Linux 操作系统及应用
第四章 — 文本编辑器 vi

唐晓晟　李亦农
txs@bupt.edu.cn　hoplee@bupt.edu.cn

Contents

- 本章部分内容来自于：*SMTH BBS, yubao.liu@gmail.com*: http://www.newsmth.net/att.php?s.731.15830.642.txt

1 vi/vim 的历史

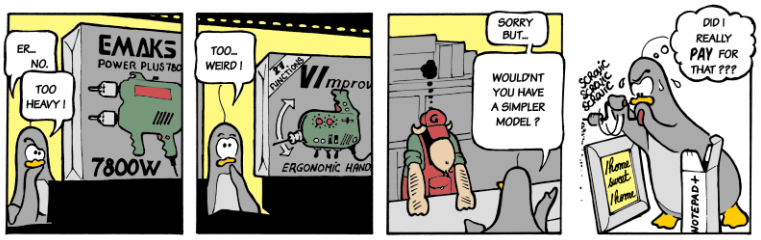
- ed
  - ed是 UNIX 上最古老最基本的编辑器，它最初是 UNIX 之父 Ken Thompson 编写的，他第一次在ed中应用了正则表达式 (regular expression)，这个创举将 RE 理论带入了实践，对 UNIX 界造成了深远的影响。实际上ed是受来自加州伯克利大学的qed编辑器的影响，Ken 便是从这所院校毕业的。ed是一种行模式编辑器，下面是一个ed会话的例子：

```
# ed greeting
0      # 因为新创建文件，所以读入了0个字节
a      # 进入编辑模式(append)
hello world, eveyone.    # 输入一行文本
.      # 回到命令模式
1p     # 显示(print)第一行
hello world, eveyone.
1c     # 最后一个词写错了，修改(change)第一行
hello world, everyone.
.      # 回到命令模式
1p     # 重新显示第一行，这回发现无误
hello world, everyone.
q      # 退出(quit)
?      # ? 表示没有保存或者命令不认识
w      # 保存(write)
23     # 提示写了23个字节
q      # 退出
```

  - 在 70 年代，许多使用 UNIX 的人都是用廉价的终端机通过电话线连到 UNIX 服务器上的，因为传输速率慢，所以这种简洁的行模式编辑就很有意义，现在虽然硬件有了长足的发展，ed编辑器很少有人用了，但是它的很多理念比如命令字符、正则表达式却在它的很多后辈身上体现出来。下面这个网址有一些ed的笑话，从中我们或多或少能体会到那个时代人们的某种精神。http://www.gnu.org/fun/jokes/ed.msg.html
  - GNU ed: http://www.gnu.org/software/ed/ed.html
  - Reference: http://en.wikipedia.org/wiki/ed, http://snap.nlc.dcccd.edu/learn/nlc/ed.html, http://cm.bell-labs.com/cm/cs/who/dmr/qed.html, http://www.answers.com/topic/ken-thompson

- ex
  - ex是ed的扩展，实际上vi构建在ex之上，vi引入了现在我们所熟悉的全屏编辑模式。

- vi
  - 随着硬件的发展，UNIX 上许多全屏模式编辑器被开发出来，pico, nano, joe, jed, jove, 但是最为广泛使用的无疑就是vi(nvi, elvis, vim等) 和emacs(GNU Emacs, XEmacs 等)。
  - 1976 年左右 Bill Joy 开发了vi，他也是伯克利大学的毕业生，后来他跟其他人一起成立了 Sun Microsystems 公司并成为了 Sun 的首席科学家。一开始 Bill 开发了ex, 尔后开发了vi作为ex的 visual interface, 也就是说vi允许一次能看到一屏的文本而非一行, vi也因此得名。在当今的 UNIX 系统上，可能没有缺省安装emacs，但是肯定会有vi，当然，ed编辑器也必定存在:-)



  - Reference:

    * vi History,
    * Bill's Work

- vim
  - 技术不断进步，需求也在不断提高，vi的各种变种也不断涌现，知名的有nvi, elvis, vim, vile,yzis, 其中移植性最好、特性最多、使用最广的当属vim, vim主要是 Bram Moolenaar 创作的。
  - 一开始，**VIM**表示**Vi IMitation**（模拟）的简称，1992 年 1.22 版本的vim被移植到了 UNIX 和 MS-DOS 上。从那个时候开始，**VIM**的全名就变成**Vi IMproved**（改良）了。
  - 下面是vim的重要版本历史:

Date	Version	Milestone
2013 Aug 10	Vim 7.4	A new, faster regex engine
2010 Aug 15	Vim 7.3	Persistent undo/ redo, Blowfish encryption, Lua/Python support
2008 Aug 09	Vim 7.2	Floating point in scripts
2006 May 07	Vim 7.0	Spell Checking, multi-tabpage, code completion
2001 Sep 26	Vim 6.0	Folding (and more), plugin, multilingual
1998 Feb 19	Vim 5.0	Syntax highlighting, scripting
1996 May 29	Vim 4.0	Graphical User Interface (Robert Webb).
1994 Aug 12	Vim 3.0	Support for multiple buffers and windows.
1992	Vim 1.22	Port to Unix. vim now competes with vi. This was when **VIM** became **Vi IMproved**.
1991 Nov 02	Vim 1.14	First release (on Fred Fish disk #591).

  - Reference:
    * *Learning the vi Editor*, 6th Edition, O'Reilly. http://www.mamiyami.com/document/oreilly/books/unix2/vi/index.htm (中国电力出版社有中译版)
    * http://www.vmunix.com/vim/hist.htmlHistory
    * http://www.vmunix.com/vim/vim
    * http://www.vim.orgHome of vim

```
1 vim :help vi_diff.txt
2     :help version4.txt
3     :help version5.txt
4     :help version6.txt
```

2 进入 vi

- vi是一个较大的 UNIX 命令，在启动的时候也有它自己的选项和参数
- 基本语法：vi [-options] [+[n]] [file]

- 常用选项：

-r	用于恢复系统突然崩溃时正在编辑的文件
-R	用于以只读方式打开文件
+n	用来指明进入vi后直接位于文件的第n行，如果不指定n，则位于最后一行

3 vi 的模式

vi的常用模式有四种：命令模式、插入模式、行模式和可视化模式：

1. 命令模式（Normal Mode）
   - 无论用户处于什么模式，按Esc键或组合键Ctrl+[就可以进入命令模式。当用户初始进入vi时，也处于这个状态。
   - 用户在命令模式下可以输入各种合法的vi内部命令，这些命令不会回显，而且不需要回车就会被执行
2. 插入模式（Insert Mode）
   - 在命令模式下使用某些命令会导致vi马上进入文本插入模式，这些命令包括：i, I, a, A, o,O等
   - 在这种模式下用户的任何模式都被当作是文件的内容看待，并将其显示在屏幕上
3. 行模式（Command-line Mode）
   - 行模式又称为最后行模式，通过在命令模式下键入冒号可进入这个模式，在这种模式下vi会在屏幕的最后一行显示一个冒号并等待用户输入命令
   - 行模式下用户输入的命令将回显在最后一行上，并且直到用户输入回车vi才会去执行它
   - 命令执行完毕后，vi自动进入命令模式，或者在输入命令时敲Esc键、或用退格键删除所有的输入之后回到命令模式
4. 可视化模式（Visual Mode）
   - 在命令模式下按Ctrl-v就可以进入可视化模式
   - 可视化模式类似于 UltraEdit/EditPlus 中的列模式，可以按列的方式操作文本区域
   - 敲Esc键回到命令模式

- 在vi里面用最后行命令:help mode可以看到vi的所有模式
- 由于区分了模式，导致vi的命令非常简洁，而无模式编辑器比如emacs，所有的命令都需要加上控制键Ctrl或Alt
- **VI vs. Emacs**: vi继承了ed的理念，另外也有历史原因，vi追求的是快捷 —启动程序迅速、编辑文本高效、功能专注，而Emacs追求的是功能的丰富强大以及集成带来的方便，在Emacs里头可以发邮件、上新闻组、听 mp3、浏览网页、玩游戏，几乎可以**login→emacs→logout**了:-), vi和Emacs都是程序员的编辑器，相比而言，Emacs更是提供了一种程序员的生活氛围。

4 基本编辑命令

- 移动光标（cursor movement）

Bksp	左移一个字符
Space	右移一个字符
Enter,+	下移一行
h	左移一个字符
l	右移一个字符
j	下移一行
k	上移一行

- vi中 “字” 的概念：

  广义： 两个空格之间的任何内容为一个字

  狭义： 空格是 “字” 的定界符，并且所有标点和非字母字符均被当成一个 “字”

- 例：`printf("Hello, world\n");` 广义时为两个字，狭义时为 11 个字。

- 命令列表

w,W	跳到下一个词的词首（狭义/广义）
b,B	跳到上一个词的词首（狭义/广义）
(,)	跳到前/后一句的句首
{,}	跳到前/后一段的段首
e,E	跳到下一个词的词尾（狭义/广义）
nG	跳到第n行（最后一行）
gg	跳到第一行（与1G等效）
^	跳到行首（第一个非空字符）
0	跳到行首（第一列）
$	跳到行尾
zt	将当前行滚卷至窗口首行
zz	将当前行滚卷至窗口中间
zb	将当前行滚卷至窗口底部
nH	将光标移到屏幕的第n行（最上行 Highest）
M	将光标移到屏幕的中间（Middle）
nL	将光标移到屏幕的倒数第n行（最下行 Lowest）
Ctrl-u	向下翻滚 1/2 屏
Ctrl-d	向上翻滚 1/2 屏
Ctrl-f	向下翻滚 1 屏
Ctrl-b	向上翻滚 1 屏
Ctrl-n	向下移动一行
Ctrl-p	向上移动一行

- 上述命令（除nG外）的前面均可增加一个数字来控制移动的次数

- 修改文本命令（modification）

nr<char>	用字符char代替光标处的字符，n指定字符char出现的次数。n缺省为 1
nR<text>	用text的内容替换当前行n次。n缺省为 1。命令执行完后处于插入方式，必须按Esc键回到命令方式

- cW（广义），cw（狭义），cc（整行），C（c$）替换一个词或替换到行尾
- 编辑中的每一行正文都有自己的行号，用下列行命令可以移动光标到指定行：
  :n将光标移到第n行
- 行命令模式下，可以规定命令操作的行号范围。
  - 数值用来指定绝对行号；
  - 字符 “.” 表示光标所在行的行号；
  - 字符 “$” 表示正文最后一行的行号；
  - 字符 “%” 表示所有行；
  - 简单的表达式，例如 “.+5” 表示当前行往下的第 5 行；“:345” 将光标移到第 345 行
- 在命令模式下正确定位光标之后，可用以下命令切换到插入模式（Insert）：

i	在光标左侧输入正文
I	在光标所在行的开头输入正文
a	在光标右侧输入正文
A	在光标所在行的末尾输入正文
o	在光标所在行的下一行增添新行
O	在光标所在行的上一行增添新行

- 删除文本命令（delete）

x	删除当前光标所在位置的字符
X	删除光标前的字符
d<cursor_movement>	删除从当前光标位置直到<cursor_movement>参数指定的位置
dd	删除当前行
D	删除从当前光标位置直到行尾的所有字符

- 字符串搜索（search）

/[pattern]/[offset]<CR>
/[pattern]<CR>
?[pattern]?[offset]<CR>
?[pattern]<CR>

  - 上述四个命令分别表示向下/向上搜索指定的正则表达式[pattern]，并将光标停留在相对搜索结果距离为[offset]行处；搜索得到结果后，可以使用重复命令n或N沿着相同或相反的方向重复上一次的搜索
  - f<char>命令在当前行搜索指定的字符并将光标停在匹配字符上
  - t<char>命令在当前行搜索指定的字符并将光标停在匹配字符的左边

  - % 搜索与当前字符匹配（括号对、某种编程语言的语句对）的位置并跳到匹配处

- 字符串替换（Substitute）
  - 此命令要求先进入行方式
  - 命令的语法为：
    [addr1,addr2|g]s/find_exp/repl_exp/[g|n][c]
  - 上述命令表示在第addr1行到addr2行的范围内将字符串find_exp用repl_exp代替。
  - n表示替换每行的第n个匹配，c表示需要用户确认
  - g放在命令末尾，表示行全程，不加g，表示只对搜索字符串的首次出现进行替换
  - g放在命令开头，表示对正文中所有包含搜索字符串的行进行替换操作。
- 重复前一命令：.
- 取消上一命令：u
- 重画屏幕：Ctrl-l
- Ctrl-G命令显示当前编辑文本的状态，包括文本共有多少行、文件名以及目前光标停在多少行。
- 文件的保存：行方式下使用w命令
- 退出vi：行方式下使用q命令将退出vi，如果文件做过改动但还没有保存，系统将做出提示并取消此次退出动作。行命令x相当于wq命令。在命令方式下使用命令ZZ等效于:x命令。
- 如果由于读写权限或是更新方面的问题，导致vi拒绝执行保存文件或退出vi的命令，那么可以在命令后加一个!号表示强制执行。

5 高级命令

- 文件操作命令 [addr1,addr2]w[!] [filename]
  将编辑缓冲区的addr1行到addr2行之间的内容写回到存储介质上的名为filename的文件中去，w后的!表示强制写入

r filename	将文件filename读入编辑缓冲区
e filename	编辑已存在于缓冲区中的、名为filename的文件
f filename	将当前文件重命名为filename
f	打印当前文件名称和状态，如文件的行数、光标所在的行号等

- 上述命令属于行方式
- 文本的移动和复制
  [addr1,addr2]m[addr3] [addr1,addr2]t[addr3]
  上述行方式下的命令分别表示将第addr1行至第addr2行之间的内容移动/拷贝到第addr3行的后面
- 行的合并
  命令方式下的命令J将当前行的下面一行合并到当前行的末尾
- 寄存器操作
  - vim 中有九类寄存器:

无名寄存器	"	最近一次删除/修改/替换操作的文本都会放入这个寄存器
10 个数字寄存器	0-9	拷贝或者删除的文本存入这些寄存器, 这些寄存器是循环使用的, 在每次存入内容到寄存器 1 时, 原有的内容会依次存入到后一个寄存器中
小删除寄存器	-	删除内容少于一行时放入这个寄存器
26 个命名寄存器	a-zA-Z	大小写无关。这些寄存器可以在拷贝或者删除等操作中指定使用
4 个只读寄存器	:.%#	特殊用途
表达式寄存器	=	特殊用途
选择和拖放寄存器	*+~	用于与系统剪切板交互, 以及接收拖放操作的内容
黑洞寄存器	_	放到这里面的内容都被丢弃，这样可以删除或拷贝时不影响其它寄存器
最后一次搜索模式寄存器	/	保存最后一次搜索的正则表达式

  - 将文本内容送入寄存器的命令为：
    ["char][n]y[<cursor_movement>|y|w|l]
  - char表示寄存器的名字，命令的小写表示覆盖写入，大写表示追加写入，n表示重复次数
  - <cursor_movement>表示想要写入寄存器的文本的范围，从光标的当前位置算起。
    * y表示以行为单位
    * w表示以词为单位
    * l表示以字符为单位
  - 从寄存器中提取其内容并将其插入在当前位置的命令为：
    ["char]p|P
  - P表示放在当前位置之前，p表示放在当前位置之后
  - 使用:reg 命令可以看到所有寄存器中的内容
  - Reference: :help registers

- 在vi内使用shell
  - 用户在运行vi的过程中不用退出vi就可以运行任何 UNIX 命令：
  - :!<unix_cmds><CR>
  - 在unix_cmds参数中，可以使用%作为当前文件名的缩写，用#作为上次编辑文件的缩写，用!作为上次命令的缩写。

- 宏（macro）
  - Normal mode 下输入q<reg>，<reg>指 {a-zA-Z0-9"} 共 37 个寄存器中的一个，然后可以进行任何操作，包括在模式间切换，最后在 Normal 模式下按q可以结束宏录制，用@<reg>命令可以应用这个宏，命令前可以带数字前缀表示执行多少次这个宏。

  - Reference: :help q

- 书签（bookmark）
  - Normal mode 下输入m<reg>制作书签，<reg>指 26 个命名寄存器中的一个，然后可以用'<reg>或者`<reg>跳到指定的书签处。
  - Reference: :help m

- 缩写（abbreviation）
  - :abbr|ab str1 str2
  - 用自定义的字符串str1来代替字符串str2
  - :unab|una str1
  - 取消缩写str1
  - 上述四个命令都是行方式下的命令
  - Reference: :help ab

- 映射（map）
  - :map key cmds_list
  - 此时key必须是一个单独的字母。或是一个Ctrl加上一个字母
  - 此时定义的是映射命令
  - :map! key string此时key必须是一个单独的字母，或是一个Ctrl加上一个字母
  - 此时定义的是映射字符串，用于文本输入方式
  - :unmap key将取消key的映射定义
  - 要注意的是在输入Ctrl+key的转义序列时必须先按下组合键Ctrl-v
  - 如果cmds_list是行方式下的命令，那么在其末尾必须加上一个回车，输入方法为先输入Ctrl-v，然后输入Ctrl-m。Ctrl-m代表回车。
  - Reference: :help map

- :s//和:g//, :!g//
  - 这两个命令加上正则表达式，常常能完成非常复杂的编辑任务，可以毫不夸张地说是vim的两柄瑞士军刀。
  - :s是替换操作，:g是查找匹配模式的行，:!g是查找不匹配模式的行。
  - http://www.vim.org/tips/tip.php?tip_id=1063这个 tip 可以把:g找到的行拷贝到一个新的缓冲区中。
  - Reference: :help :s :help :g

- 插件（plug-in）
  - vim自己有脚本语言，另外也支持用 Perl/Python/Tcl/Ruby 编写插件，这些插件极大地丰富了vim的功能。

- 配色方案（color scheme）
  - vim有许多配色方案，下面这个链接有许多配色方案效果的图样:http://www.cs.cmu.edu/maverick/VimColorSchemeTest/

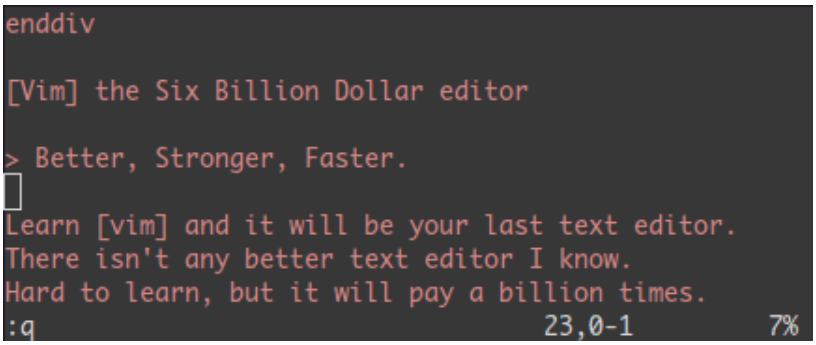


Figure 1: 分屏示意图

6 分屏

- 示意图：
- 分屏启动vim
  - 使用大写的O选项来垂直分屏。
    ```
    vim -On file1 file2 ...
    ```
  - 使用小写的o选项来水平分屏。
    ```
    vim -on file1 file2 ...
    ```
  - 注释: n是数字，表示分成几个屏。
- 关闭分屏
  - 关闭当前窗口。
    ```
    Ctrl+W c
    ```
  - 关闭当前窗口，如果只剩最后一个了，则退出 Vim。
    ```
    Ctrl+W q
    ```
- 打开分屏
  - 上下分割当前打开的文件。
    ```
    Ctrl+W s
    ```
  - 上下分割，并打开一个新的文件。
    ```
    :sp filename
    ```
  - 左右分割当前打开的文件。
    ```
    Ctrl+W v
    ```
  - 左右分割，并打开一个新的文件。
    ```
    :vsp filename
    ```
- 移动光标

  vi中的光标键是 h, j, k, l，要在各个屏间切换，只需要先按一下 Ctrl+W
  - 把光标移到右边的屏。
    ```
    Ctrl+W l
    ```
  - 把光标移到左边的屏中。
    ```
    Ctrl+W h
    ```

  - 把光标移到上边的屏中。
    `Ctrl+W k`
  - 把光标移到下边的屏中。
    `Ctrl+W j`
  - 把光标移到下一个的屏中。.
    `Ctrl+W w`

- 移动分屏

  这个功能还是使用了vim的光标键，只不过都是大写。当然了，如果你的分屏很乱很复杂的话，这个功能可能会出现一些非常奇怪的症状。

  - 向右移动。
    `Ctrl+W L`
  - 向左移动
    `Ctrl+W H`
  - 向上移动
    `Ctrl+W K`
  - 向下移动
    `Ctrl+W J`

- 屏幕尺寸

  下面是改变尺寸的一些操作，主要是高度，对于宽度你可以使用`Ctrl+W <`或是 `Ctrl+W >`，但这可能需要最新的版本才支持。

  - 让所有的屏都有一样的高度。
    `Ctrl+W =`
  - 增加高度。
    `Ctrl+W +`
  - 减少高度。
    `Ctrl+W -`

7 编程辅助

- ctags, cscope
  - 现在的 IDE 都提供了类、函数的索引功能，可以方便的找到某个类或者函数在哪里定义的，vim这方面可以利用ctags、cscope做到
  - Exuberant ctags 支持的语言种类非常多，UltraEdit 的 tags 功能也是利用的 ctags
  - cscope 只支持 C，它能实现 Source Insight 的一些功能，比如查找某个函数调用了哪些函数，某个函数被哪些函数调用
  - vim对这两个工具集成的非常好，利用它们就可以在源文件中方便地跳转搜索类和函数了
  - Reference: http://ctags.sourceforge.net Exuberant ctags http://iamphet.nm.ru/cscope/ cscope for Win32 `:help ctags :help cscope`

- multi window, multi buffer, multi tab page
  - 一个 buffer 对应一个文件，它可以对应多个 window，这样可以方便地对照编辑一个文件的不同部分

  - tab page 跟现在许多编辑器上常见的标签页意义并不一样，可以将 tab page 理解为一个 windows 的容器，这样如果想新建一个窗口编辑文件但又不想打乱现在的多窗口布局，那么就可以新开一个 tab page，把新窗口放到这个新的 tab page 里头
  - tab page 是vim 7.0 新增的特性
  - Reference: `:help windows` `:help buffers` `:help tabpage`

- 语法高亮
  - vim发行版里带了 450 多种语言的语法高亮, 在其主页vim.org上还可以找到更多
  - Reference: `:help syntax`

- 自动缩进
  - 在打开自动缩进选项后，vim会自动的控制缩进
  - 输入{自动向右缩进一个 tab 字符（具体用什么缩进可以配置）, 输入}自动回退缩进
  - 使用=命令可以对选择的程序块排版缩进
  - 对选择的块用>>和<<命令可以很方便的控制一个程序块的缩进
  - Reference: `:help cindent` `:help autoindent` `:help smartindent`

- 类和函数列表
  - taglist 等插件可以提供很方便的类和函数列表功能。

- 自动完成（auto complete）
  - vim在 Insert mode 下输入一个单词的前几个字符，然后用Ctrl-p或者Ctrl-n就可以列出以这些字符开头的单词，特别在配置了 ctags 后也能列出头文件中的符号
  - vim还有行自动完成、文件名自动完成等
  - 如果设置了如下选项:

```
1 :set wildmode=list:full
2 :set wildmenu
```

    则在命令行打开文件或者输入命令时按Tab（以输入字符为前缀补全）或者Ctrl-d（列出包含输入字符的所有匹配项目）自动补全
  - 另外有许多插件可以实现许多 IDE 中的自动完成类成员的功能
  - Reference: `:help 'complete'`

- 折叠（folding）
  - vim支持折叠代码，还可以根据文件中特殊的标记对文件中的行折叠，可以实现大纲（Outline）视图
  - Reference: `:help fold`

- 集成开发（quickfix）
  - 许多人对于 UNIX 开发的印象都是 “编辑代码，退出编辑器，编译，发现错误，记录出错信息，遍辑代码，退出编辑器，编译，用gdb调试，再编辑代码……”，这是古老的vim时代的事情了，有了vim的 quickfix 特性，我们可以在vim里编译然后直接跳到编译出错的行，这个反复的过程无需退出编辑器，而且vim的 quickfix 特性可以经配置后支持不同的编译器以及不同的语言：只要编译器在出错信息里包含文件名和行号。
  - 不过比较遗憾的是vim对于集成调试支持还不好，有些项目尝试集成gdb到vim，比如http://skawina.eu.org/mikolaj/vimgdb/和http://www.volny.cz/zellerin/gdbvim/，在http://www.vim.org/search.php中的 scripts 处搜索 gdb 也可以找到一些插件。这方面 Emacs 的 GUD(Grand Unified Debugger) 调试界面要更强大，毕竟 GCC/GDB/Emacs 是一家。
  - Reference: `:help quickfix`

8 示例

vi的命令可以非常快捷的做到一些复杂的编辑操作，下面是几个示例:

1. 将 “(1), ..., (2), ..., (100)” 替换成 “(2), ..., (3), ..., (101)”。

   - 在文本中一处处找到并修改是很累的，在vi下一条命令就可以搞定:

```
:%s/(\(\d+\))/\="(".(submatch(1)+1).")"/g
```

   - 命令解释如下:

%	替换范围为所有行（“%” 是 “1,$” 范围的缩写）
s	替换
/	搜索字符串开始
(	左括号
\(	开始记录匹配
\d+	一个或多个数字
\)	结束记录匹配
)	右括号
/	搜索字符串结束
\=	把后面的表达式计算出来作为替换字符串
"("	左括号
.	字符串连接运算符
(submatch(1) + 1)	把第一个匹配的结果加一作为一个整体返回
.")"	添上右括号
/g	替换字符串结束，g表示替换每一行的所有匹配结果

2. 有一个 log 系统对于输出行长度有限制，因此在输出很长的 log 时需要断行，在断行时以单行的 “-$-” 标记，现在的需求是把这些行连起来，在vim中也可以很方便地做到：

```
:%s/\n-\$-\n//g
```

   命令的含义就是把 “\n-$-\n” 替换成空，另一个办法是：

```
:g/-\$-/norm ddkJx
```

   命令g/-\$-/的含义就是找到所有的断行标记, 然后norm表示在找到的每一个行上执行后面的命令，dd删除这个断行标记，k移动到上一行，J合并当前行和下一行，由于J合并后会留一个空格（只对于英文情况下），所以x来删除这个空格。

3. 在 Fortran 代码

```
integer,dimension(:,:),allocatable ::  &
  short_var, &
  A_very_long_name_var, &
  other
```

   中，想把每行的 & 符号定位到固定的一列，比如第 78 列，手动对齐也是很麻烦的，这个问题可以用vim的 Align 插件（http://www.vim.org/scripts/script.php?script_id=294）解决，另外利用宏也可以：

:set ve=all	使得光标可以定位到屏幕任何位置（缺省下vim 的光标只能放在行内的字符上）
gg	到文件第一行
/\s\+&\s*$	找以 & 结尾的行, & 前至少一个空白字符, 后 0 或多个空白字符
qa	记录宏到寄存器 a
D	删除到行尾
78|	定位到第 78 列
i&<ESC>	插入 & 并返回到命令模式
n	查找下一个 &
q	停止记录宏
10000@a	执行一万遍寄存器 a 中保存的命令
:set ve=""	恢复 ve 缺省值

4. 连续插入 72 个等号：

   按ESC进入 Normal mode，输入72i=再按ESC即可。

5. 在多行开始插入//：

   移动光标到需要注释掉的第一行开头，然后按Ctrl-v（如果开启了vim的 mswin 行为，则Ctrl-v表示粘贴，这时需要用Ctrl-q代替）进入 Visual blockwise 模式，这个模式是 Visual mode 的一种，相当于 UltraEdit 中的块选择。然后按j选择上所有需要注释行的行首 (看起来效果是选择了第一列)，输入I//再按 ESC 就可以在每一行开头插入 “//” 了。

9 vim 的配置

- vim的配置共有三种方法，一种是在运行vim时使用行命令set来设置；一种是使用EXINIT环境变量；最后一种是使用用户主目录下的.exrc文件（或.vimrc文件）。
- vim的显示是输出到终端上的，所以终端的类型会对vim的显示造成影响。终端类型阿设置是使用TERM环境变量：

  $TERM=ansi;export $TERM

- 此外，由于vim可能会有很多发行版本，为了得知当前的vim发行版本的安装位置，可以在命令行方式使用echo $VIM获取vim配置文件的地址，使用echo $HOME获取vim的安装主目录。

9.1 行方式下的设置

- 为控制不同的编辑功能，vim提供了很多内部选项。在行方式下使用命令set可以显示和修改vim的各种内部环境变量。
- 基本语法：:set argument[=value]
- 命令set的常用参数及其功能如下：

all	列出所有选项设置的情况
term	设置终端类型
ignorecase	在搜索中忽略大小写
list	显示制表符（^I）和行尾标志（^M）
number	显示行号

report	显示由面向行的命令修改过的行数
ruler	在屏幕底部显示光标所在行、列的位置
terse	显示简短的告警信息
warn	显示简短的未保存告警
nomagic	取消元字符在搜索字符串中的特殊性
nowrapscan	搜索时不回绕
mesg	允许vim显示其他用户用write写到自己终端上的信息
shiftwidth	指定自动缩进的制表位
autoindent	自动缩进
directory	指定编辑缓冲区的路径
showmode	显示当前模式
window	设置显示的文本行数
tabstop	设置按Tab键跳过的空格数。例如:set tabstop=n，n默认值为 8

9.2 永久设置方式

- 上面所说的:set命令在退出vim后就失效了，下次进入vim还需要重新设置。
- 我们可以将上述:set命令写在一个名为~/.exrc或~/.vimrc 的文件中，这样每次进入vim就会自动执行。
- 或者将需要用到的:set命令保存在EXINIT环境变量中。

10 帮助系统

- 使用了那么多软件，只有vim和 Emacs 的帮助系统给我方便快捷的感觉，大部分软件的帮助往往是摆设而已，而vim的帮助的确是考虑到了自己 “help” 的身份，利用它能很方便地找到想要的东西。
- vim的帮助是超链接形式的，它使用的就是 tags，所以可以跟 ctags 功能一样按Ctrl-]跳转到链接所指处，按Ctrl-o返回。

:help	打开帮助首页, 这个首页分类非常清楚
:help cmd	查找 normal mode 命令, 比如:help dd
:help i_cmd	查找 insert mode 命令, 比如:help i_Ctrl-y
:help :cmd	查找 command-linemode 命令, 比如:help :s
:help 'option	查找选项, 比如:help 'tabstop

这些信息都在:help打开的帮助首页上

- 如果你记不清命令或者选项的全称, 那么可以利用Tab或者Ctrl-d的自动补全功能。:help options可以找到所有的选项说明
- 查看某一个选项的值（实际上选项是vim中的一种变量，类似 shell 的变量以$符号引用，vim的选项以&引用，另外vim的寄存器以@引用）：:echo &tabstop

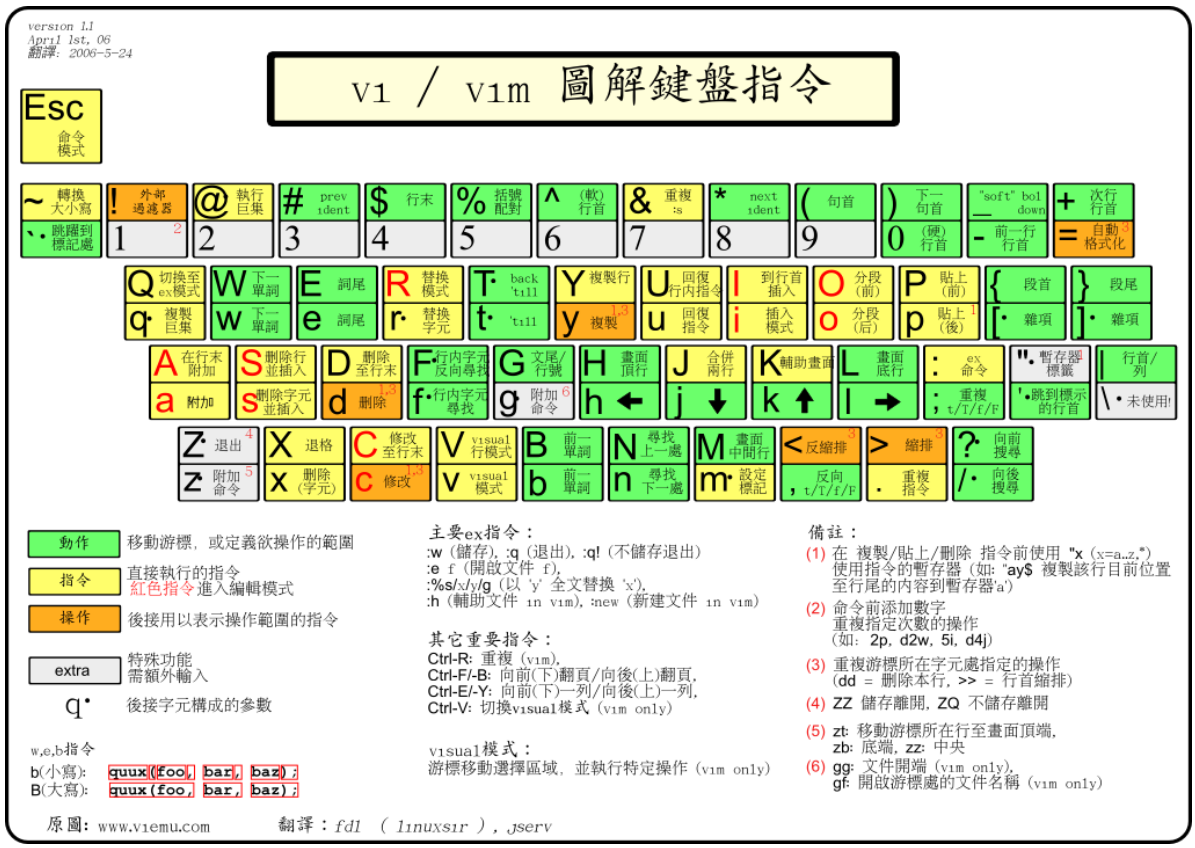
11 vim 资源

- http://www.vim.org, vim主页，有许多 scripts 和 tips，查找插件的第一去处。
- http://newsmth.net, newsmth 的vim版，有很多vim爱好者可以讨论
- http://vimdoc.sf.net, vim文档工程

- http://vcd.gro.clinux.org/, vim中文文档
- http://tnerual.eriogerg.free.fr/vim.html, vim Quick Reference Card
- http://edyfox.codecarver.org/, newsmth vim版版主的 wiki
- 《学习 vi 编辑器（第六版）》机械工业出版社译，O'Reilly
- http://weitz.de/regex-coach/, Regex Coach
- http://jregexptester.sourceforge.net/, JRegexpTester
- http://www.yzis.org/, yzis是一个新的vi变种，它支持变宽字体，目前功能还不够丰富
- http://ex-vi.sourceforge.net/, 传统vi的源代码

12 小结

- 前面已经提到，vim在自动完成和集成调试方面还比不上现代的许多 IDE
- vim对二进制编辑还没有 UltraEdit 强大，但是作为一个文本编辑器而言，堪比的只有 Emacs，
- vim也可以嵌入到 Visual Studio 中作为编辑器，另外 Code Forge, Eclipse 等也提供了一定的vim支持或者键绑定。
- vim一定要多用才能掌握。



The End of Chapter IV.