下面是我总结的人家的一些经验：

先稍为介绍一下 vim. vi 是 unix/linux 下极为普遍的一种文本编辑器, 大部分机器上都有. vi 有各种变种, 在不同的机器上常用不同的变种软件. 其中 vim 比较好用也用得比较广泛. vim 是 Vi IMproved 的缩写, 表示更好的 vi. 我个人觉得它是非常好的编辑器(为了避免 Emacs 粉丝挑眼, 就不说最好了). 没用过的也建议试试看, 当然 vim 对编写文本文件很方便好用, 比如编写程序, html 文档等等, 却不能用来编写 word 文档.

关于 vim 的安装, 基本使用方法等网络上能搜出许多, 就不在这里罗嗦了, 要是对 vim 有兴趣, 那就看看这里(中文文档): http://vcd.cosoft.org.cn/pwiki/index.php

本文就说些其中比较有用, 比较常用的命令, 若能熟练运用这些命令, 那么会发现编辑文件很舒服.

说明:
以下的例子中 xxx 表示在命令模式下输入 xxx 并回车
以下的例子中 :xxx 表示在扩展模式下输入 xxx 并回车
小括号中的命令表示相关命令.
在编辑模式或可视模式下输入的命令会另外注明.

进入与离开 FecLinux 联盟

要 进入 VIM 可以直接在系统提示下键入 VIM ＜档案名称＞，VIM 可以自动帮你载入所要编辑的文件或是开启一个新文件。进入 VIM 后屏幕左方会出现波浪符号，凡是行首有该符号就代表此列目前是空的。要离开 VIM 可以在指令模式下键入 :q, :wq 指令则是存档後再离开（注意冒号）。要切换到指令模式下则是用 [ESC] 键，如果不晓得现在是处於什麼模式，可以多按几次 [ESC]，系统会发出哔哔声以确定进入指令模式。
VIM 输入模式 FecLinux 联盟
要如何输入资料呢？有好几个指令可以进入输入模式：FecLinux 联盟
新增 (append)FecLinux 联盟

a 从光标所在位置后面开始新增资料，光标后的资料随新增资料向后移动。FecLinux 联盟
A 从光标所在列最后面的地方开始新增资料。
FecLinux 联盟
插入 (insert)
i 从光标所在位置前面开始插入资料，光标后的资料随新增资料向后移动。FecLinux 联盟
I 从光标所在列的第一个非空白字元前面开始插入资料。
FecLinux 联盟
开始 (open)
o 在光标所在列下新增一列并进入输入模式。FecLinux 联盟
O 在光标所在列上方新增一列并进入输入模式。
也许文字叙述看起来有点繁杂，但是只要实际操作一下马上可以了解这些操作方式。实务很重要，尤其是电脑方面的东西随时可以尝试及验证结果。极力建议实际去使用它而不要只是猛 K 文件，才有事半功倍的效用。（注：此段为废话。）

档案指令 FecLinux 联盟
FecLinux 联盟
档案指令多以 : 开头，跟编辑指令有点区别。例如前面提到结束编辑的指令就是 :q。现在就简单说明一下作为本篇故事的结尾：FecLinux 联盟

:q 结束编辑(quit)FecLinux 联盟
如果不想存档而要放弃编辑过的档案则用 :q! 强制离开。FecLinux 联盟
:w 存档(write)FecLinux 联盟
其后可加所要存档的档名。FecLinux 联盟
可以将档案指令合在一起，例如 :wq 即存档后离开。FecLinux 联盟
zz 功能与 :wq 相同。FecLinux 联盟

另外值得一提的是 VIM 的部份存档功能。可以用 :n,mw filename 将第 n 行到第 m 行的文字存放的所指定的 filename 里去哩。

1. 查找

/xxx(?xxx)	表示在整篇文档中搜索匹配 xxx 的字符串, / 表示向下查找, ? 表示向上查找.其中 xxx 可以是正规表达式,关于正规式就不多说了. 一般来说是区分大小写的, 要想不区分大小写, 那得先输入 :set ignorecase 查找到以后, 再输入 n 查找下一个匹配处, 输入 N 反方向查找.
*(#)	当光标停留在某个单词上时, 输入这条命令表示查找与该单词匹配的下(上)一个单词. 同样, 再输入 n 查找下一个匹配处, 输入 N 反方向查找.
g*(g#)	此命令与上条命令相似, 只不过它不完全匹配光标所在处的单词, 而是匹配包含该单词的所有字符串.
gd	本命令查找与光标所在单词相匹配的单词, 并将光标停留在文档的非注释段中第一次出现这个单词的地方.
%	本命令查找与光标所在处相匹配的反括号, 包括 () [] { }
f(F)x	本命令表示在光标所在行进行查找, 查找光标右(左)方第一个 x 字符. 找到后: 输入 ; 表示继续往下找 输入 , 表示反方向查找

2. 快速移动光标
在 vi 中, 移动光标和编辑是两件事, 正因为区分开来, 所以可以很方便的进行光标定位和编辑. 因此能更快一点移动光标是很有用的.

w(e)	移动光标到下一个单词.
b	移动光标到上一个单词.
0	移动光标到本行最开头.
^	移动光标到本行最开头的字符处.
$	移动光标到本行结尾处.
H	移动光标到屏幕的首行.
M	移动光标到屏幕的中间一行.
L	移动光标到屏幕的尾行.
gg	移动光标到文档首行.
G	移动光标到文档尾行.
c-f	(即 ctrl 键与 f 键一同按下) 本命令即 page down.
c-b	(即 ctrl 键与 b 键一同按下, 后同) 本命令即 page up.
''	此命令相当有用, 它移动光标到上一个标记处, 比如用 gd, * 等查找到某个单词后, 再输入此命令则回到上次停留的位置.
'.	此命令相当好使, 它移动光标到上一次的修改行.
`.	此命令相当强大, 它移动光标到上一次的修改点.

3. 拷贝, 删除与粘贴

在 vi 中 y 表示拷贝, d 表示删除, p 表示粘贴. 其中拷贝与删除是与光标移动命令结合的, 看几个例子就能够明白了.

yw	表示拷贝从当前光标到光标所在单词结尾的内容.
dw	表示删除从当前光标到光标所在单词结尾的内容.
y0	表示拷贝从当前光标到光标所在行首的内容.
d0	表示删除从当前光标到光标所在行首的内容.
y$	表示拷贝从当前光标到光标所在行尾的内容.
d$	表示删除从当前光标到光标所在行尾的内容.
yfa	表示拷贝从当前光标到光标后面的第一个 a 字符之间的内容.
dfa	表示删除从当前光标到光标后面的第一个 a 字符之间的内容.

特殊地:

yy	表示拷贝光标所在行.
dd	表示删除光标所在行.
D	表示删除从当前光标到光标所在行尾的内容.

关于拷贝, 删除和粘贴的复杂用法与寄存器有关, 可以自行查询.

4. 数字与命令

在 vi 中数字与命令结合往往表示重复进行此命令，若在扩展模式的开头出现则表示行号定位. 如:

5fx	表示查找光标后第 5 个 x 字符.
5w(e)	移动光标到下五个单词.
5yy	表示拷贝光标以下 5 行.
5dd	表示删除光标以下 5 行.
y2fa	表示拷贝从当前光标到光标后面的第二个 a 字符之间的内容.
:12,24y	表示拷贝第 12 行到第 24 行之间的内容.
:12,y	表示拷贝第 12 行到光标所在行之间的内容.
:,24y	表示拷贝光标所在行到第 24 行之间的内容. 删除类似.

5. 快速输入字符

在 vi 中，不要求你输入每一个字符，可以有很多种方法快速输入一些字符.

使用 linux/unix 的同学一定有一个经验，在命令行下输入命令时敲入头几个字符再按 TAB 系统就会自动将剩下的字符补齐，假如有多个匹配则会打印出来. 这就是著名的命令补齐(其实 windows 中也有文件名补齐功能). vi 中有许多的字符串补齐命令，非常方便.

c-p(c-n)	在编辑模式中，输入几个字符后再输入此命令则 vi 开始向上(下)搜索开头与其匹配的单词并补齐，不断输入此命令则循环查找. 此命令会在所有在这个 vim 程序中打开的文件中进行匹配.
c-x-l	在编辑模式中，此命令快速补齐整行内容，但是仅在本窗口中出现的文档中进行匹配.
c-x-f	在编辑模式中，这个命令表示补齐文件名. 如输入: /usr/local/tom 后再输入此命令则它会自动匹配出: /usr/local/tomcat/
abbr	即缩写. 这是一个宏操作，可以在编辑模式中用一个缩写代替另一个字符串. 比如编写 java 文件的常常输入 System.out.println，这很是麻烦，所以应该用缩写来减少敲字. 可以这么做: :abbr sprt System.out.println 以后在输入 sprt 后再输入其他非字母符号，它就会自动扩展为 System.out.println

6. 替换

替换是 vi 的强项, 因为可以用正规表达式来匹配字符串.以下提供几个例子.

:s/aa/bb/g 将光标所在行出现的所有包含 aa 的字符串中的 aa 替换为 bb

:s/\<aa\>/bb/g 将光标所在行出现的所有 aa 替换为 bb, 仅替换 aa 这个单词

:%s/aa/bb/g 将文档中出现的所有包含 aa 的字符串中的 aa 替换为 bb

:12,23s/aa/bb/g 将从 12 行到 23 行中出现的所有包含 aa 的字符串中的 aa 替换为 bb

:12,23s/^/#/ 将从 12 行到 23 行的行首加入 # 字符

:%s= *$== 将所有行尾多余的空格删除

:g/^\s*$/d 将所有不包含字符(空格也不包含)的空行删除.

7. 多文件编辑

在一个 vim 程序中打开很多文件进行编辑是挺方便的.

:sp(:vsp) 文件名 vim 将分割出一个横(纵)向窗口, 并在该窗口中打开新文件.
从 vim6.0 开始, 文件名可以是一个目录的名称, 这样, vim 会把该目录打开并显示文件列表, 在文件名上按回车则在本窗口打开该文件, 若输入 O 则在新窗口中打开该文件, 输入 ? 可以看到帮助信息.

:e 文件名 vim 将在原窗口中打开新的文件, 若旧文件编辑过, 会要求保存.

c-w-w vim 分割了好几个窗口怎么办? 输入此命令可以将光标循环定位到各个窗口之中.

:ls 此命令查看本 vim 程序已经打开了多少个文件, 在屏幕的最下方会显示出如下数据:

```
1    %a       "usevim.html"          行 162
2    #        "xxxxxx.html"          行 0
```

其中:

1 表示打开的文件序号, 这个序号很有用处.

%a 表示文件代号, % 表示当前编辑的文件, # 表示上次编辑的文件

"usevim.html" 表示文件名.

行 162 表示光标位置.

:b 序号(代号) 此命令将指定序号(代号)的文件在本窗口打开, 其中的序号(代号)就是用 :ls 命令看到的.

:set diff 此命令用于比较两个文件, 可以用
:vsp filename
命令打开另一个文件, 然后在每个文件窗口中输入此命令,就能看

到效果了.

8. 宏替换

vi 不仅可以用 abbr 来替换文字, 也可以进行命令的宏定义. 有些命令输起来很费劲, 因此我把它们定义到 <F1>-<F12> 上, 这样就很方便了.这些配置可以预先写到 ~/.vimrc (windows 下为 $VIM/_vimrc) 中, 写进去的时候不用写前面的冒号.

```
:nmap <F2> :nohls<cr>                取消被搜索字串的高亮
:nmap <F9> <C-W>w                    命令模式下转移光标到不同窗口
:imap <F9> <ESC><F9>                 输入模式下运行<F9>
:nmap <F12> :%s= *$==<cr>            删除所有行尾多余的空格.
:imap <F12> <ESC><F12>               同上
```

:java 中: (注, 这里为什么说 java 中, 因为以下定义对其他文件格式不起作用, 下文会说到如何实现这一点)

```
:nmap <F3> :comp javac<CR>:mak -d . %<CR>
```

此命令用 javac 编译 java 文件, 它会自动将光标定位到出错点. 不过这需要定义一个 javac.vim 文件在 $VIM/compiler 下, 在 javac.vim 里面只有两行字:

```
setlocal makeprg=javac
setlocal errorformat=%A%f:%l:\ %m,%-Z%p^,%-C%.%#
```

```
:nmap <F4> :comp ant<CR>:mak<CR>
```

此命令用 ant 编译 java 文件, 它会自动将光标定位到出错点. 一般来说, 安装 vim 后已经有了 compiler/ant.vim 文件, 因此这个命令可以直接使用. 但是需要在当前目录下有 build.xml 文件, 当然还必须安装 ant 才行.

```
:nmap <F5> :cl<CR>                   此命令用于查看所有的编译错误.
:imap <F5> <ESC><F5>

:nmap <F6> :cc<CR>                   此命令用于查看当前的编译错误.
:imap <F6> <ESC><F6>

:nmap <F7> :cn<CR>                   此命令用于跳到下一个出错位置.
:imap <F7> <ESC><F7>

:nmap <F8> :cp<CR>                   此命令用于跳到上一个出错位置.
:imap <F8> <ESC><F8>
```

```
:nmap <F11> :JavaBrowser<cr>
```

此命令用于在窗口左部分割出一个新窗口, 里面的内容是 java 的资源树, 包括本文件中出现的类, 类的成员变量及成员方法, 就好像 JCreator 表现的那样. 在这个窗口中输入 ? 会看到帮助. 嘿嘿, 很好用, 不过需要 ctags 支持.

```
:imap <F11> <ESC><F11>
```

9. TAB

TAB 就是制表符，单独拿出来做一节是因为这个东西确实很有用.

<<	输入此命令则光标所在行向左移动一个 tab.
>>	输入此命令则光标所在行向右移动一个 tab.
5>>	输入此命令则光标后 5 行向右移动一个 tab.
:12,24>	此命令将 12 行到 14 行的数据都向右移动一个 tab.
:12,24>>	此命令将 12 行到 14 行的数据都向右移动两个 tab.

那么如何定义 tab 的大小呢？有人愿意使用 8 个空格位，有人用 4 个，有的用 2 个. 有的人希望 tab 完全用空格代替，也有的人希望 tab 就是 tab. 没关系, vim 能帮助你.以下的设置一般也都先写入配置文件中，免得老敲.

:set shiftwidth=4	设置自动缩进 4 个空格，当然要设自动缩进先.
:set sts=4	即设置 softtabstop 为 4. 输入 tab 后就跳了 4 格.
:set tabstop=4	实际的 tab 即为 4 个空格，而不是缺省的 8 个.
:set expandtab	在输入 tab 后, vim 用恰当的空格来填充这个 tab.

10. autocmd

这个命令十分的强大，可以用这个命令实现对不同的文件格式应用不同的配置；可以在新建文件时自动添加上版权声明等等. 这些命令一般定义在 ~/.vimrc 这样的配置文件里面. 由于他很强大，所以我不能给出很具体的说明，只能举几个例子，详细的请看帮助.

:autocmd!		删除所有之前的自动命令.
autocmd FileType	java	source ~/.vim/files/java.vim
autocmd FileType	java	source ~/.vim/files/jcommenter.vim

以上两条命令让我在打开 java 文件时才应用后面提到的两个配置文件.

autocmd BufNewFile	*.java	0r ~/.vim/files/skeletons/java.skel

以上这条命令让我在新建 java 文件时自动加入 java.skel 文件的内容.

autocmd BufNewFile	*.java	normal gnp

以上这条命令让我在新建 java 文件时自动运行 gnp 命令，这个命令进行一些特殊化处理，比如将新 java 文件中的 __date__ 替换成今天的日期什么的.

11. 常用脚本

在 vim.sf.net 你可以发现很多脚本(script)，这些脚本常常有让你意想不到的作用. 我常用的有:

jcommenter.vim	自动加入 javadoc 风格的注释.
JBrowser.vim	类资源浏览. C, C++ 等可以用 Tlist

还有许多有用的，比如 checkstyle.vim 可以检验你的编程风格，jad.vim 可以直接反编译 .class 文件等等.

12. 常用配置

在~/.vimrc 配置文件中你常常需要一些个性化配置. 比如上面写的一些宏定义，一些 autocmd 定义等等. 比如:

```
set suffixes=.bak,~,.o,.h,.info,.swp,.aux,.bbl,.blg,.dvi,.lof,.log,.lot,.ps,.toc
```

这样在 vim 中打开文件时，按 tab 键补齐文件名时它会忽略上述文件.

```
set nu                  显示行号
set ai                  设置自动缩进
map Y y$                让 Y 和 D 一样，要不然 Y 的本意和 yy 一样.
```

13. 其他

还有许多有意思的命令，记录在这里免得忘记.

```
.                                                   重复上次编辑命令.
:g/^/exec "s/^/".strpart(line(".")." ", 0, 4)      在行首插入行号
:runtime! syntax/2html.vim                          转换 txt 成 html，会按照你的
                                                    颜色配置来转
```