

实用技巧

级别： 初级

吴 咏炜 (wuyongwei@gmail.com), 软件架构师

2006 年 3 月 22 日

本系列文章分三部分详细阐述了 Vim 的使用技巧、插件、定制。第一部分主要是深入分析了用。

0. Vim 简介

作为开源世界最重要的编辑器之一（另一个是 Emacs），Vim 以其强大的功能和可定制能力被众多不过，也许就是因为 Vim 的功能太强大了，要真正用好 Vim 并不容易。本文作者在多年的实际使用些实用技术，在此介绍给大家。——本文并不企图对 Vim 作全面而系统的介绍，但也绝非零星地点望通过介绍一些重要特性和提供相关参考信息，引起大家的兴趣，去深入挖掘其能力，真正把这一张

下面首先对 Vim 做一下最基本的介绍，并给出一些参考信息，以方便对 Vim 不熟悉的读者也能够理一步信息。

与大部分其它编辑器不同，进入 Vim 后，缺省状态下键入的字符并不会插入到所编辑的文件之中。（mode，可以简单地理解为“状态”）概念非常重要。需要知道，Vim 有以下几个模式：

- 正常（normal）模式，缺省的编辑模式；下面如果不加特殊说明，提到的命令都直接在正常模其它模式中都可以通过键盘上的 Esc 键回到正常模式。
- 命令（command）模式，用于执行较长、较复杂的命令；在正常模式下输入“:”（一般命令搜索）或“?”（反向搜索）即可进入该模式；命令模式下的命令要输入回车键（Enter）才算
- 插入（insert）模式，输入文本时使用；在正常模式下键入“i”（insert）或“a”（append）式（也有另外一些命令，如“c”，也可以进入插入模式，但这些命令有其它的作用）。
- 可视（visual）模式，用于选定文本块；可以在正常模式下输入“v”（小写）来按字符选定，写）来按行选定，或输入“Ctrl-V”来按方块选定。
- 选择（select）模式，与普通的 Windows 编辑器较为接近的选择文本块的方式；在以可视模选定文本块之后，可以使用“Ctrl-G”切换到另一模式——该模式很少在 Linux 上使用，本文了。

Vim 带有完整的帮助文档。在当前的 Vim 6.4 的标准发布中，有一百多章、近六十万英文词的帮助。后输入“:help”（命令模式中输入的命令要敲回车键才结束输入，下面不再说明这一点）即可访问。时，对文档中已经说明得很详细的内容只会提纲挈领地加以简短说明和提供应用范例，并提供访问相的命令。

一般的发布版中还常常带有一个简单的 30 分钟的 Vim 教程，新手在操作系统的命令行上输入“vin 开始学习。除上面的简单说明外，本文并不介绍最基本的 Vim 命令，Vim 的新手应该先通过教程熟继续往下阅读。

建议所有的 Vim 用户经常访问 Vim 的主站点 [1]。上面除了基本的发布、安装、下载等信息外，最户可以上传自己写的 Vim 脚本（script）和撰写自己认为有用的提示（tip），供其他 Vim 用户使用时候，Vim 站点上已有一千三百多个脚本，提示数刚好超过了一千。对于序号为 nn 的脚本，直接访问 http://www.vim.org/scripts/script.php?script_id=nn；对于序号为 nn 的提示，直接访问的 URL 是 http://www.vim.org/tips/tip.php?tip_id=nn。

不另加说明的话，本文讨论的内容适用于 Vim 版本 6（即从 6.0 到 6.4）。建议认真的 Vim 用户最好是自己编译升级所有的补丁包。相关信息网站上都有，此处不再赘述。

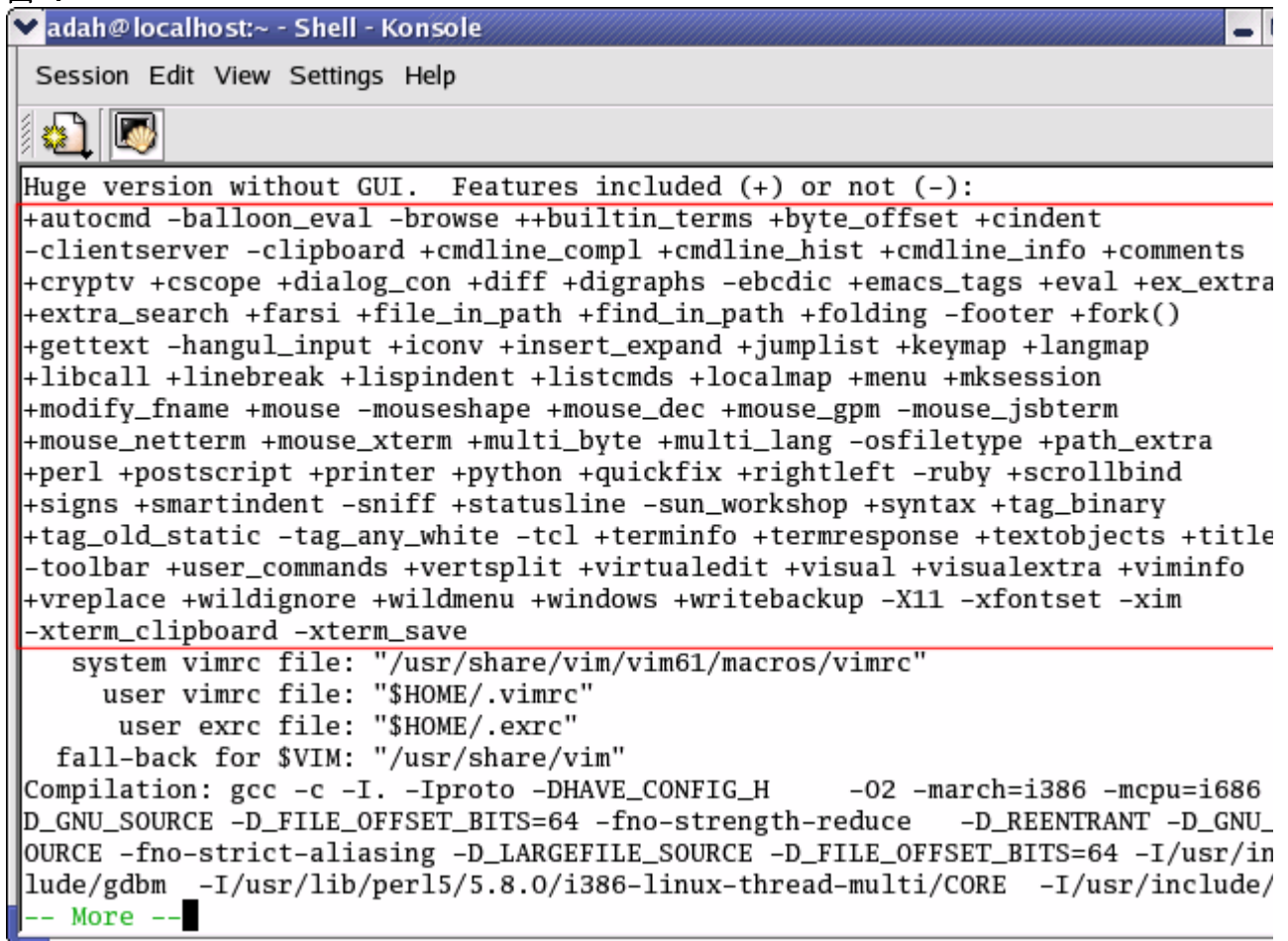
1. 实用技巧

1.1 安装

如果从 Linux 发布版直接安装 Vim，需要注意的一点是，缺省情况下系统并不一定为你安装了一个完整的 Vim。例如，在 Red Hat（以及后来的 Fedora Core）的发布版中，Vim 被拆成了四个包：vim-common（公共部分），vim-minimal（最小安装），vim-enhanced（除 X Window 支持外的完整安装），和 vim-X11（X Window 支持）。最小安装不能完整展示 Vim 的优点，通常只是作为 vi 的替代品出现，缺少很多重要的特性如 X Window 支持、鼠标支持和脚本支持。如果装了 X Window 的话，图形界面的 gvim 也比文本模式的 vim 具有优势。建议大家尽可能安装完全的 Vim。

如果愿意稍稍费一点功夫，自己编译 Vim 的话，可以更好地定制 Vim。——附带的另一个好处是，如果你遇到错误的话，你就可以自己动手来修复这个错误，或至少找到错误所在的位置，让 Bram（Vim 的作者）来帮你解决问题。图 1 是在 Vim 中执行“:version”的结果的一部分，可以看到 Vim 有很多不同的特性（features）可以随时打开或关闭。如果自己编译的话，就可以选择打开需要的功能，关闭不需要的功能，来获得一个既小巧快速的 Vim 定制版本。

图 1



```
adah@localhost:~ - Shell - Konsole
Session Edit View Settings Help

Huge version without GUI. Features included (+) or not (-):
+autocmd -balloon_eval -browse ++builtin_terms +byte_offset +cindent
-clientserver -clipboard +cmdline_compl +cmdline_hist +cmdline_info +comments
+cryptv +cscope +dialog_con +diff +digraphs -ebcdic +emacs_tags +eval +ex_extra
+extra_search +farsi +file_in_path +find_in_path +folding -footer +fork()
+gettext -hangul_input +iconv +insert_expand +jumplist +keymap +langmap
+libcall +linebreak +lispindent +listcmds +localmap +menu +mksession
+modify_fname +mouse -mouseshape +mouse_dec +mouse_gpm -mouse_jsbterm
+mouse_netterm +mouse_xterm +multi_byte +multi_lang -osfiletype +path_extra
+perl +postscript +printer +python +quickfix +rightleft -ruby +scrollbind
+signs +smartindent -sniff +statusline -sun_workshop +syntax +tag_binary
+tag_old_static -tag_any_white -tcl +terminfo +termresponse +textobjects +title
-toolbar +user_commands +vertsplits +virtualedit +visual +visualextra +vminfo
+vreplace +wildignore +wildmenu +windows +writebackup -X11 -xfontset -xim
-xterm_clipboard -xterm_save

    system vimrc file: "/usr/share/vim/vim61/macros/vimrc"
     user vimrc file: "$HOME/.vimrc"
   user exrc file: "$HOME/.exrc"
 fall-back for $VIM: "/usr/share/vim"

Compilation: gcc -c -I. -Iproto -DHAVE_CONFIG_H      -O2 -march=i386 -mcpu=i686
-D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -fno-strength-reduce -D_REENTRANT -D_GNU_
SOURCE -fno-strict-aliasing -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -I/usr/in
clude/gdbm -I/usr/lib/perl5/5.8.0/i386-linux-thread-multi/CORE -I/usr/include/
-- More --
```

1.2 中文支持

Vim 支持世界上的主要语言，当然也包括中文。如果你用 Vim 编辑中文，而中文不能正确显示，那一是使用的 Vim 不完整，不含多字节语言支持（multi_byte 特性）；二是某个配置出了问题。

说到多语言支持，最基本的概念有两个：一是文件的语言编码，而是环境的内部编码。在较老的操作系统 Linux 还是 Windows，这两个编码都是一样的，也就意味着，一次只能处理一种编码的文件：要么是 Latin1，即 ISO-8859-1 [5]，要么只能处理中文编码（GB2312 [2]）。而在新的操作系统中，是这样的。在 Linux 上，常见的情况是环境的内部编码使用 UTF-8 [6]，而 UTF-8 可以同任何一种语言换，这就保证了系统的多语言处理能力。Vim 这方面秉承了 Unix/Linux 的传统，在内部编码使 UTF 同时处理不同意语言编码的文件。

以下列出了和语言编码的相关的设置：

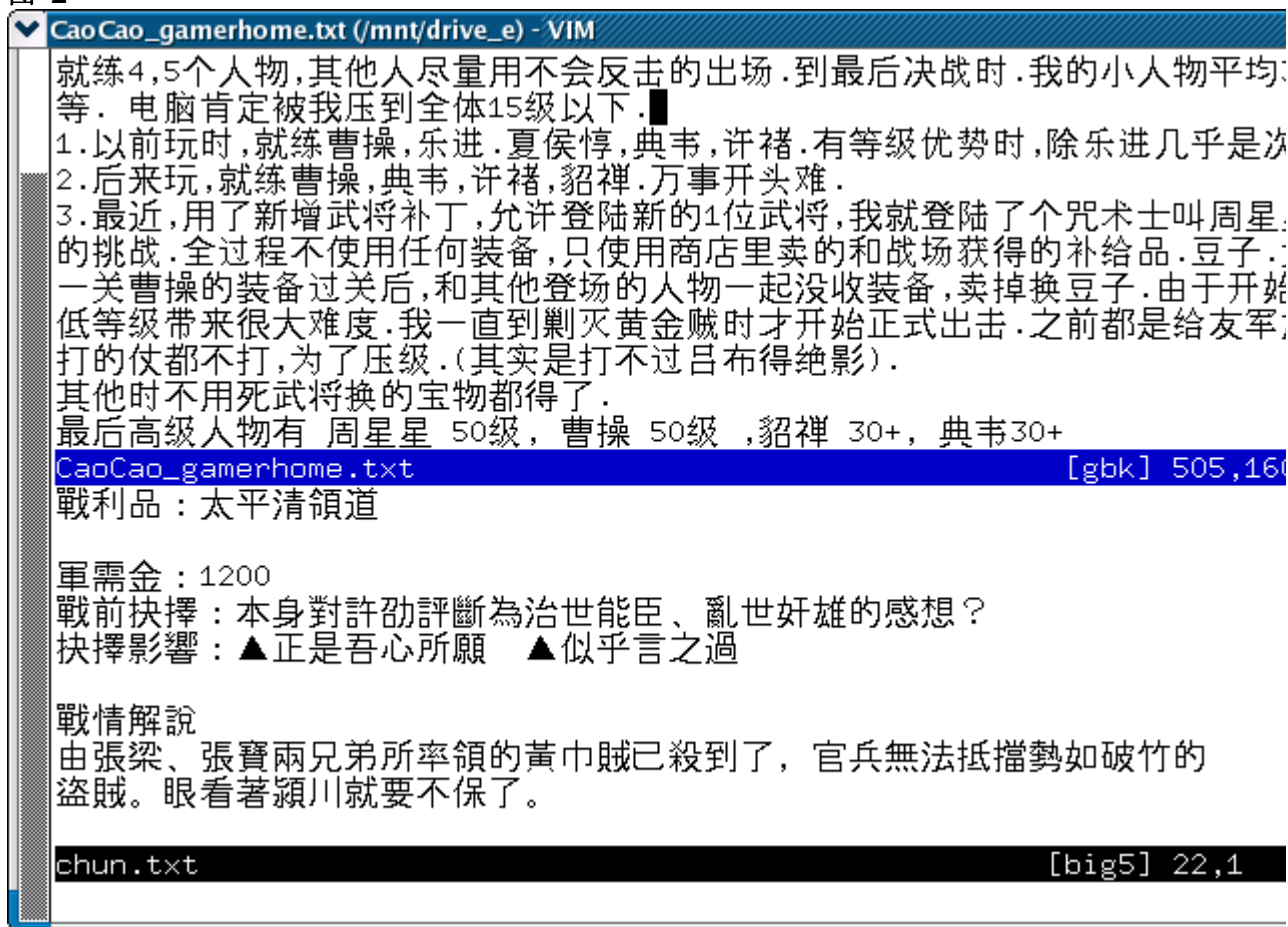
- 环境变量 LANG（使用的语言）；
- 环境变量 LC_CTYPE（使用的内部编码）；
- Vim 选项 encoding（Vim 的内部编码）；

- Vim 选项 `termencoding` (Vim 在与屏幕/键盘交互时使用的编码)；
- Vim 选项 `fileencoding` (Vim 当前编辑的文件在存储时的编码)；
- Vim 选项 `fileencodings` (Vim 打开文件时的尝试使用的编码)；
- Vim 选项 `ambiwidth` (对“不明宽度”字符的处理方式；Vim 6.1.455 后引入)。

如果你的环境只需要处理简体中文的话，那么，最简单的方式就是所有的设定全部使用简体中文。只设定 `LANG=zh_CN.GB2312`，不设定 `LC_CTYPE`（默认跟 `LANG` 一样），不设定与编码相关的 Vim 选项（`LANG` 和 `LC_CTYPE` 决定），也无需设定 Vim 选项 `ambiwidth`。也就是说，我们把语言设定为中文（zh），编码为 GB2312（注意：Vim 内部并不识别国标 GB18030 [3]，所以此处只能设 GB2312，关于 UTF-8 的讨论）。

不过，如果按照目前 Linux 下的惯例，内部编码一律使用 UTF-8 的话，会有一些额外的好处，其中情况下 Vim 支持同时编辑多种不同编码的文件，如简体中文和繁体中文（参见图 2）；另外，此时编码转换支持 GBK [4] 和 GB18030 了。这样，众多关于语言编码的 Vim 选项就有了用武之地了。一下这些选项和推荐设定（如果适用的话）：

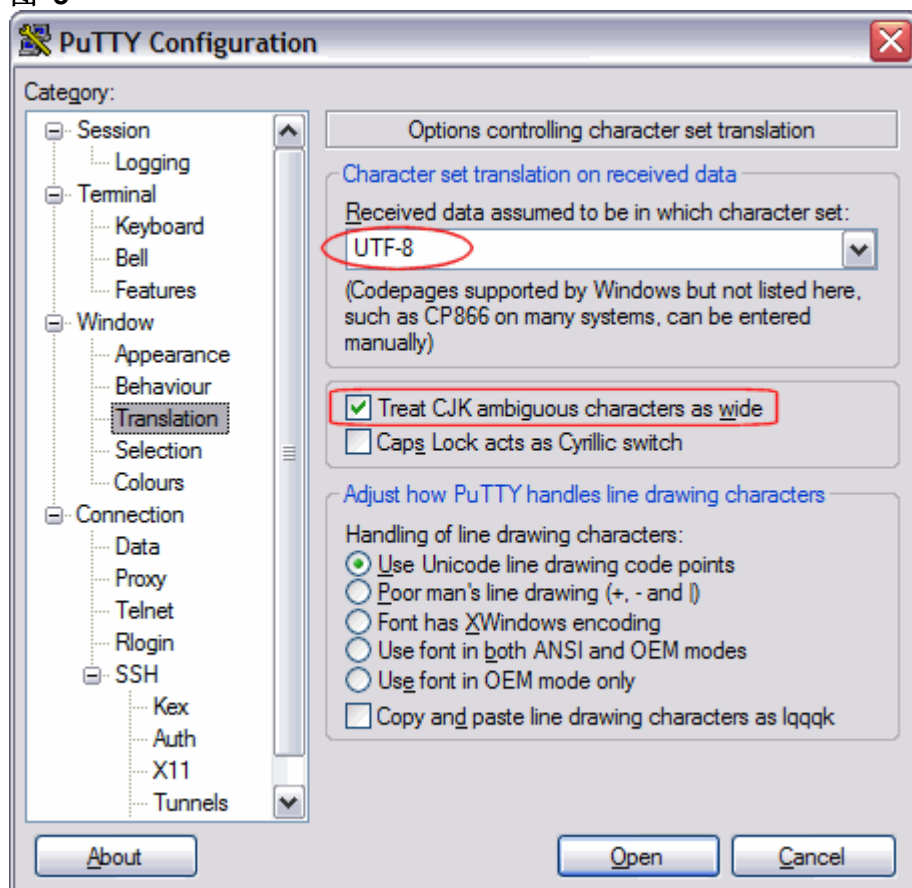
图 2



- `encoding=utf-8`: 不管文件的编码如何，不管如何显示和输入，Vim 内部使用的编码是 UTF-8 的基础。
- `termencoding`: 取决于实际的终端或 X Window 的设定。举例来说，如果选择语言简体中文至 X Window，或者正在使用 `CXTERM` [10] 的话，那么该选项应被设为 `GB2312`；如果使用缺省值（`LANG=en_US.UTF-8`）登录到 X Window，或者使用 `PuTTY` [11] 远程访问 Linux 机器、并编码（配置中 `Window-Translation`）设为 UTF-8 的话，该选项就应该设为 `utf-8`。从 `Window` 或 `PuTTY` 远程连接 Linux 的请特别注意，测试表明，仅在使用 UTF-8 的情况下，`PuTTY` 才能正确地显示和输入（显示字体必须设成中文字体）。
- `fileencoding`: 文件载入时，该选项被置为 Vim 认定的文件编码，因此，存储时文件的编码不下面 `fileencodings` 可使用的编码为 `libiconv` 支持的所有几百种编码（如果编译时包含了 `iconv` 与中文相关的有 `gb2312`、`gbk`、`gb18030`、`hz-gb-2312`、`iso-2022-cn`、`big5`、`cp936`、`cp950` 文件，你又不希望使用 UTF-8 作为文件编码时，那么，你可能需要手工设定该选项，如 “`:set fileencoding=gb2312`”。需要注意的一点是，使用 “`set`” 来设定该选项的话会改变以后新建文件的编码，而使用 “`setlocal`” 的话则只影响当前文件（参考 “`:help setlocal`”）。

- `fileencodings=ucs-bom,utf-8,chinese`: Vim 会首先判断文件的开头是否是一个 Unicode [7] 的 order mark) 字符 [8], 是的话则把文件的其余内容解释成相应的 Unicode 序列; 否的话再试释成 UTF-8 的序列; 再失败的话, 则把文件解释为简体中文 (chinese 是一个跨平台的简体中文名, Linux 下相当于 gb2312 和 euc-cn; 此处也可以根据需要进行以 gb2312、gbk 或 gb18030; 需要注意的是, 该顺序不能颠倒, 并且在后面再添加其它编码如 big5、latin1 也是没有意义的能识别 8 比特编码中的错误, 因此这些编码后列的编码永远不会被用到。
- `ambiwidth=double`: 把所有的“不明宽度”字符 [9]——指的是在 Unicode 字符集中某些同时使用的字符, 如省略号、破折号、书名号和全角引号, 在西方文字中通常字符宽度等同于普通在东方文字中通常字符宽度等同于两倍的普通 ASCII 字符, 因而其宽度“不明”——的宽度量 (中文字符宽度)。此数值只在 `encoding` 设为 `utf-8` 或某一 Unicode 编码时才有效。需要额外如果你通过终端使用 Vim 的话, 需要令终端也将这些字符显示为双宽度。比如, XTERM [12] 的选项 “-cjk”, 即使用命令 “`uxterm -cjk`” 来启动使用双宽度显示这些字符的 Unicode X 终端远程连接的话则应在配置的 Window-Translation 中选中 “Treat CJK ambiguous characters as wide (图 3)。

图 3



需要设定的选项通常放在用户的 Vim 资源配置文件中, 即在 `~/.vimrc` 文件中加入:

```
set encoding=utf-8
set fileencoding=chinese
set fileencodings=ucs-bom,utf-8,chinese
set ambiwidth=double
```

如果想进一步了解这些选项的话, 可以使用 “:help '选项’” 查看帮助文档中的相关 (英文) 信息。帮这些选项 (以及命令) 的缩写: 本文中为方便理解, 除一些极少有人使用完整拼写的命令如 “:e(edit)(ubstitute)” 等之外, 一般使用完整拼写而不说明或使用缩写。关于配置文件 `.vimrc`, 可以使用 “:h相关信息”。

在使用内部编码 UTF-8 的情况下, 如需编辑 `fileencodings` 之外 (其不能自动识别) 的文件, 则可以 “:e ++enc=编码 文件名”。详情可参考 “:help ++enc”。

1.3. 鼠标支持

不管是文本界面还是图形界面的 Vim，都支持鼠标。不过，在文本界面中，鼠标支持缺省没有被激活。在终端上使用鼠标，所有的功能仍和没有使用 Vim 时相同，并不受 Vim 影响。要激活文本界面中的鼠标，只需要执行一句“`:set mouse=a`”即可。

启用了鼠标支持之后，Vim 主要支持的鼠标操作有：

- 单击移动光标到点击的位置；
- 在帮助的关键字上双击显示该关键字相关的帮助信息；
- 在普通文本上双击选中点击位置的单词；
- 拖动鼠标选中文本；
- 使用鼠标滚轮滚动当前缓冲区中的文本；
- 多窗口编辑时可以拖动窗口分栏的位置。

进一步的信息可参看“`:help 'mouse'`”、“`:help mouse-using`”和“`:help scroll-mouse-wheel`”。

特别需要值得一提的是，在远程访问 Linux 系统时也是可以使用鼠标的。如果使用 X Window 系统而使用 SSH 远程连接时，大部分 Linux 下的终端客户程序，如 XTERM、GNOME-Terminal [13]、Konsole [14]，以及 Windows 下的 PuTTY，支持鼠标的使用：你只需简单地启动 Vim、执行一句就可以了（当然，也可以把上面的语句去掉起始的冒号放到 .vimrc 文件中）。

1.4. 空格、制表符和缩进

对于编写代码，缩进是最基本的概念之一。至于缩进是使用空格还是制表符（Tab），或者缩进是否用制表符来表示，很多程序员，特别是 Windows 开发出身的程序员，很容易混淆。幸好，Vim 对于这些的支持，足以应付各种复杂的情况。以下是相关的主要 Vim 选项：

- `shiftwidth`（缩进的空格数）；
- `tabstop`（制表符的宽度）；
- `expandtab`（是否在缩进和遇到 Tab 键时使用空格替代；使用 `noexpandtab` 取消设置）；
- `softtabstop`（软制表符宽度，设置为非零数值后使用 Tab 键和 Backspace 时光标移动的格数实际插入的字符仍受 `tabstop` 和 `expandtab` 控制）；
- `autoindent`（自动缩进，即每行的缩进值与上一行相等；使用 `noautoindent` 取消设置）；
- `cindent`（使用 C 语言的缩进方式，根据特殊字符如“{”、“}”、“:”和语句是否结束等缩进；在编辑 C/C++ 等类型文件时会自动设定；使用 `nocindent` 取消设置）；
- `cinoptions`（C 语言缩进的具体方式，请参考“`:help cinoptions-values`”）；
- `paste`（粘贴模式，会取消所有上述选项的影响来保证后面的操作——通常是从剪贴板粘贴代码的风格；使用 `nopaste` 取消设置）。

下面给出一些常用的组合：

- `shiftwidth=4 tabstop=4`：很多 Windows 出身的程序员会习惯这样的设置，让缩进等于制表符
- `shiftwidth=4 tabstop=8`：很多 Unix 程序员的设置，仍使用较常用的 4 格缩进，但制表符宽度
- `cinoptions=>4,n-2,{2,^2,:2,=2,g0,h2,p5,t0,+2,(0,u0,w1,m1 shiftwidth=2 tabstop=8`：标准的（设置，对 Vim 缺省的 C 缩进风格作了很多微调，比如，if 语句下的“{”、“}”要在“if”后数定义部分“{”、“}”仍和函数名一行对齐。开源软件经常使用该种缩进风格。

在编辑代码时一个很有用的命令是调整代码缩进，可以很方便地增加（或减少）若干级缩进，并自动用正确的空格或制表符。只需要使用“V”选中你要调整的代码行，然后键入“<”（或“>”）即可一级缩进；在键入“<”（或“>”）之前键入数字则可以指定增加（或减少）的缩进级数。

我们要讨论的最后一个相关的命令是“`:retab`”。在设定了 `expandtab` 选项时，该选项会把所有的制表符。在没有设定 `expandtab` 选项时，使用“`:retab!`”可把空白字符转换成制表符（可能误转换，慎用）。“`:retab n`”可以把 `tabstop` 重置为 n，并转换含制表符的连续空白字符为适当的制表符和空格的组合。行的行看起来没有任何变化。详细信息请参看“`:help :retab`”。

1.5. 模式行（modeline）

没人愿意每次都手工输入一大堆的 Tab 和缩进设定。可是，放在 .vimrc 文件中似乎也不是个好主意。代码不止一种风格呢？——考虑一下，如果你参加开源软件项目，你能保证你参加的所有项目，还有项目，代码风格都一样吗？——Vim 是我用过的第一个支持在文件中记录代码风格设定的编辑器。这中叫做模式行，实际上，它所做的是在打开文件时根据文件中的 Vim 指令设定相关的 Vim 选项。下

C 源代码中的模式行：

```
/* vim: set tabstop=4 shiftright=4 expandtab: */
```

模式行有好几种形式。本文只介绍上面的这种形式（其它形式类似，请自行参考“:help modeline”和尾部的“*/”告诉 C 编译器这是一行注释，不是代码的一部分；而 Vim 可通过后面的“vim:”识别（必须出现在行首或前面有一个空白字符）；后面则是“set”和空格间隔开的一串 Vim 选项；“束”。

这种方式非常简单，功能也非常强大。另外请注意，出于安全的考虑，模式行中的选项只影响当前文件（“modeline-local”），也不能做任何设置选项以外的工作。

1.6. 寄存器

通常的编辑器有一个剪贴板，以存储复制和剪切的内容。Vim 中的类似概念叫做寄存器（register）。有名寄存器外，Vim 还有一大堆有名的寄存器，可以通过“”（参见“:help ””）或“Ctrl-R”（参见“R”和“:help c_CTRL-R”）加寄存器名（字母、数字和某些特殊字符，参见“:help registers”；名字是“”）来访问。比如，你先使用“ayy”复制了一行，然后使用“dd”删掉了一行，然后移到的位置，就可以使用“aP”把先前复制的内容粘贴上去了。手工编辑是有名寄存器的作用还不是让 Vim 通过类似于宏的方式自动完成工作时，有名寄存器就变成不可缺少的重要功能了。下面我们

在使用 X Window 系统时，有两个特殊的寄存器是需要注意一下的：“”访问的寄存器是 X 的主（primary selection），“+”访问的寄存器是 X 的剪贴板（clipboard）。如果你要在 Vim 和其它间复制文本内容，你可以试一下这两个寄存器。

还有一个很特殊的“寄存器”：“=”。在插入模式或命令模式中，键入“Ctrl-R=”，Vim 会提示作式，普通的整数运算在此完全有效。如果想要进行浮点运算，请参见第 3.2 节中的技巧。

1.7. 搜索、替换和正则表达式

大家应该都已经知道 Vim 里使用“/模式”（或“?模式”）进行搜索，使用“:s/模式/字符串/标志”的“模式”是一个正则表达式。关于正则表达式，不熟悉的话可以边用边学，本节也不打算对 Vim 完整的阐述（那可能可以专门写一本小册子了），而只抛砖引玉式地给出一些有用的例子加以说明，巧。

先说一点点搜索。搜索里最最有用的一个快捷方式是“*”（向下完整匹配光标下的单词）。把光标下的词（变量名、函数名等）上，比如“test”，然后按“*”，Vim 将自动产生一个对“<test>”（参见“:help <”）和“:help >”）的搜索，也就是说，搜索完整的单词“test”。不要小看这个技巧，它经常可以大大提高速度。事实上，这是 Vim 网站上公布的第 1 号技巧，也是被评价最高的技巧。相似的技巧还有“#”（匹配光标下的单词）、“g*”（向下部分匹配光标下的单词）等，请自行查看（“:help #”等）。

Vim 在搜索和替换时会对匹配成功的文本进行加亮，在已经完成搜索和替换任务后，这种加亮有时烦 Vim 专门提供一个命令取消这种加亮（直到用户再一次使用搜索或替换命令）：“:nohlsearch”。该键盘映射（key mapping）加入到 .vimrc 中，如：

```
nmap <F2> :nohl search<CR>
```

以上命令表示，在正常模式下按 F2 键相当于输入“:nohlsearch”后面跟一个回车，即取消搜索加亮。再看几个搜索替换的实用例子。

- 去掉所有的行尾空格：“:%s/\s+\$//”。 “%”表示在整个文件范围内进行替换，“\s”表示空制表符，“+”对前面的字符匹配一次或多次（越多越好），“\$”匹配行尾（使用“\s”表示字符）；被替换的内容为空；由于一行最多只需替换一次，不需要特殊标志。这个还是比较简单。
- 去掉所有的空白行：“:%s/\s*\n\|^\n//”。这回多了“\n”、“\n”、“\n”、“\n”和“*”。前面的字符（此处为“\s”）匹配零次或多次（越多越好；使用“*”表示单纯的“*”字符），符，“\r”代表回车符，“\n”和“\n”对表达式进行分组，使其被视作一个不可分割的整体。式的完整意义是，把连续的换行符（包含换行符前面可能有的连续空白字符）替换成为一个单一很特殊的地方是，在模式中使用的是“\n”，而被替换的内容中却不能使用“\n”，而只能是历史造成的，详情如果有兴趣的话可以查看“:help NL-used-for-Nul”。

- 去掉所有的“//”注释：“:%s!\s*//.*!!”。首先可以注意到，这几分隔符改用了“!”，原因是部分使用了“/”字符，不换用其他分隔符的话就得在每次使用“/”字符本身时写成“V”，上“:%s!\s*V.*//”，可读性较低。命令本身倒是相当简单，用过正则表达式的人估计都知道“.”行符之外的任何字符吧。
- 去掉所有的“/* */”注释：“:%s!\s/*\s*\{-\}\s*!lg”。这个略有点复杂了，用到了几个不太的表达式特性。“\{-\}”匹配包含换行在内的所有字符；“\{-\}”表示前一个字符可出现零次或多。表达式可以匹配成功的前提下，匹配的字符数越少越好；标志“g”表示一行里可以匹配和替换。结果是个空格的目的是保证像“int/* space not necessary around comments */main()”这样的后仍然是合法的。

希望上面的这些简单的例子能够引起你使用 Vim 的正则表达式高效完成任务的兴趣。进一步的信息“regexp”。

1.8. 自动完成和路径设定

Vim 支持单词的自动完成。比如，你前面使用了一个很长的变量名，叫 `aLongVariable`，下面你在编辑键入了。很可能，你只需要键入“aL”，然后按下“Ctrl-P”（向前搜索可匹配的单词并完成）就变量名（没有得到想要的结果的话，多按几下“Ctrl-P”；或者前面多输入几个字符，如“aLongV”还有“Ctrl-N”（向后搜索可匹配的单词并完成）、“Ctrl-X Ctrl-L”（搜索可匹配的行并完成）、“（搜索可匹配的文件名并完成）等，具体可参看“:help ins-completion”。

如果你并不熟悉这些功能，但也并不觉得这有什么稀奇的话，下面这个例子可能会让你觉得吃惊。请白的 C 文件（vim test.c），并输入：

```
#include <stdio.h>
int main()
{
    pri
```

最后一行不要回车，直接在“pri”后面输入“Ctrl-P”，你将看到“printf”出现。是的，虽然文件里但 Vim 知道到哪里去寻找它！在作关键字匹配完成时，如果当前文件和其它打开的文件中没有想要自动到“#include”的文件中进行进一步的搜索（为什么是“#include”呢？请查阅“:help 'include'”录则由选项 `path` 决定，其缺省值在 Unix（含 Linux）下为“./usr/include,,”，代表搜索的目录依、/usr/include 和当前目录。根据实际情况，你可能需要在 .vimrc 文件中设置该选项，加入项目相注意一般要保留最后的“,,”，除非你不需要在当前目录下搜索。

设置了合适的 `path` 后，另外带来的一个便利就是可以使用“gf”命令方便地跳转到光标下的文件名中。在上面的例子中，把光标移到“stdio.h”的任一字符上，键入“gf”，则 Vim 会自动打开 /usr/文件。使用“Ctrl-O”（参见“:help CTRL-O”）可返回到原先的文件中。

1.9. 文件跳转和 Tags

大家一般都知道，在 Vim 的帮助窗口中的关键字上双击鼠标或者键入“Ctrl-]”即可跳转至该关键字题。不过，“跳转至匹配的关键字”这一功能并不仅仅局限于帮助文件。只要有合适的 `tags` 文件（`tags-file-format`），我们同样可以在源代码中使用这个方便的功能，跳转到与关键字匹配的“标记代码中某一函数、类型、变量或宏的定义位置”。

要产生 `tags` 文件，通常我们使用 `Exuberant Ctags` [15]。一般的 Linux 发布版中均带有这一工具。项数量极多，此处我们仅简单介绍如何在一个典型的多文件、多层目录的项目中使用其基本功能：到根目录处键入“`ctags -R .`”，`Ctags` 即可自动在文件中查找、识别支持的文件格式、生成 `tags` 文件。`Exuberant Ctags` 支持多达 33 种编程语言 [16]，包括了 Linux 下常用的 C、C++、Java、Perl、Ptags 文件，以下的 Vim 命令就可以方便使用了（进一步的信息可参考“:help tags-and-searches”）。

- `:tag` 关键字（跳转到与“关键字”匹配的标记处）
- `:tselect` [关键字]（显示与“关键字”匹配的标记列表，输入数字跳转到指定的标记）
- `:tjump` [关键字]（类似于“:tselect”，但当匹配项只有一个时直接跳转至标记处而不再显示列
- `:tn`（跳转到下一个匹配的标记处）
- `:tp`（跳转到上一个匹配的标记处）
- `Ctrl-]`（跳转到与光标下的关键字匹配的标记处；除“关键字”直接从光标位置自动获得外，功同）
- `g]`（与“Ctrl-]”功能类似，但使用的命令是“:tselect”）

- **g Ctrl-]**（与“Ctrl-]”功能类似，但使用的命令是“:tjump”）
- **Ctrl-T**（跳转回上次使用以上命令跳转前的位置）

当我们在项目的根目录下工作时，上面这些命令工作得很好。但如果我们进到多层目录的里层再运行时，这些命令的执行结果通常就变成了错误信息“E433: No tags file”。这是因为缺省 Vim 只在当前目录下寻找 tags 文件，而我们前面只在项目的根目录下生成了 tags 文件，Vim 无法找到该文件。一种，我认为一般较简单的做法是对每个项目都在 .vimrc 文件中增加一个路径相关设定。假设我们置分别是 /home/my/proj1 和 /home/my/proj2，那我们可以使用：

```
au BufEnter /home/my/proj 1/* setlocal tags+=/home/my/proj 1/tags
au BufEnter /home/my/proj 2/* setlocal tags+=/home/my/proj 2/tags
```

Vim 选项 tags 用于控制检查的 tags 文件，缺省值为“./tags,tags”，即前面所说的文件所在目录下的 tags 文件。上面两行自动命令告诉 Vim，在打开项目目录下的文件时，tags 选项中的内容要增加项路径。进一步信息可参看“:help 'tags'”。

1.10. Make 和 grep

Make [17] 和 grep [18] 应当算是 Unix 世界里无人不晓的基本工具了吧。很自然的，Vim 对它们有着支持主要通过访问一个特殊的快速修订窗口（quickfix window）来实现。直接在 Vim 的命令模式里：make 或 grep 命令（如“:grep foo *.c”）即可将命令的执行结果放入该窗口，同时根据返回的结果误（make 的情况；在使用 grep 时是匹配成功之处）。以下是常用的“快速修订”命令：

- **:cn**（显示下一个错误）
- **:cp**（显示上一个错误）
- **:cl**（列出所有的错误及其编号）
- **:cc**（跳转到指定编号的错误）
- **:copen**（打开快速修订窗口，在其中显示所有错误，可在错误上双击鼠标或按回车键跳转至该图 4）

图 4


```
adah@test81:~/athene/knids/test/MultiTcpSession
    ipSrc.s_addr = option.getSrcRandIpAddress();
    ipDst.s_addr = option.getDstRandIpAddress();
}

pCloneTcpSession->modify_IpAddress(ipSrc, ipDst);
pCloneTcpSession->do_checksum();

g_multiSession.add(*pCloneTcpSession);

if (g_nVerbose >= 2) {
    // Prints first 10 sessions.
    if (g_nVerbose == 2 || i < 10) {
        cout << "=====" << i + 1 << endl;
        pCloneTcpSession->test_PrintPacketData();
    }
    // Prints all sessions.
    if (g_nVerbose == 3) {
        cout << "=====" << i + 1 << endl;
    }
}

SessionMain.cpp [utf-8] 227,12-21
MyPacket.cpp|94| printf("CTcpPacket::do_checksum:Not enough data! \n ");
MyPacket.cpp|105| printf("build_tcp_ipfrag: libnet_do_checksum failed\n");
MyPacket.cpp|110| if(1 != libnet_do_checksum(pIpData, IPPROTO_IP, (pIpHdr->
    <<2))) {
MyPacket.cpp|111| printf("build_tcp_ipfrag: libnet_do_checksum failed\n");
MyPacket.cpp|203| bool CTcpSession::do_checksum()
MyPacket.cpp|207| if( (*iterator)->do_checksum() != true ) {
SessionMain.cpp|232| pCloneTcpSession->do_checksum();
MyPacket.h|122| virtual bool do_checksum();
MyPacket.h|203| bool do_checksum();
[Error List] [-] [utf-8] 9,1-39
:
```

- :cclose（关闭快速修订窗口）

Vim 的这个特性也可以与 `make` 和 `grep` 以外的程序一起工作（事实上，在 Windows XP 上，“:gr 起的是“findstr /n”）。具体调用那个程序由选项 `makeprg`（Linux 下缺省为“make”）和 `greppr` 为“`grep -n $* /dev/null`”）控制，而如何解析返回的内容则由选项 `errorformat` 和 `grepformat` 控制 Unix/Linux 下一般不需更改这些选项的内容，此处不再详述。

1.11. 执行外部命令

在“:make”这样的命令中，Vim 会自动调用外部的程序。用户当然也可以自己执行外部的程序：估计知道了用“:!命令”可以在 Vim 中执行一个外部命令。不过，估计大部分人都不知道，还有其它一外部命令，并且，即使“:!”命令里面也有一些技巧可以使用。

最正规的执行外部命令的方法，如前所述，就是“:!”。比如，我们想要显示当前目录下的所有文件行：“:ls”。Vim 会在一个终端窗口中进行文件列表，然后提示我们按键返回 Vim 中。事实上，这“cp”、“rm”这样基本不需要输出的命令比较实用，而对于“ls”这样关注于输出的命令并不太足

如果想把外部命令执行的结果插入到当前编辑的缓冲区中，可以考虑使用“:r!”。比如，我们使用“把“ls”命令的执行结果插入到缓冲区中光标所在行下面。在使用宏时，这可能会特别有用。

Vim 的“:!”命令还有一个特别强大的技巧可以使用。拿一个实际例子，我们需要对在一个文件的每个编号，该怎么做呢？——用 Vim 的宏或者脚本可以完成这一工作，但这不是最高效、最灵活的工一般带有的 GNU 的 nl，可以用非常灵活的方式来完成这一任务——要对所有的非空行进行编号，!nl”；要对包含空行的所有行进行编号？OK，“:%!nl -ba”。

稍作一点解释。当使用可视模式选中文本行后然后键入“:!”（命令行上将出现“:!<,>!”，表示命令文本），或者使用“:%!”（表示命令的范围是整个缓冲区中的文本），Vim 在执行后面的命令时，:的文本行作为后面执行的命令标准输入，并用命令执行后的标准输出替换当前缓冲区中的这些文本行命令行的工作原理。

1.12. 定宽文本排版

在传统的 Unix 环境下，文本文件的定义是具有一定长度限制的文本行的组合 [19]。虽然 Vim 本身没有任何实际的限制，但有一些工具有这样的限制。为了最大程度的兼容性，也为了在显示、打印等处理一般推荐在邮件和源代码中一般不要超出 72 列（最多不超出 80 列）。Vim 在处理定宽的文本方面具力。下面是一个在 Vim 中把行宽（使用选项 `textwidth`）设为 40 后输入 `Harry Potter and the Half-` 第一句话的结果：

```
It was nearing midnight and the Prime
Minister was sitting alone in his
office, reading a long memo that was
slipping through his brain without
leaving the slightest trace of meaning
behind.
```

输入时我只使用了英文字母和空格，换行符都是 Vim 自动插入的。如果在某一行加入或删除了一些齐了吗，该如何处理？很简单，把光标移到要重新格式化的文本开头，使用“`gq`”命令后面跟一个重新格式化的范围。比如“`gq}`”（格式化一段），“`gq5j`”（格式化 5 行），“`gqG`”（格式化至

除了选项 `textwidth` 外，选项 `formatoptions` 确定了跟文本格式化有关的基本选项，常用的数值有：

- `t`: 根据 `textwidth` 自动折行；
- `c`: 在（程序源代码中的）注释中自动折行，插入合适的注释起始字符；
- `r`: 插入模式下在注释中键入回车时，插入合适的注释起始字符；
- `q`: 允许使用“`gq`”命令对注释进行格式化；
- `n`: 识别编号列表，编号行的下一行的缩进由数字后的空白决定（与“`2`”冲突，需要“`autoin`”）；
- `2`: 使用一段的第二行的缩进来格式化文本；
- `l`: 在当前行长度超过 `textwidth` 时，不自动重新格式化；
- `m`: 在多字节字符处可以折行，对中文特别有效（否则只在空白字符处折行）；
- `M`: 在拼接两行时（重新格式化，或者是手工使用“`J`”命令），如果前一行的结尾或后一行的符，则不插入空格，非常适合中文

上面提到的注释，可以是 C/C++ 中的“`//`”和“`/*`”，也可以是邮件中引用原文使用的“`>`”等字符（`comments` 选项控制；参见“`:help 'comments'`”）。Vim 在遇到这些字符时，能够相当智能地进行日常编辑源代码和邮件的需要。在使用一些处理纯文本不够强大的邮件客户端时，我通常使用 Vim（是英文邮件），然后把结果贴回到邮件编辑窗口中进行发送。

Vim 中 `formatoptions` 的缺省值是“`tcq`”，一般我会在 `.vimrc` 文件中加入一行“`set formatoptions+=`”。Vim 能在中文字符之间折行而不要求空格的存在，并且在大部分情况下可以正确地处理中文重新格式

1.13. 其它小技巧

也许你会觉得这些很有用：

- `%`（跳转到与之匹配的括号处）
- `.`（重复上次的修改命令）
- ```（跳转到最近修改过的位置）
- `ZQ`（无条件退出）
- `ZZ`（存盘退出）
- `ga`（显示光标下的字符在当前使用的 `encoding` 下的内码）
- `guw`（光标下的单词变为小写）
- `gUw`（光标下的单词变为大写）
- `:TOhtml`（根据 Vim 的语法加亮的方式生成 HTML 代码；在图形界面中也可以使用菜单“`Syntax to HTML`”达到同样效果）

无聊的时候，还可以试试（呵呵！）：

- `:help!`
- `:help 42`
- `:help holy-grail`