教材（tutorial）。使用 EMACS 自学教材的方法很简单，只要键入 ``Ctrl-h t'' (help-with-tutorial) 即可进入 EMACS 的自学的状态了。

EMACS 的自学教材可分成以下几部份：

1. 介绍 EMACS 指令的键入方法，即介绍 Ctrl 与 Meta 键 。
2. viewing screenfuls
3. basic cursor control
4. Ctrl-g 的用法
5. EMACS 的 window 与 multiple windows
6. inserting and deleting
7. undo
8. EMACS 档案的处理
9. EMACS 的 buffers
10. extending the command set
11. 简介 EMACS 的 mode line 与 echo area
12. searching
13. recursive editing levels
14. getting more help
15. leaving EMACS
16. 有关 EMACS 的版权问题

EMACS 的自学教材是以编辑功能的介绍为主。虽然 EMACS 的功能不只如此， 但编辑是 EMACS 的最基础的功能，要了解 EMACS 当然要从它的基本著手。所以 EMACS的自学教材也以编辑的介绍为主，至於 EMACS 其它的工件环境，是无法从 此自学教材中得知。所以，本文会在第三章简介 EMACS 的工作环境。

Ctrl-h 的用法

EMACS 除了自学教材可供参考外，它还提供了其它的线上辅助说明功能，让使用者可以随时查阅需要的相关讯息。 EMACS 的线上辅助说明都是以 ``Ctrl-h'' 为开端，其种类 有以下二种：

- Ctrl-h
- Ctrl-h Ctrl-h

键入 Ctrl-h (help-command)，萤幕的最下端会出现如下的讯息： C-h（Type ? for further options）- 此时的 ``Ctrl-h" 只是用做前置字（prefix key），它是用来等候使用者输入其它的指令。若输入 ``?"，则萤幕的下方会出现所有可使用的选择，使用者可根据需要来选择合适的选项。若键入两次的 ``Ctrl-h"Ctrl-h (felp-for-help)，萤幕下方 会出现： type one of the options listed or Space to scroll 此时 EMACS 会另外开启一个视窗，将所有与求助的选项都列出来，且会做一简要的说明，要卷动此视窗则键入 Space。此新开启的视窗共有二十一个选项，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. 使用这二十一个选择项的方法非常简单，只要在 ``Ctrl-h" 之後输入任一个选择就可以了。例如，要选择 ``a" ，则执行 ``Ctrl-h a" 即可。

现在将 EMACS 常用的线上辅助说明一一作解释。EMACS 常用的线上辅助说明有：

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

``Ctrl-h c" 与 ``Ctrl-h k" 的功能相似， 二者都是在寻求与 Hotkey 有关的讯息。二者唯一的差别，就在於对指令的解释详细与否而已。 `` Ctrl-h c" 是简述与 Hotkey 连结的命令，而 ``Ctrl-h k"则详述连结 Hotkey 的命令。二者都有一个前题的预设，那就是都是先知道 Hotkey 为何，而想进一步知道此 Hotkey 所使用命令的名称。今举 ``Ctrl-x Ctrl-c" 的例子来说明二者的差别。

键入 ``Ctrl-h c RET" (RET，亦即键盘上的 Enter 键。任何一个指令输入完毕时，必需 紧跟著一个 Enter。此作用是用以告知系统，指令输入已经结束，可以开始执行相关的动作了。) 则萤幕下方会出现 Describe key briefly: - 在 ``:" 的後面键入 ``Ctrl-x Ctrl-c"，则在原先出现 Describe key briefly: - 的地方则出现 C-x C-c runs the command save-buffers-kill-emacs" 键入 ``Ctrl-h k RET"，则萤幕下方出现 Describe key: - 在 ``:" 的後面键入 ``Ctrl-x Ctrl-c"，则 EMACS 会另以一个视窗显示如下的讯息： 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. 此讯息的第一行是 Hotkey 所连结命令的全名，以後的行数则是对此命令的详细说明。 `` Ctrl-h w"w (where-is) 的用法与 `` 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对应的 Hotkey 时所使用的。例如键入 ``Ctrl-h w"，萤幕的下方会出现

where is command:

在 ``:" 之後键入 ``save-buffers-kill-emacs"，则在原处会出现

save-buffers-kill-emacs is on C-x C-c 所以想知道命令是否有相对应的 Hotkey，可以此方法查知。 ``Ctrl-h w" 可以使用 EMACS 的 *completion* ，但其最大的 不便处，就是必需给予指令的第一个字元。如 ``save-buffers-kill-emacs"， 必需先给予以 s 开头的子字串，才能逐步使用 EMACS 的 *completion* 。若不键入 s 开头的子串，而键入 s 之後的 任何字串，则无法找到相对应的指令，所以使用 ``Ctrl-h w" 必需要记著指令的第一个字。除此之外， EMACS 的线上辅助说明还提供了另一种帮助，可让使用者键入任意的子字串，都可以找到相对应的指令，那就是 `` Ctrl-h a"。 键入 ``Ctrl-h a" 则萤幕的下方会出现 command-apropos (regexp): 此时只要给予与命令相关的任一子字串或 `` regular expression" 6.3 节会讨论何谓 regular expression）再按下 RET， EMACS 会另开一个视窗，将所有涵盖此子字串或 regular expression 的指令全列出来。 此指令与 `` Ctrl-h w" 最大不同处如下：

1. 使用 `` Ctrl-h a" 所键入的子字串，并不限於指令的第一个字元, 而 `` Ctrl-h w" 则必需以指令的第一个字为起始字。 使用 `` Ctrl-h a"，可给予指令中任何位置的子字串。
2.
3. `` Ctrl-h a" 无法使用 *completion*，而 `` Ctrl-h w" 可使用 *completion*。
4.

   `` Ctrl-h a" 主要的目的是当使用者无法正确的键入指令的 第一个字元时，可以借此将所有包含使用者记得的部份子字串 的指令都列举出来。
   EMACS 除了以上几个常用的线上辅助说明之外， 还有一个非常实用的资料查阅中心，那就是 `` Ctrl-h i"。 `` Ctrl-h i" 执行 ``Info program" ，它主要是用来浏 已建构成树状结构的文件档案。目前所有与 EMACS 有关的文件档案都可 透过 Info 来阅览，最终所有与 GNU 有关的文件资料，将可以由此 而窥得其文件档案全貌。
   使用 info 模式（info mode）的方法很简单，只要键入 ``Ctrl-h i" 就可查阅所有与 EMACS 相关的文件资料。进入了 info 之後要如何 有效的使用它呢？会在下一节详细讨论。
   以上所谈的，就是较常使用的线上辅助说明种类。 若线上辅助说明的讯息，是以另一个视窗显示出来，此时的讯息 又无法一「幕」了然。想参考其它部份的资料，就必需卷动视窗。 视窗卷动的指令，可参考如下的方法：
   - Ctrl-x 1 （delete-other-windows）

- (本文所有与 Hotkey 相对应的指令都放於括号中。 保留游标所在的视窗，而将其它的视窗关闭。
- Ctrl-x o （other-window）
- 可使游标在不同的视窗间切换。换言之，如果游标在工作的视窗， 可以此指令将游标移出显示线上辅助说明讯息的视窗，反之亦然。
- Ctrl-v （scroll-up）
- 将萤幕向上卷，如此则可看清萤幕下方的讯息。
- Meta-v （scroll-down)
- 萤幕向下卷，如此可以重复参考已经看过的资料。

Emacs 的 info 使用说明

前已略述 info 的使用法，现在就更进一步详述之。在 EMACS 的线上辅助说明功能中，info 的内容可说是最为丰富的。因为， 任何线上辅助说明的文件都可在 info 模式中找到。因为，info 就是用来放置整个 EMACS 手册。

info 对於 EMACS 文件的安排是采取树状的结构， 所以是以根部（root）为出发点。info 执行 info program， 使用 info program 的方法有二：

- Ctrl-h i
- ESC-x info

info 对於档案的编排，是以 Hypertext 的方法来处理所有的 相关文件。当键入 ``Ctrl-h i" 或 ``ESC-x info" 後，会先进入 info 树状（tree）结构的最顶端。如下就是进入 info 时的第一个画面： -*- Text -*- This is the file .../info/dir, which contains the topmost node of the Info hierarchy. The first time you invoke Info you start off looking at that node, which is (dir)Top. ? 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" 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. * Texinfo: (texi.info). With one source file, make either a printed manual (through TeX) or an Info file (through texinfo). Full documentation in this menu item. * Termcap: (termcap). The termcap library, which enables application programs to handle all types of character-display terminals. * Regex: (regex). The GNU regular expression library. * Cpp: (cpp.info). C pre-processor. * Gcc: (gcc.info). GNU C Compiler --- an ANSI C Compiler developed by FSF. * Gzip: (gzip.info). GNU zip program --- an compress package developed by FSF. * Ispell: (ispell.info). A spelling checker. * Libg++: (libg++.info). G++ libraries. * Gmake: (make.info). A make utility developed by FSF. * Bison: (bison.info). GNU Yacc. * Gawk: (gawk.info). GNU awk --- pattern scanning and processing language. * Gdb: (gdb.info). GNU debugger. * Info-stnd: (info-stnd.info). Stand along GNU info. * Makeinfo: (makeinfo.info). Program for producing *.info file from *.texi file. * Graphics: (graphics.info). A set programs for producing plot files and display them on Tektronix 4010, PostScript, and X window system compatible output devices. * m4: (m4.info). m4 is macro processor, in the sense that it copies its input to the output, expending macros as it goes. GNU m4 is mostly compatible with system V, Release 3 version. * Hyperbole: (hypb.info). GNU Emacs-based everyday information management system. Use {C-h h d d} for a demo. Include Smart Key context-sensitive mouse or keyboard key support, a powerful rolodex, and extensible hypertext facilities including hyper-links in mail and news messages. * Standards Coding Style: (standards.info). GNU Coding Style. 此时，出现在萤幕的第一列是标头（header），它包含此结点 （node）的基本讯息。表头所提供的讯息，最多可有五件事情：

1. 结点所在的档案（File）
2. 结点的名称（node）

3. 此结点的下一个结点（Next）
4. 此结点的上一个结点（Prev）
5. 此结点的上一层结点（up）

此时的画面，只显示了二件事情，是因为此画面为 info 树状结构 的最上层。

在表头之下的资讯，是用来告知如何用 info 。它提供了五件事情，现一一说明。

- h
- 不知如何使用 info 模式者，可在进入 info 模式後，使用 info 的线上 辅助说明。欲使用 info 的线上辅助说明，只要在进入 info 模式後，键入 ``h" 即可。此不只是一个线上辅助说明，还是一个教学指引。跟随著它的说 明，即可明了如何使用 info 模式.
- d
- ``Ctrl-h i" 指令，会先进入 info 树状结构的根部。任何情况下， 可键入 ``d" 回到此根部。
- ?
- 想知所有与 info 有关的指令，只要键入 ``?"，就可以得知所有 指令的全貌。
- q
- 欲离开 info，只要键入 ``q" ，就可以离开 info 而回到先前的 缓冲区。
- m Texinfo< Return >
- 这一个指令是用来使用 info 的 Menu Item。使用的方法如下：
  1. 键入 ``m" 。
  2. 键入所欲参考的文件名称，也可使用 *completion* 的功能来简化输入的工作。
  3. 键入 RET。

  何为 Menu Item 会再解释。

在这五个指令之下的是，此结点可以使用的 Menu Item。 使用者可以直接至所要参考的 Menu 之下，键入 ``RET"。此时， info 会将此 Menu 的相关文件显示在萤幕上。使用 info 除了以上的指令之外，还有几个必需知道的指令。 现介绍如下：

- n
- 将结点移至下一个与此结点相连的结点。
- p
- 将结点移至上一个与此结点相连的结点。
- u
- 将结点移至上一层的结点。
- m
- 以上结点间移动的指令，必需结点间有相连接的关系。 若想做跨越的移动，这几个指令是无法做到的。此时，必需透过 Menu 来做夸越结点的文件阅读。 使用 Menu 的方式有二：
  - ``mTexinfo"
  - 这种使用 menu 的方法，在进入 info 的第一个画面就可以看到了。 此方法由三件事共同组成：
    1. m 键入 m 是使用 *menu* 的指令。
    2. Texinfo 键入 m 时，*minibuffer* 会出现
    3. menu item: 此时，可在其後输入所欲阅 的文件名称。所以，Texinfo 意 指文件的名称。此时，就可以键入画面上以 ``*" 为开头 的名称。如果是刚进入 info ，可使用的 Texinfo 名称有： Info、 Emacs、 VIP、 Forms、 GNUS、 CL、 Gcc、 Cpp、 Makeinfo、 Info-stnd、 Texi、 Hyperbole、 Octave

4. RET
5. 待文件名称输入完毕後，要按下 RET ，以告知系统文件名称已经 输入完成了，可以开始执行的动作了。

- 直接键入 ``RET"
- 如果在阅 某一个结点时，文件中出现以 ``* Menu:" 为首的 文字，就表示此列以下若有以 ``*" 为开头的列，均为一个 可以使用的 Menu Item 。此时，可以直接键入 ``RET" 参考 另一个结点的文件。

使用 Menu 有些条件，就是所要使用的 Menu，必需在此结点 的文件中有明列出来的才可以使用。要使用其它结点的 Menu， 必需先到有要使用的 Menu 的结点，才可以使用它的 Menu。 Menu 有其一定的结构。所有 Menu 的第一列，都是以 `` Menu:" 为开端。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 `` *" 为开始的列，就表示一个可以 使用的 item。 Menu Item 的表示法如下所述： * Info: (info). Documentation browsing system. 它主要由四个部份主成：

1. *
2. 每一个 Menu Item 都是以 ``*" 为开头。
3. Subtopic:
4. ``*" 之後就是 Subtopic 的名称，再加上一个 ``:"。 info 就是根据此 Subtopic 找到所对应的结点名。此例子的 Subtopic 的名称就是 ``Info"。 如果要以指令 ``m" 的方式， 使用Menu Item 所要给予的 Texinfo 的名称就是此 Subtopic 的名称。
5. node name
6. 在 Subtopic : 之後就是所使用的结点名。一般为了使用上 的方便，会尽可能的使用相同的 Subtopic 和结点的名称。 若二者的名字相同时，在 Subtopic 後的结点名会省略而以 两个 ``:"表示之。如 ``* Info::"。
7. node description
8. 结点後面的内容是选择性，它主要是用来概略描述此结点。

- l
- 如果想要回到前一个（last）所参考过的结点，可以使用指令 l 一步一步的往回走。
- b
- 指令 b 可以使游标移至文件的最前端。
- SPC
- 若文件太长可键入空白键（ SPC）来卷动萤幕。
- Ctrl-g
- 任何时候想要取消所键入的指令，可键入 Ctrl-g

回主画面

Emacs 的整合环境

前面不断的强调，EMACS 不只提供一个编辑的环境，而提供一个整合的工作环境。所以，在未进入本文的正题－ EMACS 的编辑环境之前，先对 EMACS 的工作环境做一个简介。

EMACS 所能提供的工作环境如下

1. EMACS 可执行 Shell 的指令。
2.
3. EMACS 可做为 Directory Editor（Dired）。
4.
5. EMACS 可以编辑、编译及除错程式。
6.
7. EMACS 具有编辑其它 host 上档案的能力。
8.
9. EMACS 可以列印档案。
10.
11. EMACS 具有年历、日记的功能（Calendar、Diary）。
12.
13. EMACS 具有读 Man Page 的能力。
14.
15. EMACS 可以收发电子邮件（Mail、Rmail）。
16.
17. EMACS 可以阅读网路上的电子布告栏（GNUS）。
18.
19. EMACS 具有 version control 的功能
20.
21. EMACS 对於档案的处理，具有 Outline 的能力。
22.
23. EMACS 具有资料库的处理能力。
24.
25. EMACS 可以提供电子计算机的功能。
26.
27. EMACS 提供了娱乐的环境（game），让工作者可以暂时放下工作的压力。
28.

由以上的分析可知，只要进入 EMACS 的工作环境，就可以在
其下完成所有的工作。所以 EMACS 的理想，是离开 EMACS
就是离开电脑工作的时候，因为它的最终目标，就是要完成一个
以编辑器为轴心的作业系统。
EMACS 所提供的这些功能，都是先唤起代表此功能的
模式*mode*， EMACS 的模式，分成主要模式 *major mode*
与次要模式*minor mode*。每一次只能使用一个主要模式，
而且必须要的。但一个主要模式可以搭配一个以上的次要模式。
现在就来简述 EMACS 工作环境的功能，至於如何使用这些 EMACS
的工作环境，本文就不多做介绍。

如何在 Emacs 中执行 Shell 的指令

在 EMACS 中有两种执行 shell 指令的方法：一种是进入*shell command mode*， 另一种是进

入 ***shell mode***。二者都可以执行 **shell** 的指令， 其最大不同之处是，进入 ***shell mode*** 的状态，执行 **shell** 指令的同时，仍可以切换到其它模式处理别的工作，但如果使用 ***shell command mode***，就必须等指令执行完後才可以做其它的事。 使用 ***shell command mode*** 时，使用者在萤幕的最下方 输入欲执行的指令， **EMACS** 会开启一个名为 ``**Shell command output*'' 的视窗，将 **shell** 指令执行的结果显示在此视窗中.***shell mode*** 则是执行一个 **subshell**，其输入与输出都是透过同一个缓冲区，所以输入与输出是在同一个地方，它不似 ***shell command mode***， 指令输入与结果的显示在不同的地方。 ***shell command mode*** 又可以有两种模式，一种就是很单纯的执行一个 **shell** 的指令；另一种是对某一特定区域的资料执行 **shell** 的指令。 ***shell command mode*** 容许 执行後的结果，直接输入到目前所使用的工作区内。有了如此的功能，使用者可以很轻易的将 **shell** 指令执行的结果，直接放入适当的位置，而不需另外从事剪贴的工作。 要如何使用 **EMACS** 所提供的 **shell** 功能呢？以下是最基本 的方法，至於高阶的用法则请自行参考 **GNU EMACS**所提供的 ``**GNU Emacs Manual**'' 。

1. *shell command mode*
2.
    - ESC-!（shell-command）
    - 唤起 *shell command mode*。
    - ESC- （shell-command-on-region）
    - 针对某一特定区域执行shell command mode 的 shell 指令。
    - (特定区域，是指缓冲区的某一范围(region)而言，所以此指令只是 针对缓冲区的某一部分运作的资料，)
    - Ctrl-u ESC-! 与 Ctrl-u ESC- 在 ESC 前加上 Ctrl-u ，可以将 shell 指令执行的结果，输出到 游标所在的位置。
3. *shell mode*
4. ESC-x shell\indexESC-x shell 是唤起 *shell mode*的指令。

有关目录的编辑方法

Dired 是专门针对目录来运作的编辑功能。进入***Dired mode*** 後， **EMACS** 会根据使用者所指定的目录来列出其下的档案及次目录，此时可根据需要**EMACS** 对这些档案及次目录作些运作。 **EMACS** 所提供可操作 **Dired** 的种类如下：

1. 可阅读、编辑 Dired 所列举出来的档案
2. 操作 Dired 下的档案
3. (a)在 Dired 可以删除（delete）档案 此功能可以很容易的将EMACS 的备份档（其档名以 ~结尾）、 暂存档（档名在两个 # 中间）或具某一特殊档名模式的档案 删除。

    (b) 档案的拷贝

    (c) 档名的更新

    (d) 改变档案的 mode

    (e) 改变 gid、uid

    (f) 档案的列印

    (g) 档案的压缩、解压缩

(h) 载入、编绎 EMACS 的 LISP file

(i) 可产生 hard links 与 symbolic links

(j) 可将档名换成大写或小写的英文字母

4. 可在 Dired 中执行 shell 的指令
5. 可使用 UNIX 的 diff 指令比较档案间的异同
6. 可隐藏次目录
7. 可使用 find 的公用程式来寻找档案

以上所列举的就是**Dired的功能,有人说它类似 PC 上的 PCTOOLS，**
读者是否有相同的感觉呢？

进入 Dired 模式的方法很简单，只要键入
``ESC-x dired''即可。此时的 *minibuffer* 会显示出如下
的文字：

Dired (directory):

``:'' 之後是目前所在的目录，此时可以修改目录名。确定所要使用的目录，按下 RET 後，系统会另开启一个视窗来显示此目录下的所有档案。之後就可以对这些档案做运作。 Dired 所使用 的缓冲区是一个唯读（read-only）的缓冲区，所以 mode line 会出现二个 %% 来表示其为唯读的状态。如果要缓冲区的唯读状态改为可读，可以键入Ctrl-x Ctrl-q 的指令，将缓冲区的状态改变。运作此缓冲区有其特别的方法，因此对此缓冲区做编辑并无实质上的作用。以下就介绍运作 Dired 的方法：

- **在 Dired 中删除档案**
- **Dired 最基本的指令，就是将要删除的档案做上旗标（flag）之後， 再将有旗标的档案删除。**
  - **d**
  - **将游标移至所欲删除的档案列，键入 d。此列的最前方会出现 D，这就是删除的旗标。此时的游标会移至此列的下一列。**
  - **u**
  - **若想放弃已定好的旗标，可以键入 u 使萤幕上的 D 消失。**
  - **x**
  - **键入指令 d 只是将要删除的档案先做上旗标，并未真正执行 删除的动作。只有键入 x 才会将所有做上旗标的档案删除。 执行删除档案之前，会先询问是否真要删除的意见。此时如果 回答 ``yes''则执行删除的动作，若回答 ``no'' 不执行删除的动作， 但旗标依然存在著。**
- **在 Dired 中将多个档案同时做上旗标**
-
  - **-#**
  - **键入 #，系统会自动将所有的自动储存的档案（auto-save file） 做上删除的旗标。**
  - **~**
  - **键入 ~，系统会自动将所有的备份档（backup file）做上删除 的旗标。**
  - **%d regexp RET**
  - **将所有适合的 regular expression 档案做上删除的旗标。**
  - **所有做上旗标的档案要执行删除的动作，都必需执行指令 x。**
- **在 Dired 中访问档案**
-

- **f（dired-find-file）**
- 如果想要访问目前游标所在列的档案，只要在此列上键入 **f** 即可。 此时，档案的内容会显示在原先显示 **Dired** 缓冲区的视窗上。 使用此方法访问档案，就如同以 **Ctrl-x Ctrl-f** 访问档案一样。
- **o（dired-find-file-other-window）**
- 此方法也是用来访问档案，但与键入 **f** 有些不同之处。键入 **o** 後，所访问的档案会出现在另一个视窗上而游标也会移至 所访问的视窗，显示 **Dired** 缓冲区的视窗并未消失在萤幕上。
- **Ctrl-o（dired-display-file）**
- 此方法与键入 **o** 雷同，二者不同之处在於键入 **Ctrl-o** 後所访问的档案会出现在另一个视窗上，但游标不会移至所访问 档案的视窗，依然留在显示 **Dired** 缓冲区的视窗上。
- **v（dired-view-file）**
- 此指令仅供流 档案之用，因为以此而开启的档案是唯读档案。

- 将 **Dired** 的档案做上标记
  - **m（dired-mark）**
  - 将目前游标所在的档案做上标记 ``*''。如果给予数值引数， 则做上标记的档案数目会依所给予的数目而定。
  - ***（dired-mark-exectables）**
  - 将所有的可执行档（**executable files**）做上标记 ``*，若给予 数值引数。则会将所有做上标记的可执行档的标记取消（**unmark**）。
  - **@（dired-mark-symlinks）**
  - 将所有的 **symbolic files** 做上标记 ``*，若给予数值引数。则会将 所有做上标记的 **symbolic files** 档的标记取消**(unmark**）。
  - **/（dired-mark-directories）**
  - 将所有为目录的档名，但除了 ``.'' 与 ``..'' 之外， 均做上标记 ``* 。若给予数值引数，则会将所有做上标记的 目录名称的标记取消（**unmark**）。
  - **ESC-DEL *markchar*（dired-unmark-all-files）**
  - 消除所有以字元（**character**）（ **markchar**） 做为标记的记号。 如果给予数值引号，则在消除每一个记号时，会询问是否要 消除记号。 回答 ``**y**'' 则表示要将记号消除，回答 ``**n**'' 则表示 不要消除已做好的记号。若此时键入 **!**，则表示消除其余的记号 不再询问意见。
  - ***c old new*（dired-change-marks）**
  - 使用此指令，可将原本以 **old** 为标记的记号，换成以 **new** 为标记的记号。
  - **%m *regexp* RET（dired-mark-files-regexp）**
  - 可使用***regular expression*** ，将具有某一类型的档案做上标记。
- 在 **Dired** 中的运作方式，有几件事情必需注意：
-
  1. 如果给予指令数值引数 **n** 时，此时指令所运作的档案是从 目前游标所在的档案起往後算 **n** 个档案（包括游标所在的档案）。 如果给予负数的数值，则往游标所在处之前算 **n** 个档案（包括 游标所在的档案）。
  2. 如果不给予任何的数值引数，则指令的运作范围会以做了 标记的档案为主。
  3.
  4. 如果不给予数值引数也不对任何档案做标记，则指令只对 目前游标所在列的档案运作。
  5.
  6. 所有运作 **Dired** 缓冲区的指令都是大写的英文字母，所有的 指令都是使用 **minibuffer** 来接收所需的讯息。
  7.

以下就是运作 **Dired** 缓冲区的指令：

- C new RET （dired-do-copy）
- 拷贝档案。若有多个档案同时要拷贝，则引数 new 代表 档案所要拷贝到的目录。若只拷贝一个档案，可利用此引数 new 将档案从新命名。
- R new RET （dired-do-rename）
- 更换档名。若有多个档案同时要换档名，则引数 new 代表档案 换名称後所要放置的目录。若只有一个档案，此引数 new 代表更换的档名。当档名更换完毕，Dired 缓冲区的档案名称 会自动跟著更换。
- H new RET （dired-do-hardlink）
- 将档案标上 hard links 的标记。引数 new 代表 hard links所要连接 的目录。若只有一个连接时，此引数 new 代表连结的名称。
- S new RET （dired-do-symlink）
- 将档案标上 symbolic links 的标记。引数 new 代表 symbolic links 所要连接的目录。若只有一个连接时，此引数 new 代表连结的名称。
- M midespec RET （dired-do-chmod）
- 更改特定档案的模式（mode，permission bits）。此程式使用 chmod 的程式，所以 式适用的引数。
- G newgroup RET （dired-do-chgrp）
- 改变特定档案的团体（group）为新的团体（ newgroup）。
- O newowner RET （dired-do-chown）
- 改变特定档案的拥有者（owner）为新的拥有者（ owner）。
- P command RET （dired-do-print）
- 列印特定的档案，可利用 minibuffer 输入列印的指令 command。
- Z （dired-do-compress）
- 压缩或反压缩特定的档案。如果档案已被压缩则将其反压缩， 反之则将档案压缩。
- L （dired-do-load）
- 载入特定的 EMACS Lisp 档案。
- B （dired-do-byte-compile）
- 位元编译（byte compile）特定的 EMACS Lisp 档案。

- Dired 在编辑远方档案与使用 ftp 的用法 使用 EMACS 的远方编辑或使用 EMACS 来做 ftp 时，若只 给予目录名，系统会进入 Dired 的模式。此时可使用运作 Dired 缓冲区的指令来操作所要的档案。至於何为远方的编辑会在下一节 中讨论。

如何编辑远方机器上的档案

EMACS 除了提供了一般编辑器所具有的功能之外，它还提供了一般编辑器所有的功能，那就是编辑远方 host 的档案。 EMACS 编辑远方 host 的档案，是使用了 ftp 的技巧，将所欲编辑 的档案 ftp 到目前的 host 上，待编辑完毕再以相同的技巧，把档案 ftp 传回远方的 host 而已。往昔要编辑远方的档案只有两种做法，一是签入（login）到档案所在的 host 去，另一种就是以 ftp 的方法将档案先传回目前所在的地方，修改完後再 ftp 回去。Remote Editing 也可以用到 ``anonymous ftp'' 上，它可以进入 远方的目录下，使用者即可根据需要挑选要 ftp 的档案。 使用 remote editing 的方法非常简单，在键入 `` Ctrl-x Ctrl-f'' 後， 再根据语法给予适当档名， EMACS 就会处理自行 ftp 远方 host 上的档案，其语法如下： Find file:/host:filename *host* 是指远方 host 的名称， filename 是指存放在远方 host 的档案。例如： Find file:/user1@gate.sinica.edu.tw :.login 就是编辑 host 为 ``gate.sinica.edu.tw'' 的机器，而使用者为 ``user1''，档案的名称为 ``.login'' 的档案。

程式的编辑,编译与测试

EMACS 是一个整合的环境，在提供程式编辑的同时，自然会提供一个可供程式执行的环境。以下就要谈谈 EMACS 可以为程式 撰写者提供那些服务。 EMACS 对於不同的语言提供不同的编辑模式。 EMACS 提供的服务有程式内缩的安排、括号对应的提示、程式注解的安排、游标移动的方式与程式的删除等等。基本上，EMACS 是提供一个 撰写程式的格式，只是此格式可根据使用者的需要而自行设计。 EMACS 选择适合的语言模式，是根据所编辑的档案名称附名来判断的。如附名为 .c 的 C 语言程式，EMACS 会自动给予 C 语言模式，而不需使用者自行处理。EMACS 提供的程式语言模式有 LISP、SCHEME、C、C++、FORTRAN、MAKEFILE、AWK、 PERL、ICON 与 MUDDLE 等。编辑好的程式可以直接进入 EMACS 的编译模式，不需离开 EMACS 到 UNIX 的 shell 下进行编译的动作。进入 EMACS 的 编译模式很简单，只要键入 ``ESC-x compile'' 即可。 EMACS 预设的编译指令是 make ，执行 ESC-x compile指令的结果如下所示：

compile command: make -k 若要使用其它的编译器，只需在 ``compile command :'' 的後面给予适当的编译指令即可，此指令与在 UNIX shell 下使用编译的 方法完全相同。 除了编辑、编译之外，程式撰写者还需要的功能是 Debugger 的提供。EMACS 也提供了此项的服务。EMACS 提供了四种 debugger，分别为 gdb、dbx、xdb 与 sdb，使用者可根据需来选择合适的 debugger。此处，只将使用 debugger 的指令列举如下：

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

如何在 Emacs 中列印文件

除了以上的功能外， EMACS 还提供了列印的功能。 EMACS 的列印可针对个整缓冲区或某部份的区域列印，其相关的指令如下：

1. ESC-x print-buffer
2. 列印整个缓冲区的内容。EMACS 处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。
3. ESC-x lpr-buffer
4. 此指令与上一个指令相似，只是不透过 pr 而直接使用 lpr。
5. ESC-x print-region
6. 与 ESC-x print-buffer 相似，唯一不同之处在於，此指令只列印 部份的区域。
7. ESC-x lpr-region
8. 与 ESC-x lpr-buffer 相似，唯一不同之处在於此，指令只列印 部份的区域。

在 Emacs 中如何收发信件

在 EMACS 众多的整合功能中，信件收发的功能自然是不可或缺的。 EMACS 对电子邮件的设计却与一般的电子邮件系统背道而驰，它是在以编辑为前提的条件下来提供电子邮件的子系统；换言之，电子邮件为以编辑器为主导的一个子功能。 EMACS 所提供的电子邮件系统分成二个部份，一部分为发送信件（ mail），另一部为收取信件（ rmail）；收取信件的同时也能发送信件，它的作法是使用发送信件的功能将信件发送出去。 EMACS 读取电子邮件，是将作业系统存放电子邮件的档案拷贝至 EMACS 自己的档案中，此档案名为

RMAIL， EMACS 在读取信件时会至此档（RMAIL）中读取所要的信件。 EMACS 如此设计档案的读取有它的理由，其理由如下：

1. 作业系统存放电子邮件的格式不一，读取电子邮件的软体 也是变化万千。EMACS 的 RMAIL 就是要将如此复杂的事情简单化。
2. RMAIL 记录了信件所有的相关资料，但作业系统所提供的 信件档案，并未有做如此详细的记录。
3. 一般作业系统为了要确保信件读取时的安全性，必需经由 一套繁杂的方法来保障信件读取的安全以及资料的不流失。 EMACS 的 RMAIL 所采取的措失就是既安全又简单。 RMAIL 的方法是，先把存於系统内的信件读出後，再拷贝至 RMAIL 的 档案内，待一切就绪才将存在系统内的信件删掉。如此作的好处 是，即使系统当掉只会使信件多做一份拷贝，也不会有流失信件 的事件发生。

   EMACS 对於发送信件与读取信件提供了许多有用的 操作功能，读者可自行参考相关手册，此处只告诉读者如何 进入送信件与读信件的模式：
   - ESC-x mail （发送信件）
   - ESC -x rmail (读取信件）

其它与 Emacs 相关的工作环境

EMACS 除了提供以上种种的工作环境境，还有其它的环境可以运用 ，现在让我们一一道来。

- The Calendar and the Diary
- EMACS 的 CalendAr与一般的月历功能相似 。 Diary 更可以适时提醒使用者该注意的事情。 进入 Calendar 的方法如下： ESC-x calendar Diary 的使用可以在进入 calendar 的模式中後，再来设定相关的资料。
- Reading Man Page from EMACS
- 阅读 manual page 的方法非常简单，其用法如下： ESC-x manual-entry RET unix-command-name RET
- Reading News with GNUS
- 读送网路新闻为现今交换电脑资讯的重要媒体与管道， GNU EMACS 也提供了此一子系统，称为 GNUS. EMACS 的 GNUS 是将 ``.newsrc'' 档的内容显示出来，它的内容 包括所有被订阅的(subscribe）的 newsgroups，以及未被阅读的文章。在 GNUS 中还可以看到或隐藏未被订阅的 newsgroups， 并可以再订阅未订阅的 newsgroup 或取消订阅某一个newsgroup。 当然 GNUS 也提供了可游走於各 newsgroups 间的指令。在 EMACS 中使用 GNUS这一个子系统只需键入如下的指令即可。 ESC-x gnus RET
- version control
- 管理原始档案（source files）也是EMACS 所提供了服务项目之一。 功能。 version control 是一个套装软体，它可以记录一个原始档案 (source file）所有改变的版本（multipe versions），它保留所有 改变的记录且存放於一个档案中，对於每一版本重复的部份只会 保留一分记录。version control 也会 记录每一版本被创造、谁创造 了它等等的相关资料。目前 EMACS 是透过 VC，来使用作业系统 所提供的 RCS 或 SCCS 的 version control 软体。若作业系统提供 RCS，EMACS 会先使用 RCS，若无 RCS 则会使用 SCCS。 EMACS 允许使用者自行决定使用 RCS 或 SCCS 的 version control。 透过 EMACS 的 VC，所能使用的 version control 的功能并不多， 它只提供最基本的 version control 的功能，但确是最常为人使用的 功能。若想使用 SCCS 或 RCS 所提供的所有功能，就必须进入 EMACS 的 shell mode 中了。以下就来简介 EMACS version control 的功能。EMACS 的 VC 提供的功能如下：

    - 将档案注册於 **version control** 之下。
    - 可将注册的档案从 **version control** 的控制中取出与放入。
    - 放入 **version control** 的每一个版本都可以随时取出。
    - 可比较任一版本间的异同。
    - 可将一组相关的档案，置於 **version control** 之下。
    - 可自行设计标头（**version header**），此标头可置於 **version control** 下的档案中。

  至於其它的 **version control** 功能，则必需进入 **EMACS** 的 **shell** 中直接使用 **RCS** 或 **SCCS**。例如，将数个版本合并、使用 **help** 协助使用 **version control** 等的功能，就从 **EMACS** 所提供的 **VC** 得到解答。

- **Outline Mode**
- 以 **editor** 为基石的 **EMACS** 当然少不了制作文章大纲的功能。 **EMACS** 的大纲模式（**outline mode**）可以使文章的部份主体 暂时隐藏起来，只呈现文章的大纲部份。如此一来，吾人不需 维护一套为本文，另一套为大纲部份的两套系统了。因为， 制作本文的同时就已经隐含了大纲的部份了。

前面已经将 **EMACS** 可以做的事以及它的特质都做了简要的介绍。 以下就开始讨论 **EMACS** 的基本要素 － 编辑功能。

回主画面

与 Emacs 有关的议题

上一章简介了 EMACS 的整合环境，在讨论编辑之前，先介绍 如何启动 EMACS 与离开 EMACS； EMACS 对於萤幕的安排如何；以及缓冲区与视窗在 EMACS 中角色定位等等的问题。EMACS 可用在 text-only 的终端机与 X window System 的视窗环境，但本文只针对 text-only 的终端环境来说明任何有关 EMACS 的介绍。

如何起动 Emacs

启动 EMACS 的方法非常简单，只要在 shell 的提示下键入 ``emacs'' 五个英文字就可以启动 EMACS 了。例如：

```
$ emacs RET
```

启动後的 EMACS 做了如下的启始（initialize）动作：

1. 清除目前的萤幕，开始一个全新的 EMACS 萤幕。
2. EMACS 会在这个全新的萤幕，显示一些与 EMACS 有关的 基本讯息。其中包括，目前使用的 EMACS 版本、基本的线上 辅助说明讯息以及有关 EMACS 版权的相关资讯等等。
3. 此时若不输入任何指令， EMACS 会在一段时间之後（约二分钟）自动将萤幕重新清除成一个空白的萤幕。
4.
5. 若在萤幕自动重新清除之前键入指令，EMACS 会根据 所给予的指令来做适当的运作。

   启动 EMACS 的方法，不需要给予任何的档名，只要 输入 emacs。因为 EMACS 是要建立一个能同时开启多个档案的 编辑环境；更进一步希望开启的档案，能彼此共享一些讯息。 所以，在键入 ``emacs'' 的同时，给予所要编辑的档名，就变得 不实际了。

如何离开 Emacs

知道如何启动 EMACS 後，接下来就要探讨如何离开 EMACS 了。 离开 EMACS 的方法有两种，一种是暂时离开 EMACS（suspending EMACS）， 另一种是永远离开 EMACS （killing EMACS)。其使用方法如下 （为求统一，以後的各章节都先列出 EMACS 的 Hotkey，其相对 应的命令则列举在小括号内，若无 Hotkey 则直接列出其命令。 要使用EMACS 的命令，通常要在每个命令前加上 Meta-x 或 ESC-x ）：

- Ctrl-z （suspend-emacs）
- 暂时离开 EMACS 回到其上一层的状态，一般是回到 shell 的状态。 若想回到 EMACS 的状态，只要键入 ''%emacs''，则可以回到 EMACS 了。
- Ctrl-x Ctrl-c（save-buffers-kill-emacs）
- 永久离开 EMACS。以此方法离开 EMACS，除了重新启动 EMACS 方外（即在 shell 的提示下键入 emacs），没有其它的方法可以 再回 EMACS 了 。

暂时离开 EMACS 意思是回到上一层的状态（parent process）， 一般是指 shell。使用者可以随时回到原先所启动的 EMACS 下， 对於所使用的缓冲区、kill ring以及 undo history等相关资讯， 仍保持与离开前相同的状态(有关 kill ring 、undo history 等相关 资料会在以後的各章陆续提及。以 Ctrl-z (suspend-emacs) 指令暂时离开的 EMACS ，可以在 shell 的提示下，以 ``%emacs'' 回到 离开前的 EMACS 下。有些系统或 shell 并不提供这种暂时离开的功能， 此时只能永远离开 EMACS 而无法暂时离开 EMACS 了。

要永远离开 EMACS 则必需键入 ``Ctrl-x Ctrl-c''(save-buffers-kill-emacs) 或 ``ESC-x save-buffers-kill-emacs'', EMACS 接收此指令後 会展开如下的动作：

1. EMACS 会主动提醒使用者，储存所有修改过的档案。
2.
3. 当使用者对需要储存的档案做了适当的处理後， EMACS 对於所有仍在执行的 subprocess，也会主动提醒使用者是否要结束 它们。因为离开 EMACS 的同时也就是结束这些 subprocess 的时侯。

在永久离开 EMACS 前， EMACS 会再三的提醒使用者 有关档案的储存与仍在执行的程式等等。因为，一旦永久离开 EMACS 之後，所有未存档或尚在执行的 subprocess 都会随之消失。 EMACS 对於所有未储存的档案与仍在进行的程序，会利用 *echo area* 一一提醒遗忘它们的使用者。 *echo area* 会提示需要储存的档案，同时也提供可处理这些档案的方法。 所以 *echo area* 除了显示要存档的档案名称外，还会 在档名之後出现如下的讯息： （y, n, !, ., q, C-r or C-h） 这些讯息提供，就是要让使用者对於档案或程序有适当处理的机会。 现在就对这些讯息做一讨论。

1. y
2. 同意对 *echo area* 所显示的缓冲区存档，并徵询对於 其它档案是否存档的意见。
3. n
4. 放弃对 *echo area* 所显示的缓冲区存档，但徵询对於其它 档案是否存档的意见。
5. !
6. 同意对 *echo area* 所显示的缓冲区存档，且对其它的 缓冲区也一并存档，不再徵询其它档案是否存档的意见。
7. .
8. 同意对 *echo area* 所显示的缓冲区存档，但对其它的 缓冲区则不再徵询是否存档的意见，直接放弃其它缓冲区的存档， 且离开此存档的状态。
9. q
10. 放弃存档的状态而不执行任何存档的动作。
11. C-r
12. 此指令可用来流　目前所要储存的档案内容，当离开此流　状态 即回复存档的模式，系统会再度询问与存档有关的讯息。
13. C-h

    对於以上的选项若有不明白的地方，可以此功能查阅其意思。

EMACS 的萤幕安排

在 text-only 的终端机启动 EMACS 时， EMACS 会占据整个萤幕， 此时的萤幕称为 *frame*。再一次的强调，本文只讨论 text-only 的 终端机，至於 X Window 的环境则不在讨论的行列中。

text-only 的 *frame* 又由数个 window 所组成。 启动 EMACS 时，会产生二个预设的视窗，一个视窗用来输入一般 的文件，在未有文件输入前先用来展示前面提到的 EMACS 版本、 线上辅助说明以及有关版权等讯息；另一个视窗用来输入指令 或是用做讯息的回应，称为 *minibuffer*或 *echo area*。

若终端机提供反白的功能，在反白区域以上的地方是用来输入 文件的视窗；反白区域以下的地方则是 *minibuffer* 或 *echo area*。 此反白的长条型则称为 *mode line*，它是用来描述输入文件视窗 的一些讯息。现在就来谈谈组成 EMACS *frame* 的这三个部份。

有关文字视窗的部份，因为还牵涉到缓冲区的问题， 现在先略过不谈，下一节再行讨论。现在先讨论与其有关的 *mode line*和位於 *mode line*下的*minibuffer* 或 *echo*

area。
mode line 出现在每一个文字视窗的最後一列，其描述此 视窗的相关资讯。 *mode line* 所描述的讯息如下: --ch-Emacs: buf （major minor） --pos--------------
现在则分别解释其所代表的意义。

- rl-h 代表缓冲区的状态（何谓缓冲区会在下一节讨论）。
-
    - -- 表示缓冲区未被修改过。
    -
    - ** 表示缓冲区已被修改过。
    -
    - %% 表示缓冲区为 read-only 的缓冲区。
    -
    - %* 表示 read-only 的缓冲区被修改过。
    -
- buf
- 表示此视窗缓冲区的名称，一般即为所编辑的档案名称。
- major minor
- 此缓冲区所有使用的模式（ mode）都列举在此括号内。其中 包括一个主要模式(major mode）和数个次要模式（ minor mode）。 EMACS 允许一个缓冲区有数个次要模式，但只能有一个主要模式。
- pos
- 表示文件在视窗显示的情形。其表示的种类如下：
    - All
    - 如果资料很少可以一「幕」了然，则会以 All 来表示。
    - Top
    - 若资料无法一「幕」了然，但出现的位置在最前面，则以 Top 来表示。
    - 无法一「幕」了然的资料，出现的位置是在最尾端，则以 Bot 来表示。
    -
    - nn%
    - 若资料出现的位置不在第前端也不在最後端，则以百分比 来表示资料出现的情形。

介绍完了 *mode line* ，现在来谈谈 *echo area* 与 *minibuffer*。在 *frame* 的最後一列，也就是 *mode line* 的下一列就是 *echo area*或 *minibuffer* 出现的地方。二者使用同一区位但所代表的意思却不相同。 Echoing 的意思就是将键入的字元在萤幕上回应出来。 EMACS 对於只有一个字元的指令并不会把它 Echoing 出来，例如 `` Ctrl-e''。 对於多个字元的指令，只要在键入指令的时候给予稍许的停顿， *echo area* 就会把键入的指令回应出来。等第一次的回应 产生时，再输入的部份就不需要再给予停顿的时间，其回应会在 键入的同时立即产生。 *echo area* 除了回应键入的指令， 也会将指令所产生的讯息显示出来；错误讯息的显示也是利用此区域。
minibuffer 所使用的地方与 *echo area* 相同。它本身也是一个视窗，是用来输入执行指令所需的引数（argument）。 使用 *minibuffer* 的同时也会使用 *echo area*。
minibuffer 输入引数的地方，是在 *echo area* 回应字串的 ``：'' 之後。因为 *echo area* 的回应是以 ``：'' 的出现做为结束。换言之， ``：''冒号之後就是 *minibuffer* 输入引数的地方。
例如，要访问一个档案，键入指令 ``Ctrl-x Ctrl-f'' 时， *echo area* 会出现 Find file: ``Find file'' 就是 *echo area* 的回应字，而此回应字串以 ``：'' 做为结束。 所以 ``：'' 之後，就是 *minibuffer* 的地盘了， 也就是 *minibuffer* 输入引数的地方。
使用 *minibuffer* 时，游标会自动移至 *minibuffer* 所在处，当游标在 *minibuffer* 时，就表示可以输入引数了。 若游标因为某些原因不出现在 *minibuffer* 的位置，此时可以 `` Ctrl-x o （other-window)使游标在视窗间移动，直到游标出现在 minibuffer 所在的视窗为止。 若已在 *minibuffer* 的状态， 但不想输入任何引数，此时可以 `` Ctrl-g（keyboard-quit）) 离开 *minibuffer*。输入 ``Ctrl-g'' 後，游

标会移至其它的视窗。
minibuffer 也是一个视窗，所以可以从别的视窗 移至此视窗； `` Ctrl-x o'' 的指令就是用来使游标在各个视窗间 移动的。一般的 *minibuffer* 都只有一列的高度，但有时 一列的高度无法将资料显示完毕，此时的 *minibuffer* 就需要 调整其大小了。至於如何将 *minibuffer* 的视窗做调整， 就是下一节所要讨论的重点之一了。

Emacs 的缓冲区与视窗

EMACS 的缓冲区与视窗的关系密不可分，缓冲区是用来存放 编辑文件的，但视窗却是用来显示缓冲区的文件。现在就来谈谈 缓冲区和视窗。 缓冲区（buffer）是 EMACS 编辑文件时，暂时存放文件的地方。 这个地方只用来暂时存放文件，要想永久保留这些文件，必需将 暂时存放的文件储存起来，一般是使用硬碟来安置缓冲区的文件。

在 EMACS 中所做的任何事情，都是先暂放於缓冲区内。 EMACS 处理档案的方式，也是先将档案从硬碟中取出後， 再放於缓冲区内。所以不论是删减、修改与新增文件，都是在 缓冲区内进行，除非将缓冲区内的文件存回硬碟，否则硬碟的内容 都不会因缓冲区内容的改变而改变。
文件未存回硬碟而离开 EMACS（kill EMACS），将永远消失 。但 EMACS 有一个自动储存文件的功能，称为 `` auto save'' 。 每当键入一定数量的字元（通常是三百个字元），EMACS 就会 自动做储存的动作；经过一段停置的时间（通常是三十秒），EMACS 也会做自动储存的动作。
EMACS 自动储存的功能并非将文件直接存回该档案 所在的硬碟中，而是将缓冲区的文件存入一个暂存档内。 只有以存档的指令，例如 `` Ctrl-x Ctrl-s'' 的指令，将缓冲区的 文件存回硬碟时，缓冲区内的文件才会存回硬碟中。只有当文件 存回硬碟中，EMACS 才会自动清除此暂存档。若缓冲区的内容 一直未存回硬碟，此暂存档就会一直存在著，直到存回硬碟才会消失。
EMACS 如此安排暂存档有两个好处，第一个好处是可以 确保编辑的档案资料不会流失；第二个好处是可预防机器意外关机 或当机，档案不及存回硬碟，所造成的损失。
EMACS 命名此暂存档 的方式，是以缓冲区所使用的档名为依据。在档名的前後各加上一个 ``#'' ，就是暂存档的名称。举例说明，若所编辑的档名为 ``emacs.doc'' ，其产生的暂存档即为： #emacs.doc# 若所编辑的档案未存回硬碟时， EMACS 会自动产生一个暂存档。 下次编辑此档时， EMACS 允予使用者从暂存档中将流失的资料回复。例如编辑的档案为 ``emacs.doc'' ，在离开 EMACS 时未存回硬碟 ， EMACS 会自动产生一个 ``#emacs.doc#'' 的自动储存档。当重新启动 EMACS 且编辑 ``emacs.doc'' 档时， EMACS 会提示使用者此档案 已被更改过但未给予适当的储存。此时，使用者可自行决定是否 要从自动储存的档案中（#emacs.doc#）将 ``emacs.doc'' 档中 未被储存的资料找回。
如何从自动储存档中将资料找回呢？想要从自动储存的档案中， 恢复原始档案中流失的资料，可以使用 `` Meta-x recovery-file'' 的指令。若存放於硬碟中的档案，有相对应的自动储存档时， 可经由如下的步骤将资料找回：

1. 键入 ``Ctrl-x Ctrl-f RET''
2. Find file: ~/ filename
3. 在 Find file: 处输入所欲编辑的档案後後， *echo area*
4. 会出现如下的讯息： Auto save file is newer: consider M-x recovery-file
5. 键入 ``Meta-x recovery-file RET''
6. 此时 echo area 会自动出现相对应的自动储存的档案名称， 若愿意执行恢复的动作，只要直接按下 RET 即可。否则， 以 ``Ctrl-g'' 指令，放弃此命令的执行。

除了暂存档外， EMACS 对於每一个编辑的档案， 都会在编辑前做一份备份，以防止在编辑的过程中因一时的疏忽 而将档案毁损。备份档的设计是，当档案被存回硬碟後，备份档 也不会因此而消失。
EMACS 命名备份档的方式，是在所要编辑的档名之後加上 ``~'' 。例如，

``emacs.doc'' 的备份档就为 ``emacs.doc~''。
以上的设定是可以改变，因为它们都是变数。下面 列出相关的变数，使用者可自行决定其所需。

- auto-save-visited-file-name
- 设定自动储存档案的种类。可以设为暂存档也可设为正在 使用的原档案。
- delete-auto-save-file
- 设定档案被存回硬碟後，自动储存的暂存档是否会自动删除。
- auto-save-interval
- 设定自动储存时的字元数。
- auto-save-timeout
- 设定自动储存时的时间。

想知道如何设定变数吗？在 EMACS 中任何设定变数的方法都是以 `` Meta-x set-variable（或 ESC-x set-variable）''的指令来完成 变数的设定。变数值的设定，可以只设定真假值或设定数值或是 设定字串。

若只是设定变数的肯定或否定值时， EMACS 有一个遵循 的规则。EMACS 中以任何 ``non-nil'' 的值来代表肯定，习惯上是以 ``t'' 来表示肯定；而以 ``nil'' 来代表否定。
在设定新的变数值之前，若想知道目前变数的值， EMACS 可以 `` Ctrl-h v''(describe-variable) 来查阅变数的值。 现在就举设定 ``auto-save-visited-file-name'',

``和 auto-save-interval'' 二个变数来说明变数设定的方法。

1. 以 `` Ctrl-h v'' 查阅 auto-save-visited-file-name 的变数。
2. 以 `` Meta-x set-variable'' 来设定变数 。
3. 再以 `` Ctrl-h v'' 来查阅所设定的 auto-save-visited-file-name 变数。

现在来看看设定 auto-save-visited-file-name 这一个变数的实际过程：

1. 键 ``Ctrl-h v RET''
2.
    1. echo area 处会出现 Describe variable:
    2. 在 Describe variable: 後键入 auto-save-visited-file-name
    3. 萤幕上会另开一个视窗，显示如下的讯息:
    4. 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.

3. 键入 ``Meta-x set-variable''
4.
    1. echo area 处会出现 Set varaible:
    2.
    3. 在 Set variable: 後键入 auto-save-visited-file-name RET
    4.
    5. echo area 处会出现 Set auto-save-visisted-file-name to value:
    6.
    7. 此时可以利用在 ``:'' 之後的 *minibuffer*, 输入变数 的值。此变数的值不是肯定就是否定的。目前的值是 nil，要改 其值为肯定的可以输入 t。
    8.
5. 再以 ``Ctrl-h v'' 来检视变数设定的情形。

上一个例子是设定肯定与否定值的例子，现在来看看设定变数值 为数字的例子。auto-save-interval 实际执行的过程：

1. 键入 ``Ctrl-h v RET''
2.
    1. echo area 处会出现 Describe variable:
    2. 在 Describe variable: 後键入 auto-save-interval
    3. 萤幕上会另开一个视窗，显示如下的讯息：
    4. 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.

3. 键入 ``Meta-x set-variable''
4.
    1. echo area 处，会出现 Set varaible :
    2. 在 Set variable: 後键入 auto-save-interval RET
    3. echo area 处会出现 Set auto-save-interval to value:
    4. 此时可以利用 ``:'' 之後的 *minibuffer*，输入变数 的值。此变数的值为数字。目前的值是 300，使用者可根据需要 输入适当的数字。
5. 再以 ``Ctrl-h v'' 来检视变数设定的情形。

在 EMACS 执行过程中所设定的变数值，只对目前所执行的 EMACS 有用，一旦离开此 EMACS，所有的设定就恢复成原来的 预设值。要想永久保留此设定的变数值，就必需将所设定的变数值 储存在档名为 ``.emacs'' (.emacs 档为 EMACS 的启始档，进入 EMACS 时会先执行此档内的指令， EMACS 的设定也是根处此档 而来的的档案中。因为启动 EMACS 时， EMACS 会先执行 .emacs 档，所有存於此档案的变数会被重新设定一次。

在 .emacs 档中设定变数 ``auto-save-visited-file-name'' 与 ``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 可以容许多个缓冲区的同时存在，既然如此，自然有其处理每个缓冲区的方法。现在就来看看 EMACS 如何 处理缓冲区。

- Ctrl-x b buffer RET （switch-to-buffer） 此指令用来选择不同的缓冲区，其预选的缓冲区是目前所使用 的缓冲区之外，最近被使用过的缓冲区。此指令可以使用 completion 。 使用此指令， echo area 会出现如下的讯息： Switch to buffer: (default filename) 若所要选择的缓冲区不是系统所预设的，可以利用 *minibuffer* 将所要选择的缓冲区名称键入。
- Ctrl-x k buffername RET （kill-buffer） 此指令是用来删除 *minibuffer* 所显示的缓冲区。若只键入 RET ，则删除目前的缓冲区，否则，删除所输入的缓冲区名称。 同样的，此指令可以使用 completion。
- Ctrl-x Ctrl-b （list-buffer） 将目前 EMACS 所使用过的缓冲区显示出来。以下的就是执行 ``Ctrl-x Ctrl-b'' 时，视窗所显示的资料：

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
```

以上资料的每一栏位各有其所代表的意思，详述如下：

- 栏位 MR，标记缓冲区的状态，其可能的状态如下所示：
  - ``*''
  - 表示此缓冲区被修改过。
  - ``.*''
  - ``.'' 表示此缓冲区为目前被选择的缓冲区， ``.*''表示此选用 的缓冲区被修改过。
  - ``%''
  - ``%'' 表示此缓冲区为 read-only 的缓冲区。
  - ``%*''
  - 表示此 read-only 的缓冲区被修改过。
- 栏位 Buffer，显示所使用的缓冲区名称。
-
  - Buffer 中的资料若为档案名称时，则表示缓冲区所放置 的资料为一个档案。
  -
  - 若 Buffer 中的资料前後加上了 ``*''，则表示此缓冲 区不是任何被访问的档案。
  -
- 栏位 Size，显示缓冲区的大小。
-
- 栏位 Mode，显示缓冲区所使用的主要模式。
-
- 栏位 File，表示所访问档案的绝对名称。若缓冲区的资料 不是来自访问的档案，亦即栏位 Buffer 的名字前後加上 ``*'' 时， 则以空白表示。
-

- Meta-x buffer-menu
- 此指令好似 Dired\indexdired 的功能，应用在缓冲区上。 此指令可对列出来的缓冲区各别做运作。其运作内容包括，储存 缓冲区、删除缓冲区、显示缓冲区以及编辑缓冲区等等。其实 运作於 `` Meta-x buffer-menu'' 的指令同样也可用在 `` Ctrl-x list-buffer'' 上， 只是使用 `` Meta-x buffer-menu'' 指令时， echo area 处会显示出 可运用的选项。其可运用的选项内容如下所示：Command: d, s, x, u; f, o, 1, 2, m, v; ~, %; q to quit; ? for help. 现举较常使用的选项说明，至於其余的选项，使用者可键入 ``?'' ，来使用其所提供的线上述助。
  - d
  - 标示所欲删除的缓冲区。 在 MR 栏位的最前方会出现 D。此时并未真正删除缓冲区， 只是将要删除的缓冲区做上标记，直到下达执行标记的命令时， 才会真正将标示 D 的缓冲区删除。此执行的指令为 `` x''。
  - s
  - 标示所欲储存的缓冲区。 在 MR 栏位处标示上 S 。此时并未真正做储存的动作，只是 在要储存的缓冲区做上标记，直到下达执行标记的命令时， 才会真正将标示 S 的缓冲区存档。
  - x
  - 对做好标记的缓冲区，下达执行的命令。也就是对标示有 D 与 S 的缓冲区，做执行的动作。
  - u
  - 将设好的标记取消。

- f
- 选择目前游标所在处的缓冲区。此时的视窗会将此缓冲区的内容 显示出来、

谈完了 EMACS 的缓冲区，现在来谈谈与其关系密切的视窗。 前已略述，进入 text-only 的 EMACS，即进入一个 frame。 一个 *frame* 由数个视窗组成，每一个视窗显示一个 EMACS 的缓冲区，且一次只显示一个缓冲区的内容。

EMACS 在任何时候，总有一个视窗为选择的视窗 （ selected window）。此视窗所显示的缓冲区，则称为目前的 缓冲区（ current buffer）.point（或称为游标） 所在的视窗，就是 EMACS 的选择视窗。EMACS 是透过游标来示 point 所在的位置。 所以说，若想要知道目前的选择视窗，观察游标所在的位置就可知道。
至於什麼是 point 呢？ point 就是用来标示目前 所使用的视窗或缓冲区所在的位置。EMACS 的每一个视窗， 各有其所专属的点位置（ point location）.每一个缓冲区 也有属於它自己的点位置。每一个缓冲区或视窗 point 的位置， 并不会随著视窗或缓冲区的改变而变动。换言之，point 的位置会 随时被记录下来，当再次访问其它的视窗或缓冲区时，游标仍会 回到离开前的位置。所以，任何移动 point 的指令，只会对所选择 的视窗产生影响，对於其它视窗的 point 是不会有任何影响的。 EMACS 下的每个视窗除了有各自的 point 外，也各自有其相对应的 *mode line*。
EMACS 视窗的大小是容许重新调整的。除了大小是 可以调整的，一个视窗也可以再分成两个视窗。其分割的方法， 可以做水平或垂直的化分。视窗的操作，除了分割视窗之外， 也可以使游标在不同的视窗间移动；当然，将不需要的视窗删除 也是基本的功能。现在就来看看与视窗相关的指令。

- Ctrl-x 2（split-window-vertically)
- 将一个视窗分成上下两个视窗。此时化分出来的两个视窗，分享 著化分前视窗的缓冲区。换言之，此时两个视窗的缓冲区内容是 一样的。因为共享著同一个缓冲区，所以改变其中一个视窗缓冲区 的内容，也会改变另一个视窗缓冲区的内容。分割成两个视窗的 好处之一是，可以编辑一个缓冲区，将另一个缓冲区作为参考的依据。
  - Ctrl-x ^ （enlarge-window） 将目前游标所在的视窗拉长一列。此指令，只有在 *frame* 存在一个以上的视窗时才有效果。若只有一个视窗，此视窗就 占据了整个 *frame*，此时自然就无多余的空间可以放大了。
  - Ctrl-u n Ctrl-x ^ （enlarge-window nn） 与 Ctrl-x ^ 指令相似。不同之处在於，此指令可以将目前游标 所在的视窗拉长 n 列以上。
- Ctrl-x 3（split-window-horizontally）
- 将视窗分成左右两个视窗。此时的两个视窗依然拥有相同的 缓冲区，所以，改变一个视窗缓冲区的内容，同时也会改变 另一个视窗缓冲区的内容。
  - Ctrl-x （enlarge-window-horizontally）
  - 将目前游标所在的视窗拉宽一行。对 *frame* 存有一个 以上的视窗才有效。若只有一个视窗，此视窗已经占据整个 *frame* 了，即使想放大，恐怕也爱莫能助了。
  - Ctrl-u n Ctrl-x （enlarge-window-horizontally nn ） 与 Ctrl-u 指令相似。此指令可以将目前游标所在的视窗 拉宽n 行。
- Using Other Windows and Deleting Windows
-
  - Ctrl-x o（other-window）
  - 此指令是用来选择所欲使用的视窗。注意，此 `` o’’ 是英文字的 `` o’’，而非数字的 ``0’’。
  - Ctrl-x 0（delete-window）
  - 将目前游标所在的视窗删除。提醒大家注意，此 `` 0’’ 是 阿拉伯数字的 `` 0’’。
  - Ctrl-x 1（delete-other-window）
  - 保留目前游标所在的视窗，其余的视窗全部删除。

视窗的大小是有限的，但缓冲区的内容却经常超过视窗 所能显示的范围。接下来就是要告诉各位，如何在有限的空间中， 以窥缓冲区的全貌。 想要在视窗的局限下，洞悉缓冲区的全貌，其最基本动作就是 卷动萤幕。所谓萤幕的卷动，就是萤幕上下左右的移动。除了 卷动萤幕之外，还需考虑萤幕的清除。在那些情形之下要清除 萤幕呢？例如，远方送来的 message， 在萤幕的显示久久不退； 系统送来的讯息也在萤幕上不会消失。这些情况的发生，都值得 将萤幕做清除且重新显示的动作。以下就来看看这些相关的指令。

- Ctrl-l（recenter）（清除萤幕）
- 清除萤幕且重新显示萤幕。
- 使萤幕上下卷动
-
    - Ctrl-v（scroll-up）（向上卷动萤幕一列）
    - 向上卷动萤幕，且将目前萤幕的最後二列做为卷动後萤幕的 前二列。 *point* 出现在萤幕的第一列。
    - Ctrl-u n Ctrl-v （向上卷动萤幕 n 列）
    - 萤幕向上卷动 n 列。 若指定卷动的列数 n ，不超过 point 在此萤幕上所在 的列数，卷动後的 point 仍会留在原处不动，否则 *point* 移至萤幕的第一列。
    - Meta-v（scroll-down）（向下卷动萤幕一列）
    - 向下卷动萤幕， 且将目前萤幕的前二列做为卷动後萤幕的後二列， point 出现在萤幕的最後一列。
    - Ctrl-u n Ctrl-v（向下卷动萤幕 n 列）
    - 萤幕向下卷动 n 列。 若向下卷动的列数 n，不超过 point 在此萤幕所在位置以下 的列数， 卷动後的 point 仍会留在原处不动，否则 *point* 移至萤幕的最後一列。
    - Meta < （beginning-of-buffer）（萤幕卷至缓冲区的最前端）
    - (beginning-of-buffer)将萤幕卷至缓冲区的最前端， point 也移至第一列。
    - Meta >（end-of-buffer）（萤幕卷至缓冲区的最尾端） 将萤幕卷至缓冲区的最尾端， *point* 也移至最後一列。
    - Ctrl-ESC-v（scroll-other-window）
    - 前面六个卷动萤幕的指令，都是针对游标所在位置的视窗而言， 若要卷动其它的视窗，则必需使用 `` Ctrl-ESC-v''。 `` Ctrl-ESC-v'' 指令是用来卷动游标所在位置下一个视窗的萤幕。 所以，如果开启了两个视窗，可以使用此指令来参考非游标所在 位置的缓冲区资料。此指令方便之处在於，可以省略移动游标 的步骤，就可以卷动其它的视窗。 如果今开启了 A 与 B 二个视窗，游标在视窗 A 处，以 `` Ctrl-ESC-v'' 可以卷动视窗 B。如果开启二个以上的视窗， `` Ctrl-ESC-v'' 指令所卷动的视窗，就必需看那一个视窗最 接近游标所在的视窗了。
- 使萤幕左右卷动
-
    - Ctrl-x <（scroll-left)
    - 萤幕向左卷动。
    - Ctrl-x >（scroll-right）
    - 萤幕向右卷动。
- 上下移动游标
-
    - Ctrl-p 或 ↑（previous-line）
    - ↑Ctrl-p 可以使游标向上移动一列。若有设定功能键，则可以用 来移动游标。
    - Ctrl-u n Ctrl-p
    - 使游标向上移动 n 列。
    - Ctrl-n 或 ↓next-line） Ctrl-n 可以使游标向下移动一列。若有设定功

能键，则可以用 ↓ 来移动游标。
- Ctrl-u n Ctrl-n
- 使游标向下移动 n 列。

回上页

Emacs 的基本编辑指令

前面的四个章节已经把 EMACS 的环境与架构做了扼要的介绍， 现在开始讨论 EMACS 做为编辑器（editor）所能 提供的服务。

如何载入档案与储存档案

使用编辑器最基本的需求，就是要能载入档案以便编辑。现在就先来看看 EMACS 是如何处理档案的载入。 EMACS 载入档案的方法很简单，只要在键入 "Ctrl-x Ctrl-f" 之後，再利用 *minibuffer* 输入所要编辑的档名即可。 档名的输入可以与 *completion* 相互搭配。 键入 ``Ctrl-x Ctrl-f'' 命令之後，在 *mininbuffer* 处输入 已存在硬碟的档案， EMACS 会执行如下的过程：

1. 产生一个新的缓冲区。
2. 将所欲编辑档案的内容，拷贝至缓冲区内。
3. 将缓冲区的内容显示出来，以便编辑。

这整个事件的过程，在 EMACS 的编辑系统中称为「访问档案」（ *visiting file* ）。
虽然载入档案的方法很简单，但仍有几件事情值得讨论的，
现说明如下：

- 键入 ``Ctrl-x Ctrl-f'' 後， *echo area* 会显示目前缓冲区 的目录，此时若编辑档案的目录与 *echo area* 所显示的目录 相同，可迳在其後输入档名即可。
- 若档案所在的目录与目前缓冲区的目录不同，输入档名的方法 有如下几种：
-
    - 档案在同一个机器上的做法：
    -
        - 无视 *echo area* 所显示的目录，由使用者重新输入档名。
        - 键入新档名的方法，是在 *echo area* 显示目录的最後方键入 ``/'' ,``/'' 代表， 忽略``/'' 之前面的路径，新的路径从 ``的路径 必须键入绝对路径（absolute file name）。 现举一实例说明。 Find file: ~/work/emacs/emacs.tex 所要编辑的档案在 ~/work/doc/text.tex 此时输入正确档名的做法如下所示： Find file:~/work/emacs/emacs.tex//~/work/doc/text.tex
        - 以 EMACS 提供的删除功能，将不必要的字删除之後，
        - 再输入正确的资料. 除非显示的目录与所欲编辑的目录相差无几，使用删除的方法 才有意义，否则不如放弃所显示目录，重新开始新的生涯才是正途。 至於如何删字，会在 5.2.5 一节中说明，此处就不赘言。

      所欲编辑的档案，不在目前所在的机器上的做法，如下所示： 前已述及 EMACS 可以 FTP 的方式来编辑远方的档案，但 EMACS 是如何以 FTP 的方法来编辑远方档案呢？想要编辑远方的档案， 只要给予正确的语法， EMACS 就会根据其语法来决定是否要使用 FTP 来编辑此档案了。其语法很简单，只有 /host:filename 而已。 如何来使用其语法呢？ 由其语法可知，其语法是由四组元素所组成的字串，包括二组子字串 （host 和 filename）以及二个符号（/ 和 :）。在 *minibuffer* 处 输入 / 後，紧接给予档案所在的机器名称（host），在其後立即给予 ``:'' ，在 ``:'' 後则输入所欲编辑档案，整个访问档案的过程就完成。注意此四组资料间，不可以留有任何的的空白。现举一实例子来说明之。 Find file:~/work/emacs.tex//hsko@gate.sinica.edu.tw:~/work/text.tex

    - 键入 ``Ctrl-x Ctrl-f'' 当 *echo area* 显示出目前缓冲区

- 的目录後，只给予 RET 而别无它物时， EMACS 以目前缓冲区 所使用的档案为预设档案。
- 键入 ``Ctrl-x Ctrl-f'' 後，突然改变心意，想知此目录以外 其它目录的档名，除了使用 ``Ctrl-x d''（使用目录的编辑指令－dired） 指令外，还可以利用现有的目录名称来得到想要的讯息。其作法 是直接修改 *echo area* 所显示的目录，直到所要的目录 出现後，键入 RET，此时会另开启一个视窗来显示出此目录下 的所有档名。若想操作这些档案，其操作方式与运作 Dired 的 方式相同。若对 Dired 的印象已经模 的人，请参阅 3.2 节 。
  以上是介绍档名的输入方式。在编辑资料的同时，是否 可以轻易查得目前所在的目录？" Meta-x pwd" 指令可以满足 这种需求。键入 " Meta-x pwd" 後，EMACS 会假借 *echo area*将目前所在的目录显示出来。
  以上谈的都是 EMACS 输入档案的方式，但 EMACS 又是如何为其缓冲区命名的呢？其实使用者并不需要为缓冲区命名， 因为 EMACS 会自动给予缓冲区合适的名称。EMACS 为缓冲区的 命名，可以从 *mode line*上得知。
  EMACS 命名缓冲区的方式是根据所键入的档案名称而来， 它舍弃了所有的目录名称只保留编辑档案的档名。所以若全名为 ``/user/work/emacs.tex'' 的档案，其缓冲区的名称则为 ``emacs.tex''。 除了将档案从硬碟直接载入缓冲区外，在编辑档案的同时， 有时需要参考其它的档案，甚至需要引进其它的档案到目前所使用 的缓冲区内。 EMACS 的 `` Ctrl-x i'' 指令，就是让使用者能随时 引进其它的档案到目前的缓冲区内。 ``Ctrl-x i'' 的 `` i'' 是 insert 的意思，相信了解意思後，对於该指令应有较深刻的印象，否则 怎有知已知彼，百战百胜之说呢？ 以下是 EMACS 有关档案载入指令的整理：
  - Ctrl-x Ctrl-f（find-file）
  - Ctrl-x i（insert-file）
  - Meta-x pwd

  文件编辑完後，最重要的事情就是要能将其保留下来。以下就 来谈谈文件储存的方法。档案储存不外乎将缓冲区的内容以原名 或易名存回磁碟；此储存方式可以选择一次存一个档案或一次 存数个档案；最後的考量是存完档案後是否要直接离开 EMACS。 下面就来看看EMACS 所提供的相关指令。

  - 存档但不离开 EMACS
  -
    - Ctrl-x Ctrl-s（save-buffer
    - 将目前缓冲区的内容，存回磁碟中。存回的档案名称与缓冲区 的名称相同 。此指令只对目前所使用的缓冲区做存档的动作。若缓冲区 的内容未有任何的变动则 *echo area* 会显示如下的文字： （No changes need to be saved）
    - Ctrl-x Ctrl-w（write-file）
    - 此指令与上一个指令相似，不同之处在於可以使用与缓冲区 不同的档案名称存档。。换言之，可另行指定存回磁碟的档名。 键入 ``Ctrl-x Ctrl-w'' 後， *echo area* 会显示出目前 缓冲区所在的目录， 此时，使用者可根据需要输入档名。若不输入任何档名只键入 RET ，系统仍会将缓冲区内的资料存回原先访问的档案中。此指令 也只对目前所使用的缓冲区做存档的动作。
    - Ctrl-x s（save-some-buffers）
    - 此指令可用来储存所有被修改过的缓冲区。使用此指令时， *echo area* 除了显示 档案名称外，还会在档名之後出现 （y, n, !, ., q, C-r or C-h） 这些讯息是提供给使用者做参考的。现在就告诉使用者这些 讯息所代表的意义。
      1. y
      2. 同意对 *echo area* 所显示的缓冲区存档，进一步 徵询其它档案是否存档的意见。

3. n
4. 放弃对 *echo area* 所显示的缓冲区存档，但徵询 其它档案是否存档的意见。
5. !
6. 同意对 *echo area* 所显示的缓冲区存档且一并 对其它的缓冲区存档，此时不再一一徵询其它档案是否 存档的意见。
7. .
8. 同意对 *echo area* 所显示的缓冲区存档，但放弃 对其它未存档的缓冲区存档，且直接离开此存档的状态。
9. q
10. 离开存档的状态而不执行任何存档的动作。
11. C-r
12. 可以此指令流　目前所要储存的档案内容，当离开此流　　状态即回复存档的模式，系统会再度询问与存档有关的讯息。
13. C-h
14. 对於以上的选项若有不明白的地方，可以此功能查阅其意思。

- 存档後直接离开 EMACS
- Ctrl-x Ctrl-c（save-buffers-kill-emacs） 此指令容许用者在决定是否将缓冲区的内容存档後，立即 离开 EMACS。此指令实际上是先执行 ``save-some-buufers'' 的动作再离开 EMACS。

讨论完了载入与储存档案的功能之後，相信已经迫不急待想要 知道如何编辑一份心目中想要的文件。现在就是介绍如何编辑 档案的时候了。

Emacs 的基础编辑指令

编辑器做些什麼事呢？现在先谈谈编辑器的基本功能，至於 编辑器的进阶功能则在下一章讨论。此处所谈的基本或进阶的功能， 所指的都是 EMACS 可以提供的功能，所区分的基本和进阶也只是 为了讨论上的方便而已。

所谓的编辑器就是用来编辑文件的器具。一张纸与一支笔， 就可以满足文件的编辑，这也是最简单且最原始的编辑器。 将纸笔产生文件的动作以电脑来代步，就是电子编辑器 （以下简称编辑器）主要的功能，也是本文所要讨论的编辑器。 文件的编辑不外乎文件的键入、游标的移动、文件的搬移 与删除以及文件资料的找寻与取代等等。现将 EMACS 可提供 的基本编辑功能简列如下，其详细的用法会在以下的各节中一一叙及。 EMACS 所能提供的基本编辑功能如下：

- 字元输入与显示在萤幕上的方式
-
    - 字元的显示方式，可使用插入法（insert）或覆盖（overwrite）的方法。
    - 除了一般的字元（ASCII Character）外，还可以显示特殊字元
    - （Special Character）以及任何八进位超过 200 的字元。
    - 文件在萤幕上所能显示的范围。
- *point*（cursor） 的移动方式（move *point*）
-
    - *point* 能左右移动一个或数个字元（character）
    -
    - *point* 能左右移动一个或数个字（word）
    -
    - *point* 能移至一列的开头或结尾
    -

    - *point* 能上下移动一列或数列（line）
    -
    - *point* 的设定，使游标能上下移动至指定的栏位
    -
    - *point* 的移动能以页为单位（page）
    -
    - *point* 能至萤幕的前端或尾端
    -
    - *point* 能移至缓冲区的前端或尾端
    -
- 显示缓冲区大小与 *point* 所在的位置
- 删除萤幕上所显示的文件
-
    - 向左或向右删除一个字元（character）
    -
    - 向左或向右删除一个字（word）
    -
    - 删除游标所在位置以後的所有文件
    -
    - 删除一个区块的文件（region）
    -
    - 删除的文件可以再使用（yank）
    -
- 文件的搬移与拷贝（move and copy）

编辑器具备以上的功能，就可以编辑出想要的文件。 现在就来看看如何在EMACS所提供的编辑环境，来编辑出 想要的文件。

在 Emacs 中如何加入与显示文件

EMACS 允许输入文件时，将资料直接输入在 *point* 所在位置 的正前方，此方法称为 insert mode；或将输入的资料以覆盖的 方式取代 *point* 所在位置的字元，此方法则称为 overwrite mode。 现举一实例来说明 insert mode 与 overwrite mode 的异同。

- 字串 food， *point* 所在位置为 d 处，此时的状态为 insert mode。
- 在 *point* 所在的 d 处键入 t ，原字串变为 footd。
- 字串 food， *point* 所在位置为 d 处，此时的状态为 overwrite mode。在 *point* 所在的 d 处键入 t ，原字串变为 foot。

EMACS 对输入模式的预设值是 insert mode，若想将模式转换成 overwrite mode， 指令 `` Meta-x overwrite-mode RET'' 可满足此一需求。若想恢复 insert mode，只需再使用一次 `` Meta-x overwrite-mode RET''就可以了 。当输入模式转为 overwrite mode 时，萤幕下方的 mode line 会显示 ``Ovwrt'' 的讯息，用以提示目前是使用 overwrite 的模式。 指令 `` Meta-x overwrite-mode''是用来转换 insert mode 与 overwrite mode

。在此前提下，原为 insert mode ，经转换 则为 overwrite mode。反之，若原为 overwrite mode 则转换成 insert mode。

EMACS 的 overwrite mode 只针对从键盘输入的文件有效， 若文件不是从键盘输入，而是以别的方式产生的，则一律失去 overwrite的效用。例如，拷贝而来的文件或以 `` Ctrl-x i'' 得来的文件， EMACS 一律使用insert mode。

除了 insert 与 overwrite 的显示方法外， EMACS 还允许 使用者输入一些从键盘上无法输入的字，那就是一些控制码 和八进位超过 200 的字元。要输入这些特殊的文字时，只要在 这些字的前方加上 `` Ctrl-q''即可。例如，要输入分页码 （formfeed ，ASCII Ctrl-L， octal code 014）， 则输入 ``Ctrl-q Ctrl-l'' 即可。此时萤幕会出现 C此时萤幕会出现 ^L 的符号。当输入文件的长度，超过 EMACS 视窗宽度所能显示 的范围，EMACS 对此情形的处理如下所示：

- 若文件太长需要换列时，在换列处键入 RET ，其後的文字 会自动转到下一列且以第一个栏位为新列的起始点。若其後没有 文件而键入 RET ，游标会会停在下一列的第一个栏位。
- 不理会文件是否会超过视窗的宽度，也就是不键入 RET 而 继续输入文字。EMACS 会自动在视窗的最後加上 ``\''，而将其余 的文字移至下一列；若下一列还是无法显示出所有的文字，会在 此列的最後再加上一个 ``^续将多余的文字移至下一列。 EMACS 就是不断重复如此的动作，直到所有的文字都能完全显示出来为止。 所代表的意思与键入RET 并不相同。键入 RET 表示重新使用 一个新列；不键入 RET 而令 EMACS 自动加入所产生的文件， 仍代表著同一列，只是这一列太长， EMACS 无法以其视窗的 宽度来一次穷尽，必须分为数次来表示。
- 不键入 RET ，也不使 EMACS 自动产生 ，而使超过萤幕宽度 的部份暂时隐藏起来。EMACS 处理这种情形，是在视窗的最後 加上一个 ``$''。 ``$'' 表示其後的内容在视窗上暂时看不到， 但仍安在缓冲区内。EMACS 的基本预设是自动加入 ``\''。要使 多余的文字隐藏起来，必须设定 ``truncate-lines'' 变数的值为正值。 变数设定的方法请参考 4.4 节。4.4 节曾谈过一个视窗可以分成左右二个小视窗，此视窗可以 做水平的卷动，此情形下的视窗在处理太长的列时，就是将 truncate-line 变数的值设成正值，使超过宽度的文件隐藏起来。

前面所谈的都是「文字」的插入方式，但如何插入一个「非文字」 的空白列呢？在编辑的过程中，若想在某列之前加入一个新列， 只需将游标移至此列的最前端 ，随後再按下 RET 即可。此时 EMACS 会在游标所在处的前一列， 加入一空白列。 EMACS 为何要将空白列加在游标之前而不是游标 之後呢？因为将空白列加在游标之前，有一个最大好处，就是可以 很轻易的在缓冲区的最前端加入一个空白列。 此时所键入的 ``RET''，代表著 newline 。若不键入 RET， 也可以使用 EMACS 所提供的 Hotkey ，也就是 Ctrl-j (tex-terminate-paragraph) 来获得新的一列。

---

*point*的移动

文字的键入及显示是编辑过程不可或缺的。除此之外，移动游标到 适当的位置，也是编辑过程不可缺的功能。现在就来看看在 EMACS 下如何移动 *point* 或可称为如何移动游标。移动 *point* 也就是移动 游标，因为 *point* 是透过游标来显示的。所以在本文会将游标与 *point* 交互使用。 游标移动不外乎以固定的单位，将其做上下左右的移动。此固定 的单位可能是字元（character）、字（word）、列（line）或页（page）。 现在就来看看如何将游标以这些单位来移动。

- 左右移动一或数个「字元」（character）
-
    - Ctrl-f（forward-char）
    - 游标往前（右）移动一个字元。
    - Ctrl-u n Ctrl-f
    - (Ctrl-u n Meta-x forward-char) 游标往前（右）移动 n 个字元。
    - Ctrl-b（backward-char）
    - 游标往回（左）移动一个字元。

    - Ctrl-u n Ctrl-b （Ctrl-u n Meta-x backward-char）
    - 游标往回（左）移动 n 个字元。
- 游标左右移动一或数个「字」（word）
-
    - Meta-f（forward-word）
    - 游标往前（右）移动一个字。
    - Ctrl-u n Meta-f （Ctrl-u n Meta-x forward-word）
    - 游标往前（右）移动 n 个字。
    - Meta-b （backward-word）
    - 游标往回（左）移动一个字。
    - Ctrl-u n Meta-b （Ctrl-u n Meta-x backward-word）
    - 游标往回（左）移动 n 个字。
- 游标移至一列的「最前端」或「最尾端」
-
    - Ctrl-a （beginning-of-line）
    - 游标移至一列的最前端。
    - Ctrl-e （end-of-line）
    - 游标移至一列的最尾端。
- 游标上下移动「一列」或「数列」（line）
-
    - Ctrl-n （next-line）游标向下移动一列。
    - 下移一列的游标其所在的水平位置，与移动前的水平位置 相同。若游标在文件的最後一列时， Ctrl-n 会自创新列再 移到此新列的第一个栏位。
    - Ctrl-u n Ctrl-n(Ctrl-u n Meta-x next-line)游标向下移动 n 列。
    - Ctrl-p （previous-line)游标向上移动一列。
    - 上移一列的游标其所在的水平位置，与移动前的水平位置相同。 若游标已在文件的第一列时，再使用 `` Ctrl-p’’ 时 *echo area* 会显示如下的讯息： Beginning of buffer
    - Ctrl-u n Ctrl-p （Ctrl-u n Meta-x previous-line） 游标向上移动 n 列。
- 上下卷动缓冲区
-
    - Ctrl-v（scroll-up）
    - 向上卷动缓冲区，且将目前视窗所显示的最後二列，做为 卷动後的前二列， *point* 出现在视窗上的第一列。 EMACS 如此做只是让使用者有一个承先启後的感觉罢了。
    - Ctrl-u n Ctrl-v
    - 缓冲区向上卷动 n 列。 数字 n 若小於目前 *point* 在此视窗所在位置的列数，卷动後的 *point* 位置，仍留在卷动前的同一列上；否则 *point* 移至视窗 的第一列。
    - Meta-v（scroll-down）
    - 向下卷动缓冲区，且将目前视窗上的前二列做为卷动後的後 二列，此时的 *point* 出现在萤幕的最後一列。
    - Ctrl-u n Ctrl-v
    - 缓冲区向下卷动 n 列， 若数字 n 小於*point* 在此视窗以下的列数，卷动後的 *point* 位置，仍留在卷动前的同一列处。否则 *point* 移至视窗的最後一列。
- 左右卷动缓冲区
    - Ctrl-x <（scroll-left） 缓冲区向左卷动。
    - Ctrl-x >（scroll-right） 缓冲区向右卷动。
- 游标移至缓冲区的前端尾端
-

    - Meta < (beginning-of-buffer)
    - 将游标移至缓冲区的最前端， *point* 也移至视窗的第一列。
    - Meta > (end-of-buffer)
    - 将游标移至缓冲区的最尾端， *point* 也移至视窗的最後一列。
- 页数的移动
- 还记得讨论 `` Ctrl-q'' (quoted-insert) 时，提过分页的控制码。 有了分页的设定，自然会有以页数做为移动游标的单位了。
    - Ctrl-x [(backward-page)
    - 将 *point* 移至上一页分页指标（^L）之後且紧邻 ^L。 若*point* 已紧邻在 ^L 之後了，则会略过此 F在 ^L 之後了， 则会略过此 ^L 到上一个 ^L 之後。
    - Ctrl-x] (forward-page)
    - 将 *point* 移至下一页分页指标（^L）之後且紧邻 ^L。 若*point* 已紧邻在 ^L 之後了，则会略过此 F在 ^L 之後了， 则会略过此 ^L 到下一个 ^L 之後。
    - Ctrl-x Ctrl-p (mark-page)
    - 以页为单位做上记号之後，再配合其它的指令对此页做处理。 此指令类似mark region（5.4 节会讨论其意思）。例如 Ctrl-x Ctrl-p Ctrl-w 是将游标所在的那一页删掉。此指令可以配合 numeric arguments（5.3 节会讨论其意思）使用。

除了以移动游标的方法来游走缓冲区外， EMACS 还提供一个可以直接将游标移到指定的行列。所要到达的目的地， 不论是以字元数或列数为移动的单位，其起算点都是以缓冲区 的第一列第一个栏位为起算的标准。以下就是此方法的介绍。

- Meta-x goto-char RET
- 键入此指令後再按下 RET ， *echo area* 会出现 ``游标移动的总字元数即可。再一次的提醒各位，字元的 计算是以缓冲区的第一列的第一个字元为起算点。例如， 在 ``Goto char:'' 给予 100 的数字，则游标会从缓冲区的 第一字元移动到第 100 个字元。
- Meta-x goto-line RET 键入此指令再按下 RET 後，*echo area* 会出现 ``， 在其後输入想要游标移动的列数即可。再一次的提醒各位， 列数的计算是以缓冲区的第一列为起算列。例如，在 ``Goto line:'' 给予 100 的数字，则游标会从缓冲区的第一列开始往下移动 10 列。

何谓数值引数

在上一小节 *point* 移动一文中不断的看到 Ctrl-u n ，它是什麽呢？ 它是 EMACS 的数值引数（numeric argument），但数值引数 又是什麽的呢？

数值引数适用於所有 EMACS 的指令。它的用途可分成 以下数类：

- 对某一指令做重复执行的动作
-
    - 用法： Ctrl-u n command 或
    - Meta- n command
    - 说明：
        - n 代表重复的次数； command 代表要重复执行的指令。
        - 若键盘提供 Meta 键（个人电脑可以使用 ESC 键），则 Meta 键是使用数值引数最方便的方法。
        - n 若为负值的数字，则表示以反方向运行指令。
    - 例子：
    -
        - n 为正值
        - Ctrl-u 5 Ctrl-f 或 Meta-5 Ctrl-f 表示游标往前（右）移动

5 个字元。
        - n 负值
        - Ctrl-u -5 Ctrl-f 或 Meta-5 Ctrl-f 表示游标往回（左）移动 5 个字元。
    - 重复动作四次
    - 用法：
    - 说明：
    -
        - Ctrl-u 後面所接的不是数字而是字元，它代表著一个 Ctrl-u 重复指令的动作为一个四的倍数。换言之，一个 Ctrl-u 执行四次，二个 Ctrl-u 则执行 16 次，以此类推。
    - 例子：
    -
        - Ctrl-u Ctrl-f
        - 表示游标往前移动 4 个字元。
        - Ctrl-u Ctrl-u Ctrl-f
        - 表示游标往前移动 16 个字元。
- 利用引数变数自动键入相同的「字元」
-
    - 用法： Ctrl-u n char 或
    - Meta- n char
    - 说明：省略 n 与否，代表不同的意义
    -
        - Ctrl-u 後紧接著数字与字元，表示重复字元 n 次。
        - Ctrl-u 後面不接数字而紧接字元，表示重复字元四次。
    - 例子：
        - Ctrl-u 10 r 或 Meta-10 r 则萤幕上出现 10 个 r。
        - Ctrl-u r 或 Meta-4 r 则萤幕上出现 4 个 r。
- 使用引数变数自动产生相同的「数字」的方法
-
    - 用法： Ctrl-u n Ctrl-u n 或 Meta- n Ctrl-u n
    - 说明： Ctrl-u 後面有二个 n ，第一个 n 表重复的次数， 第二个 n 表重复的数字。第二个 n 之前一定要再给予一次 Ctrl-u，若不给第二个 Ctrl-u ，系统会以为重复 nn 次。
    - 例子： Ctrl-u 5 Ctrl-u 4 或 Meta-5 Ctrl-u 4 萤幕上出现 5 个 4。 Ctrl-u 5 4 ，则系统以为重复 54 次。

数值引数的用途很广，且适用任何一个 EMACS 的指令， 当需要重复执行某一指令的动作时别忘了它。有人说电脑最强大 的功能之一，就是能不厌其烦的执行重复的事情。以下有关 EMACS 指令的介绍，不再特别强调 numeric arguments，但并不表示它就此而消失了，因为数值引数适合所有的指令， 所以没有必要每次都重复强调。只需记得，需要时可随时使用 numeric arguments。

如何得到与 *point* 有关的讯息

前面谈过 EMACS 移动 *point* 的方法，现在介绍探知 *point* 位置 的方法。*point* 所在的栏位、列数与页数等讯息，可从整个或部份 的缓冲区而得知。现在就开始讨论 *point* 位置。

- Meta-x what-page
- 告知 *point* 所在的页数与列数。若缓冲区没有以分页（^L）符号 分页，则 *point* 所在的页数永远为第一页。 若 *point* 在第一页的 第 200 列，其显示的讯息如下： Page 1, line 200

- Ctrl-x l （count-lines-page）
- `` Meta-x what-page’’ 用来告知 *point* 所在位置的页数与列数， 但无法从中得知此页的总列数。若想知道某一页的总列数， 必需靠 Ctrl-x l 来得知。 此指令除了得知某页的总列数外，同时还知道 *point* 所在位置之前与之後的尚有的列数。使用此指令时， *echo area* 会出现 如下的讯息：Page has 23 lines（20 + 4） *echo area* 出现的第一个数字为总列数，括号内的二个数字 分别代表*point* 所在处之前与之後的列数。括号内的两个数 是以*point* 为分界点 而得来的。所以当 *point* 不出现在某一列的第一个栏位时， 括号内二个列数的总和会比总列数多出一列，因为 *point* 所在的列 被重复计算了两次. 此指令在决定如何分页时可以帮上大忙， 因为可以轻易得知 *point* 前後的列数。
- Meta-x what-line
- 告知 *point* 在缓冲区的列数。若此时 *point* 在第 200 列， 则 *echo area* 会出现如下的讯息：
- Ctrl-x =
- 指出游标所在栏位的字元资料。这些资料包括字元的八进位码、 字元所在的位置占整个缓冲区的比例（此比例以字元为基本单位） 以及字元所在的栏位。例如以此指令来得知此 ``a’’ 字元的资料， *echo area* 会出现如下的资讯： Char: a (0141) *point*=23905 of 38784 (62) column 19 临时想知某一字元的八进位，也不妨试试此法。
- Meta-=
- 用来得知某特定区域（ region）的总列数与总字元数。 至於如何设定区域会在 5.3 节讨论。以此指令得知的结果如下所示： Region has 200 lines, 2000 characters
- Meta-x line-number-mode
- 以上所讨论的指令，只在使用指令时才会显示想要的讯息， 讯息的出现是无法长存的。 基於此，EMACS 提供了一个可使 讯息永久存在 line 的方法，那就是使用 `` Meta-x line-number-mode’’。 此指令可以在 mode line 上显示 *point* 所在的列数，直到离开 此状态或离开此视窗才会消失。若想使每次进入EMACS 都能 显示列数，最好的方法是在 ``.emacs’’ 档加上如下的叙述： (setq line-number-mode t) 如此一来，只要进入 EMACS ， mode line 会自动将列数显现出来。 至於自动显示栏位的方法，目前的 EMACS 尚未提供此服务。

文件的删减

修改文件不外乎将原有的文件删除後，再加入新的内容； 或将现有的资料做重新的排列组合。现先讨论文件的删除部份， 文件的重组就留待 5.4 节再进行讨论。

EMACS 中的删除有两种形式，一种是指文件的 *killing*， 另一种则指文件的 *deletion*。在 EMACS 中所谓的 *killing* 是指将文件从目前的缓冲区移到一个称为*kill-ring* *kill-ring* 为一个变数的地方去。文件在缓冲区中是消失了， 但却储存在 kill-ring ,variable这一个变数中。 EMACS 可以有许多的缓冲区，但却只有一个 *kill-ring* 的储存变数。 也就是说，多个缓冲区彼此共享一个 *kill-ring*，而且也只有一个 *kill-ring*。 EMACS 所设计共享的 killing-ring 的用意是让被遗弃的文件可以找回， 而且各缓冲区彼此也可借由 killing-ring 来建立一个互通的管道。 所谓的互通就是把甲缓冲区的东西给乙，反之，也可把乙缓冲区的 东西给甲。此模式在 EMACS 中就是透过 *killing-ring* 来完成。 因此，想从甲缓冲区中得到某些文件给乙缓冲区，只要将 甲缓冲区的文件放入 *kill-ring* 中，乙缓冲区就可以至此共享的 *kill-ring* 中将文件取出。如此一来，就可以共享资源了。

另一种模式的删除，在 EMACS 中称为 *deletion*。此种删除， 并不将删除後的资料放入 *kill-ring* 中，而是将删除的文件遗弃。 此种情形的删除是无法失物复得的。 以 *deletion* 删除的资料，可以键入

Ctrl-x u（undo）来找回。使用一次 `` Ctrl-x u，恢复前一个指令的景象； 使用二次 Ctrl-x u，则恢复前二个指令的容貌，如此周而复始 的使用 Ctrl-x u ，可恢复更改前的全貌。至於 `` Ctrl-x u'' 更详细的 说明会在 5.5 节进一步讨论。

前已说过，不放入 *kill-ring* 的指令称为 *deletion* 的指令。 EMACS 删除字元、空白字元以及空白列的指令都不放入 *kill-ring* 中的。具体说来就是，Ctrl-d、 DEL、 Meta-\、 Meta-SPC 与 Ctrl-x Ctrl-o 等指令。 现就为各位说明这些指令。
文件的修改不是重组资料就是删减与新增资料，新增资料已 论述过了，而重组资料即将在 5.4 节登場，现在就开始谈文件的删减 。删除文件也不外乎删除一个字元、一个字、一行或一个区块； 删除的方向可以选择左右删除的方式。现在就来看看如何以 EMACS 所提供的指令，来做删除的工作。此处将删减的指令分成 *deletion* 和 *killing*而大类。

- 属於 *deletion* 的指令集
-
    - 删除「字元」（character）的方法
    -
        - Ctrl-d（delete-char）
        - 删除 *point* 所在位置的字元。
        - DEL（delete-backward-char）
        - 删除 *point* 之前的字元。 此指令与 Ctrl-d 为最基本的删除指令，只要耐心够，任何 的删除都可以此二个指令完成。
    - 删除 spaces 和 tabs 的方法
    -
        - Meta-\（delete-horizontal-space）
        - 输入资料时，常会不自觉的输入无意的空白（space）和 tab 。当合并上下列而为一列时，也常会出现 space 或 tab 从中作梗。 为此 EMACS 提供了 Meta-\ 的指令，让使用者可以很容易将不必要的 space 和 tab 删除。当然其它的删除指令，也可以将不需要的 space 和 tab 删除，此指令只是更方便删除 space 和 tab 而已。 Meta-\ 可删除 *point* 前後所有的 space 和 tab 。 例如：abc def g，此时的游标在 f 与 g 之间。 键入了 Meta-\ 其结果如下所示： abc defg
        - Meta-SPC（just-one-space）
        - 删除 *point* 前後的 space 和 tab 时，若希望留下一个 space 或 tab 做为彼此的分隔，就必须使用 `` Meta-SPC'' 来完成了。 例如：abc def g，此时的游标在 f 与 g 之间。 键入了 `` Meta-SPC 其结果如下所示： abc def g
- 属於 *killing*的指令
-
    - 删除「字」（word）的方法
        - Meta-d（kill-word）
        - 此指令往前（右）删除 *point* 所在位置的字。 其所删除字的范围端赖 *point* 所在的位置而有不同。 若*point* 在一个字的第一个字元，则会删除此字；若 *point* 在此字的 其它位置，则删除 *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. 游标若在 test 这个字的 t 处，使用此指令的结果如下： This is a . 例（二）： This

is a test. 游标若在 test 这个字的 e 处，使用此指令的结果如下： This is a t. 若想删除一个以上的字，可以使用 numeric argument 。
        - Meta-DEL (backward-kill-word)
        - 此指令往回删除 *point* 所在位置的字。 其所删除字的范围端赖 *point* 所在的位置而有不同。 若 *point* 在一个字的第一个字元，则会删除此字之前的字； 若 *point* 在此字的其它位置，则删除 *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. 游标若在 test 这个字的 t 处，使用此指令的结果如下： This is test. 例（二）： This is a test. 游标若在 test 这个字的 e 处，使用此指令的结果如下： This is a est. 同样的，回想一下 numeric argument 吧。
    - 删除列的方法
    -
        - Ctrl-k (kill-line)
        - 删除列的指令。所删除列的范围，以 *point* 所在的位置为 准则， 以 *point* 所在位置为起始点，以此列的结束为终点。 被删除掉的文件会在视窗上留下一空白，若想将空白列也一并删除， 必需再使用一次 Ctrl-k。此时的 Ctrl-k 是用来删除换列指令 的控制码。
        - Ctrl-x Ctrl-o (delete-blank-lines)
        - 编辑的过程中会不经意的加入许多空白列，此种情形常发生在缓冲区的最尾端。每按下一个 RET ，就输入了一个新列，但在缓冲区 的尾端按下 RET ，此时所加入的新列并不易为人查觉。此指令可以 将*point*所在位置「前後」的空白列删除只留下一列。当然并非 一定要以此指令来删除空白， Ctrl-k 自然也可以用来删除空白列， 也可以将空白列以删除区块的方式删除之，提供此指令只是方便满足 使用者的需求而已。
    - 删除区块的方法
    -
        - *Ctrl-w (Kill-region) EMACS 允许删除某一特定的区块。要删除区块，必需先标示区块， 此标示的动作称为 mark\。如何标示区块,就是下一节的主题。

EMACS 对於被 *killing*而删除的资料，都是放在 *kill-ring* 中。 *kill-ring* 其实只是一个变数而已，所有 *killing*删除掉的文件， 就是此变数的值。其已述及查阅变数值的方法，现不厌其烦的再论述一次：

1. 键入 ``Ctrl-h v'' 後， *echo area* 处会出现：
2. 在 Describe variable:後输入变数的名称， EMACS 会另开一个 视窗来显示此变数的值。此时所要键入的变数名为 ``*kill-ring*''

此节讨论 EMACS 编辑的基本指令，有了这些指令之後，编辑的工作 就不是难事了。接下来各节则讨论其它的编辑功能。

何谓 Yanking

将 kill ring 的内容取出的动作，称为 *yank*。 *yank* 除了可 *yank* 最新 killing 的资料，也可 *yank* 早先 *killing*的内容。现在就以二种不同的 *yank* 做为讨论的对象。 将最新 *killing*的文件从 *kill-ring*中取出的方法 很简单，只要使用 `` Ctrl-y'' 即可。 但在 *yank* 时，一定要确保 在 *killing-ring* 中存有被删除的资料。 想要 *yank* 最新 *killing*之前的文件，就较为复杂了。所谓

的 较为复杂，只是多了一个移动指向 *kill-ring* 变数值的程序而已。 因为 *yank* 指令所要*yank* 的内容，全视指标指向 *kill-ring* 的位置而定。此指标一般都是指向最新放入的文件。若想 *yank* 其它的内容，就必需先移动指标了。

移动 *kill-ring* 指标的方法是使用 `` Meta-y'' (*yank*-pop)， 但在使用 `` Meta-y'' 之前，一定要先使用 `` Ctrl-y''。 换言之， `` Meta-y'' 的使用，一定要紧跟在 `` Ctrl-y'' 之後。
`` Meta-y'' 的运作是将指向 *kill-ring* 入口的指标，向前移动， 再将`` Ctrl-y'' *yank* 出来的资料 ，以此时指标指向的文件取而代之。 这就是为何使用 `` Meta-y'' 指令之前，一定要先执行 `` Ctrl-y''。 以 `` Meta-y'' 来移动指标，并不会影响 *kill-ring* 变数值的内容次序。

如何在文件中做上标记

前面有些指令是专门运作区块的，如 `` Ctrl-x Ctrl-p'' 与 `` Ctrl-w'' 等等。 但什麼是区块呢？ EMACS 所言的区块，是指从标记（ mark）处到 *point* 所在位置间的范围，此范围就是区块（ region)了。 换言之，区块的范围是指，标记所在位置之後（包括标记所在位置本身） 到 *point*之前（不包括 *point* 所在的位置）的所有文件。 知道区块的定义後，接下来就介绍如何定义区块了。

既然区块是指从标记处到 *point*的范围，自然设定区块也意味著 设定标记和 *point*。设定 *point*的方法很简单，只要移动 *point*至目的地即可。此时的 *point* 代表著区块范围的终点。 至於设定标记的方法也不难，只要在想要设定标记的地方，输入以下的 任一个指令即可。

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

所以设定区块的步骤如下所示：

1. 设定区块的始位置，也就是所谓的标记（ mark）。
2. 可使用 `` Ctrl-@'' 或 `` Ctrl-SPC'' 任一个指令来设定标记。
3. 设定区块的终结位置，也就是移动游标至区块的尾端。

在标记与 *point*之间的范围就是所谓的区块。

为何会提供二个设定标记的 Hotkey 呢？ Hotkey 是用来连结 EMACS 的命令，EMACS 会将常用的命令给予一个 Hotkey 与之连结。 此连结的Hotkey，常会受所使用的终端机而不同。因此有的终端机 可以使用 `` Ctrl-@''，但有的终端机则必需使用 `` Ctrl-SPC''。更甚者， 有的终端机却二者都无法使用，例如笔者以个人电脑模拟成的终端机， 却必需键入 `` Ctrl-2'' 才能达到设定标记的效果。 EMACS 可以允许使用者 重新设定所使用的 Hotkey ，但这已经超过本文讨论的范围。如果无法 使用以上任何一个 Hotkey ，就使用命令

`` ESC-x set-mark-command''。

使用 EMACS 的指令（ command）永远可以达到目的的。

EMACS 的 *point*，是透过视窗上的游标来显示的。 text-only 视窗只有一个游标，所无法同时表示标记及 *point*。 若想观测区块的来龙去脉，可以 `` Ctrl-x Ctrl-x''

（exchange-*point*-and-mark）

将标记与*point*做交换，从交换的过程可以观察标记与 *point*的位置。 `` Ctrl-x Ctrl-x'' 所能做的事，只互换标记与 *point*而已。

那些指令是使用在区块上呢？以下指令就是适用在区块上的。 有些是曾经提过的，有些是将要谈及的，有些可只能只是列举出来 并不会在本文中被讨论。

- 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

接下来讨论与编辑有切身关系的搬移与拷贝。

文件的移动与拷贝

变换文件在缓冲区出现的位置称为搬移（在缓冲区的其它地方重复出现 称为拷贝（ Copy）。

前已论及放於 *kill-ring* 的文件，可在需要时拿出来使用， 此动作称为 *yank*\index*yank* 。使用 *yank* 的方法 很简单，只要键入 ``Ctrl-x y''即可。使用此方法可将某特定区块的文件 做「搬移」的动作，只是所需的步骤比较烦琐而已。首先将要搬移的文件 重复以 `` Ctrl-x k'' 的指令，将其放入 *kill-ring* 中，再将 *point* 移至文件欲搬移的位置，以 `` Ctrl-y'' 将其 *yank* 出来。
「拷贝」一个区块的方法与「搬移」一个区块的方法累同，唯一 不同是要执行二次 *yank* 。第一次使用 *yank*，是将删除的 原文件再放回被删除的位置，第二次使用 *yank* 是做拷贝的动作， 也就是将 *kill-ring* 的文件放入要拷贝的地方。
以上的方法当然可以用来做搬移和拷贝之用，但如要搬移和拷贝的文件 有数十甚至数百数千列，岂不要重复 `` Ctrl-k'' 的动作数十甚至数百数千次。 因为 `` Ctrl-k'' 基本上是删除列的指令，当然删除列的指令，可以配合 numeric argument。但在使用数值引数之前还必需确定所欲搬移或拷贝的确实列数， 如此一来岂不使事情愈来愈复杂呢？所以，要搬移与拷贝文件最好的方法， 就是使用标记与区块。下面所谈的是针对区块的搬移与拷贝的方法。
不论搬移或拷贝，只要触及区块的使用，首要之事就是先把区块标示出。 区块定好之後，要搬移就以 `` Ctrl-w'', 将整个区块删除之後，再以 `` Ctrl-y'' 将其 *yank* 出来。要拷贝则以 `` Meta-w'' 将区块的内容 拷贝一份放在 *kill-ring* 中，尔後再以 `` Ctrl-y'' 将其 *yank* 出来。现说明搬移与拷贝的实际操作过程。

- 不设定区块，以删列的方式，将文件一列列先删除後,再 *yank* 出来。 此法可搬移与拷贝 *kill-ring* 内的文件。
-
    1. 重复执行 `` Ctrl-k''（删除文件，可使用 numeric argument）
    2. Ctrl-y（将放置在 kill ring 中的文件，*yank* 至缓冲区内。）
    3.
        - 若做搬移的动作，只需使用一次 *yank*。
        - 若做拷贝的动作，则要使用二次的 *yank*。
- 以设定区块的方式，「搬移」区块的文字。

- 1. Ctrl-@（设标记，也就是设定区块的起始值）
  2. 移动游标以便设定 *point*的位置（设定区块的终点）
  3. Ctrl-w（删除所标示的区块）
  4. 移动游标至要搬移的位置（确定文件搬移处）
  5. Ctrl-y（将放置在 kill ring 中的文件，*yank* 至缓冲区内。）
- 以设定区块的方式，「拷贝」区块的文字。
- 1. Ctrl-@（设标记 (mark)）
  2. 移动游标以设定 *point*的位置（设定区块的终点）
  3. Meta-w（不删除标示区块的内容，将此内容拷贝至 *kill-ring* 中）
  4. 移动游标至要拷贝的位置（确定文件拷贝处）
  5. Ctrl-y（将放置在 *kill-ring* 中的文件，
  6. *yank* 至缓冲区内。）

为了更清楚搬移与拷贝後的真实结果，现举实例来说明。

实例：现有二段文件，第一段文字是做搬移与拷贝用的， 第二段文字是用来接收搬移後的文字。 第一段：标记设在第一列的 A 处，而 *point*设在 B 的地方。
1111111111A22222222222 33333333333333333333 44444444444444444444
55555B5555555555555555 第二段： *point*在 Y 处 00000Y0000000000000000
00000000000000000000 00000000000000000000 结果一、将第一段的文件搬移至第二段，搬移後的游标仍在 Y 处。 1111111111B5555555555555555
00000A22222222222 33333333333333333333 44444444444444444444
55555Y0000000000000000 00000000000000000000 00000000000000000000 结果二、将第一段的文件拷贝至第二段，搬移後的游标仍在 Y 处。
1111111111A22222222222 33333333333333333333 44444444444444444444
55555B5555555555555555 00000A22222222222 33333333333333333333
44444444444444444444 55555Y0000000000000000 00000000000000000000
00000000000000000000 执行搬移与拷贝的动作时， EMACS 所采取的一律是 insert-mode。此时，即使设定为覆盖（ overwrite-mode）， 覆盖的效果也会暂时失效。
区块的范围以标记（mark）为起始值，以 *point*所在位置之前 的字元做结束。所以，从以上的实例可以很清础的看到，搬移或拷贝之後 的文件会包括标记所在的字元，但不会包含 *point*所在的字元。 搬移或拷贝的文件会出现在 *point*之前，原先所有的文件会向後移动， 此时的 *point*仍在移动前的字元处。
区块是以标记为开始而以 *point*为终点所构成的。标记与 *point*所在的列，可能包含所在列的一部分，其它区块所包含的列都是完整的 一列。若搬移或拷贝的范围，为文件中某一个长方形的区域，那该如何来设定 其范围呢？设定好的范围又该如何来操作呢？ EMACS 对於这种形状的区域有其它 的处理方式，称为长方形的区块 （ rectangle region）。
设定长方形（ Rectangle)的方法与设定区块（ Region）的方法是一样的 ，只是在理解上有所不同。长方形设定的方法，也是设定二个标记， 此两个标记位於长方形相对的两个直角上。所以，设定了左上角， 另一个标记一定要设在右下角；反之，如果设定了右上角，另一个标记就一定 要在左下角了。。此所设好的长方形， EMACS 称为 rectangle 。 长方形与区块的设定方法是一样，所以单从设定标记与 *point*的方式 ，是无法辨识二者的差别的。要知是使用区块或长方形，只有等到使用运作 於此范围的指令时才会知道。 若是以删除长方形的指令将长方形的区块删除时，此时删除掉的长方形文件 并不放在 一般的 *kill-ring* 中，而是放於别的地方。因为删除长方形的 运作方式与删除区块的方式不同，所以将长方形删除後的内容，与放置区块 删除後的内容分隔，以便管理。 长方形运作的种类与区块运作的种类累同，可分为二大类，一类 为删除与插入，另一类则专门用来处理空白。 删除长方形可以采取摒弃的方式 （ Ctrl-x r d）或将其储存於

某处（ Ctrl-x r k)，以利事後的 *yank*（ Ctrl-x r y）（*yank*-rectangle)。删除掉的长方形资料的保存，只能保留最新删除的资料，所以只有最新 删除掉的长方形资料，才可以被 *yank* 出来。
长方形区块的运作种类，除了能将资料做删除与搬移外，还可以 在文件的某处加入固定区域的空白。因为从事编辑时，常需要在某一 区域加入一些空白（ Ctrl-x r o）(open-rectangle)。 例如在文件加上空白做为边缘（margin）；制作图表时留些空白；或将 某一区域的文件以空白取代（ Meta-x clear-rectangle）等等。长方形 的运作与区块的运作最大不同处在於，无法直接做拷贝的动作，若要拷贝 必需先删除後再执行二次的 *yank*。现在就说明长方形区域的用法。

- 长方形区域的 *yank* 方法
  1. Ctrl-@（设定长方形区域的第一个对角）
  2. 移动 *point*至第二个对角处（设定长方区域的第二个对角）
  3. Ctrl-x r k（kill-rectangle）
  4. 删除设定好的长方形，以便 *yank* 使用。
  5. 移动 *point*到要 *yank* 的地方。
  6. Ctrl-x r y（*yank*-rectangle）
  7. 将删除的长方形，从储存处取出。
- 永远删除长方形的内容
-
  1. Ctrl-@（设定长方形区域的第一个对角）
  2. 移动 *point*至第二个对角处（设定长方区域的第二个对角）
  3. Ctrl-x r d（delete-rectangle）
  4. 删除设定好的长方形区域。此删除不会储存在某一定地方。换言之，一旦摒弃此区域，就无法将其唤回，除非使用 undo 的指令。
- 在长方形区域内插入空白，原文件向前（右）移动。
-
  1. Ctrl-@（设定长方形区域的第一个对角）
  2. 移动 *point*（设定长方区域第二个对角）
  3. Ctrl-x r o（open-rectangle）
  4. 填入空白在设定好的长方形区域内。此时区块内的文字会自动往右 移动。使用 overwrite mode，原有的文件也不会被加入的空白覆盖。
- 将设定的长方形区域以空白覆盖
-
  1. Ctrl-@（设定长方形区域的第一个对角）
  2. 移动 *point*至第二个对角处（设定长方区域第二个对角）
  3. Meta-x clear-rectangle
  4. 将此长方形区块内的文件以空白取代。
- 在设定的长方形区域内填充某一类型的字串。
- 此方法常可用在程式的撰写或测试上。例如，测试 ``.emacs'' 程式时，常因测试的过程中，需要将设定做增减的工夫， 使用此方法可以随时将测试条件做增减。
  1. Ctrl-@（设定要填充字串的起点）
  2. 移动 *point*至第二个对角处（设定要填充字串的终点）
  3. 键入 Meta-x string-rectangle RET ，此时 *echo area* 会出现
  4. String rectangle: 利用 *minibuffer* 键入要填充的字串，即可将字串填入长方形区内。 值得注意的是，此时长方形的宽度由字串的宽度来决定， Ctrl-@ 与 *point*只用来决定长方形的长度。

何谓 Undo

在 EMACS 中想要将已经做过的动作放弃，以恢复旧观。 EMACS 将其 称为 undo，以下就介绍如何 *undo*：

- Ctrl-x u（*undo*）
- Ctrl-_ （*undo*）

以上两个 Hotkey 都是用来执行 *undo* 的指令。 提供二个 Hotkey，是因为有些键盘并无明显使用 `` Ctrl-\_'' 的方法， 为了弥补无法以一个字元达到 *undo* 效果的键盘，故另行提供 `` Ctrl-x u'' 给无法使用 `` Ctrl-_'' 的使用者。
EMACS 所提供的 *undo*，可以连续恢复最近使用过的指令。 *undo* 的顺序是最新使用过的指令最先被 *undo* ，第二次使用 *undo* 则恢复第二新的指令，任何指令的输入（除了 *undo* 本身之外） 都会使指令输入的顺序 重整， 这也同时影响 *undo* 的顺序。
使用 *undo* 有一个限制，就是 *undo* 只能 *undo* 对缓冲区内容造成改变的指令。对於只是改变游标动作的指令，是无法以 *undo* 来恢复旧观。若所有修改过内容的指令，都以 *undo* 恢复原状後， 再一次使用 *undo* 的指令，*echo area* 会出现如下的讯息： no furter *undo* information 当使用了 *undo* 之後，还想要在 redo 这个已被 *undo* 的动作时， 有一个技巧可以达成如此的效果。

1. 首先键入一个不会改变缓冲区内容的指令（如游标移动的指令），使原来
2. 存放指令的顺序因新指令的加入而改变。
3. 再使用一次 *undo* 的指令，就可以达到 redo 的效果了。

现举一实例来说明此 > 视窗上现有的资料为： This is a test.

- 如下为所执行的一连串指令，括号内表示所用过的指令：
-

1. 执行四次 DEL ，视窗显示如下的讯息：
2. This is a 所使用的指令集如下： DEL DEL DEL DEL
3. 将游标至字元 ``i'' 处，视窗显示的讯息并无改变：
4. This is a 所使用的指令集如下： DEL DEL DEL DEL
5. 键入 Meta-d，视窗显示如下的讯息：
6. This a 所使用的指令集如下： DEL DEL DEL DEL Meta-d
7. 键入 Ctrl-x u，视窗显示如下的讯：
8. This is a 此时恢复最新被使用过指令 ``Meta-d'' 前的状况 （Ctrl-x u 不为恢复的对象），也就是步骤二的情形。 此时使用的过指令集如下： DEL DEL DEL DEL Meta-d Ctrl-x u
9. 键入 Ctrl-f，视窗仍出现与先前相同的讯息：
10. This is a 所使用的指令集则增加如下： DEL DEL DEL DEL Meta-d Ctrl-x u Ctrl-f
11. 此时可以 Ctrl-x u 恢复第一次 *undo* 前的状况， 即步骤三的状况。视窗出现：
12. This a 因为记录指令历史的指标，此时已在第二个 Ctrl-x u 处，但 Ctrl-x u Ctrl-f 均不在 *undo* 的行列中，所以此时可以再恢 复一次 Meta-d 前的状况。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，每一个使用过的编辑指令都有一个与之相对应的 *undo* 记录。
每一个 *undo* 的记录，都只对目前的缓冲区有效。有的指令需要一个
以上的 *undo* 记录来完成 *undo* 的动作；有的指令会先汇集一群
undo 的记录，当使用 *undo* 时，会将此汇集的结果一次展现出来。
例如，单一字元运作的指令，若每次使用 *undo* ，只恢复一个字元则
非常的不经济，所以遇到此种情形，以集合体的方式处理是比较合理的作法。

回主选单

Emacs 进阶编辑指令

前面的章节谈论 EMACS 的基本用法，现在讨论 EMACS 进阶 的用法。现在先谈 EMACS 的搜寻 *search* 与字串 *string* 的取代功能。

文件的搜寻

搜寻特定的字串，并非 EMACS 所特有的功能，相信大多数的编辑器都具有如此的功能。但 EMACS 所采取的搜寻方法是，每键入一个字元就展开搜寻，EMACS 称此种方式的搜寻为 *Incremental Search* 。 当然， EMACS 也提供非 Incremental Search}，称为 Nonincremental Search。 EMACS 对於所要搜寻的字串，仍是利用 minibuffer 来输入所欲搜寻的字串。 此时，输入 *minibuffer* 的搜寻字串，若全由小写的英文字母 （lower case）组成，则 EMACS 在展开搜寻的行动时，不论字母是否有大小写的差别，会将所有与 *minibuffer* 具有相同英文字母的字串都找寻出来。例如在 *minibuffer* 处输入 abc， EMACS 会找寻abc、 Abc、aBc、 abC、ABc、aBC、ABC 等字串。 所以在 *minibuffer* 处输入小写的英文字母，就表示所要找寻的字串包括大写的字母在内。 如何直接找寻上述例子的 ABC，而不需经过 abc、Abc、aBc、 abC、 ABc、ABC 等字串呢？欲达如此的效果，必需在 *minibuffer* 处，给予大写字母（upper case）的字串。例如，在 *minibuffer* 处，给予 ABC 的字串，此时，缓冲区的内容若为abc、 Abc、aBc、 abC、ABc、aBC、ABC ，则会直接搜寻 ABC 。所以在 *minibuffer* 处输入大写的英文字串时， EMACS 所找寻字串的大小写，就会与*minibuffer* 字串的大小写 完全一样。此种对大小写极度敏感的作法，称为 *case sensitive*。 EMACS 对於大小写出现的位置也有差别，现在就先来讨论 *case sensitive* 的问题。

- *minibuffer* 的字母都是小写，则可能找到的字串，包含 所有大小写的字串。 原始文件：abc、 Abc、aBc、 abC、ABc、aBC、ABC 输入 *minibuffer* 的字串： abc search string：abc、 Abc、aBc、 abC、ABc、aBC、ABC
- *minibuffer*中的字母，若有任一个以上的字母是大写，则可能 找到的字串，就如同 *minibuffer* 所示的一样。 原始文件：abc、 Abc、aBc、 abC、ABc、aBC、ABC
    - 输入 *minibuffer* 的字串： Abc search string：Abc
    - 输入 *minibuffer* 的字串： aBc search string：aBc
    - 输入 *minibuffer* 的字串： ABc search string：ABc
    - 输入 *minibuffer* 的字串： ABC search string：ABC

如果希望 *minibuffer* 输入什麼，缓冲区就找到什麼时，
例如，在 *minibuffer*中输入 abc，所要找寻的字串就是 abc。
此时就必需修改 EMACS ``case-fold-search" 变数的值为 ``nil"。
因为EMACS 对此变数的预设值是 t，它的意思就是使搜寻成为
case sensitive。若将此变数改成非 *case sensitive* 时，在
minibuffer} 输入 abc，就只会找寻 abc。

知道了 EMACS 对大小写的处理程序後，现在就来谈谈
*incremental search*和*noincremental search*。键入
第一个字母至*minibuffer* 时，搜寻的序幕就展开，是为
incremental search。
使用 *incremental search*时，当 *minibuffer*
收到第一个搜寻字母时，搜寻行动就从游标所在位置向下开始搜索。
此时的游标，会从原先游标所在位置移至其下第一个出现此字母的
地方；当 *minibuffer* 出现两个字元时，游标也移至其下出现
此两个字元的地方。当然，这些都必需要以缓冲区中有这些文字为前题，

若找不到任何合适的文字时，
echo area 会出现 ``Failing I-search:" 的警示语。
在 EMACS 中执行 *incremental search*的方法有两种，
一种是往前的搜寻（ forward search)，另一种是回头的搜寻
（ backward search。如下，就是二种搜寻所使用的指令：

- Ctrl-s （isearch-forward）
- 游标所在处向前（右）搜寻。
- Ctrl-r （isearch-backward）
- 游标所在处往回（左）搜寻。

当使用 Ctrl-s 或 Ctrl-r指令时， EMACS 的
echo area 会出现 ``I-search:"或 ``I-serach backward:"。当出现这些
提示字时，就表示其後是要输入搜寻字串。只要键入第一个字元，
搜寻的行动就立即展开了。 虽然如此，仍有几个有关搜寻的关键字
必需先行讨论，它们是 ``RET"、 ``DEL" 与``Ctrl-g"。

- RET
- 当 *minibuffer*收到 RET 时，游标会停留在最後一个满足搜寻 条件之处，且结束搜寻的动作。这种做法的好处是，若所找的资料 就是所要的，且又是必需修改的资料，游标停留在该处，即可立即处理。 因为，有些离开搜寻状态的方法，会使游标到第一次执行搜寻的位置， 这就是下面要谈的`` Ctrl-g" 的处理原则。
- Ctrl-g
- ``Ctrl-g" 的用法有二种，其一、是离开搜寻，承如上面所言； 其二、就是当所键入的搜寻字串，无法找到完全符合的对应字串时， 可利用 ``Ctrl-g" 将*minibuffer*中所能找到的子字串留下， 而将不能找到的资料除去。 所以，若已经找到所要的字串而想离开搜寻时，键入 ``Ctrl-g" 除了可以离开搜寻外，还可以将游标带回原先的出发点。根据 *minibuffer* 处所给予的资料，完全无误的在缓冲区中找到相对应 的文字後，以 ``Ctrl-g" 可将游标归回原位。但若所给予的资料在缓冲区 中无法找到完全对应的字，此时，若想离开搜寻的状态，就必需用「两次」 的 ``Ctrl-g" 才可完全脱离搜寻的状态。

  此时，键入的第一个 ``Ctrl-g" *minibuffer* 所找到的字串留下，而将 无法兑现的字串删除。此时再键入一次 ``Ctrl-g" ，则会完全离开搜寻的 状态，而游标也会自动移回展开搜寻时的位置。

- DEL
- 想要修改 *minibuffer* 内的字串时，请用DEL（delete-backward-char） 。 还记得它吗？它是用来删除字元的，此删除的动作是往回的删除。每执行 一次的DEL，就是一个新字串的诞生， 也就是一个新搜寻的展开。值得注意 的是，此新展开的搜寻与原先的搜寻方向相反，这个设计是合理的。 因为， 只要*minibuffer* 输入任何一个字，搜寻就已展开了，此时，若以DEL 来 删除错误的输入，则表示先前所找到的字串也不符需要，所以只能重新来过 才是可行之路。这也就是为何每删掉一个字元，搜寻的行动会回走的理由了。

要使用搜寻的指令，只需键入 ``Ctrl-s" 。此时*minibuffer*会出现
``I-search:"。若键入 Ctrl-r 後，则出现 ``I-search backward:"。

使用搜寻的指令，会有若干情形出现：

- 首次搜寻，即觅得所要的字串。

- 使用者此时可根据所需来做适当的处理。例如，离开搜寻，修改搜寻到 的资料等等。
- 搜寻不利，必需一而再，再而三的努力，才能找到所要的字串。
- 此时只要重复使用 `` Ctrl-s" ，游标就移至下一个符合的字串。所以， 只要所找寻的字串不变 就不需要再给予搜寻的字串。重复使用 `` Ctrl-s" 的结果，仍无法找到所要的字串，*echo area* 会回应出 ``Failing I-search:" 的讯息。
- 若重复使用 `` Ctrl-s" 的结果，已使 echo area 出现
- ``Failing I-search"後，则表示缓冲区已到了极限。若想再查看已阅 过的资料，只要再使用一次 `` Ctrl-s" ，就会迫使搜寻从原来的启始点 再进行一次的搜寻。
- 搜寻的过程中，若想回头再查阅已看过的资料时，只要交替使用 `` Ctrl-s" 与 `` Ctrl-r" ，就可以了。

EMACS 可以将搜寻过的字串，再拿出来重复使用。因为，EMACS
将使用过的搜寻字串，都放在一个名为 *search ring*的变数中。
它与先前讨论过的 kill-ring 类似，都为变数。既为变数，其值就可以
增减与参阅的。至於如何查阅变数的值，请再行参考 4.4 节
（EMACS buffer and windows）。

再使用搜寻过的字串，只要将 ``*serach-ring"*此一变数的
值取出即可。取出其值的方法有两种，一种是往前（右）的取出，
另一种是往回（左）的撷取，其分界点是以最新使用过的搜寻字串
为分野。因为 ``*search-ring"* 为一个ring，所以撷取的方式，
不是采顺时锺的方向，就是采反时锺的方向。今举一实例说明之。

以下是以 `` Ctrl-h v" 所得到有关 ``*search-ring"* 的资料。

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

Documentation:
List of search string sequences.

此时 `` search-ring" 的变数值是：

kill、 Ctrl、 tex、 Ctrl-h、 text 、 search
最新的搜寻字串则位於最前端，此时为 ``kill" 。往前（右）所得
的字串则为``Ctrl"，往回（左）所得的字串则为 ``search"。
今以此例，将使用`` search-ring" 得取使用过的搜寻资料的步骤
说明如下：

- 首先键入 Ctrl-s 或 Ctrl-r
-
    - ESC-p
    - 当 echo area 出现 I-search: 後，再键入``ESC-p"， 则可得到 ``Ctrl"的字串。若想继续往前寻找使用过的字串， 只要重复键入``ESC-p" 即可。
    - ESC-n\indexESC-n
    - 当 echo area 出现 I-search: 後，再键入ESC-n，则可得到 ``search" 的字串。若想继续往回寻找使用过的字串，只要重复 键入 ``ESC-n" 即可。

除了 search-ring 的资料可再使用外，将拷贝技巧运用在搜寻上，
也是资料再使用的另一项运用。

拷贝在搜寻上的运用，就是将缓冲区的资料拷贝至 echo area 出现

I-search: 後的 *minibuffer* 处。经此拷贝的过程，任何大小
的字串，都可使它轻易的出现在 *minibuffer* 处。

以下就是使用的方法：

- 键入 Ctrl-y
- 若要将游标所在位置到列尾的所有资料，放入 *minibuffer* 处， `` Ctrl-y" 可完成此一任务。
- 键入 Ctrl-w
- 若只想拷贝游标所在位置之後的字（word），使用 `` Ctrl-w" 是一个很好的选择。 `` Ctrl-w" 是拷贝缓冲区上的一个字（word） 到 *minibuffer* 。其拷贝的范围是以游标所在地为准则， 若游标位於一字的第一个字元时，则拷贝整个的字到*minibuffer* 处， 若游标所在位置不在字首，则将游标之後的次字串拷贝至 *minibuffer* 处。
- 键入 ESC-y(yank-pop)
- 向 *kill-ring* 借资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" 将原先放於*kill-ring* 的资料拷贝於*minibuffer* 处。可惜的是，此方法只能将最新被删除 的资料来出用。

到此为止，所谈的搜寻都是 *incremental search*的搜寻，现在来谈谈
nonincremental search。 nonincremental search
是一般编辑器处理搜寻最常用的方法，所以又将其称为传统的用法。

non*incremental search*的使用，必需从 *incremental search*开始。
当以 `` Ctrl-s" 或 `` **Ctrl-r" 启动 incremental search後，待 *echo area***
出现 ``I-search:" 或``I-search backward:" 後，只键入 RET 而不给予
任何其它的字串，此时就启动了 *nonincremental search.*当然*echo area*
出现的讯息会不一样，此时 echo area 出现的讯息 ``Search:" 或
``Search backward:"。所以要使用 *incremental search*或
***nonincremental search*，完全取决於键入 *minibuffer* 的内容**
而定。若为只键入 RET ，就是选择 *nonincremental serach*。

在启动了 *nonincremental search*之後，在 ``Search:" 或
``Search backward:"之後键入 ``Ctrl-w"，则表示要执行字的搜寻
（Words Search）。此功能可以用来搜寻一组的字，只要在 *minibuffer*
中将所要搜寻字群以一个空白隔开，就表示所要搜寻的是一个
字群了。以下将 nonincremental serach 的相关指令整理如下：

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

还有一种搜寻不论是 *incremental search*或 *nonincremental search*
都有的，那就是 *regular expression*，简称为*Regexp*。
所谓的 regular expression就是以最少的字元组合来表示最大可能的巨集。
现先不讨论如何来表达 *regular expression* ，只讨论如何使用
***regular expression* 的搜寻。**

因为有关 ***regular expression*** 会有专节（**6.3** 节）来讨论。

- **Regexp** 的 **incremental search**
-
    - **ESC-Ctrl-s**（**isearch-forward-regexp)**
    - 使用***Regexp*** 的搜寻方法： 先按下 ``**ESC**'' 之後再将其放掉，随後再按下 ``**Ctrl-s**''， ***echo area*** 处就会出现 **Regexp I-search:** 这时就可以输入 ***Regexp*** 的表示字串了。 当然还有若干事与 ***Regexp*** 的搜寻有关，讨论如下：
        - 若要继续以 **Regexp** 的方法搜寻，只需继续键入 ``**Ctrl-s**'' 就可找到下一笔相关的资料。
        - ***Regexp*** 的搜寻，也有其独立的 ***search-ring***。 这也意谓著可以再使用 ***Regexp*** 的 ***search ring*** 。 其使用法与前所讨论的方法一样，就是以 ``**ESC-p**'' 与 ``**ESC-n**'' 来再使用***Regexp*** 的 ***search-ring***。
    - **ESC-Ctrl-r**（**isearch-backward-regexp**）
    - 往回（左）的 **Regexp** 搜寻，若想继续的往回找寻想要的字串， 只需键入 ``**Ctrl-r**'' 即可。当然，也可以 ``**ESC-p**'' 与 ``**ESC-n**'' 来再使用其 **search-ring**。
- ***nonincremental search***
- 以下是以 ***Regexp*** 来执行 ***nonincremental search***的方法。
-
    - **ESC-x re-search-forward**
    - **ESC-x re-search-backward**

搜寻缓冲区的内容，除了要参考其内容外，有很大的机率是希望
能将找到的内容以它种内容取代。取代（ **Replacement**）的方法，
就是下一节讨论的主题。

文件的取代

从事编辑工作的时候，常会为了某种需求而将某一共同的用语以另外一种语辞替换，此情形最常发生在撰写程式的时候。当然，英文字串由大写改为小写或小写改为大写也是经常可见的。编辑文件在处理这些事时，就如如下的若干问题产生。

1. 所有要修改的文件，都能如愿以偿的得到适当的修改，不会 有漏网之鱼发生。
2. 修改过文件的一致性，是必要的条件。

因为有这些问题的考量，所以使用 **replacement** 来完成如是的工作，
就成为最佳的解决之道。现在就来谈谈 **EMACS** 如何处理 ***replacement***
的问题。

EMACS 处理 ***replacement*** 的方法有一气呵成的取代，
与选择性的取代二种。
所给予的被取代文字也有二种，
一、被取代的字串完全与所给予的字串一样；
二、以 **Regexp** 来做为取代的依据。试分述之。

- 二种的取代方法
-
    - 一气呵成的取代
    - 所谓一气呵成的取代，是一鼓作气将游标之後所有符合条件的 字串，全部以新字

串取而代之。 此作法不会一一徵求是否要取而代之的意见，而是自动、全盘 且无条件的取代。 此作法称为 **Unconditional Replacement**。
    - 选择性的取代
    - 选择性的取代，是会先徵求取代的意见，只有在取得同意权时， 才会采取取代的行动。这种取代的行为称为 ***query replacement***。
- 二种被取代字串的表示法
-
    - 被取代的字串完全与所给予的字串一样。
    - 此方法所要取代的文字，与 ***minibuffer*** 中所给予的文字一样。 所以，其可能符合条件的选择最多也只有一种。
    - ***Regexp*** 来做为取代的依据。
    - 此方法就是以 ***Regexp*** 来表示所要找寻的字串集。前已论及，所谓的 ***Regexp*** 就是以最少的字元组来表达最多的巨集。此时的取代， 就不是单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选择。如何表达正确的 ***Regexp*** 会在下一节中详细讨论。

以下就是使用取代（ **replacement**）的方法。

- **Unconditional Replace**
- 进行取代时不事先徵求意见，迳行将缓冲区中游标所在位置（包括 游标所在位置本身）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**
    - 此指令是将缓冲区中，所有出现 **string** 的字串以**newstring** 取代。 其详细的步骤如下：
        1. 键入 ``**ESC-x replace-string**''，当按下 **RET** 时，
        2. ***echo area*** 会出现

           Replace string:

           此时可利用 ``**Replace string:**'' 後的 ***minibuffer***， 将所要被取代的字串输入，按下 **RET** 则表示已完成输入的工作 。

        3. 按下 **RET** 时， ***echo area*** 会出现如下讯息：
        4. **Replace string string with:** 此时可利用 ***minibuffer*** 给予所要取代的新字串。键入 **RET** 时，游标所在位置之後的所有 **string** 都会转换成***newstring***。
        5. 当转换完成後， **echo area** 会出现 ``**done**''。此时就已 大功告成了。
    - **ESC-x replace-*Regexp* RET *Regexp* RET*newstring* RET** 此方法与上一个方法雷同，不同之处在於所要取代的资料不是 某一个特定的字串，而是某一巨集的字串组。从指令的表示法， 也可以看出所给予的被取代字串是 ``***Regexp***'' 而非 ``**string**''。 其详细的执行步骤与字串的取代雷同，只是在 **echo area** 将所有 的 **string** 换成 ***Regexp***罢了。
- **Query Replace**
- ***Query Replace***顾名思义就是在取代时会徵询取代的意见， 使用者可根据需要来取决是否要进行取代。以下就介绍字串和 ***Regexp*** 的取代。
    - **ESC-x query-replace RET string RET newstring RET**
    - 使用 ***query replace*** 的方法及步骤与 ***unconditional replace***的方法雷同，不同处只在於所引用的 指令有异，以及多增了询问的选择。其详细步骤如下 （斜体字表示使用者所输入的资料，粗体字表示系统自行根据 输入资料的回应）：
        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)**

5. 此步骤是 ***query replace*** 与 ***unconditional replace*** 最大不同之处。因为所有徵询的工作都是从此展开。 不知如何使用徵询的使用者可键入 ``**?**'' 来得到线上的求助。 以下就是键入 ``**?**'' 系统给予的资讯。 **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*** 时可有多种选择项。现只介绍常用的几个选项：

   1. **Space** 或 **y**
   2. 当决定以新的字串取代原来的字串时，以 **Space** 或 **y** 来表示。 执行 ***query replace*** 时，游标会移至下一个合适的字串处， 此时若决定将其取代，则键入 ``**Space**'' 或 ``**y**''。当取代完成後， 游标会自动移至下一个合适的字串处。当然也可以放弃所找到 的字串，这就是下一个要讨论的选项了。
   3. **Delete** 或 **n**
   4. 放弃字串的取代，使游标移至下一个目的地，是 **Delete** 或**n** 所做的事情。
   5. （**Period**）
   6. 若已找到合适的字串，而想终止所有进一步的取代行为时， 键入 ``**.**''可使目前游标所在处的字串以新的字串取代，并且 在取代後立即离开***query replace*** 的状态。
   7. **!**
   8. `` **!**'' 可使 ***query replace*** 恢复为***unconditional replace***。 因此，若想放弃询问的权利而恢复 ***unconditional replace*** 的 状态，键入 ``**!**'' 就可将游标所位置及其以後所有合适的字串， 都以新字串取而代之。
   9. **RET** 或 `` **q**''
   10. 若想就此离开 **query-replace** 而不再做进一步的取代动作， 只需按下 **RET** 或 ``**q**'' 即可。

- **ESC-x query-replace-*Regexp* RET *Regexp* RET newstring RET**
- 此方法与 `` **ESC-x query-replace RET string RET newstring RET**'' 相似，但此时所取代的不是特定的字串，而是某一字串的巨集。

字串大小写（ ***case sensitive***）的问题与取代也有很大
的相关性。因为大小写的问题，对於 ***unconditional replace*** 与
query replace 均适用。所以，只举 ***query-replace***
为例说明，至於 ***unconditional replace*** 就如法泡制。
当启动 `` **ESC-x query-replace RET string RET newstring RET**''

时， **string** 与 **newstring** 的大小写，关系不同字串的取代。
其规则如下所述：

- 当 **string** 与 **newstring** 都以小写的形式出现时，
- 取代工作的进行，就有三种情形：
  - 缓冲区原始字串的第一个字母是以小写为开端时，不论 此字串是否有其它的大写字母，经取代後全转换成小写。
  - 若原始字串的每个字母都是大写时，经取代後也维持大写的形式。
  - 若原始字串以大写为开端，不论此字串是否还有其它的字母 为大写，只要不是全为大写的情形，取代後只有字串的第一个 字母为大写，其余一律为小写。

  以下举一实例，供参考： ***string*** **：abc ，** ***newstring*****： xyz** 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xyz Xyz xyz xyz Xyz Xyz xyz XYZ**

- 当 ***newstring*** 部份为大写，而 ***string*** 依然维持小写 的情形：
-
  - 若原始字串的第一个字母为大写时，被取代後的第一个字母 仍维持大写的形式。其余原始字串的大小写，就视 ***newstring*** 的大小写而定，与原始字串本身的大小写无关。
  - ***newstring*** 的字母若以大写出现，则被取代的原始字串 也会在相对应的位置以大写的形式出现。
  - 若原始字串全为大写时，取代後仍维持大写的形式。

  以下为若干实例，供参考：

  - **string： abc， newstring： Xyz**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **Xyz Xyz Xyz Xyz Xyz Xyz Xyz XYZ**
  - **string： abc， newstring： xYz**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xYz XYz xYz xYz XYz XYz xYz XYZ**
  - **string： abc， newstring： xyZ**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xyZ XyZ xyZ xyZ XyZ XyZ xyZ XYZ**
  - **string： abc， newstring： XYz**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **XYz XYz XYz XYz XYz XYz XYz XYZ**
  - **string： abc， newstring： xYZ**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xYZ XYZ xYZ xYZ XYZ XYZ xYZ XYZ**
  - **string： abc， newstring： XYZ**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **XYZ XYZ XYZ XYZ XYZ XYZ XYZ XYZ**
- 只要 ***string*** 有大写的字母出现时，取代後字串的大小写， 就完全依照 **newstring** 的大小写。换言之， ***newstring*** 为大写的地方依旧为大写，为小写的地方依然为小写。 以下为若干实例，供参考：
  - **string： Abc， newstring： xyz**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xyz xyz xyz xyz xyz xyz xyz xyz**
  - **string： aBc， newstring： xyz**
  - 原始文件为： **abc Abc aBc abC ABc AbC aBC ABC** 经过取代为： **xyz xyz xyz xyz xyz xyz xyz xyz**

- **string： ABc， newstring： xyz**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为： xyz xyz xyz xyz xyz xyz xyz xyz**
- **string： ABC， newstring： xyz**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为： xyz xyz xyz xyz xyz xyz xyz xyz**
- **string： aBC， newstring： Xyz**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为： Xyz Xyz Xyz Xyz Xyz Xyz Xyz Xyz**
- **string： Abc， newstring： xYz**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为： xYz xYz xYz xYz xYz xYz xYz xYz**
- **string： aBc， newstring： XYz**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为：XYz XYz XYz XYz XYz XYz XYz XYz**
- **string： ABC， newstring： XYZ**
- **原始文件为： abc Abc aBc abC ABc AbC aBC ABC 经过取代为： XYZ XYZ XYZ XYZ XYZ XYZ XYZ XYZ**

讨论至此，取代的部份应该可以告一段落了。接下来，就是要将
一直未正式讨论的 ***Regular Expression*** 做一详尽的说明。

Regular Expression

所谓的 ***Regular Expression***（以下简称 ***Regexp***）是用来表达一连续字元的组合，或是用来描述字样（**pattern**）的一种方法，它在概念上不同於字串。 字串是指由一连续字元（**character**）所形成的字元组，一个字元组就只代表一个特定的字串，而 ***Regexp*** 通常所代表的是一组具有共通特性的字串集，此共通特性称为 ***pattern***。 ***Regexp*** 的精神，就是希望以最少的字元，来表达最大的巨集。所以 ***Regexp*** 所能表达的字串数应不只於一个，而是一个字串的集合。这也是字串与 ***Regexp*** 最大不同处。当然， 也可将字串视为 ***Regexp*** 的一个特例。 本文一直没有讨论撰写程式的问题，但不可否认的，程式撰写使用到 ***Regexp***的机会不胜数。举凡变数、常数的重新命名，同类字串的搜寻，数个程式合并後的整理工作等等，若能灵活使用 ***Regexp*** ，则可达事半功倍之效。这是一篇 讨论 **EMACS** 使用入门的文章，并不是讨论程式语言的文章，所以只是将 ***Regexp*** 的用法告知。

撰写程式常有的经验，就是要将数个函式合并後，必需将使用这些函式的地方，以合并後的新名称取代。例如 **read_i.c**、 **read_c.c** 与**read_f** 等三个函式合并成 **read.c** 时，所有使用这三个函式的地方，都必需改成 **read.c**。 此时，若以传统的修改方法，恐会漏一挂万，若采取代法，则必需执行三次的取代，才可以大功告成。但使用 ***Regexp*** ， 则可以一举成功。既方便又快速。又则，编辑文件或程式时，注解的功夫是不可或缺的。不同的文件与语言，各有其不同的注解表示法。例如 **LISP** 语言是以 ``;'' 表示注解， **LaTex** 则以``'' 为注解。有时为了将注解分类，极H不同数目的注解符号来做区分。此时遇到的问题是，当重新分类注解时，相对应的注解符号也要调整。以 **LISP**语言为例， 重新分类後，可能将拥有一个 ``;'' 及二个 ``;''的注解符号，以三个 ``;'' 注解符号取而代之，这时若使用***Regexp*** ，问题就可迎刃而解。

程式档名的转换也可以 ***Regexp*** 来轻松完成。因为不同的程式各有其不同的副名，例如 **C** 语言的副名为 ``.c'' ， **C++** 语言的副名为 ``.C'' ，而**FORTRAN** 语言则为 ``.f'' 等等。 此时，若想将 **FORTRAN** 语言所发展软体，转换成 **C** 语言，则必需将所有 ``.c'' 的档名做适当的调整。

例如，将所有``**filename.f**'' 的档案，改为 ``**filename.c**'' 的档名。 此时，若使用 ***Regexp*** ，可

能在极短的时间就可将所有的档名转换成功。 ***Regexp*** 的运用非常的广泛，这里所举的例子只是凤毛麟角。以下就开始简介 ***Regexp*** 的使用法，至於更详细的介绍，请参照另一篇 **94019** 的技术报告。

Regexp 并非是 **EMACS** 的专利，而是 **UNIX** 系统下 的产物。 **UNIX** 系统下的许多工具程式都使用 ***Regexp*** ， 虽然彼此在表达 ***Regexp*** 的方法上，略有差别，但在概念上却是一致的。 **UNIX** 系统上使用 ***Regexp*** 的工具，除了**EMACS** 外，尚有**ed** 、**ex** 、**vi** 、**sed** 、**awk** 、**grep** 、**egrep** 等等。 这里所介绍的 ***Regexp***，在概念上与其它地方是共通的， 但在实际的运作上可能会有所出入。

Regexp 是由字元所组成，此字元分为一般字元与特殊字元两种。一般字元所组成的 ***Regexp*** ，是最简单的 ***Regexp*** 的表示法。因为它所要表达的字串与 ***Regexp*** 完全一模一样 。由特殊字元所组成的 ***Regexp***就较为复杂了。因为不同的特殊 字元，各有特殊的代表意义。**EMACS** 中***Regexp*** 的特殊字元有 **$**、 **^** 、 **.**、 ***** 、 **+** 、 **?** 、 **[**、**]** 及 ****等九个。例如， **``a''** 是一般 的字元，它也只代表 **``a''**，别无其它。但是 **``a.''**、 **``a*''** 、 **``a?''** 与 **``a+''**等所代表的意义，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。现在来谈谈几个简单的 ***Regexp*** 的表示法。 想进一步多了解 ***Regexp***，仍请参考编号为 **94019** 的技术报告。以下就开始讨论，代表 ***Regexp*** 的符号有九个。为了讨论上的方便，将其分成若干类：

- 不在乎所出现的字母为何，可以『**.**』 来表示。
- 所以『**.**』 代表除了 **newline** 之外的任何一个字元。 例如 **``a.b''** 表示任何一个由三个字元所组成的字串， 此三个字元必需符合第一个字元是 **``a''**， 第三个字元 是 **``b''**，但中间的字元只要不是 ***newline***， 任何字元都可以。所以 **``a.b''** 可以为 **aab**、**abb**、**acb**、**axb**、**a1b**、**a2b** 等等， 甚至可以是一些特殊字元，只要这些特殊字元在此时没有特别的意义。例如 **``a^b''**。
- 以特殊字元，来表示重复出现的一般字元。这些特殊字元称为
- **postfix character**。其表示符号有 『*****』、『**+**』 与『**?**』 等三种。
  - 『*****』
  - 任何字元之後加上『*****』，可表示字元重复出现的次数， 从零次到无限多次。因此 **``ab*''**，可表示， **a**、**ab**、**abb**、**abbb**、**abbb**以及 **ab...b** 到无限个 **b**。
  - 『**+**』
  - 任何字元之後加上『**+**』，可表示字元重复的出现次数， 从一次到无限多次。因此 **``ab+''**，可表示为 **ab**、**abb**、**abbb**、**abbb** 以及 **ab...b** 到无限个 **b**。
  - 『**?**』
  - 任何字元之後加上『**?**』，可表示字元重复的出现次数， 不是零次就是只有一次。因此 **``ab?''** 不是表示为 **``a''**， 就是表示为 **``ab''**。
- 若字元为属於某一特定集合中的元素时，可以字集（**character set**） 来表示。
- 此时的代表符号是 **[...]**。中括号内就是放置字集的地方。如下讨论 字集的若干变化。
  - **[...]**
  - 最简单的字集表示法，是将所有符合的字元放於在括号内。 例如，**[a@!d13]**。此时，符合的字元就只有 **a**、**@**、**!**、**1** 和 **3** 而已。 若所表示的字集，具有一定范围的连续性，为了方便表达， 可以 **[a-g]** 来代表**[abcdefg]** 。 这一类的表示法，将会有更 详细的讨论。
  - **[...]** 与其它的特殊字元的配合。>
  -
    - **[...]** 与 「*****」 的配合
    - **c[ab]d** 代表 **``cad''** 或 **``cbd''**。 **c[ab]*d** 代表头尾为 **``c''** 与 **``d''** 的字串 ， 头尾间的组合，则由 **[ab]*** 来决定。 **[ab]*** 可表示为 **cd**、**cad**、**cbd**、**caabd**、**cabbaar** 等形式。
    - **[...]** 与 **+** 的配合
    - **c[ab]+d** 表示头尾为 **``c''** 与 **``d''** 的字串 ， 头尾间的组合，则由**[ab]+** 来决

定。 **[ab]+** 可表示为 **cad、cbd、caabd、cabbaad** 等形式。
        - **[...]** 与 **?** 的配合
        - **c[ab]?d** 表示头尾为 ``**c**'' 与 ``**d**'' 的字串 ， 头尾间的组合，则由 **[ab]?** 来决定。 **[ab]?** 可表示为 **cd、cad、cbd** 等形式。
    - 特殊字元，为字集内容时的解释
    - 任何 ***Regexp*** 的特殊字元，出现在中括号内都 可将其视为一般的字元，除了二个符号之外。这二个 例外的符号，也要看其出现的位置而做不同的解释。 这两个符号是 ``^'' ， ``-''。
        1. ``^''
        2. ``^'' 若出现在括号的第一个位置时，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]** 则表示除了英文字母之外的所有集合。
        3. ``-''
        4. 简化范围性的字集的表示法。其用法是将字集的起始点 放於 ``-''的右边，而终点放於 ``-'' 的右边。 例如，**[abcdefg]** 可以 **[a-g]** 来表示， **[a-z]** 则表示二十六个字母的集合。**[0-9]** 则表示数字的集合， 要表示二位数字的集合，则可以 **[0-9][0-9]** 来表示。
- 指定 ***Regexp*** 出现的地方为列首或列尾。
-
    - ``^''，指定 ***Regexp*** 出现在列首的符号。
    - 在 ***Regexp*** 前加上 ``^'' 的符号，即表示此 ***Regexp*** 必需出现在列首。例如， ``^**The**'' ，则找寻每列以``**The**'' 为首的字串。
    - ``**$**''， 指定 ***Regexp*** 出现在列尾的符号。
    - 指定 ***Regexp*** 必需出现在列尾时，只要在 ***Regexp*** 後加上一个 ``**$**'' 即可。例如，**xxxx+$** 会将列尾以 ``**x**'' 结束的字串找出。
- ``\'' 的用法
- \ 在 ***Regexp*** 中有二种涵意：一、使特殊字元变为 普通字元，二、使普通字元转为特殊字元。
    - 特殊字前加上 \ ，此特殊字元就不具特殊意义， 只是一个普通字元而已。例如，列首要以 ^ 为开端， 则以 \^ 来表示，此时的 ^ 则为普通字元。
    - 将如下的普通字元
    - **|、 (、)、 d、 '、 `、 b、 B、< 、> 、 w、 W、 sc、 Sc** 之前加上 \ 之後，则此普通字元就已特殊化了。现在 只讨论几个常用的字元（| 、**(**、**)**、**d**、< 、> ）。
        - \| ，（表示选择的用法）
        - 欲在二个 ***Regexp*** 中择一而用，可以 \| ， 将二个 ***Regexp*** 放於其左右来表示。 例如,**the\| it** 为二选一的 ***Regexp*** 的表示法。可能的符合字串为大小写穿插其间的 **the** 或 **it**。例如，**The**、**tHe**、**thE**、**THe**、**tHE**、**THE**、**iT**、**IT** 等。为何会有不同的大小写表示法，请再一次注意 **EMACS** 对於 ***case sensitive*** 的处理方式。若不清楚者， 请参照 **6.1** 节。 以 \| 所表示的 ***Regexp*** 有一个特色， 那是在找寻合适的字串时，与 ***Regexp*** 从开始至结尾都符合的字串。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** 来表示。乍看之下，似乎没有任何的疑问。事实上， 这种表示法也没有不对，可是各位是否还记得 ***Regexp*** 的真谛，就是要以最少的字元来表示最大的字集，在这个例子 中的重复性很高，似乎不太能符合 ***Regexp*** 的精神。 下面就是改进的方法。
        - 利用 **\(...\)** 将 ***Regexp*** 的范围规范出来。
        - 利用此方法，上例就可以``**\(read\| get\)file**'' 来找寻 **readfile** 或 **getfile**。这样是不是简捷多了吗？ **\(...\)** 还可以配合 ***、+、**?** 等特殊字元 使用。 ***、+、**?** 等特殊字元在处理字元的重复性时， 只适用在特殊字元之前的一个字元，所以若想重复一组 字元时，就必需以**(...\)** 将其组合起来。例如，``**ba\(na\)***， 可以表示 **bana**、**banana** 与**bana.....na** 等无数个 **na** 的组合。

除了以上所述的表示法外，若干普通字元加上 \ 还可以 有特别的用法，以下讨论常用的几个用法。

1. **\ d**
2. 在 ***Regexp*** 的表示法中，可以只撷取部份的符合 **egexp** 。其做法是将要保留的 ***Regexp*** 暂存在缓冲区 内之後，再将其拿出来使用。例如，要将所有副名为 **.c** 的档名，换成为 **.f** 的档名，其做法如下： **ESC-x replace-*Regexp* RET \(file[0-9]\)\.c RET \1\.f** 如此，会将所有名为 **file1.c file2.c file3.c ..... file9.c** 的档名，改为 **file1.f file2.f file3.f ..... file9.f**。
3. \< ，寻找一字的开头
4. \> ，寻找一字的结尾 例如，\< **b[a-z]g**\> 会将 **beg**、 **big**、 **bag** 等字串找出。

回主选单

Emacs 的其它相关事项

这章所谈的内容较为纷杂，但都环绕在加强 **EMACS** 的编辑功能 为核心。所谈的内容包括 ***register***、 ***bookmark***、 错字的自动侦测以及如何在**EMACS** 中使用绘图功能。现在就开始讨论 ***register*** 和 ***bookmark***。

Registers and Bookmarks

EMACS 的 ***register*** 是一个可将文件与游标位置（***point loaction***）暂时存放其间的地方。***register*** 有它的缺失，那就是一旦离开目前所使用的 **EMACS**，所有存於 ***register*** 中的资料，也会随著消失。若想再使用这些放於 ***register*** 中的文件或游标位置时，必需重新将资料存入 ***register*** 中。所以 ***register*** 只适合暂时性的储存，若想永久使用文件或游标位置，必需仰赖 ***bookmark***的帮助了。 ***bookmark*** 与 ***register*** 相似，它们都用来 储存物件，但在相似中又有不一样的地方。二者不同的地方如下：

- 命名上的不同
- *bookmark* 的名称可由「一个以上」的字元（character） 组成，但 *register* 的名称只能由「一个」字元来命名
- 资料保存的时间不同
- 存於 *bookmark* 的资料具有永恒性，它可以在离开 EMACS 後还存在。但存於 *register* 中的资料，在离开EMACS 後就无效了。

  现在就分别介绍 *register* 与 *bookmark* 二者的用法。首先讨论 *register* 的用法。*register* 可存放的内容有，游标的位置、一般文件与长方形文件的内容、 档案的名称以及本文未曾讨论的视窗资料等等。现在就一一 来介绍这些内容如何存放，以及存放後如何将其取出用， 与视窗有关的 *register* 在此依然不予讨论。
  - 储存与移动至特定 point 位置的方法
  -
    - **Ctrl-x r SPC *r*（point-to-register）** 将游标目前所在的位置，存於 ***register*** 中。存於 ***register*** 的步骤如下：
      1. 将游标移至所要储存游标位置的地方。
      2. 键入 ``**|Ctrl-x r|**'' 以及 ``**SPACE**''（空白键）後，
      3. ***echo area*** 会出现 **Point to register:**
      4. 此时可输入任意一个字元（**character**），做为此 ***register*** 的名称。以後要使游标移至此位置就要靠此字元。 ***register*** 的名称，只能由一个字元组成。
    - **Ctrl-x r j *r*（jump-to-register）**
    - 根据 ***register r*** 所储存的位置，将游标移至 ***r*** 所设定的位置。使用此指令前，必需确定 ***register r***， 已设定的妥当了 以下就是设定游标位置以及使用游标位置的步骤：
      1. 先以 ``**Ctrl-x r SPC *r***''，将游标的位置储存於 ***register r*** 内。
      2. 键入 ``**Ctrl-x r j**''， ***echo area*** 会出现 ``**Jump to register:**'' 的讯息。
      3. 在 ``**Jump to register:**'' 之後，输入已设定妥善的 ***register r*** 的名称。此时游标所在的缓冲区若与 ***register r*** 所设定 的缓冲区不一样时，游标会自动移至另一个缓冲区，视窗的 内容也会换成新的缓冲区内容。所以，使用 ***register*** 所存放的游标位置，是可以跨越不同的缓冲区来做移动的。
  - ***register*** 还可用来存放经常被访问到的档案。
  - 有人或许会不解，为何耗费如此的功夫，只为了从事 ``**Ctrl-x Ctrl-f**'' 指令可以做到的事。使用 ``**Ctrl-x Ctrl-f**'' 有一个不便之处，就是若所欲访问的档案与

EMACS 的预设档案位置 不一样时，则必需告之完整的路径名。所以，此时若使用 *register* 来储存档名，就只需要给予 *register* 的名称即可。至於， 冗长的全称就交给 *register* 去处理了。以下就是以 *register* 存档案的方法：

1. 使用 *register* 来储存档名与储存游标位置，最大的 不同点在於，以 *register* 来储存档名必需借助 ``.emacs'' 档。 因为以 *register* 来储存档案名称，必需借由 ``.emacs'' 来设定。
2. 以 *register* 来储存档案名称的实际执行步骤如下：
3.
    1. 设定 ``.emacs'' 的方法
    2. 在 ``.emacs'' 档中以 ``set-register'' 函式 ，将 *register* 的名称以及其所储存的档名设定清础。以下的例子就是将 ``chap7.tex'' 的档名，放於名为 ``a'' 的 *register* 时， ``.emacs'' 档的设定方法 。 **(set-register ?a '(file . "\~/report/emacs/basic/chap7.tex"))**
    3. 当档案名称的设定已在 ``.emacs'' 档中完成後，必需离开 EMACS，再重新进入 EMACS 一次。因为只有重新执行 EMACS， 修改过的 ``.emacs'' 档才有机会被重新执行。 重新执行过的 EMACS ,就可以 `` Ctrl-x r j *r*'' 将档案从 *register r* 中拿出来使用。 以下就是使用 *register r* 的方法：
    4.
        1. 键入 ``Ctrl-x r j RET ''， *echo area* 会出现
        2. ``Jump to register:''。
        3. 待 ``Jump to register:'' 的讯息出现时，即可输入已设定好
        4. *register* 名称。以上例为例，若输入 ``a''， 则视窗会出现 chap7.tex 的文件内容。

- *register* 中储存文件的方法
- 储存於 *register* 的文件可以有二种：一种为线性的文件 （linear text），另一种为长方形的文件（rectangle text）。 所以有此区分，原於使用 *register* 储存文件，需先将所要 储存的文件做上标记，再根据标记放於 *register* 中。 线性文件与长方形文件在标记上是一样的，要区分二者的差异 只能仰赖使用时的指令了。想进一步了解长方形区域，请参考 5.4 节 ``Move Text and Copy Text'' 有关长方形部份。以下就讨论二者的用法。
    - 线性文件的 *register* 设定方法如下：
    -
        1. 将要放入 *register* 的线性文件做好标记。
        2. 设定标记的详细方法，请参考 5.3 节。现略做题示， 使用 Ctrl-@ 或 Ctrl-SPC 做为线性文件的起始点， 以游标所在的位置为终点。
        3. 使用指令 `` Ctrl-x r s *r*''，将所设定好的线性文件
        4. 放入 *register r* 中。
        5. 欲使用 *register r* 中的资料时，可以指令
        6. `` Ctrl-x r i *r*'' 将存放於 *r* 中的资料取出使用。

      以上所设定的 *register* 有一特性，就是离开EMACS 後所有的设定 也成为历史。想将 *register* 内容保留的方法，是将设定写在 ``.emacs'' 中，如下是以 *register* ``a'' 储存 *register* 文件， 以及以 *register* ``.'' 储存 ``。'' 资料的设定法： **(set-register ?a "register") (set-register ?. "。")**

    - 长方形文件的 *register* 表示法：
    -
        1. 将要放入 *register* 的长方形文件做好标记。设定标记的
        2. 方法以及所谓长方形文件的定义，请参考第五章第三节及第四节。
        3. 使用指令 `` Ctrl-x r r *r*''，将所设定好的线性文件
        4. 放入 *register r* 中。

5. 使用 ***register r*** 中的资料时，可以指令
6. `` **Ctrl-x r i *r*''** 将存放於 ***r*** 中的资料取出使用。

放於 ***register*** 中的文件，不论是线性文件或长方形文件， 将其拿出使用的指令都是一样的。不一样处在於放入 ***register*** 时的差异。会有差异是不难理解的，因为二者在处理标记的过程是一样， 只有靠存入 ***register*** 时，以不同的指令来加以区别。

谈完了 ***register*** ，接下来介绍 ***bookmark*** 。 ***bookmark*** 的一些基本概念，已在前面略做介绍了，现在就进一步详述它。 ***bookmark*** 与 ***register*** 在记录游标位置的功能是一样的， 但在对 ***register*** 的命名方面二者就显得有差异了。 ***bookmark*** 的命名可以由一个以上的字元组成。换言之，它可以有一个较长的名字， 但 ***register*** 就只能由一个字元来命名。而且以 ***bookmark*** 设定的资料，可永久储存起来，即使离开**EMACS**，***bookmark*** 的资料也不会就此消失。所以想要永久保存的资料，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，将游标所在的
2. 位置做上记号**(mark**），此记号就称为 ***bookmark***。 当第一次使用 `` **Ctrl-x r m''** 指令时， ***echo area*** 会出现 ``**Set bookmark (visited-filename) :''** 的讯息。此时，若不输入任何 字元只键入 **RET** ，系统会以所访问的档名做为 ***bookmark*** 的名称。若输入其它的字串，则 ***bookmark*** 就以此字串命名之。
3. 以 `` **Ctrl-x r b *bookmark* RET''** 指令，来移动游标的位置。
4. 键入 ``**Ctrl-x r b''** 时， ***echo area*** 会出现

   ``**Jump to bookmark (bookmark-name) :''**，

   括号内的名称为系统的预设值。若所给予的预设值不是心中所爱， 可以在 `**:'** 之後输入想要的 ***bookmark*** 名称。

资料以 ***bookmark*** 的方式储存时，使用者若不另行指定 储存的档名，系统会将有的讯息存入一个名为 ``**~/.emacs-bkmrks''\index.emacs-bkmrks** 的档案中。 接下来，讨论一个使 **EMACS** 的编辑功能更强化的议题 － **Fixing Typos** 。

文字的勘误

EMACS 所提供的这项功能，并不是内建於 **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** 环境，较常使用的拼字检查工具程式 **(spelling checker program)**有 **spell** 与 **ispell**。二者又以 **ispell** 较为方便好用。虽然如此，并不是所有的 **UNIX** 系统都 提供有 **ispell** 与 **spell** 等工具程式。使用者可以 `` **which''**、 `` **find''** 等指令，在 **shell** 下查阅 **UNIX** 系统所提供的工具为何。 不清楚如何使用 **which** 或 **find** 等指令的使用者， 请查看 **man page** 。

EMACS 提供的拼字检查介面，是针对 **ispell** 的使用法 为主。透过 **EMACS** 的介面，使用者可使用 **UNIX** 系统所安装的 **ispell**。若系统没有安装 **ispell**，可以 **ftp**到适当的地方得取。 **(Ispell is available via anonymous ftp from ftp.cs.ucla.edu in the directory /pub/ispell.)**
ispell 除了会自动侦察文件的拼字错误外，还提供可能 修改的讯息，供使用者参考。**ispell** 的除错范围，可以侦察 一个字，也可以侦察一个区域或整个的缓冲区。 使用 **ispell** 时，程式会根据所检查的内容做出适当的回应。 如果检查无误， ***echo area*** 会出

现无误的讯息。如果所检查 的字有误， **ispell** 会另开一个视窗，将所有可能的勘误组合 显示出来，以利使用者做评估。若在显示的视窗找到合适的取代文字， 可直接键入文字的编号来完成勘误的程序。
ispell 使用了两种不同的字典，一种是由 **ispell** 所提供的标准字典**(standard dictionary)**。另一种是使用者自已 建构的私有字典（**private dictionary**）。标准字典是系统所提供的； 使用者字典是在编辑文件时，因需要而随时加入的。当标准字典无法找到的字，使用者可随时将新增的字加入私有字典中。系统所使用的 标准字典，可依使用者的需要来指定。当然，若不指定标准字典， 系统会使用预定的字典。若想改变预定的字典，可使用如下的指令来设定：

ESC-x ispell-change-dictionary

若想查阅所使用的标准字典为何，可查阅变数``**ispell-dictionary**'' 来得到相关讯息。如下是笔者以 `` **Ctrl-h v**'' 指令查阅此变数 所得的结果。若已淡忘变数的查询，请参考 **4.4** 节。 **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.**
现在就讨论 **ispell** 的使用方法。

- 一般文件使用 **{ ispell** 检查错误所使用的指令，
- 可依所检查的范围分成如下三种：
  - 检查「字」的 **ispell**指令如下：
  - **ESC-\$**
  - 检查「区域」（**region**）的 **ispell** 指令如下：
  - **ESC-x ispell-region\index{ESC-x ispell-region** 使用区域的检查以前，要先将区域的范围界定清础，再使用 此指令来做区域的检查。
  - 检查「缓冲区」（**buffer**）的 **ispell** 指令如下：
  - **ESC-x ispell-buffer** 检查整个缓冲区内的文字内容。
- 信件中使用 **ispell** 检查错误的指令
- **ESC-x ispell-message**
- 以 **ispell** 检查文字内容时， ***echo area*** 会有回应
- 的讯息出现。若有错误发生时，会另开一个视窗，将所有的修改 可能显示出来。如何使用 **ispell** 来修改错误，就是现在 的话题了。
  - 若检查的范围只是一个字时，此时，所检查的字若正确无误，
  - ***echo area*** 会出现

    ***word* is correct**

  - 若 **ispell** 认为所检查的英文字有误时，会另开视窗，
  - 将 **ispell** 认为可能的正确字列举出来。使用者可参照所显示 的字，来做出合适的回应。回应的方式如下：
    - **SPC**
    - 保留现状而不更改错误，但将游标移至下一个发生错误的地方。
    - ***digit***
    - 当有错误出现时， **ispell** 会将可能的正确字一一编上号码， 供使用者参考。 若所显示的字就是所要更改的字，此时只要 键入编号，缓冲区的错误即可获得修正。 例如，检查 ***dictionory*** 这个字是否的正确时，视窗上会出现 `` **(0)dictionary** '' 的提示，此时若想修改错误的字，只需键入提示 的数字 `**0**' 即可。
    - **r *new* RET**

- 将目前找到的错字以新字 ***new*** 取而代之。文章中其它地方 若有相同的错误，不会因此而有变动。
- **R *new* RET**
- 除了将目前出错的地方修正外，同时还进行 ``**query-replace**'' 的 动作，将所有相同错误的地方也一并修正。
- **a**
- 将找出来的错误，将错就错视为是正确的。此错误适合所有目前 此 **EMACS** 下的所有缓冲区，但离 **/i> EMACS** 後就恢复原状， 错还是错。
- **A**
- 大写的 ``**A**'' 与前一个小写的 ``**a**''，唯一差别的地方就是， 将错就错的地方，只适用於目前所使用的这一个缓冲区。
- **i**
- **ispell** 所使用的字典有二种，一种是标准字典，另一种 是使用者自行订定的私有字典。将标准字典中所没有的字加入 私有字典中就是靠指令 ``**i**'' 了。 **ispell** 在检查字的正确与否时，是根据标准字典来判断对错。 此标准字典对於专有术语的搜集，并不是很周延。为了弥补此缺失， 私有字典就应运而生了。使用者可将标准字典中没有的字， 加入私有字典中，使这些原本被视为「错」的字得以正名。 当 **ispell** 检查出「错字」时，键入 ``**i**''，就可将此错字 纳入私有字典中，而成为「正确」的字。。从此若再出现此字时， **ispell** 就会将其视为「正确」的字。纵使离开 **EMACS** 後再进入 **EMACS**，此字也会是「正确」的字。 若想使新字加入私有字典时， ***echo area*** 能将加入的新字回应出来，则必需使用 **"m**'' 指令， 而非 `` **i**'' 指令。

以上所谈的是透过 **EMACS** 所提供的介面，使用 **UNIX** 的拼字检查的 工具（ **ispell**）。如果所使用的系统无法提供此工具软体时， 请以 **ftp** 取得软体後再自行装置 **ispell**。否则，就算 **EMACS** 提供了使用 **ispell** 的介面，也是无济於事的。 除了以上所谈的以软体来修正错误外，还有其它的方法可以修正所 发生的错误，详述如下：

- **Transposing Text**
-
  - **Ctrl-t**（**transpose-chars**）
  - 游标前後两个字元（**character**）互调。
  - **ESC-t**（**transpose-words**）
  - 游标前後两个字（**word**）互调。但两个中间的标点符号 （**punctuation**）则不会移动。例如， **transposing,text** 则互换成 **text,transposing** 而不是 **text transposing,**。
  - **Ctrl-x Ctrl-t**（**transpose-lines**） 前後两列互换。将游标所在的列与其上一列互换。
- **Case Conversion**
-
  - **ESC-l**
  - 将一个字的字元改成小写的字元。键入指令 **ESC-l** 会将游标所在 处之後的所有字元换成小写的字元，其范围只限於游标所在处的 那一个字。
  - **ESC-u**
  - 将一个字的字元改成大写的字元。键入指令 **ESC-l** 会将游标所在 处之後的所有字元换成大写的字元，其范围只限於游标所在处的 那一个字。
  - **ESC-c**
  - 将游标所在处的字元换成大写的字元（**capital**），所以如果要使 一个字以大写为开始可以使用此指令。

接下来所要讨论的议题是，如何使用 **EMACS** 的编辑器来绘制简单的图形。

图形的编辑

EMACS 的绘图模式所能提供的绘图功能并不多，它只予许以键盘 上出现的字元，来构思图形，也就是只能以 **ASCII** 的字元， 来编辑所要的图形。**EMACS** 的图形模式可以与任何其它的模式 一起搭配使用。所以，只要有编辑图形的需要，都可以切换至 图形模式，将所要的图形编辑出来。

EMACS 的绘图功能，主要是由八个控制游标移动方向 的指令来完成。这八个方向也是**EMACS** 构图的主体， 它们依序是，东、南、西、北、东南、西南、西北以及东北。 虽然可以设定八种游标移动的方向，但一次只能设定一个方向， 若想使游标移动的方向改变，必需重新设定游标移动的方向。 设定这八个游标移动方向的指令分别是：

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

这八个指令似乎颇为复杂，但如配合图形的解释，就可了 这八个指令的由来。因为，每个指令的表示法与所要表达的方向 有著密切关系。 北**(rignt) C-c ^ . 西北(nw) . 东北(ne) C-c ` . C-c ' . . . . C-c < ..... C-c >** 西**(left) ... 东(right) . . . . . C-c / . C-c ** 西南**(sw) . 东南(se) . C-c . 南(down)** 从以上的图示，不知能否看出这八个控制方向的指令与所表达 的方向有著密不可分的关连性。例如，往东的方向则以 「>」来表示， 往西的方向则以 「<」，往北的方向则以 「^」 来表示等等。 基本上，从使用的指令就可知道所代表的方向为何。当然，其原意 并不是要以指令来猜方向，而是用方向来诠释指令。所以，只要 掌握西南方的方向，就知道使用 「/」 来表示。

图形模式可以在编辑的过程中随时使用的。使用图形模式的 方法很简单，只要将其唤起即可。唤起图形模式的方法是： `` **ESC-x edit-picture**''。 当进入图形模式後，***mode line*** 会将使用图形模式的讯息 显示出来。进入图形模式後，也可以随时离开此模式，而回到 唤起此模式前的状态。回到先前模式的方法是`` **Ctrl-c Ctrl-c**。
进入图形模式後，就可利用设定游标移动方向的八个指令， 来控制输入字元出现的方向，而绘制出合适的图样。当选定了 游标的移动方向後， ***mode line*** 会将游标移动的方向显示出来。 此时，键入任何一个键盘上的字元，字元会依照所设定的方向出现在萤幕上。
图形模式所采用的是覆盖模式，这是有其道理的。绘图就是 要在特定的地方绘上所欲的图样，如果使用插入的模式，就会破坏整体 的设计。所以，使用覆盖模式是较合理的安排。
图形模式除了设定游标移动方向的指令外，还有其它的指令 是针对编辑功能而设。例

如，字或列的删减，空白列的增加等等， 都是在绘图时，不可或缺的编辑指令。现在就来讨论与绘图有关 的编辑指令。绘图的编辑指令与一般编辑的指令大同小异， 为了避免将一般的编辑指令与绘图时所使用的编辑指令混淆， 现将二者的差异整理列举如下：

Keystrokes	Text Mode	Picture Mode
RET	加入一行空白列	使游标向下移动，若要加入空白列， 则使 用 ``**Ctrl-o**'' 指令。
Ctrl-d	删除一个字元後，文件会向左移动	将游标所在处的字元以空白取代， 但文件不会向左移动。若要删除字元，必需使用 "**Ctrl-c Ctrl-d**" 指令。
SPACE	输入空白且将文件移向右边	游标向右移动，同时游标所到之处，均将 文字以空白覆盖。此时若想输入空白，则 必需回到 **Text Mode**，在**text Mode** 的状态输入空白 。
Ctrl-k	删除所在列的内容，使用两次 **Ctrl-k** 则删除内容与此空白列。	删除内容但不会删除列。欲删除列只有回到 **Text Mode**
TAB	输入 **TABs** 且将文件向右移动	只以**TAB** 的距离移动游标，但它所经过之处并不会输入 **TABs** 。
Ctrl-n	游标移至下一列，但游标所在的栏 位则视情形而定。	游标移至下一列，但栏位与上一列相同。
Ctrl-p	游标移至上一列，但游标所在的栏 位则视情形而定。	游标移至上一列，但栏位与下一列相同。
Ctrl-f	游标向前移动一个字元。	游标向右移动一个字元。
Ctrl-b	游标往回移动一个字元。	游标向左移动一个字元。

除了以上所谈的种种功能之外，图形模式亦可与长方形 编辑相配合。长方形编辑是专用来编辑区块文件，而图形模式 的编辑，从某个角度来看，就是区块的组合。所以，长方形编辑 的运作，就可用在图形模式的剪贴上。因此，任何可以使用在 长方形编辑的指令，都可用在图形模式中。不熟悉长方形的编辑者， 可参考本文的 **5.4** 节。
图形模式除了可使用长方形编辑的指令外，图形模式 本身，也提供了适用於此模式的专门指令。现就一一为各位介绍。

- 可利用的长方形编辑指令，如下：
-
    - **Ctrl-x r d**（**delete-rectangle**）
    - 删除设定好的长方形区块内的文件，经删除後的文件无法再将其 **yank** 出来。除非是执行 **undo** 的指令。
    - **Ctrl-x r k**（**kill-rectangle**）
    - 与图形模式 `` **Ctrl-c Ctrl-k**'' 的作用相同。
    - **Ctrl-x r y** （**yank-rectangle**）
    - 与图形模式 `` **Ctrl-c Ctrl-y**'' 的作用相同。
    - **Ctrl-x r o** （**open-rectangle**）
    - 在设定好的长方形区块内，插入空白後使文件向右移动。
    - **Ctrl-x r r *r*** （**copy-rectangle-to-register**）
    - 将设定好范围的区块资料，存入 ***register*** 中， 此时区块内的资料并不会消

失。
    - **Ctrl-x r i *r*** （**insert-register**）
    - 将存入 ***register*** 中的资料取出。
    - **ESC-x clear-rectangle**
    - 将设定好的长方形区块内的文件以空白取代。
    - **ESC-x string-rectangle RET *string* RET**
    - 设定好的长方形区块插入新的字串（ ***string***）。此时的 区块只决定加入新字串的长度，宽度则由所给予的字串长度 来决定。原来区块内的文件会向右移动。
- 图形模式自行开发的编辑指令，如下：
-
    - **Ctrl-c Ctrl-k**（**picture-clear-rectangle**）
    - 长方形指令 `` **Ctrl-x r k**" 与此指令具有相同的效果。 此指令经常被使用，所以将长方形的指令予以精简，使其在 图形模式中更易於被使用
    - **Ctrl-c Ctrl-y**（**picture-yank-rectangle)** 长方形指令 `` **Ctrl-x r y**" 与此指令具有相同的效果。此指令经常 被使用，所以将长方形的指令予以精简，使其在图形模式中更 易於被使用。
    - **Ctrl-c Ctrl-w r** （**picture-clear-rectangle-to-register**）
    - 将区块内的资料存入 ***register*** 中。此时区块内的资料会以空白 取代，这是与长方形指令 `` **Ctrl-x r r *r*** 最大不同之处。
    - **Ctrl-c Ctrl-x *r*** （**picture-yank-rectangle-from-register**） 将存入 ***register r*** 中的资料取出。

长方形指令的使用在图形模式中非常有用。因为，图形的构成就是 一块块的区域。所以善用长方形指令会使图形的编辑更加灵活。

回主选单

中文编辑环境

以上所谈的种种，都较适合英文的编辑环境，现在来讨论一个适用於中文编辑的环境。在 EMACS 下可使用的中文环境有二种，一个就是 EMACS 本身，另一个是 EMACS 的姐妹品－ Mule。现在就开始讨论这二个中文的编辑环境。

Emacs 下的中文编辑 .emacs 档的设定

在 EMACS 的环境下要使用中文，必需先做些设定。因为，中文内码是使用 8 位元位元组，所以必需将系统所使用之位元组型式设为 8 位元，才可以在中文环境下使用 EMACS 。设定系统使其适合使用中文的环境，必需视所使用的作业系统与所使用的 Shell 的 script file 有关，现简述如下：

- 若使用 C Shell/TC Shell 时， .cshrc 的设定方式又因所用的
- UNIX 作业系统不同可分为如下二种方式，
    - 使用 Sun OS 作业系统的设定
    - setenv LC_CTYPE ISO_8859_1
    - 使用 HP-UX 作业系统的设定
    - setenv LC_CTYPE american.iso88591
    - 在萤幕上显示 8 位元的中文时，设定 8 位元的方式如下：
    - stty cs8 -istrip -parenb
- 若使用 Bourne Shell/Korn Shell 时， .profile 的设定方式 又因所用的 UNIX 作业系统不同可分为如下二种方式，
    - 使用 Sun Solaris 作业系统的设定
    - LC_CTYPE=ISO_8859_1 export LC_CTYPE
    - 使用 HP-UX 作业系统的设定
    - LC_CTYPE=american.iso88591 export LC_CTYPE
    - 在萤幕上显示 8 位元的中文时，设定 8 位元的方式如下：
    - stty cs8 -istrip -parenb

除了以上针对 Shell 的 script file 的设定外，要在 EMACS 的环境
输入与显示中文尚必需在 EMACS 的起始档 ``.emacs" 中加入
如下的设定:

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

虽然经由这些设定，可以使 EMACS 在中文的环境下使用，
但其处理方式还是以英文的模式来处理中文。换言之，就是将二个
byte 所组成的中文字，依然视为二个 byte 的英文字来处理。
这种处理方式，自然有其不便之处。例如，在处理换列时就会
遇到一些问题。因为 EMACS 将中文视为是二个 byte 的字元共同组合
而成。所以，当一列只剩下一个 byte 时， EMACS 会将中文切割
成二个独立的 byte 分别处理之。此时中文字的第一个 byte 会在
此列的最後一个栏位，而下一列的第一个栏位则出现此字的第二个 byte 。
所以，在 EMACS 的环境下中文字很容易被切割，而不知其为何物。
因为 EMACS 处理中文的方式与处理英文无异，所以删除一个
中文字，等於删除二个 byte 的字元（character）。换言之，
一个中文字若使用删除字元的指令 （ **Ctrl-d）来删除，必需**
使用两次的 Ctrl-d 才能将一个中文字删除。

因为 EMACS 在处理这些七位元之外的字型有其基本上
的问题，所以就有 Mule 的诞生。以下就介绍 Mule 的使用法。

中文化的 EMACS － Mule

Mule 是 "The MULtilingual Enhancement of GNU Emacs" 的简称，它是针对非英语系国家的使用者而设计的 EMACS。 对中文的使用者有很大的助益。 要使 EMACS 能真正的中文化，最好的方法就是安装 Mule。目前最新的 Mule 版本是 Mule 2.0 版（1994 年 8 月 6 日）。 Mule 的设计是以 EMACS 为基础。至於 EMACS 的启始档案 － ``emacs''，也就是 Mule 的启始档案。换言之，使用 Mule 不需在 ``.emacs'' 档中做任何的设定。所以 Mule 可将 EMACS 的 ``.emacs'' 档做为其启始档。若无特殊需求，也可不使用 ``.emacs'' 档。想要取得 Mule 的软体，可以 ``anonymous ftp'' 至以下几个地方取得：

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''。 它不只处理七位元（7 bits）的 ASCII 字型（ASCII Characters） 以及 ISO Latin-1 的八位元字型（8bits）。它更能处理日文、中文（GB、Big5）、韩文（16 bits，ISO2022 标准）、泰文 （TIS620）以及越文（VISCII 及 VSCII）。目前适合吾人使用的繁体中文系统是以 BIG-5 为主。 Mule 有如此多的语文可供使用，所以在一个文字档案中，使用者可以混合使用这些不同的语言。使用这些语文，只需借由 Mule 所提供的输入法，将各种不同的文字输入即可。如果是在 terminal emulator 下使用 Mule（如 cxterm、exterm 或 kterm），可使用此 emulator 所提供的输入方法。其实使用 Mule 的方法与使用 EMACS 的方法是一样的， EMACS 的指令在 Mule 中都可以使用。二者最大的差别，就是 Mule 加入了输入法的使用。欲在 EMACS 的环境下使用中文，首先要起动中文系统（例如倚天中文系统），而输入法就是使用中文系统所提供的输入法了。使用 Mule 时也是先起动中文系统(此处仍是以text-only 的终端机为讨论的重点，若使用中文的 X-window 则不需先行启动中文 系统)。，但中文的输入法，则由 Mule 来提供。换言之， Mule 有自已的中文输入法。现在就来讨论如何使用 Mule 的输入法。 要在 Mule 中可以方便的使用输入法，最好将如下的资料加入 ``.emacs'' 档中。 **(set-default-file-coding-system *big5*) (set-display-coding-system *big5*) (set-file-coding-system-for-read *big5*) (quail-mode 1) (quail-use-package "zozy")** 以上的资料会在使用 Emacs 时产生错误的讯息，解决的办法是将以下的资料加入% ``.emacs'' 档中。 **(cond ((boundp 'mule-version) ;;** 给 mule 用的设定区 **;; mule** 设定开始 **(set-default-file-coding-system *big5*) (set-display-coding-system *big5*) (set-file-coding-system-for-read *big5*) (quail-mode 1) (quail-use-package "zozy") ;; mule** 设定结束 **) ;;** 给 emacs 用的设定区 **(t ;; emacs** 设定开始 **(set-input-mode (car (current-input-mode)) (nth 1 (current-input-mode)) 0) (standard-display-8bit 160 255)) ;; emacs** 设定结束 **)** Mule 所提供适合中文使用的输入法有，仓颉、注音、拼音、倚天注音以及标点符号等五种。使用这些输入法的方式如下：

1. **Ctrl-x - Ctrl-k - Shift-m RET**
2. 连续键入 ``Ctrl-x'' ``Ctrl-k'' ``Shift-m'' 三个指令後，按下 RET， *echo area* 会出现

Quail Package： 此时就可键入所要使用的输入法。

3. 输入法的名称如下所示：
4.
    - 仓颉：**cj**
    - 注音：**zozy**
    - 拼音：**pinyin**
    - 倚天注音：**etzy**
    - 标点符号：**punct-b5**

    要使用任何一种输入法，只要在 *echo area* 处的 **"Quail Package:"** 後，键入所欲使用的输入法的名称即可。

5. **Ctrl-]**（**self-insert-command**）
6. 在任何一种的输入法的状态下，都可切换至英文的状态下。 只要键入 ``Ctrl-]'' 就可使中英文互换。

Mule 对使用中文的人来说，可说是中文化的 **EMACS**，而 **Mule** 的诞生，确实给予中文的编辑者很大的方便。**Mule** 的使用方法，除了输入法与 **EMACS** 略有不同之外，其它与 **EMACS** 一样。所以，已熟悉 **EMACS** 的人，使用 **Mule** 可说是易如反掌。如果只安装 **Mule** 而无安装 **EMACS** 的系统，想要使用 **EMACS**，只要进入 **Mule** 後一直维持英文的状态就与使用 **EMACS** 一样了。

EMACS 的入门手册介绍到此，已可暂告一段落。这并不意谓著 **EMACS** 的介绍已经穷尽了。其实，有关 **EMACS** 的话题还有许多，这里只是环绕在编辑的层面略作解说。

回主选单

Conclusion

Emacs 入门简介到此已近尾声了，希望读者能有所获。不论是从此开始使用 **Emacs** 或从此不再使用 **Emacs**，都是一种收获。撰写这本手册除了要介绍如何使用 **Emacs** 外，更希望读者能从中得知 **Emacs** 能解决何种问题。毕竟问题得以解决才是大家关切的重点。当然 **Emacs** 能解决的问题，其它的软体或许也能提供相同的解答。那麼使用 **Emacs** 的理由是否就变得薄弱了呢？其实，天生我材必有用，重点在於要适材而用。如果只是为了编辑一个简短的档案而且只从事一次的编辑，就不一定非锺情於 **Emacs** 不可。 如果在编档的同时又想从事编译的工作，在工作的同时还要兼顾信件的收发，这时候选择 **Emacs** ，就是个良策。因为 **Emacs** 本身就是一个整合的环境，它对於需要整合性的工作，非常的有助益。许多使用过 **Emacs** 的人，都觉得 **Emacs** 的指令太过於复杂而使人却步。关於这一点笔者从不反驳，但是对於 **Emacs** 的了解不能有因噎而废食的反应。毕竟，指令只是用来实践软体的工具而已，软体的本质才是要费神研究的重心。所以在使用 **Emacs** 的时候，不要因它五花八门的指令而怯步，希望读者能多留意 **Emacs** 所能解决的问题。只有如此，才能见到它的真谛。至於 **Emacs** 能做那些事，在前文中或多或少已经介绍过了，读者可以自行研判它是否可以解决问题。当然，有些 **Emacs**能做 的事在文中并未论及，除了限於篇幅外，还因为本文一直环绕在编辑器的层面。所以，对於一些程式的编撰与除错的问题就略过不提。对於 **Emacs** 巨集指令的安排，也因为超过「入门」太远也暂时将其放下。这些未曾提及的主题，并不是意味著它们不重要。而是因为不适合於此时此地出现。文章结束前再一次的强调 **Emacs** 是以编辑为基石，而以发展整合性的作业系统为志业的软体。所以，只要进入了 **Emacs** 的工作环境，就理应可以在其中完成所有的工作。离开 **Emacs** 的环境就是离开电脑工作的时候。最後，如果各位在看完此手册，觉得心动者，请就开始行动吧。只有勤加使用它，才会越发觉得它的好用。开始时的不顺，是必然的现象。开始动手吧。如果仍不知从何开始的使用者，请先进入 **Emacs** 後，使用它的线上辅助说明吧！任何时候请不要忘记 **``Ctrl-h''**，只要知道使用 **``Ctrl-h''** 那使用 **Emacs** 就有希望了。

回主选单

Powered by xiaoguo’s publishing studio
QQ:8204136