

# Emacs 使用手册

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) 启动.

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 20 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 10 C-x u UNDO 10 次

给 C-L 一个引数会怎么样:

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 10 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`, 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. 然後把游标移到另一端, 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. 编辑不同类型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 incremental search, 就是你打字的同时, emacs 就直接帮你找. 一个是 forward, 一个是 backward. 找到了怎麽办? 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 什麽意思? 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 incremental 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, 怎麽办?

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` 编辑模式. 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)

强烈建议改用 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 mailing 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,

基本上和文字模式的 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 开头的的所有 key-binding. 虽然说 Emacs

可以定义按键, 可是 Ctrl- 开头的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) 表示这个函式会和 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 "\\..*")
return a list of files under directory X
```

(load "xxxx.el") 同#include <stdio.h>  
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 "\\..*")
return a list of files under directory X
```

(load "xxxx.el") 同#include <stdio.h>

(setq load-path (cons "~/emacs" load-path)); load 的 search path.  
(autoload ...) 不像 load 会直接 evaluate 整个档案, 而是需要时再 load.

```
(local-unset-key [(control c)])
(local-set-key [(control c) a] 'forward-sexp)
```

sexp 就是一个 expression, n 个 expression 如果用括号括起来就算一个.

(expression 的定义随语言的不同而有不同, 在 C, lisp tex, html, fortran 下皆有差异)