You are looking at the Emacs tutorial.

Emacs 的命令通常包括控制键(就是上面标有 Ctrl 或 Ctl 的那个)或者是 META 键(上面标有 EDIT 或 ALT)。为了方便起见我们将用下面的缩写来代替这些键的全称：

C-<chr> 意思是当敲入字符<chr>时同时按住控制键，因此，C-f 表 示：按住控制键并且按 f 。

M-<chr> 表示当键入<chr>时按住 META 或 ALT 或 EDIT 键。如果没有 META 或 ALT 或 EDIT 键，则用 ESC 键代替。<ESC>表示 ESC 键

注意：退出 Emacs，按 C-x C-c(两个字符)。在文本左边区域的 “>>”符号表示让你试着使用一个命令。比如：

>> 现在键入 C-v(观看下一屏)移动到下一屏。(就象前面说的，按 v 的同时也按住控制键)。从现在开始，每当你读完一屏的时候都可以用它来翻屏。

注意在翻屏后会保留上屏的最后一行；这是为你继续阅读文本提供某些连贯性。

你所需要知道的第一件事是如何把光标从一个地方移动到另一个地方。你已经知道了如何向前翻一屏--用 C-v。要向后翻一屏，键入 M-v。

>> 试着键入 M-v 和 C-v 几次。

* 摘要(SUMMARY)

下面几个命令对整屏观看时有用：

C-v 向前翻一整屏。
M-v 向后翻一整屏。
C-l 清除屏幕并重新显示所有的文本，然后把光标移动到屏幕的中央。 (注意是 Control-L,而不是 Control-1)。

>> 寻找光标，并且注意它在文本里的位置。然后键入 C-l。再寻找光标你会注意到光标现在会出现在同样的文本附近。

* 基本光标控制(BASIC CURSOR CONTROL)

整屏整屏的移动是很有用的，可是如何把光标移动到屏幕上文本里的一个指定的地方呢?

有好几个方法可以实现。最基本的方法是用命令 C-p，C-b，C-f，和 C-n。这些命令每个都使光标在屏幕上往特定的方向移动一行或者一列。下面是一个图表显示了这四个命令和它们所移动的方向：

```
                    上一行，C-p
                         :
                         :
向前，C-b ..... 当前光标的位置 ..... 向后，C-f
                         :
```

:
下一行，C-n

>> 用 C-n 或 C-p 把光标移动到图表中间。然后键入 C-l 会看到整个图表出现在屏幕的中央。

你也许会发现这些字母很容易记住：P 代表上面的(previous)，N 代表下一个 (next)，B 代表向前(backward)，F 代表向后(forward)。这些是基本的光标位置命令，你将经常会用到它们。所以现在学习它们很有好处。

>> 用几次 C-n 把光标向下移动到这一行。

>> 用 C-f 把光标移动到行里，再用 C-p 把光标上移。观察当光标在行的中间时 C-p 做了些什么。

每一个文本行都以一个换行符结尾，它用来当作行与行之间的分格。你的文件的最后一行的尾部应该有一个换行符(但 Emacs 并不要求一定要有一个)。

>> 试着在行的开头使用 C-b。它将会把光标移到上一行的末尾。这是因为它向后移的时候穿过了换行符。

C-f 也能象 C-b 一样穿过换行符。

>> 使用几次 C-b，使您能知道光标在哪。然后用 C-f 移动到行的末尾。然后再用一次 C-f，使光标移动到下一行。

当你移动超过屏幕的顶部或底部，光标回移动到下一屏的中间，这叫做“滚屏 (scrolling)”。它使得 Emacs 滚屏移动到文本上指定的部位而不是移出屏幕。

>> 试着用 C-n 把光标移过屏幕的底部，看看回发生什么。

如果觉得一个一个字符的移动太缓慢，可以一个单词一个单词的移动。M-f(Meta-f) 向前移一个单词，M-b 向后移一个单词。

>> 键入几个 M-f 和 M-b。

当光标在一个单词的中间，M-f 移动到单词的末尾。当光标在两个单词间的空白部分 M-f 移动到后一个单词的末尾。M-b 与 M-f 一样，只是移动的方向相反。

>> 键入 M-f 和 M-b 几次，中间穿插一些 C-f 和 C-b 以使你能观察到 M-f 和 M-b 在单词中和单词间的不同行为。

注意比较 C-f，C-b 与 M-f，M-b。通常情况下 Meta 键用于有关语言单位(词，句，段落) 的操作；而控制键用于编辑时的基本单位(字符，行等)。

这是句与行的比较：C-a 和 C-e 移动到一行的开头和末尾，M-a 和 M-e 移动到一个句子的开头和末尾。

>> 键入一对 C-a，再键入一对 C-e。 键入一对 M-a，再键入一对 M-e。

你会看到重复键入的 C-a 什么也不做，而重复键入的 M-a 则会移动一个以上的句子。

光标在文本中的位置也叫“点(point)”。在段落里，光标标示出了点在屏幕上文本里的位置。

下面是简单的光标移动命令的总结，包括单词和句子的移动命令：

C-f　向前移动一个字符。
C-b　向后移动一个字符。

M-f　向前移动一个单词。
M-b　向后移动一个单词。

C-n　移动到下一行。
C-p　移动到上一行。

C-a　移动到行首。
C-e　移动到行尾。

M-a　向前移动到句子的开头。
M-e　向后移动到句子的末尾。

>> 试着对每一个命令都实践几次，它们都是经常要用到的命令。

另外两个重要的光标移动命令是 M-<(Meta 小于)，它移动光标到整个文本的开头，M-> (Meta 大于)它移动光标到整个文本的末尾。

在多数终端上，“<”在逗号的上面，所以你必须用 Shift 键来输入它。在这些终端上，你也必须用 Shift 键来输入 M-<；没有 Shift 键，你可以输入 M-逗号。

>> 现在就试试 M-<，移动到本教程的开头，然后再用 C-v 移回这里。 现在就试试 M->，移动到本教程的末尾，然后再用 M-v 移回这里。

你也可以用方向键来移动光标，如果你的终端有方向键的话。我们建议学习 C-b， C-f，C-n 和 C-p 有三个原因。第一，它们能在所有类型的终端上工作。第二，你获得了使用 Emacs 的锻炼，你将会发现输入这些 CTRL 加字符比按方 向键要快(因为你不必把你的手从键盘上移开)。第三，一旦你养成了使用这些 CTRL 加字符的命令的习惯，你就能一样容易的学习其他高级的光标移动命令。

大多数 Emacs 命令接收一个数字参数；对大多数命令来说，这表示命令重复的次数。输入重复命令次数的方法是在输入命令之前按 C-u 和数 字。如果你有 META(或 EDIT 或 ALT)键，则有另一种方法输入数字参数：在按住 META 键的时候输入数字，我们建议学习 C-u 方法，因为它能在任何终端上工作。

例如，C-u 8 C-f 向前移动 8 个字符。

>> 试着使用带数字参数的 C-n 或 C-p，只用一个命令就把光标移动到与本行相邻的 行上。

绝大多数命令把数字参数当作重复次数，但也有几个例外。C-v 和 M-v 就是。当给出一个参数，只是上滚或下滚数字指定的行数而不是屏数。比如，C-u 4 C-v 滚动 4 行屏幕。

>> 现在试试 C-u 8 C-v。

这将使屏幕滚动 8 行，如果你想往回滚动的话，键入一个带参数的 M-v。

如果你正在使用 X 窗口，在 Emacs 窗口的左手边有一个叫做滚动条的矩形区域。你能通过用鼠标点击滚动条来滚动文本。

>> 试着在滚动条顶部的高亮区域点击中键。这将使文本滚动，滚动的位置取决于 你点击的长短。

>> 试着按住鼠标中键上移或下移鼠标，你将看到当你移动鼠标时文本会上下滚动。

* 当 EMACS 挂起时(WHEN EMACS IS HUNG)

当 Emacs 停止响应你的命令时，你能用 C-g 把它安全的停止。当一个命令执行了太长的时间时你可以用 C-g 把它终止。

你也可以用 C-g 来取消数字参数和输入后又不想执行的命令。

>> 键入 C-u 100 产生一个值为 100 的数字参数，然后按 C-g。再按 C-f。它只会移动 一个字符，因为你用 C-g 取消了参数。

如果错误的输入了一个<ESC>，你能用 C-g 消掉它。

* 禁止命令(DISABLED COMMAND)

一些 Emacs 命令是“禁止”的，所以新手不会因偶然而执行它。

如果你键入了一个禁止命令，Emacs 会显示一条消息说明这条命令是干什么的，并且问你是否需要执行它。

如果你真的想要执行，敲空格键继续。通常，如果你不想执行禁止命令，用“n”来回答。

>> 输入<ESC>:(一条禁止命令)，然后用 n 来回答。

* 窗口(WINDOWS)

Emacs 能有好几个窗口，每一个显示自己的文本。我们将在后面解释怎样对多窗口操作。现在我们要解释怎样去除多余的窗口屏回到基本的单窗口编辑状态。这是一个例子：

C-x 1 一个窗口(也就是除去其他所有的窗口)。

因为 Control-x 跟了数字 1。C-x 1 使包含光标的窗口占满整个屏幕，屏删除其他所有窗口。

>> 把光标移动本行并输入 C-u 0 C-l。

>> 键入 Control-h k Control-f。 看这个窗口如何缩小，并在按 Control-f 的时候出现了一个新的文档窗

口。

>> 键入 C-x 1 并且看到那个文档窗口消失了。

* 插入和删除(INSERTING AND DELETING)

如果你要插入文本，只须输入文本。输入的字符你能见到，比如 A，7，*等等。Emacs 会立即把它们插入。键入<Return>(回车键)插入一个换行符。

你能用<Delete>删除你输入的最后一个字符。<Delete>就是键盘上标着“Del”的键。在某些情况下，“Backspace”键作用和<Delete>一样，但不总是这样！

通常，<Delete>立即删除光标前面的那个字符。

>> 输入几个字符，然后用<Delete>删除它们。不必担心这个文件回被改变；你不会 替换主教程。这只是你的个人拷贝。

当一行文本太长而超过屏幕宽度时，这一行会在屏幕的下一行被“继续”。文本的右边会有一个反斜杠“\”表示它被继续。

>> 插入文本直到最右边，然后再插入。你将看到一个继续了的行。

>> 使用<Delete>删除文本直到行的长度在屏幕的宽度以内。继续的行将会消失。

你能像删除其他任何字符一样删除换行符。删除两个行间的换行符会使它们合并为一行。如果这一行很长屏幕显示不下的话，将会用一个继续的行来表示。

>> 把光标移动到一行的开头按<Delete>这将使本行和上一行合为一行。

>> 按<Return>重新插入你删除的换行符。

记住大多数的 Emacs 命令能接收一个重复次数。这包括文本字符，把一个文本字符重复的插入几次。

>> 键入这个-- C-u 8 * 来插入 ********

你现在已经学习了 Emacs 的大多数输入和排错的方法。你也能一样的删除单词或行。这是删除操作的摘要：

<Delete>	删除光标前面的字符
C-d	删除光标后面的字符
M-<Delete>	除去光标前面的单词
M-d	除去光标后面的单词
C-k	除去从光标位置到行尾的内容
M-k	除去到当前句子的末尾

注意比较<Delete>，C-d 与 M<Delete>，M-d 和 C-f，M-f(<Delete>不是一个控制字符，但不用担心)。C-k 和 M-k 就象 C-e，M-e。

当你一次除去不止一个字符时，Emacs 将保存着这些文本，所以你可以恢复它们。恢复那些被除去的文本称作“拉(yanking)”。你能在除去文本的同一地方拉回它们，或是在文本的其他地方。你能对文本拉上几次以产生它们的多个拷贝，拉的命令是 C-y。

注意“除去(killing)”与“删除(Deleting)”之间的区别，被除去的东西能被拉回来，而被删除的不能。通常除去能除去很多的文本屏保存，而删除只能除去一个字符，或是空行或空格，并且不保存。

>> 把光标移到一个空行的开头，键入 C-k 除去这一行。

>> 按第二次 C-k，你将看到剩下的空行也被除去了。

注意单个的 C-k 除去行的内容，第二个 C-k 除去行本身，并且使后面的所有行上移。特别要注意数字参数：它除去很多行和它们的内容，这不仅仅是重复。C-u 2 C-k 除去两行和它们剩下的空行；而按两次 C-k 并不会这样做。

要在当前光标处找回上次被除去的文本；按 C-y

>> 试一试，用 C-y 把文本拉回来。

把 C-y 考虑为你把某人从你这里拿走的东西再拿回来。注意你如果在一行上按了几次 C-y，所有被除去的文本是存在一起的，所以按一次 C-y 将拉回全部的行。

>> 现在就试一下，按几次 C-k。 现在找回被除去的文本；

>> 按 C-y。然后把光标下移几行再按一次 C-y，你现在会看到怎样拷贝这些文本。

当你要拉回一些被除去的文本该怎样做呢？C-y 只能拉回最近被除去的文本。但以前的文本并没有消失。你能用 M-y 来恢复它。当你用 C-y 拉 回最近被除去的文本后，换成 M-y 可以拉回以前被除去的文本。键入一次又一次的 M-y 可以拉回更早以前被除去的文本。当你找到要寻找的文本，不必做任何事来保持它，只须离开拉文本的 地方继续你的编辑。

如果你 M-y 了足够多的次数，你会回到开始点(最近被除去的)。

>> 除掉一行，移开，再除掉另一行。 然后用 C-y 拉回第二行。 然后换成 M-y 拉回被除掉的第一行。 再按一次 M-y 看看得到了什么。继续按直到拉回被除去的第二行；然后再做几次。 如果原意的话，你可以给 M-y 加正的或负的数字参数。

*** 撤销(UNDO)**

如果你对文本作了一些改动，然后又发现这样做是错误的，你能用撤销命令，C-x u 撤销这些改变。

通常，一次 C-x u 撤销一个改变；如果你在一行上重复几次 C-x u，就会重复几次撤销操作。

但有两个例外：不改变文本的操作(包括光标移动和滚屏命令)不算在内；只能处理 20 次。

>> 用 C-k 除去这一行，然后按 C-x u 它将重现出来。

C-_是一个可选择的撤销命令；它所作的工作和 C-x u 完全一样，只是更容易输入。 C-_的缺点是有些键盘上没有它，这就是为什么还提供 C-x u 的原因。在某些终端上你可以按住 CTRL 的时候再敲/来输入 C-_。C-_或 C-x u 把数参数字当作重复次数。

* 文件(FILES)

为了永久保存你编辑的文本，你必须把它放到一个文件里。否则当你退出 Emacs 的时候它就会消失。你通过“查找(finding)”文件，把你编辑的内容放到文件里。(也称为 “访问(visiting)文件”)。

(译注：为了保持与原文的一致性，把 find 译为“查找”，但是这里和后面出现的 “查找文件”指的都是打开文件的意思。)

查找(finding)一个文件意味着你在 Emacs 里看文件的内容，在多数情况下，也就是你在编辑它。但是，你用 Emacs 对它作的改变 并不是永久行的，除非你“保存(saving)”它。所以你可以避免把一个改了一半的文件留在系统上。甚至你保存了文件，Emacs 也会把原始文件换个名字保留下来，以防过后你发现对文件的改动是错误 的。

如果你观察屏幕的你将看见一个开始和结尾都是破折号的行，并且以“--:**-- TUTORIAL”或之类的东西开始。屏幕的这部分通常显示你正在访问的文件的名字。现在，一个叫做“TUTORAL”的文件，它是你的 Emacs 教程的 个人拷贝。当你用 Emacs 查找一个文件，文件名会出现在同样的位置。

查找和保存文件命令不像前面学的那些命令。它们都以字符 Control-x 开始。以 Control-x 起头的是一个完整的命令系列；它们中的许多都是对文件，缓冲，和相关的东西进行操作的。这些命令有两个，三个或四个字符长。

关于查找文件命令的另一件事是你必须给出你需要的文件的文件名。我们说这个命令 “从终端读取一个参数”。(在这种情况下，参数是文件的名字)；当你键入命令 C-x C-f 后，Emacs 会提示你输入文件的名字。你输入的文件名会出现在屏幕底部的行上。这个底部的行称为微型缓冲(minibuffer)用于这类较短的 输入。你能用 Emacs 本身的编辑命令来编辑文件名。

当你正在输入文件名(或其他任何微型缓冲区输入)，你能用命令 C-g 来取消。

>> 键入命令 C-x C-f，然后输入 C-g。这将取消微型缓冲，也取消了 C-x C-f 命令所使 用的微型缓冲，所以你不查找任何文件。

当你输完文件名后用<Return>来结束。然后 C-x C-f 开始工作，并开始寻找你所选择的文件。当 C-x C-f 命令结束后微型缓冲区也消失了。

过一小会儿文件的内容就会显示在屏幕上，然后你就能对它进行编辑了。当想永久保留你的改动时用命令：

C-x C-s　保存文件(save the file)。

这个操作会把 Emacs 里的文本拷贝到文件里。在你第一次作的时候，Emacs 把原始文 件改为一个新名字以使它不至于丢失。新名字是在原来名字的后面加一个“~”。

保存结束后，Emacs 打印出被写的文件的文件名。你应当经常的保存，万一系统崩溃 的话你不至于丢失太多的工作。

>> 键入 C-x C-s 来保存你的教程的拷贝。屏幕的底部会打印出“Wrote.....TUTORIAL”。

注意：在某些系统上，输入 C-x C-s 会把屏幕冻结住使你从 Emacs 看不到更多的输出。这 表示这个操作系统的“特性”叫做“控制流程”，它拦截了 C-x 不让它到达 Emacs 那里。 要使屏幕解冻，输入 C-q，然后看 Emacs 手册里的“Spontaneous Entry to Incremental Search”一节，按上面的建议来对付这种“特性”。

你能查找一个已存在的文件，来查看它或编辑它。你也可以查找一个尚未存在的文件。这是 Emacs：里创建文件的方法：查找文件，将会出现一个空白，然后插入文件的文本。当你 “保存(saving)”的时候，Emacs 将会用你插入的文本创建文件。从那时候起，你可以认为你在编辑一个存在的文件了。

* (缓冲)BUFFERS

如果你用 C-x C-f 查找第二个文件，第一个文件仍然留在 Emacs 里。你可以再用 C-x C-f 查找一次来切换回去。用这种方法你在 Emacs 里有很多文件。

>> 输入 C-x C-f foo <Return>来建立一个名为 foo 的文件。然后插入一些文本，编辑它，并 用 C-x C-s 来保存“foo”。 最后输入 C-x C-f TUTORIAL<Return>以回到本教程。

Emacs 把每个文件的文本都保存在一个叫“缓冲(buffer)”的东西里。查找(打开)一个文件就会在 Emacs 里产生一个新的缓冲。要看你当前运行的 Emacs 里存在的缓冲列表，输入：

C-x C-b　列出缓冲(list buffers)

>> 输入 C-x C-b

观察每个缓冲都有一个名字，它可能也有一个它所保存的文件的文件名。一些缓冲不对应文件。比如，叫“*Buffers List*”的缓冲没有任何文件。这个缓冲只包含由 C-x C-b 产生的缓冲列表。你在 Emacs 窗口里看到的任何文本都是某个缓冲的一部分。

>> 输入 C-x 1 消除缓冲列表。

如果你对一个文件的文本作了改动，然后查找另一个文件，第一个文件并不保存。它的改变保存在 Emacs 里，在那个文件的缓冲里。被建立或编 辑的第二个文件的缓冲并不影响第一个的。这一点很有用，但这也意味着要有一个便捷的方法来保存第一个文件的缓冲。如果要用 C-x C-f 切换回去只是为了按 C-x C-s 保存它将会是一件令人讨厌的事。所以我们用

C-x s　保存缓冲(save the buffer)

C-x s 向你询问每个改动过但未存盘的缓冲，对每个这样的缓冲都询问是否保存。

>> 插入一行文本，然后按 C-x s。 将会问你是否保存叫 TUTORIAL 的缓冲。 输入“y”来回答是。

* 扩展命令集(EXTENDING THE COMMAND SET)

有太多的 Emacs 命令，大大超过了 Contorl 和 meta 加上字符所能表示的数量。Emacs 用 X(扩展 eXtand)命令来解决这个问题。有两种风格：

C-x　字符扩展，后跟一个字符。
M-x　名字命令扩展，后跟一个长名字。

这些命令通常有用，但不如你已经学过的那些命令使用的频繁。你已经见过了它们中的两个：文件命令 C-x C-f 用于查找和 C-x C-s 用于保存。

另一个例子是结束 Emacs 的命令 C-x C-c(不必担心你所作的改动会丢失，在退出 Emacs 之前，C-x C-c 会提示你保存每一个改动过的文件)。

C-z 命令用于*临时*退出 Emacs，所以你能回到原来运行的 Emacs 里。在允许这样做的系统上，C-z 把 Emacs“挂起”；就是说回到外壳(shell)下，但并不破坏运行的 Emacs。在大多数外壳上，你能用‘fg’命令或‘%emacs’来继续 Emacs。

在不支持挂起的系统上，C-z 建立一个子外壳(subshell)运行于 Emacs 下以使你能运行其他程序然后回到 Emacs；这并不是真 正的“退出”Emacs。在这种情况下，通常从子外壳回到 Emacs 的外壳命令是‘exit’。 有很多 C-x 命令，这是已学过的一个列表：

C-x C-f　查找文件
C-x C-s　保存文件
C-x C-b　缓冲列表
C-x C-c　退出 Emacs

C-x u　撤销操作

被称作扩展命令的命令的使用频率都不太高。或者是只在某些模式下使用。一个例子是替换字符串的命令，它在全文里把字符串替换为其他的。当你 键入 M-x， Emacs 会在屏幕的底部提示你输入命令；在这种情况下，是“replace-string”。比如输入“repl s<TAB>”， Emacs 会把命令补全。用<Return>来结束命令。

替换字符串命令要求两个参数--要被替换的字符串和用来替换的字符串。你必须用<Return> 来结束两个参数。

>> 把光标移上两行，然后输入 M-x repl s<TAB><Return>changed<Return>altered <Return>。 注意现在这一行改变了：你把光标初始位置后的所有单词 c-h-a-n-g-e-d 替换为了 “altered”

* 自动保存(AUTO S***E)

当你改动了一个文件还未存盘的话，所作的改动也许会由于系统崩溃而丢失。为防止这种情况发生，Emacs 在编辑时为每个文件提供了“自动保 存(auto save)”。自动保存的文件的文件名前后都有一个#号；例如，如果你编辑的文件名叫“hello.c”，自动保存的文件的文件名就叫 “#hello.c#”。当你正常的保存了文件后，Emacs 会删除这个自动保存的文件。如果遇到死机，你能打开那个文件后按 M-x recover file<Return>来恢复你的编辑，(是你编辑的文件而不是自动保存的文件)。当提示确认时，输入 yes<Return> 来继续恢复自动保存的数据。

* 回显区域(ECHO AREA)

如果 Emacs 发现你输入命令的速度很慢的话它会在屏幕底部为你显示出来，这个区域叫 “回显区域”。

* 模式行(MODE LINE)

回显区域上面的一行称为“模式行(mode line)”。模式行显示与下面类似的东西：

--**-Emacs: TUTORIAL (Fundamental)--L670--58%----------

这一行给出了有关你在编辑的文件和 Emacs 状态的有用信息。

你已经知道了文件名意味着什么。--NN%--指出你现在在文本里的位置；它意味着上面还有 NN%的文本。如果是在文件的开头，会用 --Top-- 来代替--0%--。如果是在行的末尾，会显示--Bot--。如果你正在看的文本内容很少，可以全部显示在屏幕上，模式行会说 --All--。

前面的星号表示你已经改动过文本了。一旦你保存了文件或打开了一个新文件，模式行的这部分就不是星号而是破折号了。

模式行上括号里的部分是现在的编辑模式。现在是缺省的基本(Fundamental)模式。它是 “主模式(major mode)”的一种。

Emacs 有很多不同的主模式。有些意味着不同的语言或不同的文本。如 Lisp 模式(Lisp mode)，文本模式(text mode)等等。在任何时候有且只能有一种主模式被激活。并且它的名字会出现在现在显示 “Fundamental”的位置上。

每一个主模式都有些自己的命令。就象不同的编程语言的注释看起来不同一样。每种主模式插入的注释也不同。可以用扩展命令切换进某种主模式。例如，M-x fundamental-mode 是切换进基本模式。

>> 输入 M-x text-mode<Return>

不必担心，没有命令会给 Emacs 带来很大改变。但是你可以看到现在 M-f 和 M-b 把省略号当作单词的一部分。而先前，在基本模式里，M-f 和 M-b 把省略号当成当成分隔符。

主模式通常作诸如此类微小的变化：大多数命令在每个主模式里作“同样的工作”，但又有些微小的不同。

要观看关于你现在的主模式的文档，按 C-h m。

>> 键入 C-u C-v 一次和多次使本行接近屏幕的顶端。
>> 输入 C-h m，看看文本模式和基本模式有些什么不同。
>> 按 C-x 1 从屏幕上关掉这个文档。

主模式之所以叫做主模式是因为也存在从模式，从模式与主模式完全不同。每个从模式可以自己打开或者关闭，独立于所有其他从模式，也独立于你的主模式。所以你可以不用从模式或者同时用很多种从模式。

有一种从模式很有用，特别是在编辑英文文本时。它是自动填充模式(auto fill mode)。当这个模式打开的时候，当输入的文本过宽的时候就会自动折行。

你能用 M-x auto-fill-mode<Return>来打开自动填充模式。如果此模式已经打开 M-x auto- fill-mode<Return>则把它关闭。我们把这叫做切换开关。

>> 输入 M-x auto-fill-mode<Return>。然后插入一些“asdf”直到看到这行被分为两行。你必须在中间放一些空格，只有到空格的时候才会换行。

通常边界宽度是 70，但你能用带数字参数的 C-x f 命令来改变它。

>> 键入带参数 20 的 C-x。(C-u 20 C-x f) 然后输入一些文本看现在每行只有 20 个字符了。然后用 C-x f 把它改回 70。

如果你在一个段落的中间产生了改变，自动填充模式将不会重新填充。要想重新填充段落，当光标在段落里的时候按 M-q。

>> 把光标移到上一段按 M-q。

* 搜索(SEARCHING)

Emacs 能朝前和朝后搜索字符串(指相邻的一些字符或单词)。搜索是一个移动光标的操作，它把光标移动到字符串出现的下一个地方。

Emacs 的搜索命令和其他大多数编辑器不同，它是“增量式(incremental)”的，这意味着搜索在你键入字符串时就开始了。

开始一个向前搜索的命令是 C-s，C-r 是往回搜索。但等等，先别忙。

当你输入 C-s 是你将注意到在回显区域会出现一个字符串“I-search”。这告诉你 Emacs 开始了一个增量搜索，并在等待你输入要搜索的东西。<RET>结束查询。

>> 现在键入 C-s 开始一个搜索。慢慢的输入单词‘cousor’，在输入每一个字母的时候停顿一 下，注意看光标发生了什么。
>> 再输入一次 C-s，来搜索“cursor”出现的下一个地方。
>> 现在输入<Delete>四次看看光标移到了哪里。
>> 输入<RET>结束搜索。

看到发生什么了吗？在 Emacs 的增量搜索里，你输入多少字符串它就试着搜索这些字符出现的地方。到字符串出现的下一个地方，只须再按一次 C-s。要搜索的字符串不存在的话，Emacs 会发出蜂鸣并告诉你当前的搜索“失败(failing)”，按 C-g 也是终止搜索。

注意：在某些系统上，输入 C-s 会把屏幕冻结住使你从 Emacs 看不到更多的输出。这 表示这个操作系统的“特性”叫做“控制流程”，它拦截了 C-s 不让它到达 Emacs 那里。 要使屏幕解冻，输入 C-q，然后看 Emacs 手册里的“Spontaneous Entry to Incremental Search”一节，按上面的建议来对付这种“特性”。

如果你在搜索的过程里按了<Delete>，你将注意到要搜索的字符串的最后一个字符会被删除并且光标会回到上一个被搜索到的 地方。比如，假设你键入了“c”，将会搜索“c”第一次出现的地方。然后如果你键入“u”，光标将移到“ cu”第一次出现的地方。现在键入<Delete>。这将从搜索的字符串里把“u”删掉，这时光标回到“c”第一次出现的地方。

如果你在搜索时按了 Control 或 meta 键加字符(少数几个少数命令例外，如 C-s 和 C-r)，搜索将被终止。

C-s 向当前光标的后面搜索字符串出现的地方。如果你需要搜索前面文本里的东西，用 C-r 来代替。我们所介绍的 C-s 的每个特性 C-r 也支持，除了方向相反。

* 多窗口(MULTIPLE WINDOWS)

Emacs 有一个非常好的特性是能同时在屏幕上显示不止一个的窗口。

>> 把光标移到本行上按 C-u 0 C-l。

>> 现在按 C-x 2，它把屏幕分裂成两个窗口，每个窗口都显示本教程。光标在上面的窗口里。

>> 按 C-M-v 滚动到下面的窗口里。(如果你没有一个真正的 Meta 键，那么按 ESC C-v)

>> 按 C-x o (“o” 指 “其他(other)”) 把光标移到到下面的窗口里。
>> 用 C-v 和 M-v 滚动下面窗口里的文本。 在上面的窗口里看本教程。

>> 再次按 C-x o 使光标回到上面的窗口里。 现在光标象以前一样在上面的窗口里了。

你能一直用 C-x o 在窗口间切换。每个窗口都有它自己的光标位置，但仅有一个窗口能显示活动的光标。所有的编辑命令都发生在那个显示光标的窗口上。我们把这个窗口叫做“选中窗口(selected window)”。

当你在一个窗口里编辑文本，而用另一个窗口作参考时命令 C-M-v 非常有用。你总是能把光标留在所编辑的地方，而用 C-M-v 来翻阅另一窗口。

C-M-v 是 CONTROL-META 加字符的一个例子。 如果你有一个真正的 META 键，你能同时按住 CTRL 和 META 再按“v”来输入 C-M-v。CTRL 和 META 谁先按谁后按无所谓。

如果你没有一个真正的 META 键， 你可以用 ESC 来代替。这时候次序是有关系的： 你必须让 ESC 跟在 CTRL-v 后面; 否则 CTRL-ESC v 将不工作。 这是因为 ESC 是一个有意义的字符而不是一个修饰字符。

>> 输入 C-x 1 (在上面的窗口里) 以消除下面的窗口。

(如果你在下面的窗口里键入 C-x 1，将会把上面的窗口去掉。可以把这个命令看作是“只保留你现在在的那个窗口。)

你不必一定要在两个窗口里显示同样的缓冲。如果你在一个窗口里键入 C-x C-f 查找文件，另一个窗口的内容不会改变。你能独立的在每个窗口里查找文件。

这是让两个窗口显示不同内容的另一种方法：

>> 在你输入的文件名后再输入 C-x 4 C-f，然后用<Return>结束。会看到指定的文件出现在下面 的窗口里。光标也在那里面。

>> 键入 C-x o 回到上面的窗口，然后输入 C-x 1 删掉下面的窗口。

* 递归编辑层(RECURSIVE EDITING LEVELS)

有时候你会进入“递归编辑层(recursive editing level)”。由模式行上的方括号指示。它在主模式名的括号外面。例如你也许会看到(Fundamental)变成了[(Fundamental)]。

要退出递归编辑层，按 ESC ESC ESC。这是一个通用的退出命令，你也可以用它除去额外的窗口，或者退出微型缓冲。

>> 输入 M-x 进入一个微型缓冲； 然后用 ESC ESC ESC 离开。

你不能用 C-g 来退出递归编辑层。这是因为 C-g 只能取消在递归编辑层里面的命令。

* 获取更多的帮助(GETTING MORE HELP)

在本教程里我们试着为你开始使用 Emacs 提供了足够多的信息。但是有关 Emacs 的信息实在是太多以至于不能全部都在这里说明。但是，你 还应该学习更多有关 Emacs 的东西，因为它另外还有很多有用的特性。Emacs 提供了很多读取有关命令的文档的命令。这些“帮助”命令都以 Control-h 开头，叫做“帮助字符”。

为了使用帮助特性，输入字符 C-h，然后再输入一个字符来说明你需要哪种帮助。如果你真的不知道，输入 C-h ? 然后 Emacs 会告诉你它能给你什么样的帮助。如果你输入了 C-h 又觉得不需要任何帮助，你可以用 C-g 来取消它。

(在有的地方，C-h 的作用被改变了。如果按 C-h 在屏幕的底部没有出现有关帮助的信息的话，试试用 F1 和 M-x help RET 来代替。)

最基本的帮助特性是 C-h c。输入 C-h，然后是字符 c，然后输入一个命令字符和序列；然后 Emacs 会显示这个命令的简洁的描述。

>> 输入 C-h c Control-p.

显示的消息看起来会象这样：

    C-p runs the command previous-line

这告诉你“功能的名字”。功能的名字主要用于对 Emacs 的功能扩充和定制。但因为功能的名字指出了命令的用途，所以最好不要改动它。

C-h c 后面可跟多字符命令，比如 C-x C-s 和 (如果你没有 META 或者 EDIT 或者 ALT 键) <ESC>v 。

要获取有关命令的更多信息，用 C-h k 代替 C-h c。

>> 输入 C-h k Control-p.

这将在一个 Emacs 窗口里显示命令的文档。当你读完后可以用 C-x 1 除去帮助文本。如果不想马上离开，你可以一边编辑一边参考帮助文本，然后再按 C-x 1。

这是一些有用的 C-h 选项：

    C-h f 描述一个功能，在你输入了这个功能的名字后。

>> 输入 C-h f previous-line<Return>。 将打印出 C-p 命令所实现的所有功能。

    C-h a 命令查找。输入一个关键字，Emacs 将列出所有名字里有这个关键字的命令。 包括所有以 Meta-x 开始的命令。对有些命令，C-h a 也将列出实现同一功能的 几个命令序列。

>> 输入 C-h a file<Return>.

这将在窗口里显示所有名字里有单词“file”的 M-x 命令。

>> 输入 C-M-v 来滚动帮助窗口，做上几次。

>> 输入 C-x 1 来删除帮助窗口。

Emacs 启动:
直接打 emacs, 如果有 X-windows 就会开视窗. 如果不想用 X 的版本,
就用 emacs -nw (No windows)起动.

符号说明
C-X 表示按住 CTRL 键, 然後按 X, 再把 CTRL, X 一起放开.
M-X META META

在没有 META 键的电脑上, M-X 等於先按 ESC 键, 接著按 X 键.
Sun 上面 META 键就是菱形的那个键.
有些系统 META 键就是 ALT 键.(或者某一边的 ALT 键)

C-X 或 M-X 的 X 没有大小写分别.
Emacs 按键命令基本上是一串 C-<chr>和 M-<chr>组成的.
超过两个以上的按键命令, Emacs 会在萤幕最下面一行显示你按过什麼.
这一行叫作 mini buffer

结束 Emacs 按 C-x C-c

取消执行 C-g
有些 Emacs 命令会跑很久, 可以用 C-g 中断之. 按错键也可以按 C-g 取消.

上下移动 C-p 向上 (previous line) C-n 向下(next line)
左右移动 C-f 向右 (forward) C-b 向左 (backward)

其实 Emacs 内部没有行的概念, 把一篇文章放在一个大 buffer
里面, 所以 C-f (forward)就是向档尾移动, C-b (backward)
是移回去的意思, 一次一个字.

翻页 下一页 C-v (view next screen)
上一页 M-v

翻页时,上一页末尾会留一点在萤幕最上面,以维持连续性.

Emacs 在游标接近萤幕最下方时会自动跳半页, 把档案往前挪一点,
方便阅读.

重画萤幕 C-L

Emacs 里面游标的专有名词叫 point. point == 游标目前的 点

游标一次跳一个字(word) M-f 往後跳 M-b

  注意 C-f 与 M-f, C-b 与 M-b 的对称性.

移到行头 C-a 行尾 C-e
移到句首 M-a 到句尾 M-e
  (M-a 到上一个句点後面,一个句子的起头.
   M-e 到句点後面)

移到档头 M-< 档尾 M->

删除游标目前指的/後面的字 C-d
        前面的字 DEL (Delete 键)
DEL 的正名叫 Rubout (Rub out)

M-DEL 往回删一个字(word)
M-d   往前删 (游标後面)
C-k   删至行尾 (kill)
M-k   删到一句子结尾(删到句点) (kill)

注意 Backspace = C-h 在 Emacs 下是 help 的意思
後面有(kill)的, 表示此删除的动作是 kill, 不太等於 delete.
emacs 会把 kill 掉的东西放到 kill ring 去, 算是一种暂存的地方,
以後可以叫出来.见 yank 说明.

Undo: C-x u
C-_ 等於 C-x u   有些 DEC 终端机, C-/就是 C-_
                  有时等於 C-Shift- -

重复执行

举例, 向右移 8 个字, C-u 8 C-f
C-u 在 Emacs 里是蛮特别的,用来设定一些引数(argument/repeat count)
给其後的命令.

C-u 2 0 C-n 向下移 20 行

有一个特别的例外, C-u 3 C-v 不是翻三页, 而是整个萤幕向上移三行.
据说这比较有意义.

C-u 1 0 C-x u UNDO 10 次

给 C-L 一个引数会怎麼样:
C-u 0 C-l 会重画萤幕,并且把目前的行移到萤幕第一行.

另外, C-u 100 等於 M-100
    C-u 数字 等於 M-数字

X windows 下,
C-left C-right 一次移一个字(word).
C-up C-down 移动一段 (paragraphs/C 语言的话是 block)
Home = C-a
End = C-e
C-Home = M-<
C-end = M->
PgUp PgDn = M-v C-v
设定重覆次数更加简单,
比如要向右移 10 个字 C-1 C-0 right-arrow
 就是按住 CTRL, 然後打 10 就对了, 比 C-u 1 0 简单.

Mouse 中键用来选取有 hi-light 的地方.
    右键是 menu-button

如果不小心按两次 ESC, 等於 M-ESC, 会有一个讯息跑出来
说你按到一个被 disable 的命令. 这是高级指令, 作者认为
初学者用不道,所以会问你要不要启动它, 一般回答 no.

如果某一行太长, 萤幕显示不下, Emacs 会在萤幕最右边打个$,
表示此行未完,右边还有.

把一行拆成两行: 在想拆处按 Enter 即可.
合并两行为一行: 在行尾按 C-d (或行首按 DEL)

Yank: 吐出被删掉的(killed)东西.

只要用 kill (C-k, M-k 等) 删除, 超过一个字的资料,
emacs 就会把它存起来, 然後 C-y 可以把它叫出来.

功能跟 Cut & Paste 一样. Kill 和 delete 不一样, 只有被
kill 掉的东西才能用 yank 吐回来.

游标在同一地方不动, 连续 kill 掉的资料会被当成一次 kill 掉的,
yank 时会一起回来.

被 Kill 掉的资料是放在称作 kill ring 的资料结构上面, ring 就是个圆圈,
被 kill 掉的东西会依序摆在圆圈上. yank 会放回最近一次 kill 掉的资料.
如果不是你想要的话, 用 M-y 可以换. (M-y 就是告诉 emacs, 不对不对,
我不是要这一个,换前一个给我).

M-y 要紧接在 C-y 之後.

拷贝文字的方法== 连续 C-k 几次, 把要拷贝的行全部删掉,
然後按 C-y 弄回来. 再到想复制的地方按一次 C-y, 就成了.

把要拷贝的资料 kill 掉在 yank 回来好像很笨. 是有比较文明的
方法, 那就是 M-w, 不过较麻烦.

首先,要先设标记. Mark 用 C-SPC 或 C-@ 设. 然候
把游标移到另一端, 按 M-w 就可以把 mark 到 point 间的
字存到 kill ring 上. point 就是游标的意思.

Emacs 不会把 Mark 起来的地方用 highlight 表示, 除非在 X 下.
在 X 下, 可以用 M-w 来拷贝用滑鼠反白的文字.

kill & yank 就是 cut & paste 的意思.

以上大部份指令对 Bash 的命令列编辑也有效

档案操作
读档: Emacs 术语叫 finding a file.
C-x C-f 然後在 mini-buffer 输入档名. 输入档名时, SPC 键有
auto-complete 的功能,或者会秀出到目前为止档名前几
个字和输入一样的. (TAB 键也有类似功能)

C-x C-f 叫 find-file

C-x C-s 存档 (save current file, save current buffer)

C-x s　存所有的档

C-x i　插入档案 把另外的档案的内容读入目前编辑区内

视窗
Emacs 把档案读进来,存在 buffer 中.
我们透过 window 来看/编辑 buffer.

两个视窗会把萤幕切成两部份, 他们可以同时显示
相同的, 或不同的档案.

对初学者而言, 最需要的是记住怎样让不想要的视窗消失:

C-x 0 关掉目前的视窗
C-x 1 会让目前的视窗占满整个萤幕 (One Window),
　　取消/关掉其他的视窗.

Emacs 里面有许多功能都会开一个小视窗来和使用者沟通, 显示讯息.
有时候不会自动消失很讨厌, C-x 1 就很有用.

另一个功能是如何跳到另一个视窗.
C-x o (other-window)

C-x 2 把目前的视窗切成两个 (水平分割)
C-x 3 (垂直分割)
C-x 4 是一串与视窗有关的指令.
C-x 4 是一串与视窗有关的指令.
C-x 5 则是扩展到 X 的视窗, 称为 frame.
　　C-x 5 2 就是再开另一个 X 视窗 (frame).

多档编辑
C-x C-b 看目前有那些 buffer (buffer 就是 emacs 放开起的档案的地方).
C-x b 然後在 minibuffer 输入 buffer 的名字,可以切换编辑 buffer.
　　TAB 键也有作用. 有些内部的 buffer (就是没有档案的 buffer),
　　是用*开头和结束, 这个也要打, 如*scratch*

最候提醒:
　C-x 1 可以把多馀的视窗关掉.

Emacs 扩充指令

前面介绍的 emacs 按键大部份都是 C-<chr> 或者 M-<chr>的形式.
这是最简单的按法, 由一对按键构成一个指令.

Emacs 的按键可以超过 2 个以上. 如 C-x 1 或 C-x C-b.
一般超过一个按键组合的命令都是用 C-x 开头.

另外你也可以直接下命令. 按 M-x 之後就可以打一个 Emacs 命
令来执行. 一般这些命令名字都很长, 不过都不常用. 等一下
我们会介绍一些. 还有介绍怎麼把这些命令设成按键指令.

C-x C-c 就是结束 Emacs. 不过一般 Emacs 很笨重, 一旦起动就不轻易
退出. 所以比较常用的是 C-z

C-z 把 Emacs 暂停, 回到命令列. 当你下次再需要编辑时,打 fg %emacs
就可以把 Emacs 唤醒.

在 X 下, C-z 会把 emacs 缩成 icon

mode line

emacs 编辑画面由 编辑区(buffer) 状态列 (modeline) 和对话区 (minibuffer)
构成. 这里解释 modeline 显示的讯息.

以下是个范例:

--**-XEmacs: xemacs.qs (Fundamental)----74%-------

由後面往前解释, 74% 表示游标的位置.

(Fundamental)表示编辑模式.这是最原始的模式. 编辑不同种类的文章
可能希望用不同的模式, 比如说 C-mode, lisp-mode, tex-mode, text-mode
等等. 在不同模式下可能多一些按键出来. 举例 text-mode.
M-x text-mode
可以切入 text-mode, 这是一般人编辑文字使用的模式. 和 Fundamental mode
没什麼差异. 不过游标移动时, Emacs 对一个字的定义就有所不同, 因而
M-f M-b 等移动一个字, 一个段落的指令就可能会停在标点符号的前面.

此时状态列变为... (Text)----70%---

以上说的是 Major mode. 另外还有 minor mode, 其实就是一些额外的功能.
比如说, M-x auto-fill-mode 则状态列显示 (Text Fill).
auto-fill 就是自动断行, 让文章每行固定有 70 个字.

M-X fundamental-mode 可以变回来.

这里要说明一下, emacs 在 minibuffer 下有 auto-completion 的功能,
也就是打 M-x fund 然後按 SPC, 它会自动补全 fundamental-mode,
不用全打. 如果有两个以上的选择, 它会告诉你. 这个功能对
find-file (C-x C-f)等等档案编辑功能也有效. 前面提过.
最後解释两个**号. 右边的*表示文章被修改过了.
左边的* 表示这个编辑区(buffer)可以修改.
有一些 emacs 的 buffer 是 read-only buffer, 就会标成%
%%表示档案是 read-only.

C-x C-q 可以解开 read-only 的锁定, 无论如何你要改这个编辑区.
这是个 toggle 指令, 如果原来是可以修改的, C-x C-q 会把它切成
read-only.

Search

没有 Search 功能的编辑器简直就是小朋友的玩具. Search
是一项很重要的功能, 所以 emacs 也提供的很完善.
C-s
C-r
M-x re-search-forward
M-x re-search-backward
M-x search-forward
M-x search-backward
以上这些指令是基本的 search 指令. C-s, C-r 是 increamental search,
就是你打字的同时, emacs 就直接帮你找. 一个是 forward, 一个是 backward.
找到了怎麼办? 按 C-g 可以取消搜寻, 跳回原来的位置. 按 Enter 就让游标
停在找到的地方 -- 此时 minibuffer 显示:Mark saved where search started
什麼意思? 就是 isearch 帮你在原来的位置设了一个 mark, 然後把 point
(cursor) 移到新的位置.
想跳回去原先的地方?
C-x C-x 就可以了.(exchange-point-and-mark)

C-u C-SPC 可以依序跳回前几次设 mark 的地方.
(C-SPC 是设 mark, 给它一个 argument, 就是反动作)
(还记不记得 C-u 可以给後面的指令设一些参数.
有些指令拿这个参数来当作 repeat count,
有些指令就只拿来当作 on/off, true/false, set/clear 而已)

M-x re-search-forward 可以让你用 regular expression 搜寻.
M-x search-forward 则没有 increamental 的功能.

另外一个指令, 作用和 grep 很像:
M-x occure

和 search 相提并论的就是 replace.
M-x replace 然後按 SPC, 就知道了.

Emacs 的设定:

Emacs 的设定档是 $HOME/.emacs
你应该多少知道, emacs 是用 lisp 写成的编辑器, .emacs 档也都是
要用 lisp 的语法设定. emacs 用的 lisp 称为 elisp, 和一般的 lisp 差一点点.

有一个 info page, emacs-lisp-intro, 深入浅出的介绍 emacs lisp.
如果你还不会, 不懂 programming, 强烈建议你看这份文件. 如果你
会 texinfo, 你可以把它很漂亮的印出来. (内容一点点而已, 两三
天就看完了)

如果你把.emacs 搞砸了, 进 emacs 很奇怪, 怎麼办?
1. 用 vi 改 .emacs :>
2. emacs -q 进 emacs

Major Modes

一般常见的 emacs major mode 有
fundamental-mode
text-mode
lisp-mode 有自动对括号/重排, 直接执行 lisp code 功能.
c-mode/cc-mode c-mode 是比较旧的 c-mode, cc-mode 应该是
目前新的 c-mode. 有自动重排/对括号的功能.
也可以在 emacs 内 compile, 跳到 compiler error

修正错误. 执行程式时 debug. (配合 dbx/gdb)
compile 是透过 Makefile 进行.

tex-mode Tex/Latex 编辑模示. 可能是打一些奇怪的标点
符号比较方便.

<programming-language>-mode
同 lisp/cc-mode. 如果是 interpreter 的话,
emacs 通常都可以直接执行/debug.

<programming-language>-mode 还有 tags 的功能, 後述.

html-mode, texinfo-mode, sgml-mode: 编写 html, texi, sgml 之用.

w3-mode WWW browser. 在 x-win 上不满意,但可以接受...

Tags

Tags 是一个显为人知的功能? 所以我想提一下. 这不是 emacs 发明的,
而是 vi 原本的特异功能. emacs 只是发扬光大而已.
假设你有一个目录, 里面是一个程式的原始码, 比如说, tin 的原始码,
放在 ~/tin-1.3beta 下面. 你想看它们.

首先, 叫 emacs cd 到该目录:

M-x cd

然後, 建立 tag table.
tag table 就是一张对照表, 记录哪个符号(variable/function call)
对映到哪个档案的哪个地方. 有这张表, emacs 可以让我们快速的在程
式码内游走. 一般这张表是一个档案, 叫作 TAGS (大写)

M-! etags *.[ch]

M-! 是执行 external shell command 的意思. etags 就是 emacs 的建表程式.
你只要告诉它你的 source code 在那□即可.

vi 的话是使用 ctags 这个程式, 它建出来的档名叫 tags (小写). 因为
我们介绍 emacs, 所以不管它.

然後, 怎麼看程式? 你知道所有的 C 程式都是由 main()开始, 所以你想
找到 main()在哪个档案. 这时只要按 M-. 然後 emacs 会问你 tag table
在哪里. 因为我们已经 cd 到该目录, 直接按 enter 就好了. 然後输入 main,
emacs 就会把你带到 main(){ ... }去.

如果 你看到某个程式片断呼叫一个你没看过的函式, 你可以把游标
移到该函式的名字上, M-. ENTER 就搞定了.

如果 emacs 找错了 (比如有变数和函式同名, emacs 跳到变数去),
那你可以用 C-u M-. 找下一个.

在编辑程式码的时候, M-SPC 很有用, 它会把游标附近的空白缩成一个.
在其它地方也有效.

Emacs 的一些 package:

M-x dired (或 C-x d)
    游走/编辑 目录, 就是档案总管的意思 :)

M-x man 就是 man page
M-x shell 开个 command prompt, 不过不能跑 vi,elm, tin...
M-x gnus 读新闻/读信
M-x rmail 读信
M-x vm view mail
M-x mh-rmail 读信 (package mh-e)
M-x mh-smail 送信 (package mh-e)

强列建议改用 emacs 读 news/bbs. 世界会更美好!

读信的话就要看你的感觉. 这些读信程式都会把信从系统的 mail folder
搬到自己的目录下, 占用 quota, 我不喜欢 :p 建议 elm 或 mutt.
除非参加 mailling list 配合 procmail. 不然不实用.
用 mh-e 须要装 mh 这个外部程式, 不太好. 建议 vm 或 gnus.
写完信, C-c C-c 就可以送信.

如果你的资料用 rcs/sccs 作版本管理, emacs 自动会起动 version control
(minor mode.), c-x c-q 变成 check-in/check-out.

如何取得更多的资讯:

Emacs 的 lisp 经过多年的发展,已成为完整的 self-documenting 系统.
很多东西都可以线上找到你要的资讯.

前面说过,或者你已经不小心按 backspace 遇到了, C-h (就是 backspace

的 ascii 码) 在 emacs 里面是 help 的意思, 它可以带出一串指令.
常用的有:

C-h F Emacs FAQ
C-h t Emacs 使用教学
C-h n Emacs NEWS file, 介绍最近改版的新功能
C-h i Info system. Info 是 gnu 用来取代 man page 的系统,
基本上和文字模示的 WWW 差不多. 有许多重要的资讯
可以在这边找到. 如果你是新手, 建议你在 x-win 下
看. 不然, 按键 m (menuitem), SPC next page
l (last node: node 就是章节的意思) u (up node)
d (directory, 索引). BS (Backspace, back a page).
如果全部只按 SPC, 就跟 man 一样.
C-h k describe key, 告诉你按这个键执行那个 lisp function.
C-h f describe function. 告诉你 function 在作什麼.
如果只按 SPC, emacs 会给你所有 lisp 函数的列表, 和说明.
C-h v describe variable 同 function.
C-h a apropos 的意思(approximate). 给 lisp function 的部份
字串, emacs 帮你找.
C-h b 列出目前所有的 keybinding
C-h m mode help. 列出目前的 mode 的特殊说明.
C-c C-h 列出以 C-c 开头的所有 key-binding. 虽然说 Emacs
可以定义按键, 可是 Ctrl- 开头的所有组合大概都用光了,
只有 C-c 算是可以自定指令. 不过有些 mode 也侵犯这个空间.
目前的 convention 是 C-c <chr> 留给 user, C-c C-<chr>
留给 package.

有以上这些 help, 你的 emacs/elisp 功力会随著时间成长.

Elisp 简介:

Emacs 有三份手册.第一份是使用手册, 第二份是 Elisp 手册, 第三份是
Elisp 简介. 第三份的程度是入门级, 值得看. Elisp 手册其实也写的
很简单, 还教你 lisp, 不过有点长, 适合参考.

因为我 lisp 没有仔细学过, 所以:
以下所言, 如有巧合, 那才是真的.

Basic data type

字串 (string) "Hello, World"
字元 (char) ?a ; 问号开头
atom & list:
(1 2 3 4) 是一个 list, 由 4 个 atom 组成.
pair: 中间是句点.
(apple . 2)
alist (associated list)
就是一堆 pair 的集合,就像 perl/tcl 的 associative array.
或者说是一个资料库, 一堆 (key, value) pair.
'((Apple . 1)
(Orange . 2)
(PineApple . 3))
vector (?)
emacs 19 用 vector 来表示按键(key strok sequence)
[f1] [f2] [f1 a]

nil 就是空的 list, 或者表示 false
t true

Forms

我们写程式最好有样版让我们填空最简单了.
Form 就是样版, 不过意义不太一样.
Form 就是 Elisp 可以接受的句型.
lisp 解译器 预设是对 list 的每个元素求值(evaluate),
除非是 special form, 有特殊的定义. 比如说

```
(defun FUNC (ARG-LIST)
  BODY ...)
```

就是一个 special form, 用来定义函式, 所以 FUNC 不会被
求值, 被当成 symbol, ...

```
(quote (LIST))
```

这也是个 special form, 叫 lisp 把 (LIST)当做 symbol 就好了,
不要 evaluate.

quote 很常用, 所以有个缩写:
'(LIST) 等於 (quote (LIST))
'Asymbol 可以表示一个 Atom, 名称叫 Asymbol

set 可以产生/定义新的变数.

```
(set 'hello 1)
```

; hello = 1
; 注意我们用 'hello, 所以 lisp 不会 evaluate hello 的值.

这家伙很常用, 也有简写.
(setq hello 1)
setq 就是 set quote 的缩写. 这是个 special form, 不会对
第二个元素求值.
valuation

在 Emacs 下, C-x C-e 可以执行(evaluate, 求值)游标左边的叙述.
结果会出现在 minibuffer.

lisp-interaction-mode 中 C-j 可以 evaluate, 并且把结果 append 到 buffer.

lisp 程式由一堆 list 构成.称为 expression.
每个 expression 都回传回一个值.
有些 expression 有副作用, 如删掉一个字.
(这跟 C 的 int delete_char() 意思一样, 它传回 int,
并且删掉某个 char)

定义函式:

```
(defun NAME (ARGS-LIST)
    "注解"              ; optional
    (interactive)       ; optional
    BODY)
```

定义一个叫 NAME 的函式. BODY 是一堆 expression.
注解是用来给 C-h f 显示的.
(interactive) 表示这个函示会和 user/buffer 作用.
(interactive "B") 表示执行此函式先问 user 一个 buffer 的名字,
然後当作参数传给它. (如, 当 user 透过 key-binding
或者 M-x 呼叫此函式时)
(interactive "BAppend to buffer: \nr")
问 user buffer name 时, 提示号 Append to buffer:
此 function 有两个引数,第一个是 B, 就是 buffer
第二个是 r, region
用\n 隔开.
(interactive "p") 用 C-u 设的 prefix 把它当作参数传给我.
预设值==4. C-u C-f 向右移四个字

一些 lisp 函式:

(list 1 2 3 4) 产生 '(1 2 3 4)
(car '(1 2 3 4) 1
(cdr '(1 2 3 4) '(2 3 4)
(cons 1 '(2 3 4)) '(1 2 3 4)
(cons 1 2) (1 . 2)
(cons 0 (cons 1 (cons 2 nil)))
等於 '(0 1 2)
{list 是用 pair 串起来的,
用 C 表示:
pair: {Object *first, Object *second};
*(pair[i].first) == i;
pair[i].second == pair[i+1]; }

(cons '(1 2) '(3 4)) '((1 2) 3 4)
(setq a 1)
(1+ a) ; a+1
(+ 2 a) ; a+2
(* 1 2 3 4)
(current-buffer) ; 传回目前 buffer 的资料物件
(switch-to-buffer (other-buffer))
(set-buffer)
(buffer-size)
(setq current-pos (point))
(point-min)
(point-max)
(message "Hello") ; 在 minibuffer 显示 Hello
(if (test)
(then-part)
(else-part))
(cond ((test1) BODY1)
((test2) BODY2)
(t OTHER-WISE)
(let ((var1 value) ; local variable
var2 ; no value
(var3 value)
...)
BODY ...)
(lambda (ARG-LIST) ...) 同 defun, 但是没有名字 (anonymous).
可以存到变数去:
(setq hello (lambda () (message "Hello,World")))

(funcall hello)
(goto-char (point-max))
(defvar VAR VALUE "*注解") 如果 VAR 不存在才定义. 有注解可以用
C-h v 看. 注解打*号表是使用者可以直接改/ 这个变数本来就是
给使用者设定用的.
可以用 M-x edit-options 来线上设定 (emacs 结束就没有了,
不过 edit-options 可以给你所有可修改的变数的列表,你可以
放到.emacs 档内.

(directory-files "./" t "\\..*")