1. **Emacs 中的光标移动**

C-f 光标前进一个字符

C-b 光标后退一个字符

C-p 上移一行

C-n 下移一行

M-f 前进一个单词

M-b 后退一个单词

C-a 行首

C-e 行尾

M-a 句首

M-e 句尾

M-{ 上一个段落

M-} 下一个段落

C-v 屏幕上卷一个屏

M-v 屏幕下卷一个屏

C-M-v 另一个窗口下卷一个屏幕

C-M-V 另一个窗口上卷一个屏幕

M-< 文件头

M-> 文件尾

C-u n 或 M-n 重复执行n次后继命令

1. **Emacs中的删除**

C-d 删除光标上的字符

DEL 删除光标前的字符

M-d 删除光标后的单词

M-DEL 删除光标前的单词

C-k 从光标位置删除到行尾

C-S-Backspace 删除当前行

C-y 恢复删除最近的一个

M-y 循环恢复删除上一个

1. **文本块操作**

C-@ 标记文本块的开始

C-x C-x 互换插入点和文本标记的位置

M-h 标记整个段落

C-x C-p 标记整个页面

C-x h 标记整个缓冲区

1. **查找与替换**
2. **简单查找替换**

C-s 向前递增查找

C-r 反向递增查找

C-s C-w 向前递增查找，把光标所在的单词作为查找对象

C-r C-w 反向递增查找，把光标所在的单词作为查找对象

replace-string 查找替换光标后所有的内容

M-% 查询替换操作

对于M-%，需要注意的是其相关的一些操作：

SPACE或y 用新字符串替换原来的，并进入下一个位置

DEL或n 不替换，进入下一个位置

. 在当前位置做替换后退出查询－替换操作

, 替换，并显示替换情况（再按空格或y进入下一个位置）

! 对后面的内容全部进行替换，不再提问

^ 返回上一个替换了的位置

RETURN或q 退出查询替换操作

C-r 进入递归编辑状态

C-w 删除此处内容并进入递归编辑状态（好进行其他修改）

M-C-c 退出递归编辑状态，继续完成查询－替换操作

C-] 退出递归编辑状态和查询替换操作

1. **正则表达式查找与替换**

C-M-s Return 向前查找正则表达式

C-M-r Return 反向查找正则表达式

C-M-s 向前递归查找正则表达式

C-M-r 反向递归查找正则表达式

query-replace-regexp 查询-替换正则表达式

replace-regexp 无条件替换正则表达式（慎用）

正则表达式中的字符：

^ 匹配行首

$ 匹配行尾

. 匹配任意单个字符

.\* 匹配任意个字符

\< 匹配单词的开头

\> 匹配单词的结尾

[] 匹配[]中的任何一个字符，如[a-z]匹配任何一个小写字母

1. **Emacs 的C/C++ 模式**
2. **指定为C++模式的方法**

一般根据扩展名自动设定，不用指定，不过有时候你希 望.h文件是C++模式的（缺省是C模式），在文件第一行（或其末尾）上加入

// -\*- C++ -\*-

1. **语法高亮**

不是C模式专有，顺便提一下，M-x global-font-lock-mode RET 或 在.emacs中加入(global-font-lock-mode t)。在console下，Emacs21才支持语 法高亮。（语法高亮，不同关键字用不同的颜色来突出）

1. **编译和调试**

M-x compile RET 编译

M-x gdb RET 调试

C-x ` （出错信息中）下一个错误，一个窗口显示错误信息，另一个

显示源码的出错位置

C-u C-x ` 在compile buffer 列出同样的错误。

C-c C-c 转到出错位置

启动gdb调试器后，光标在源码文件缓冲区中时：

C-x SPC 在当前行设置断点

C-c C-c 中断

C-c C-z 中止

C-x C-a C-s step

C-x C-a C-n next

C-x C-a C-t tbreak

C-x C-a C-r continue

C-x C-a C-d 删除当前断点

C-x C-a C-l 到达最后一行

1. **显示vector元素值**

GDB 在调用成员函数的时候需要首先找到函数的代码才能执行。如果一个成员函数是内联的、并且没有在程序中被使用， 那么 GDB 将找不到其代码，也就不能执行此成员函数。

方法1 在源程序的查看之前的位置使用一次 at() 成员函数。

方法2 执行 print vec\_obj 可以列出 vector 的内部数据结构，使用这些结构也可以查看各个元素。

假设列出了如下内容：

$1 = {<std::\_Vector\_base<int,std::allocator<int> >> = {

\_M\_impl = {<std::allocator<int>> = {<\_\_gnu\_cxx::new\_allocator<int>> = {<No data fields>},

< No data fields>}, \_M\_start = 0x8095040,

\_M\_finish = 0x809505c, \_M\_end\_of\_storage = 0x8095060}}, <No data fields>}

可以这样查看第一个元素：print vec\_obj.\_M\_impl.\_M\_start[0]

为方便使用，可以定义一个宏来使用：

define print\_vector

print $arg0.\_M\_impl.\_M\_start[$arg1]

end

这样，print\_vector vec\_obj 2 就可查看第三个元素。

1. **在C/C++代码中操作**

M-a 移动到当前语句的开始

M-e 移动到当前语句的末尾

C-M-a 移动到当前函数的开始

C-M-e 移动到当前函数的结尾

M-q 若光标在注释文本中间，则进行段落重排，保留缩进和前导字符

C-M-h 把光标放在当前函数开头，把文本块标记放在函数结尾， 即把函数整个选作为一个文本块。

C-c C-q 对整个函数进行缩进

C-x h 选定整个buffer， 然后C-M-\可以对代码进行重新排版

C-c C-u 移动到当前预处理条件的开始位置

C-c C-p 移动到上一个预处理条件

C-c C-n 移动到下一个预处理条件

M-; 对当前行进行注释

C-x C-x 快速返回移动前的光标位置

C-c C-c 对一块代码进行注释；取消注释用命令 uncomment-region

1. **子模式**

auto-state 当你输入时自动缩进，自动换行

hungry-state 当你Backspace时，自动删除尽可能多的空白和空行

C-c C-t 同时转换(开/关)auto-state和hungry-state子模式

C-c C-a 转换 auto-state 子模式

C-c C-d 转换 hungry-state 子模式

C-c . 设置缩进风格（按TAB键可列出可用的风格，缺省的为gnu，其

缩进为2个字符；linux为8个；k&r为5个…）

TAB 重新缩进当前行

M-/ 自动补齐（缓冲区中能找得到的串）

M-; 行尾加入注释

C-c C-e 扩展宏

C-c C-c 注释掉整个区域

C-c C-\ 将区域中的每一行结尾都加入一个'\'字符

1. **Emacs中的register**

在多个文件中逛的时候，我们常常需要快速切换到先前访问的某个位置。因此，我们需要把文件及其光标位置暂时存放在某个地方。

在Emacs中，我们可以使用register暂时性保存这些信息。

将当前光标所在位置保存入一个register中：

C-x r SPACE + register名（一个字符，比如a吧）

然后我们就可以到处瞎逛，若要回到保存的register a位置，我们可以输入：

C-x r j a

挺好用的吧:)

如果你记性和我一样不好，恐怕会常常想看看自己保存了哪些register，我们可以输入：

M-x view-register 查看某一个register

M-x list-registers 查看所有的register

其实我常用的就是以上功能，为了使笔记稍微完整一点，下面开始Copy 《GNU Emacs Manual》的相关章节：

在下面，我们使用r来命名所有的register：

1. **Save Positions in Registers**

C-x r SPC r

Save position of point in register r (point-to-register).

C-x r j r

Jump to the position saved in register r (jump-to-register).

1. **Saving Text in Registers**

C-x r s r

Copy region into register r (copy-to-register).

C-x r i r

Insert text from register r (insert-register).

M-x append-to-register RET r

Append region to text in register r.

M-x prepend-to-register RET r

Prepend region to text in register r.

1. **Saving Rectangles in Registers**

C-x r r r

Copy the region-rectangle into register r (copy-rectangle-to-register).

With numeric argument, delete it as well.

C-x r i r

Insert the rectangle stored in register r (if it contains a rectangle) (insert-register).

1. **Saving Window Configurations in Registers**

C-x r w r

Save the state of the selected frame's windows in register r (window-configuration-to-register).

C-x r f r

Save the state of all frames, including all their windows,

in register r (frame-configuration-to-register).

1. **Keeping Numbers in Registers**

C-u number C-x r n r

Store number into register r (number-to-register).

C-u number C-x r + r

Increment the number in register r by number (increment-register).

C-x r g r

Insert the number from register r into the buffer.

1. **使.emacs立即生效**

M-x eval-current-buffer

1. **Etags**
2. **建立Etags文件：**

在代码目录中运行etags -R

如果要向现有tags表中添加，则运行etags -a

1. **访问Tag文件：**

M-x visit-tags-table

1. **常用热键**

M-. 访问tag C-u M-.访问下一个tag M-\* 返回

1. **Emacs 中的 GNU-Serv**

常常使用console的人往往也改不了这么个习惯，就是进了X就开个终端，然后在里头埋头干活，看到什么文件想修改就vi一把。我也是这个习惯。vi比emacs启动快多了。

GNU-Serv改变了这个情形。

首先要安装gnu-serv这个包，在我的机器上apt-get install gnu-serv就可以了。

接下来修改.emacs文件

(require 'gnuserv)

(gnuserv-start)

接下来修改.bashrc文件

alias e='gnuclient'

这样，在终端下就可以通过 e foo来编辑foo文件了，它会自动关联到已经打开的emacs中。

更强大的功能在于，你还可以在远程执行gnuclinet，在本机编辑文件后自动存放到远程机器上。

1. **Emacs中的bookmark**

将当前页面加入bookmark

M-x bookmark-set

修改.bashrc，将list-bookmarks绑定到F9上

(global-set-key [(f9)] 'list-bookmarks)

这样，就可以通过F9键访问书签了。

1. **常用键绑定**

F2 寻找wiki

F3 进入dired模式

F4 进入eshell模式

F12 t 进入todo模式

F12 c 访问Calendar模式

F12 p 在当前页添加Planner

F12 g 查阅当天的plan

F12 r 添加笔记

F9 列出bookmarks

1. **在Emacs 中添加Appointment**

这个主要是添加约会和提醒用的，首先修改.emacs文件打开约会提醒功能。

(setq appt-issue-message t)

然后可以通过appt-add添加新的约会提醒，到时间Emacs就会弹出窗口提醒。如果这期间要取消约会，那么可以通过appt-delete删除。

1. **Calendar和Diary**

Calendar是配合Diary进行行程安排的，两者配合可以用于规划比较长远的事情。

我已经把Calendar模式绑定到了F8上，以下列举几个常用的命令：

. 跳回当天

o 跳到某一个月

g 这一系列命令表示goto，可以跳到指定的某一天。

g d 跳到某年某月某日

g c 跳到某年某星期的星期几

g C 跳到阴历的某一天

p 这一系列命令表示print，例如p C显示显示当前的阴历日期

i 这一系列命令表示插入行程安排

i d 加入当前这一天的行程安排，类似还有m w y

i w 跳到某个星期的星期一，按下i w可以加入每个星期都必须做的事情

i a 跳到某天，按下i a可以加入周年纪念日

好习惯应该是经常打开calendar，跳到某天，按下d就可以显示这一天的行程安排。

1. **Emacs 中的 To-Do Mode**

To-do可以用来做日程安排，todo，done，top分别表示要完成的事情，已完成的事情，和优先事情。

首先要修改.emacs文件

(setq todo-file-do "~/.emacs.d/todo-do")

(setq todo-file-done "~/.emacs.d/todo-done")

(setq todo-file-top "~/.emacs.d/todo-top")

下面是一些快捷操作

i 加入一个条目

e 编辑条目

k 删除条目

+/- 在不同category之间切换

j 跳转到不同的category

f 对已完成的事情进行归档，并可以进行评论。

如果按i想把条目归入新档时出现错误，可以用todo-add-category加入

想要添加什么事情，随手就可以通过F5进入todo-mode，按寻找一个category，按I添加一个新的entry。完成之后再切换到todo去，按下d或者f删除entry。每个星期可以通过bookmark访问一下done文件，顿时觉得生活过得好充实。

todo部分转载自coda@newsmth。

1. **用planner安排日程**

这里有篇不错的入门文档[一](http://www.newsmth.net/bbscon.php%3fbid%3d573&id%3d26253)、[二](http://www.newsmth.net/bbscon.php%3fbid%3d573&id%3d26262)。

1. **安装planner**

在我的机器上只需要 apt-get install planner-el即可，非Debian的系统也不麻烦。

接下来要配置.emacs文件。

;;;planner

(require 'planner)

(global-set-key (kbd "<f12> p") 'planner-create-task-from-buffer)

(setq planner-publishing-directory "~/document/wiki/diary/plan")

;;;;Start planner together with Calendar

(planner-calendar-insinuate)

(setq planner-calendar-show-planner-files t)

1. **常用快捷键**

查询任务

M-x plan 查询所有未完成的工作

M-x planner-goto-today 查询当天未完成的工作

添加任务 C-c C-t或者M-x planner-create-task-from-buffer

标记任务完成 C-c C-x或者M-x planner-update-task

修改任务描述 M-x planner-edit-task-description

重新安排一个任务 C-c C-c

重新安排多个任务 区域选定后M-x planner-copy-or-move-task

Change a plan of task M-x planner-replan-task

Delete a task M-x planner-delete-task

Reordering task M-p 升序 M-n 降序

修改优先级 C-M-p 提升；C-M-n 降低

1. **以下是从info中复制过来的**

*Ideas for using planner more effectively:*

* Add `(plan)' to the end of your `~/.emacs' so that you are reminded about your tasks every day.
* Bind useful functions to shortcut keys and get used to creating tasks and notes from anywhere.
* Think about how you plan your day and look for ways to improve it. Ask the mailing list for tips.
* Browse the rest of this manual, the source code, and other resources on the Net for tidbits you can use.
* Have fun!

*Useful functions outside planner buffers:*

* `planner-create-task-from-buffer'
* `remember'
* `planner-goto-today'
* `planner-goto'
* `plan'

*Useful functions inside planner buffers:*

* `C-c C-t' (`planner-create-task-from-buffer')
* `C-c C-x' (`planner-task-done')
* `M-x planner-edit-task-description'
* `C-c C-c' (`planner-copy-or-move-task'), `M-x planner-copy-or-move-region'
* `M-x planner-replan-task'
* `M-x planner-delete-task'
* (`planner-raise-task') and (`planner-lower-task')
* (`planner-raise-task-priority') and (`planner-lower-task-priority'),
* `planner-replan-note'
* `planner-update-note'

--