Vim 实用技术

技巧、插件和定制

吴咏炜（http://wyw.dcweb.cn/）

最初由 IBM developerWorks 中国网站（Linux 专栏）发表，网址是 http://www.ibm.com/developerworks/cn

目录

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开发者所喜爱。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中逐渐掌握了一些实用技术，在此介绍给大家。——本文并不企图对 Vim 作全面而系统的介绍，但也绝非零星地点到即止；而是希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，真正把这一强大的工具用好。

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解并自己查阅进一步信息。

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。Vim 的模式（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式下输入；任何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）、“`/`”（正向搜索）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*）才算完成。
- 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）即可进入插入模式（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，输入“`V`”（大写）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；在以可视模式和选择模式之一选定文本块之后，可以使用“*Ctrl*-`G`”切换到另一模式——该模式很少在 Linux 上使用，本文中就不再介绍了。

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文件，进入 Vim 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。本文在介绍特性时，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，并提供访问相应的 Vim 文档的命令。

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“`vimtutor`”命令即可开始学习。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悉一下 Vim，再继续往下阅读。

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，最有用的内容是用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。在写这一段的时候，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，直接访问的 URL 是 `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 用户升级到 Vim 6.4，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。比如，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`（公用部分），`vim-minimal`（最小安装），`vim-enhanced`（除 X Window 支持外的完整安装），和 `vim-X11`（X Window 图形界面支持）。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多字节语言支持、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 具有更多的特性。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，你如果发现什么错误的话，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）可以更快地解决问题。图 1 是在 Vim 中执行“:version”的结果的一部分，可以看

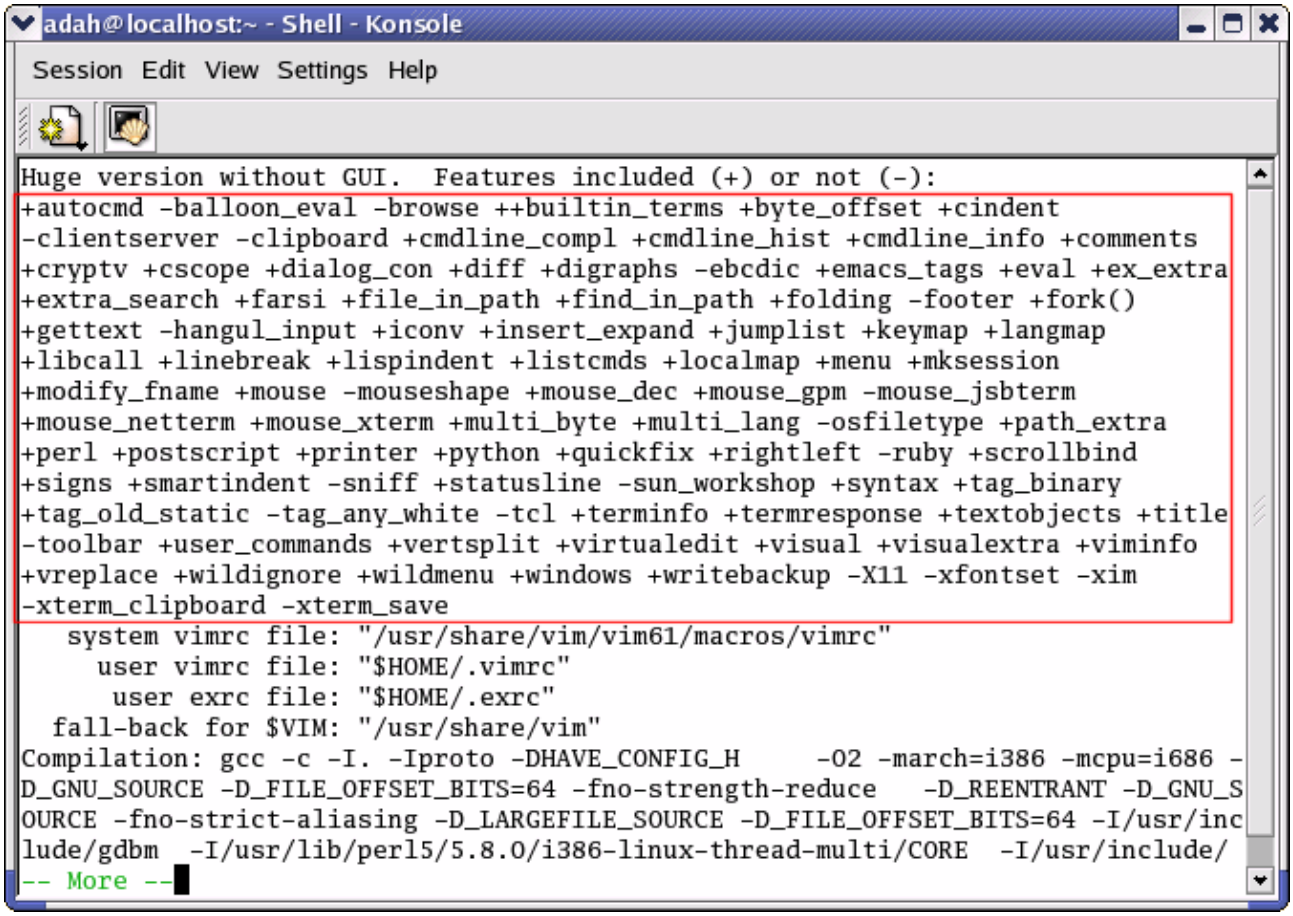

图 1

到 Vim 有很多不同的特性（feature）可在编译时打开或关闭。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，来获得一个既功能强大、又小巧快速的 Vim 定制版本。

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，那有两种可能性：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。在较老的操作系统中，不管 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：要么只能处理西文编码（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，这两者可以是不一样的。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编码作无损转换，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，在内部编码使用 UTF-8 的时候，可以同时处理不同语言编码的文件。

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。只需要：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（默认由 `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国（`CN`）使用的中文（`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；参看下面关于 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，其中之一就是在这种情况下 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

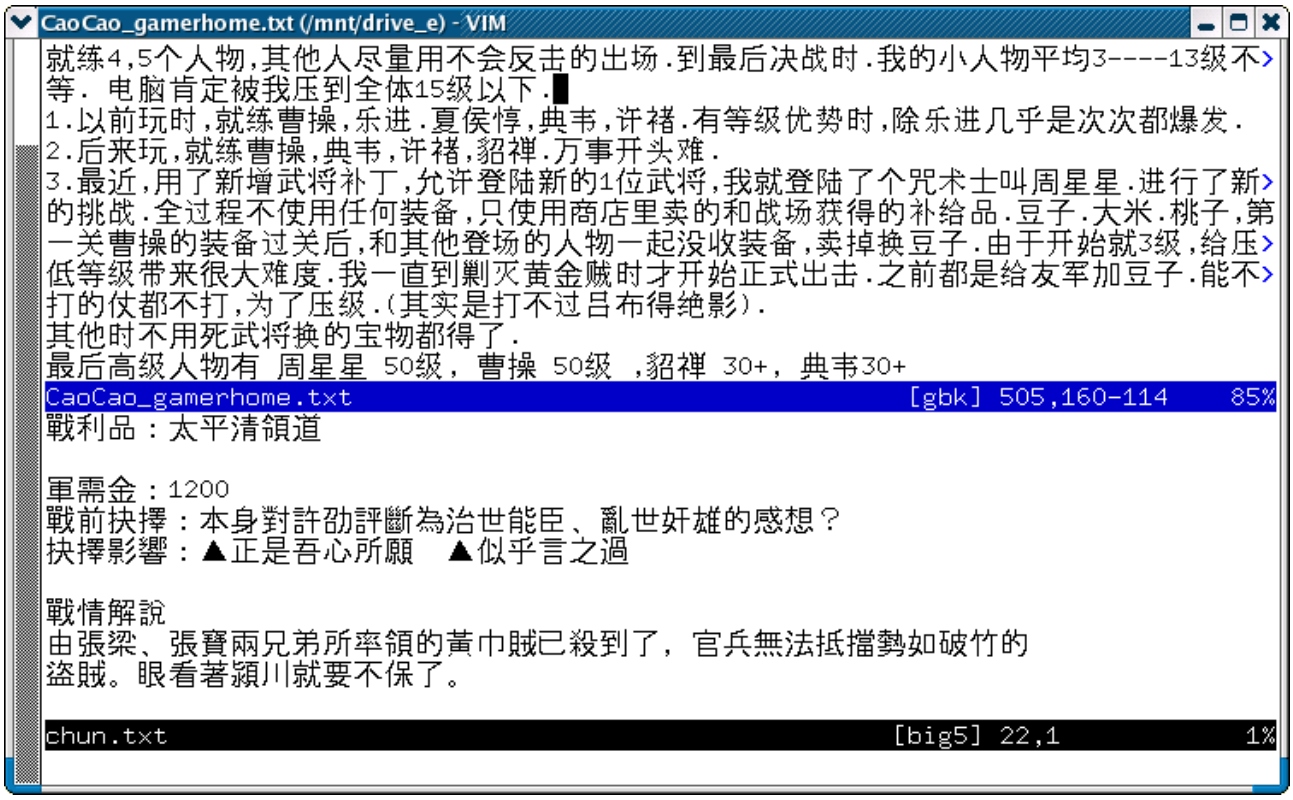

图 2

外，此时 Vim 也可以通过编码转换支持 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。下面进一步说明一下这些选项和推荐设定（如果适用的话）：

- `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；这是国际化支持的基础。
- `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登录到 X Window，或者正在使用 CXTERM [10] 的话，那么该选项应被设为 `GB2312`；如果使用缺省的语言（LANG=en_US.UTF-8）登录到 X Window，或者使用 PuTTY [11] 远程访问 Linux 机器、并且设定里的字符编码（配置中 Window—Translation）设为 UTF-8 的话，该选项就应该设为 `utf-8`。从 Windows 下使用 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 才能可靠地支持中文的显示和输入（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会改变。此处和下面 `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* 特性的话），与中文相关的有 `gb2312`、`gbk`、`gb18030`、`hz-gb-2312`、`iso-2022-cn`、`big5`、`cp936`、`cp950` 等。如果创建新文件，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缺省编码，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 BOM（byte 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；否的话再试图把文件内容解释成 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字符集的别名，Linux 下相当于 `gb2312` 和 `euc-cn`；此处也可以根据需要以 `gb2312`、`gbk` 或 `gb18030` 等编码替代）。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，因为 Vim 不能识别 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 字符集中某些同时在东西方语言中使用的字符，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 ASCII 字符，而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”——的宽度置为双倍字符宽度（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。需要额外注意的是，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] 的情况下应该使用选项“`-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；使用 PuTTY 远程连接的话则应在配置的 Window—Translation 中选中“Treat CJK ambiguous characters as wide”（参见图 3）。

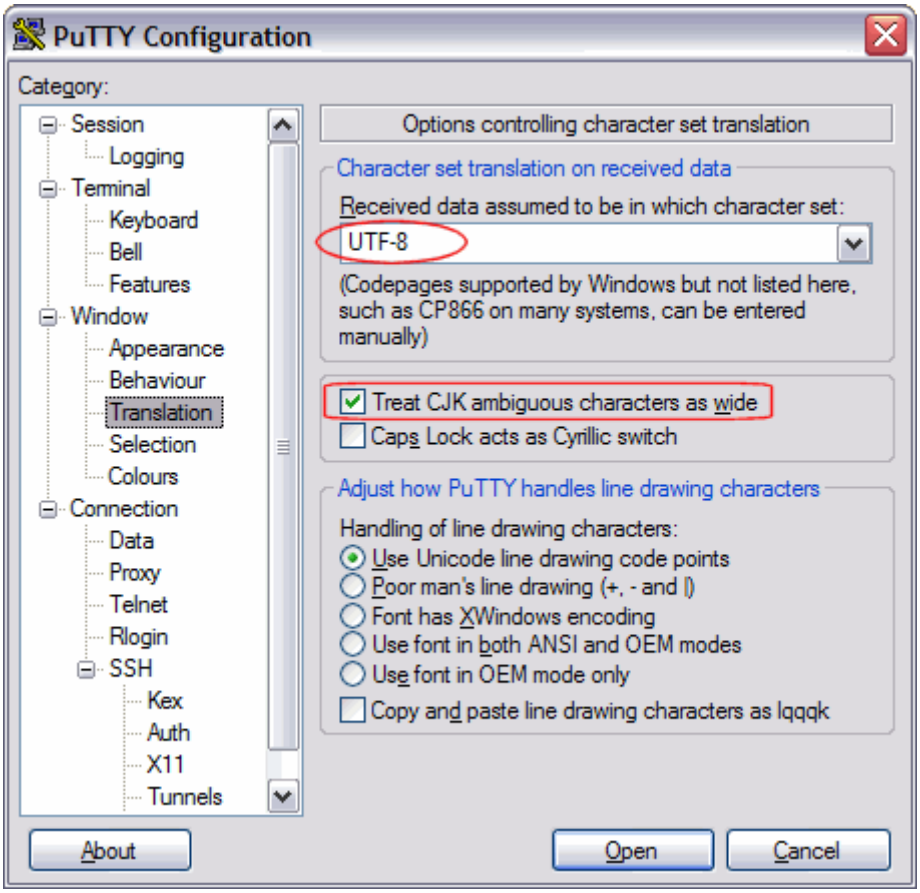

图 3

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。帮助中也可以查到这些选项（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(dit)`”、“`:s(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“`:help .vimrc`”查看相关信息。

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，则可以使用以下命令：“: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；这就意味着，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支持也很容易，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，自然不必说；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]、较新版本的 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“:set mouse=a”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），或者缩进是否正好使用一个制表符来表示，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 对于这些概念有非常完整的支持，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* 时光标移动的格数等于该数值，但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“{”、“}”、“:”和语句是否结束等信息自动调整缩进；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——保持原有代码的风格；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，让缩进等于制表符宽度。
- shiftwidth=4 tabstop=8：很多 Unix 程序员的设置，仍使用较常用的 4 格缩进，但制表符宽度为标准的 8。
- cinoptions=>4,n-2,{2,^-2,:2,=2,g0,h2,p5,t0,+2,(0,u0,w1,m1 shiftwidth=2 tabstop=8：标准的 GNU 编码风格的设置，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”后缩进两格，但函数定义部分“{”、“}”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，并自动根据选项设定使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“>”）即可增加（或减少）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转换成空格。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），使用“: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以保证含制表符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：如果我编辑的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，还有你公司里的软件项目，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。这个特性在 Vim 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。下面就是一个嵌在 C 源代码中的模式行：

```
/* vim: set tabstop=4 shiftwidth=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”）：行首的“/*”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:”识别出模式行的开始（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；“:”表示模式行结束。

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（“:help 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）。除了有一个无名寄存器外，Vim 还有一大堆有名的寄存器，可以通过“""（参见“:help ""”）或“*Ctrl*-R”（参见“:help i_CTRL-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”；“无名”寄存器的名字是“""）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，然后移动光标到要复制到的位置，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很大，但当你想让 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。下面我们还会用到。

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 的主选择区域（primary selection），“"+”访问的寄存器是 X 的剪贴板（clipboard）。如果你要在 Vim 和其它的 X 应用程序之间复制文本内容，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 会提示你输入一个表达式，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*”进行替换，其中的“*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 的正则表达式作完整的阐述（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，以及一些实用技巧。

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）。把光标移动到你要搜索的词（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，它经常可以大幅度地提高搜索的速度。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“#”（向上完整匹配光标下的单词）、“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，这种加亮有时反而会妨碍显示。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”。建议用户创建一个键盘映射（key mapping）加入到 .vimrc 中，如：

```
nmap <F2> :nohlsearch<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，即取消搜索加亮显示。

再看几个搜索替换的实用例子。

- 去掉所有的行尾空格：“:%s/\s\+$//”。“%”表示在整个文件范围内进行替换，“\s”表示空白字符（空格和制表符），“\+”对前面的字符匹配一次或多次（越多越好），“$”匹配行尾（使用“\$”表示单纯的“$”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。这个还是比较简单的。
- 去掉所有的空白行：“:%s/\(\s*\n\)\+/\r/”。这回多了“\(”、“\)”、“\n”、“\r”和“*”。“*”代表对前面的字符（此处为“\s”）匹配零次或多次（越多越好；使用“*”表示单纯的“*”字符），“\n”代表换行符，“\r”代表回车符，“\(”和“\)”对表达式进行分组，使其被视作一个不可分割的整体。因此，这个表达式的完整意义是，把连续的换行符（包含换行符前面可能有的连续空白字符）替换成为一个单个的换行符。唯一很特殊的地方是，在模式中使用的是“\n”，而被替换的内容中却不能使用“\n”，而只能使用“\r”。原因是历史造成的，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，这儿分隔符改用了“!”，原因是在模式或字符串部分使用了“/”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“\/”，上面的命令得写成“:%s/\s*\/\/.*//”，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“.”匹配表示除换行符之外的任何字符吧。
- 去掉所有的“/* */”注释：“:%s!\s*/*_.\{-}*/\s*! !g”。这个略有点复杂了，用到了几个不太常用的 Vim 正则表达式特性。“_.”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次，但在整个正则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多次。替换的结果是个空格的目的是保证像“int/* space not necessary around comments */main()”这样的表达式在替换之后仍然是合法的。

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。进一步的信息可参考“:help 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，下面你在输入时，就不用完整键入了。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）就可以得到完整的变量名（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”）。类似的命令还有“*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）、“*Ctrl*-X *Ctrl*-F”（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。请尝试打开一个空白的 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，虽然文件里没有“printf”，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的结果，Vim 会自动到“#include”的文件中进行进一步的搜索（为什么是

“#include”呢？请查阅“:help 'include'”），搜寻的目录则由选项 path 决定，其缺省值在 Unix（含 Linux）下为“.,/usr/include,,”，代表搜索的目录依次是文件所在目录、/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，加入项目相关的包含目录，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所代表的文件中。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/include/stdio.h 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相关的帮助主题。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（参见“:help 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”处（通常是源代码中某一函数、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。Ctags 带有的选项数量极多，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：我们只需在项目的根目录处键入“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。目前 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、PHP 等。有了 tags 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表）
- :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，功能与“:tags”相同）
- 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 Vim 打开文件时，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 只在文件所在目录和当前目录下寻找 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。解决方法有好几种，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。假设我们有两个项目，位置分别是 /home/my/proj1 和 /home/my/proj2，那我们可以使用：

```
au BufEnter /home/my/proj1/* setlocal tags+=/home/my/proj1/tags
au BufEnter /home/my/proj2/* setlocal tags+=/home/my/proj2/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和当前目录下的 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目的 tags 文件的路径。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 对它们有着特殊的支持。该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 的命令模式里输入相应的 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跳转到第一个错误（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错误；示例参见图 4）

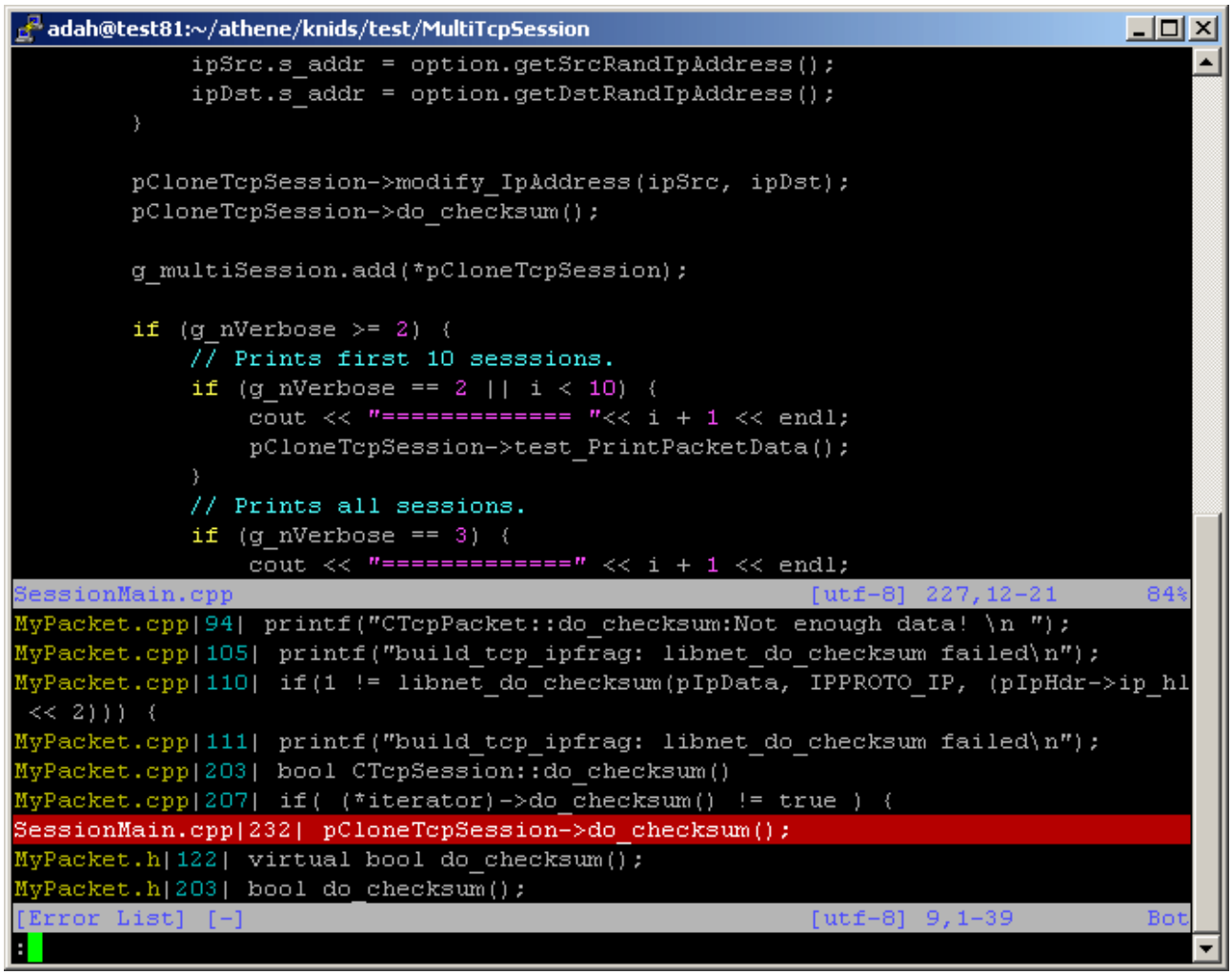

图 4

- :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，“:grep”命令一般调起的是“findstr /n”）。具体调用那个程序由选项 makeprg（Linux 下缺省为“make”）和 grepprg（Linux 下缺省为“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。鉴于在 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：估计很多的人都已经知道了用“:!*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，还有其它一些命令可以执行外部命令，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，就可以直接执行：“:!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，这种方式对于“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适用。

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“:r!ls”，就可以把“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一行之前插入一个编号，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、最灵活的工作方式。Linux 下一般带有的 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，只需要“:%!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“:'<,'>!”，表示命令的范围是选定的文本），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，将把命令范围里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。这就是上面的命令行的工作原理。

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 本身对行的长度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、打印等处理上比较方便，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有特殊的支持能力。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-Blood Prince* 的第一句话的结果：

```
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字符后行不就不齐了吗，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”命令后面跟一个光标移动命令确定重新格式化的范围。比如“gq}”（格式化一段），“gq5j”（格式化 5 行），“gqG”（格式化至文件末尾）。

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“autoindent”）；
- 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开头是多字节字符，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（具体由 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处理，足以完成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 编辑邮件（特别是英文邮件），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+=mM”来确保 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化。

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—Convert 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`

2. 常用插件

前一章介绍了一些基本的 Vim 使用技巧。掌握这些技巧可以很大地提高编辑效率，但是 Vim 的强大功能并不仅限于此。Vim 还可以通过“插件”来进行功能扩展。精确地说，是通过脚本来进行扩展，脚本类型有插件、语法加亮、配色方案、文件类型检测等多种。大部分的脚本都是由 Vim 的用户写的，解决了用户身边的问题，使 Vim 变得更加有用。本章将介绍最常用的一些脚本，其中除了一个属于“语法加亮”脚本外，其它都属于“插件”类型。关于如何写脚本的一些基础知识将在下一章进行一些介绍。

2.1 gzip（压缩文件支持）

作者： Bram Moolenar
网站脚本编号： 无（包含在 Vim 的标准发布之中）
安装说明：

无

功能说明：

该脚本使得 Vim 可以直接打开使用 gzip [20]、bzip2 [21] 和 compress [22] 压缩的文件（要求存在相应的命令行工具）。后缀为“.gz”、“.bz2”和“.Z”的文件会在打开时被动态解压缩，并在写操作时被自动重新压缩。

打开压缩文件时，屏幕上出现一个（不正确的）“`[noeol]`”的提示是正常的，不必进行理睬。

2.2 netrw（网络文件支持）

作者： Charles E. Campbell, Jr.（绰号 Dr. Chip）
网站脚本编号： 1075（Vim 的标准发布之中可能包含一个较老的版本）
安装说明：

Vim 6.4 的标准发布带的版本是 42，较老、功能不齐全，但无须安装。建议：

1. 在 Vim 网站上下载版本 62（更新的版本只能用于 Vim 7）；
2. 使用“`tar xvfj netrw.tar.bz2 -C ~/.vim`”解开；
3. 在 Vim 中运行“`:helptags ~/.vim/doc`”安装文档。

功能说明：

支持直接读写网络上的文件，支持的协议有 ftp、http、rsync、scp 等。比如，使用 FTP 协议以用户

名 adah 打开服务器 server 上 ~/temp 目录下的 test.cpp 文件，可以直接在命令行上使用：

```
vim ftp://adah@server/temp/test.cpp
```

Vim 会自动提示用户输入口令，然后打开文件。

更多的帮助内容请参考“:help netrw”。

2.3 engspchk（英文拼写检查）

作者：Charles E. Campbell, Jr.（绰号 Dr. Chip）

网站脚本编号：195

安装说明：

1. 在 Vim 网站上下载最新版本（engspchk.tar.gz）；
2. 使用“tar xvfz engspchk.tar.gz -C ~/.vim”解开；
3. 在 Vim 中运行“:helptags ~/.vim/doc”安装文档。

提示替换拼写功能需要 agrep [23]，可能需要另外下载安装。

功能说明：

一个 Vim 专用的拼写检查器，其最主要的特点是：

- 可以通过变量 spchkdialect 选择英语变体（英国、美国、加拿大），对于找不到的词、不常见的词、不在当前英语变体中的词以不同的方式加亮显示；
- 支持用户词典（保存在 .vim/CVIMSYN 目录中）和项目词典（保存在被检查文件所在的目录中）；
- 对于源代码文件，只对注释进行拼写检查，而不会对你的变量名称胡乱提抗议；
- 通过词典文件可支持除英语以外的其它语言。

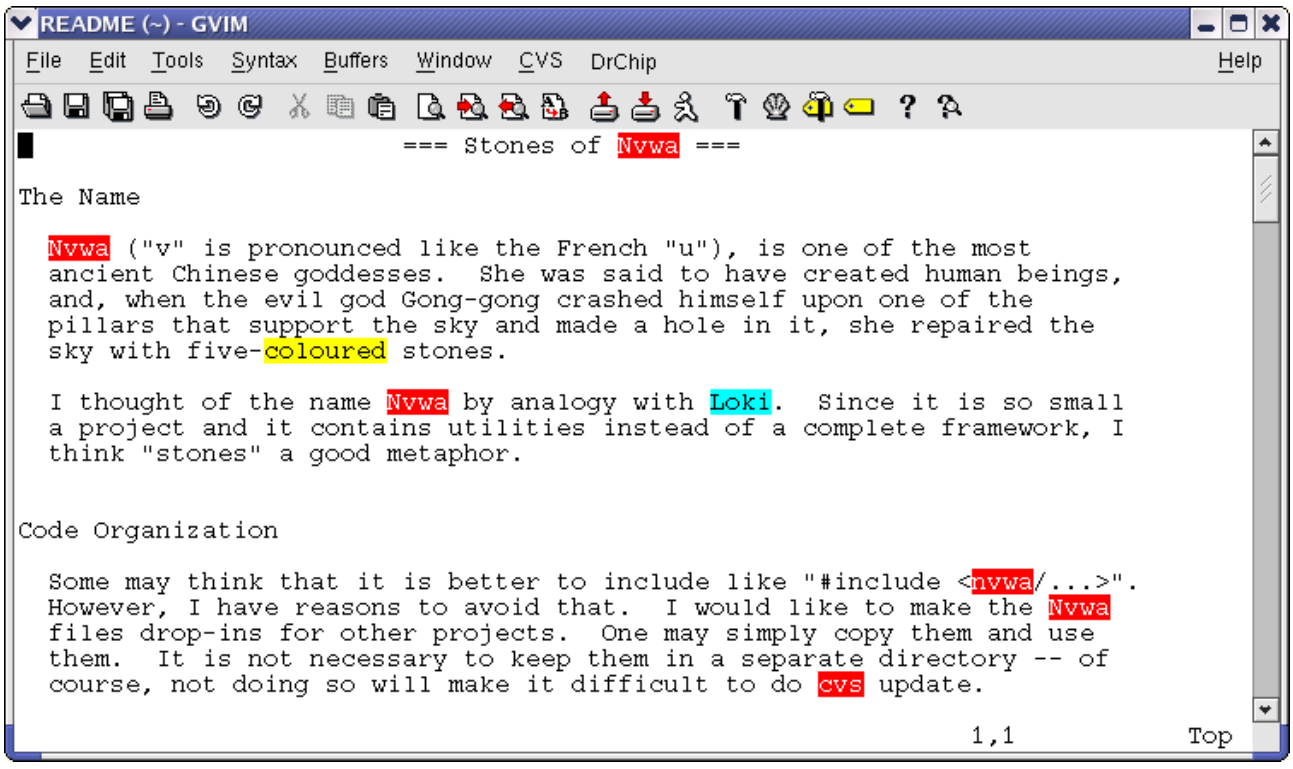

图 5

图 5 中的窗口里 engspchk 正在运行中。其中可以看到，红色部分是插件所不能识别的单词，黄色部分是当前英文变体（缺省是美国英语）中不正确的拼法，而青色部分是很少见的单词。图中运行的是包含图形界面支持的 Vim，因而还可以看到一个名为 DrChip 的菜单项（参见图 6），在其中可通过菜单选择拼写检查相关的各项功能；同时还能看到快捷键：“\ec”开始拼写检查，“\ee”结束拼写检查，“\ea”选择替换拼写，等等。在文本模式的 Vim 中，我们一般就只使用这些快捷键了。

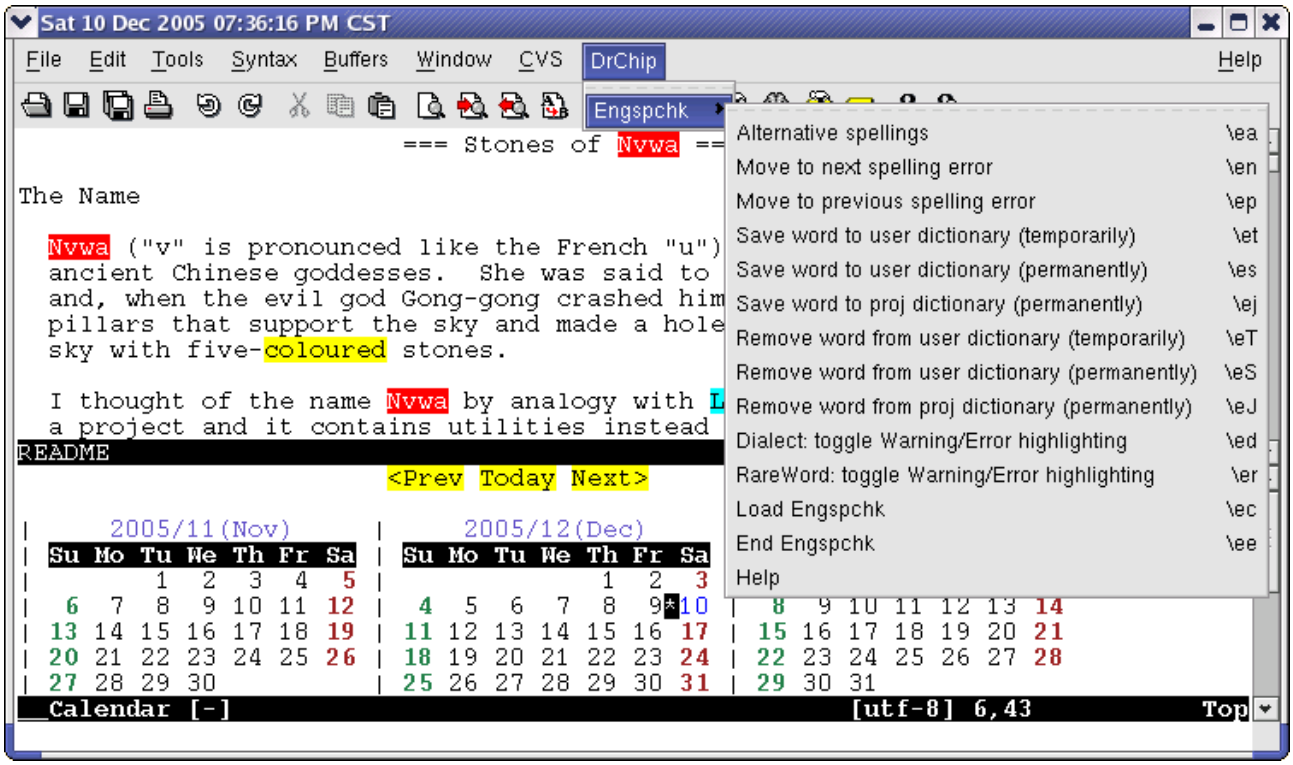

图 6

更多的帮助内容请参考“`:help engspchk`”。

2.4 calendar（日历显示）

作者： Yasuhiro Matsumoto
网站脚本编号： 52
安装说明：

在 Vim 网站上下载文件 calendar.vim，存到 ~/.vim/plugin 目录中。

功能说明：

图 6 的下半部分就是 calendar 的运行示例，是直接在正常模式下键入“`\caL`”显示出来的。光标在该窗口中时，可使用左箭头或上箭头显示前一个月，右箭头或下箭头显示后一个月，“`t`”回到“今天”，“`q`”关闭日历窗口。在有鼠标支持时，也可以使用鼠标在黄色字样的“Prev”、“Today”、“Next”上双击达到类似的效果。

除使用“`\caL`”外，“`\cal`”也可以打开一个日历窗口，但显示位置是在 Vim 的左侧而不是下方。另外，命令“`:Calendar`”效果和“`\cal`”相同，但其后可以跟参数：一个参数的话表示月份，两个参数的话则分别表示年份和月份。“`:CalendarH`”与“`:Calendar`”类似，但日历窗口的打开位置和“`\caL`”（而不是“`\cal`”）一样，是在下方而不是左侧。

2.5 project（文件的“项目”管理）

作者： Aric Blumer
网站脚本编号： 69
安装说明：

1. 在 Vim 网站上下载最新版本（project-1.3.tar.gz）；
2. 使用“`tar xvfz project-1.3.tar.gz -C ~/.vim`”解开；
3. 在 Vim 中运行“`:helptags ~/.vim/doc`”安装文档。

功能说明：

该插件可以把文件组织成一棵树的形式，以便于查找和管理。使用命令“:Project”即可打开一个用户的“项目文件”（~/.vimprojects）。项目文件采用普通的文本文件的形式，非常易于浏览和修改。图 7

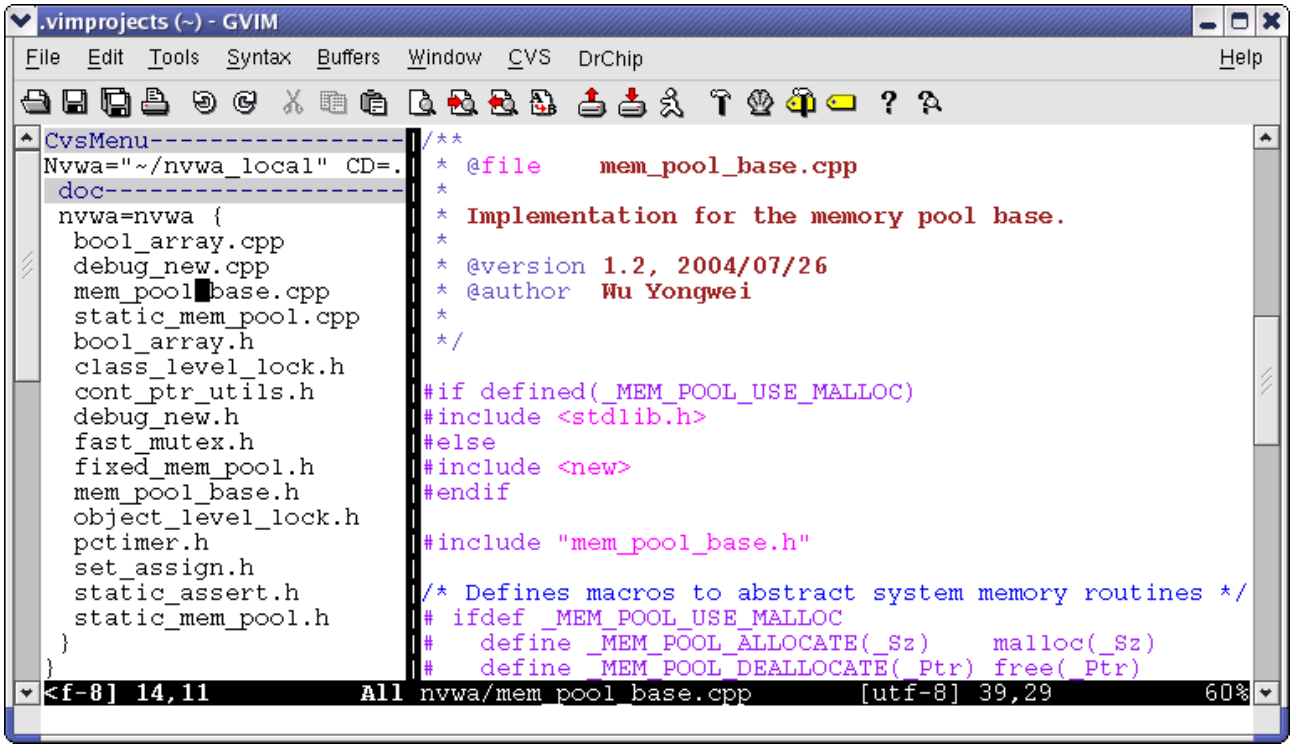

图 7

是一个示例。其中左边窗口部分的就是一棵项目树，完整内容如下：

```
CvsMenu="~/.vim" CD=. {
 plugin/cvsmenu.vim
 doc/cvsmenu.txt
}
Nvwa="~/nvwa_local" CD=. {
 doc=doc {
  ChangeLog
  README
  webupdate.sh
 }
 nvwa=nvwa {
  bool_array.cpp
  debug_new.cpp
  mem_pool_base.cpp
  static_mem_pool.cpp
  bool_array.h
  class_level_lock.h
  cont_ptr_utils.h
  debug_new.h
  fast_mutex.h
  fixed_mem_pool.h
  mem_pool_base.h
  object_level_lock.h
  pctimer.h
  set_assign.h
  static_assert.h
  static_mem_pool.h
 }
}
```

在这个项目文件中存在两个项目：CvsMenu 和 Nvwa。引号中的内容表示路径，“CD=.”则表示打开该项目中的文件时，当前目录会更改到项目所在的目录。项目下面可以再分子项目，如 Nvwa 项目下有 doc 和 nvwa 两个子项目，没有“CD=.”表示打开子项目里的文件时不再更改当前目录。

图中并没有显示出完整的内容，因为其中的内容可以折叠（缺省打开项目文件时是完全折叠起来的，即在上面的例子中，只能看到两个项目的名字“CvsMenu”和“Nvwa”）。在非文件名行上使用鼠标双击或回车键可以进行展开或折叠，在文件名行上使用鼠标双击或回车键则可以打开对应的文件。

更多的帮助内容请参考“:help project”。

2.6 taglist（源代码结构浏览）

作者：Yegappan Lakshmanan
网站脚本编号：273
安装说明：

1. 确保机器上有一个可用的 Exuberant Ctags 的版本（可以执行命令“ctags”）
2. 在 Vim 网站上下载 taglist（假设为 taglist.zip）；
3. 使用“unzip taglist.zip -d ~/.vim”解开；
4. 在 Vim 中运行“:helptags ~/.vim/doc”安装文档。

虽然 taglist 使用 ctags，但并不要求 tags 文件的存在。

功能说明：

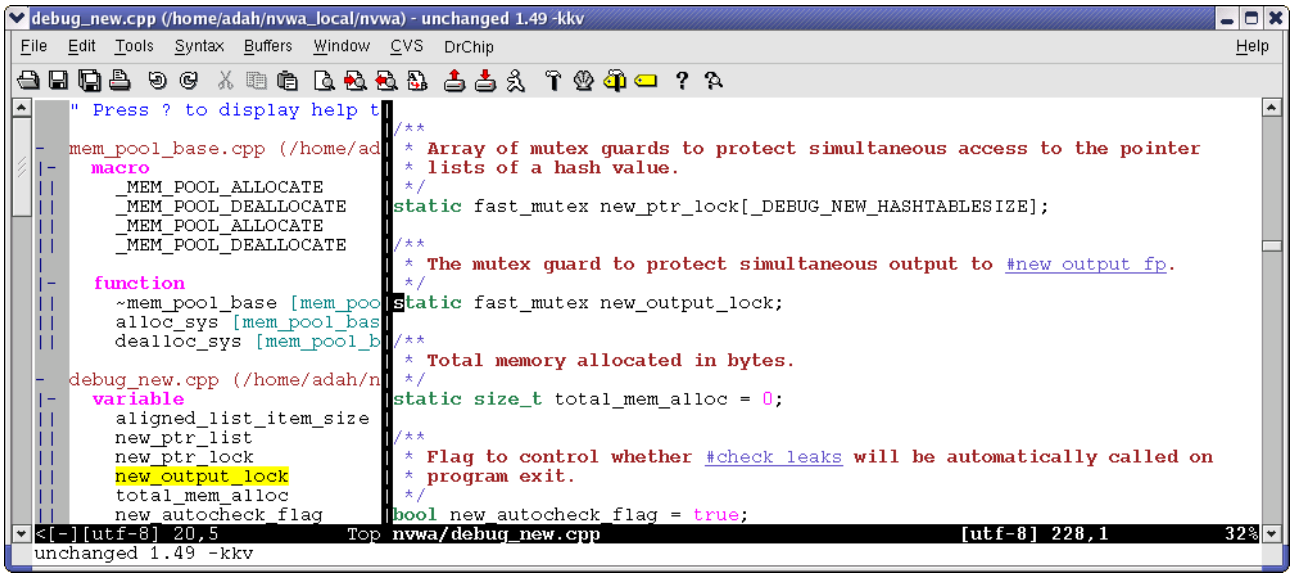

图 8

使用命令“:Tlist”启用/关闭 taglist，效果请先看一下图 8。Taglist 的主要特性有：

- 可以显示宏（macro）、函数（function）、变量（variable）、类型定义（typedef）、类（class）、结构（struct）等多种内容信息；
- 打开多个文件时，可以同时看到多个文件的结构信息；
- 在左边的 taglist 窗口显示的名称上双击鼠标或按回车键，右边主窗口中会跳转到相应的定义位置；
- 右边窗口中光标的位置改变，左边的窗口也会在 4 秒内（缺省值，参见“:help 'updatetime'”）黄色加亮显示相应的名称；
- 在启用/关闭 taglist 时，插件能够自动改变当前 Vim 窗口的大小（不管是文本模式的 Vim 还是图形界面的 Vim），除非使用“:let Tlist_Inc_Winwidth=0”关闭这一功能（当使用这一功能导致兼容性问题时）。

更多的帮助内容请参考“:help taglist”。

2.7 cvsmenu（CVS 集成）

作者：Thorsten Maerz/吴咏炜
网站脚本编号：1245
安装说明：

1. 在 Vim 网站上下载文件 cvsmenu.vim，存到 ~/.vim/plugin 目录中；
2. 启动 gvim，在菜单中选择“CVS—Settings—Install—Install updates”（文本模式的 Vim 可以使用快捷键“,cgii”），从网上（SourceForge）的 CVS 中安装最新版本和帮助文档（此步骤可选）。

功能说明：

在 Vim 中集成 CVS [24] 版本管理功能。该插件最初由 Thorsten Maerz 编写，在 2002 年以来没有再得到维护。我在 2005 年初开始使用这个插件后，非常喜欢它，修正了其中存在的错误，并一直维护该插件。如果大家发现有 bug，报告给我就可以了。

主要功能可在图 9 的菜单中看到：

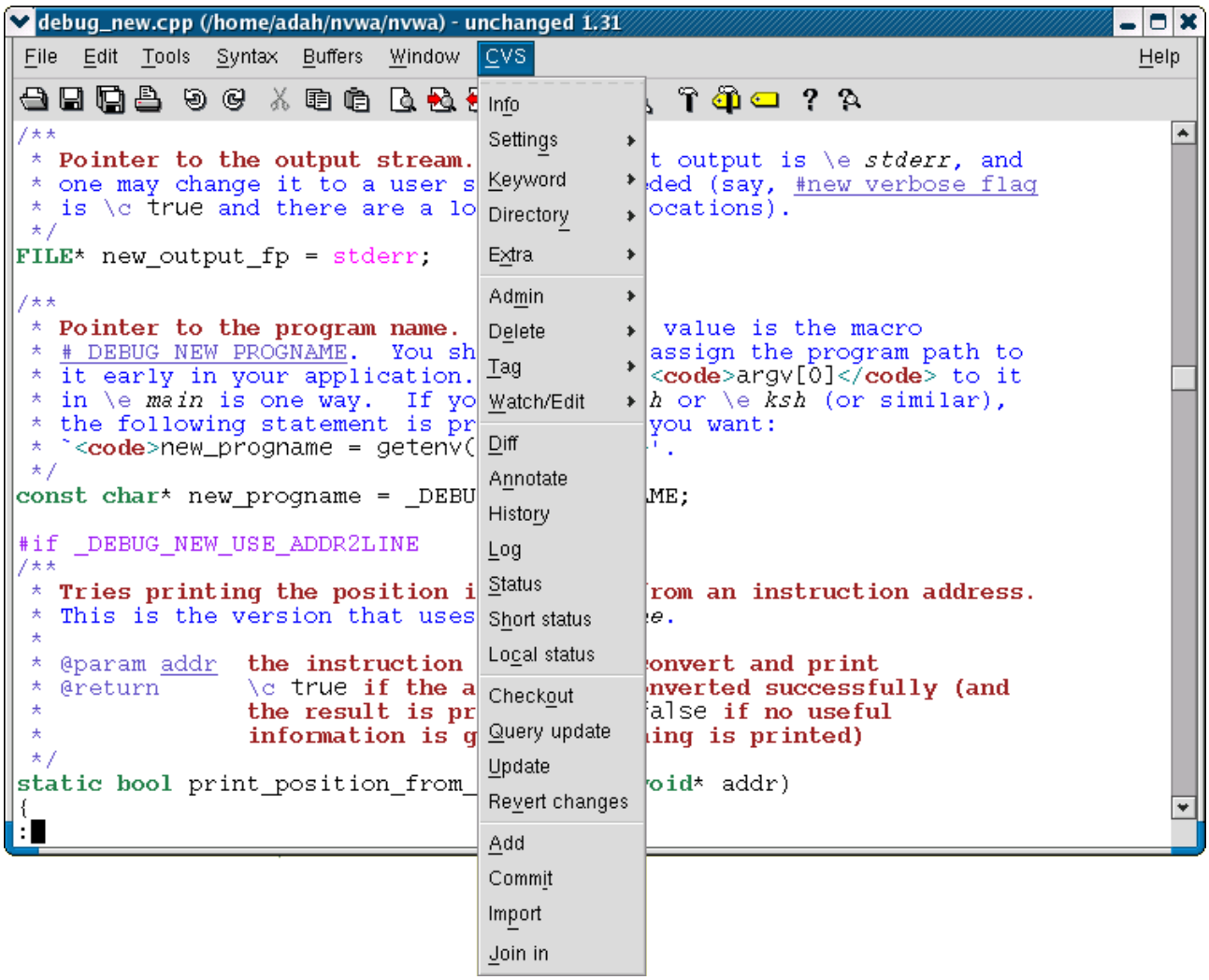

图 9

- 了解 CVS 环境的基本信息和 cvsmenu 中的变量设定（参见图 10）；
- 调整设定，安装更新版本；
- 插入 CVS 的可扩展关键字，如“$Date$”、“Id”；
- 对目录进行基本操作（cvs update 等）；
- 接受附加参数的基本操作（文件比较等）；
- 管理功能，如登录；
- 删除类操作；

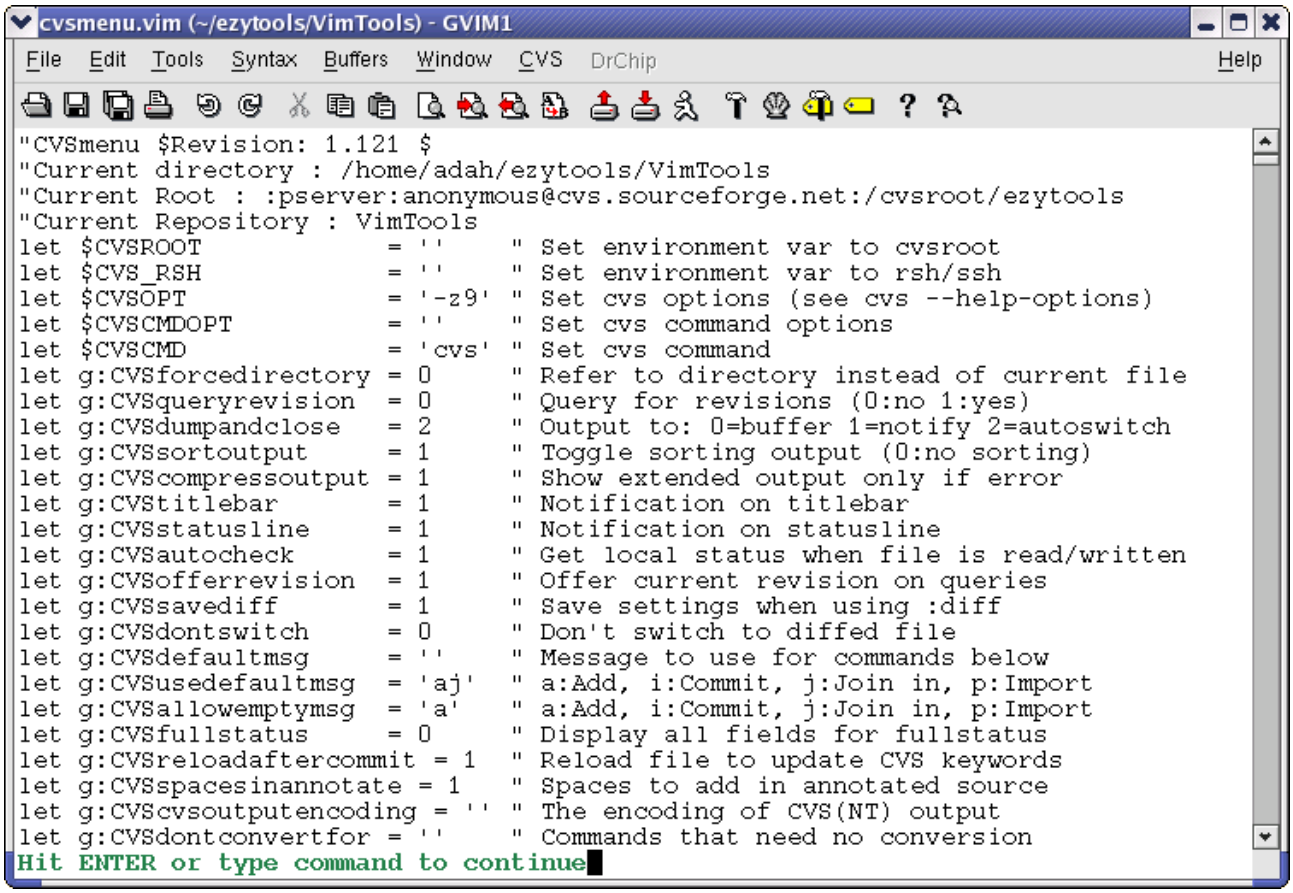

图 10

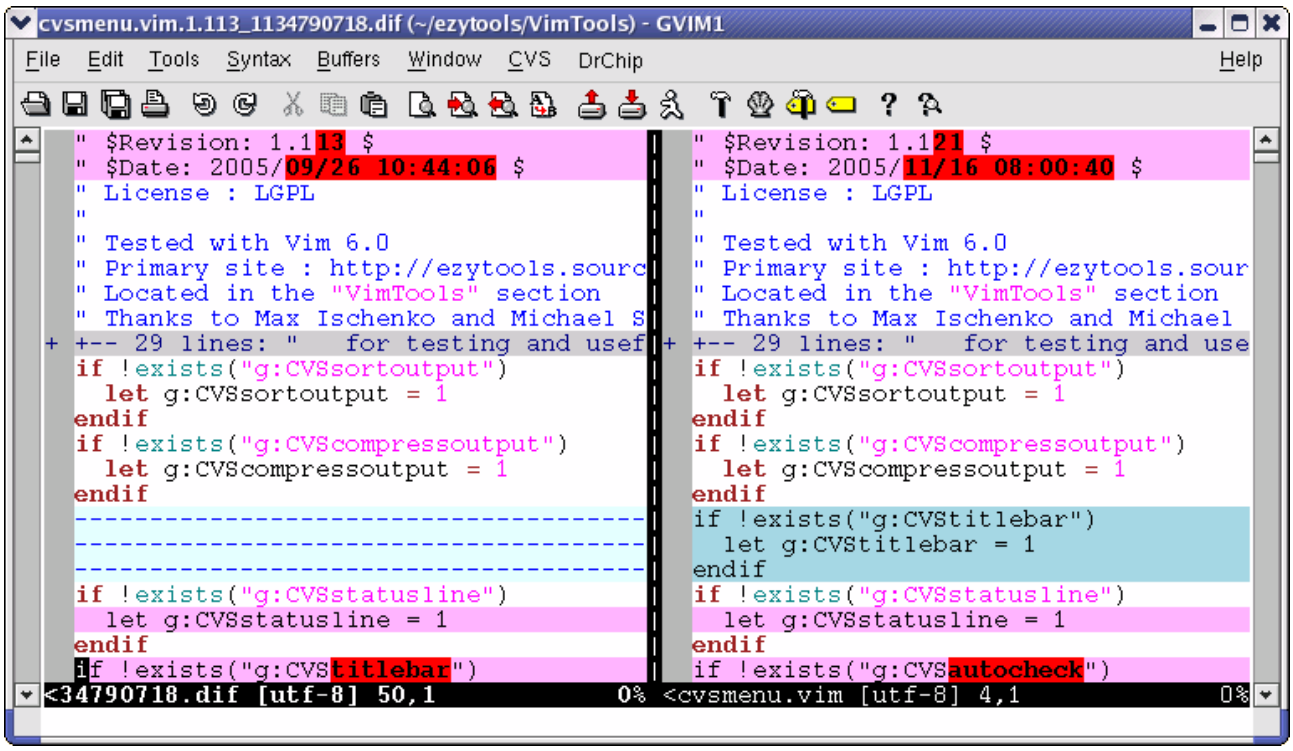

图 11

- 打标签；
- 观察（跟踪）和编辑功能；
- 将本地文件和 CVS 中的文件进行比较（以 Vim 的分左右两列使用颜色加亮显示修改、增加、删除部分的比较方式；效果参见图 11）；

- 显示文件每一行的更改人和更改时间（cvs annotate，参见图 12）；

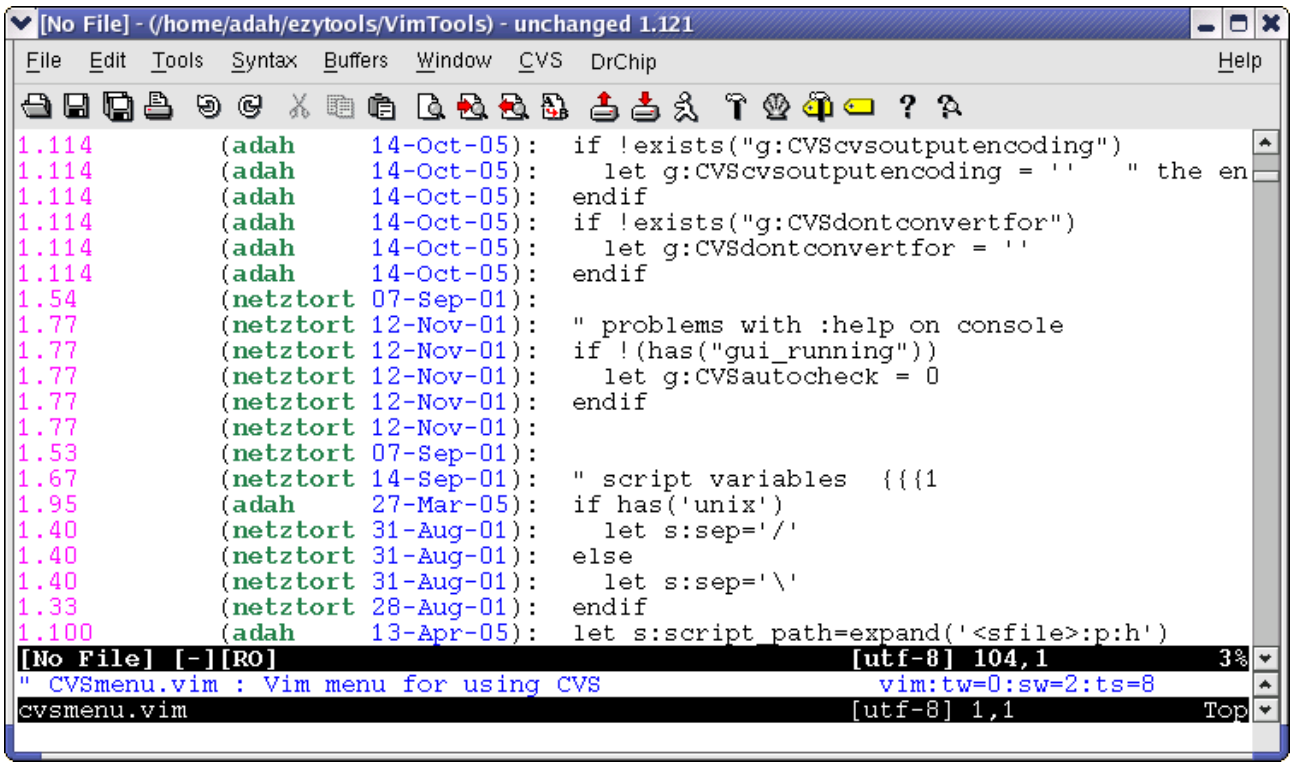

图 12

- 显示库的访问历史（cvs history）；
- 显示文件修订日志（cvs log；参见图 13）；

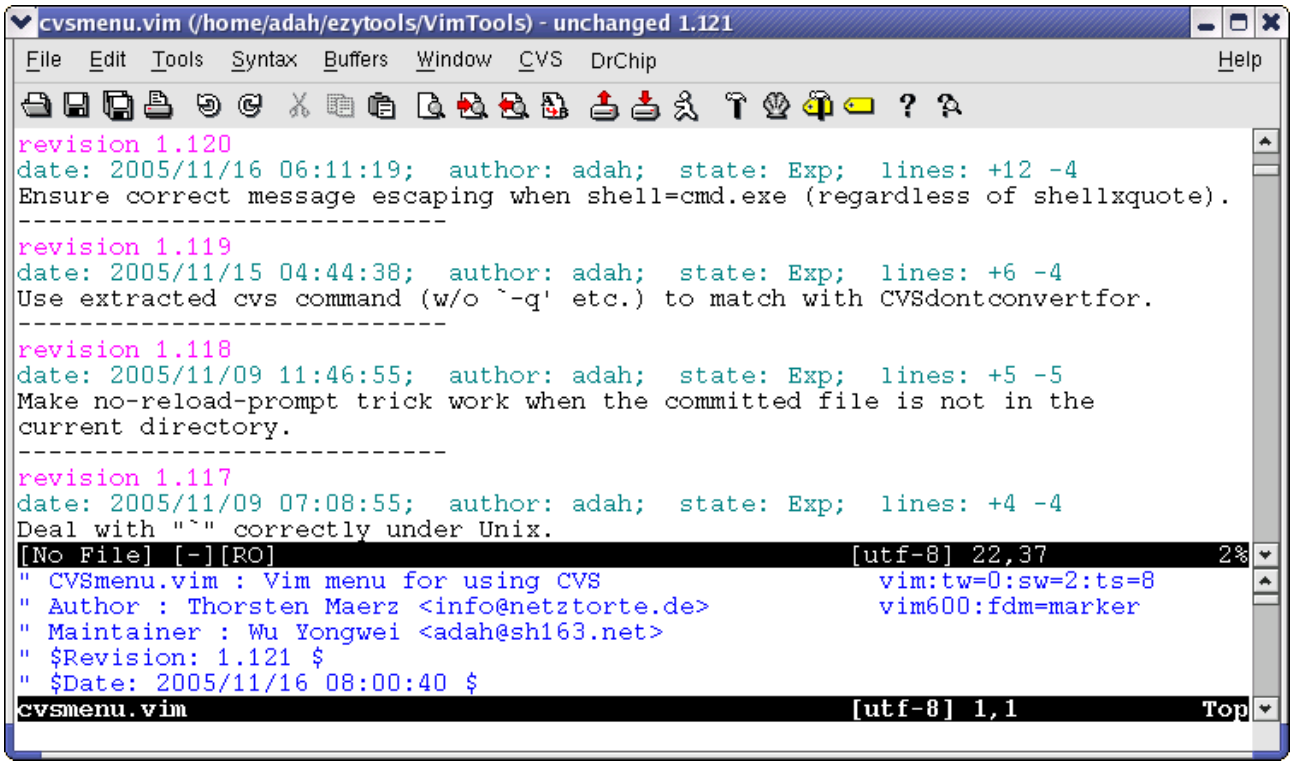

图 13

- 显示文件状态；
- 显示较短的文件状态；
- 根据本地信息显示文件状态（不访问库）；
- 签出模块（cvs checkout）；

- 查询更新；
- 更新，如有冲突会高亮显示（cvs update）；
- 取消修改；
- 添加当前文件到 CVS 中（cvs add）；
- 签入当前文件（cvs commit）；
- 导入文件（cvs import）；
- 添加并签入当前文件。

根据实际使用的需要，除了修订错误之外，我加入了一些编码相关的支持。拿一个最实际的情况，如果使用“`set encoding=utf-8`”，但源代码中仍使用了 GBK 编码的中文字符，那么，必须在 .vimrc 中加入一行“`let g:CVScvsoutputencoding='gbk'`”才能保证“cvs annotate”操作的结果是正确的。

Thorsten 把该插件的易用性设计得相当好。举例来说，在作文件比较时，按 *Tab* 可跳转到下一个不同处，左侧的比较窗口中按一下“`q`”即可关闭该窗口。更多的帮助内容请参考“`:help cvsmenu`”。

2.8 doxygen（文档注释语法加亮）

作者：Michael Geddes
网站脚本编号：5

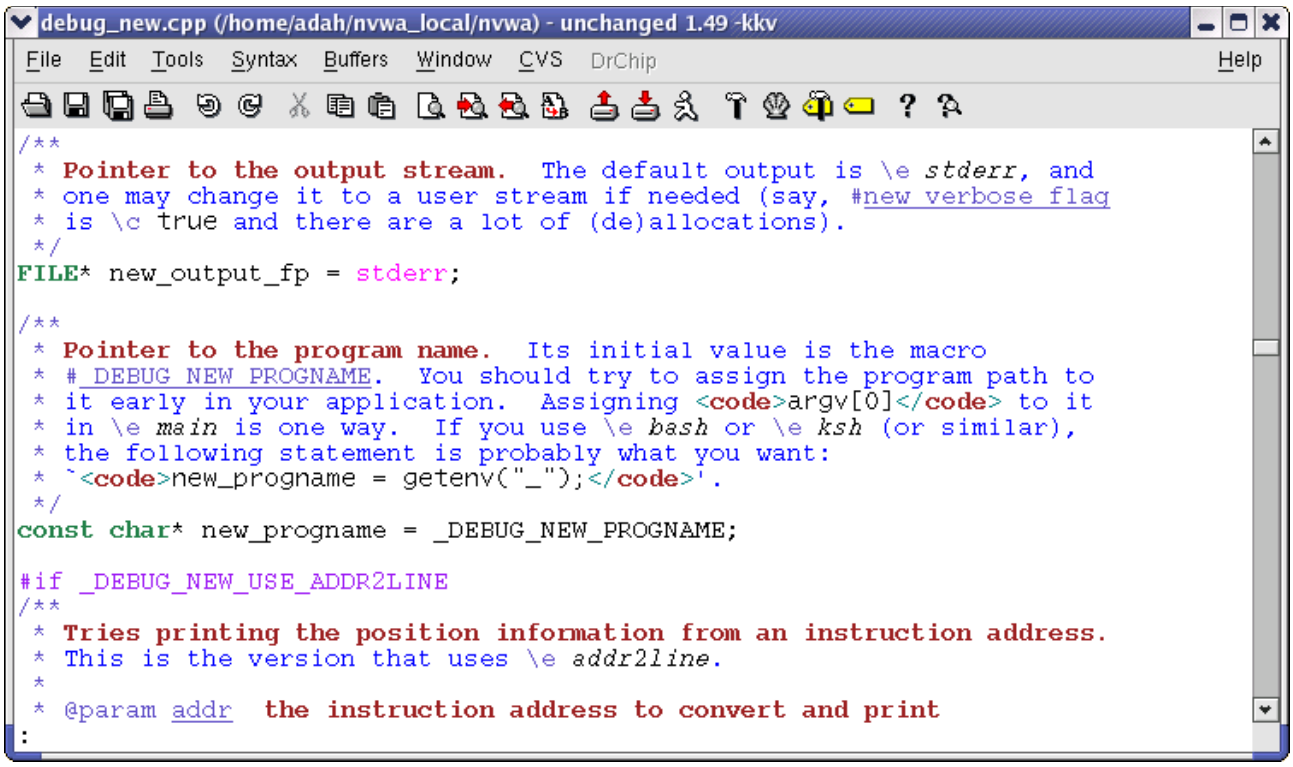

图 14

安装说明：

1. 在 Vim 网站上下载最新版本（doxygen.zip）；
2. 使用“`unzip doxygen.zip`”解开；
3. 执行“`cp -p doxygen.vim ~/.vim/syntax`”和“`cp -p doxygen.txt ~/.vim/doc`”复制文件到 Vim 目录下；
5. 在 Vim 中运行“`:helptags ~/.vim/doc`”安装文档。
4. 如果 ~/.vim/after/syntax 目录不存在，使用“`mkdir -p ~/.vim/after/syntax`”创建该目录；
5. 进入 ~/.vim/after/syntax 目录，使用下面三行创建语法文件之间的关联，使得在 C、C++ 和 Java 文件中可以识别文档注释：

```
ln -s ../../syntax/doxygen.vim c.vim
```

```
    ln -s ../../syntax/doxygen.vim cpp.vim
    ln -s ../../syntax/doxygen.vim java.vim"
```

6. 可选地，看一下我的个人编程网页 [25] 下有没有对 doxygen.vim 的更新（里面包含有作者尚未并入到标准发布中去的我的更改——好奇的话，你可以在 doxygen.vim 文件中数一下，看一看 Wu Yongwei 的名字出现了几次☺）。

功能说明：

如果你用过文档注释，相信我不需要多说，看一下图 14 你就知道这个脚本的功能了。如果你没有用过文档注释，强烈建议你到 Doxygen [26] 的网站上看一下，了解一下这个非常有用的工具。作为示例，可在 http://nvwa.sourceforge.net/doc/debug__new_8cpp.html#a17 看一下图中的代码使用 Doxygen 自动生成的 HTML 网页。

2.9 matrix（！）

作者： Don Yang
网站脚本编号： 1189
安装说明：

在 Vim 网站上下载文件 matrix.vim，存到 ~/.vim/plugin 目录中即可。

功能说明：

是的，说的就是电影 *Matrix*！Vim 的脚本并不是都干“正经事”的。想看一下电影里那些酷酷的字符在 Vim 的窗口里翻滚吗？发一个命令“`:Matrix`”即可（按任意键退出）。效果见图 15。

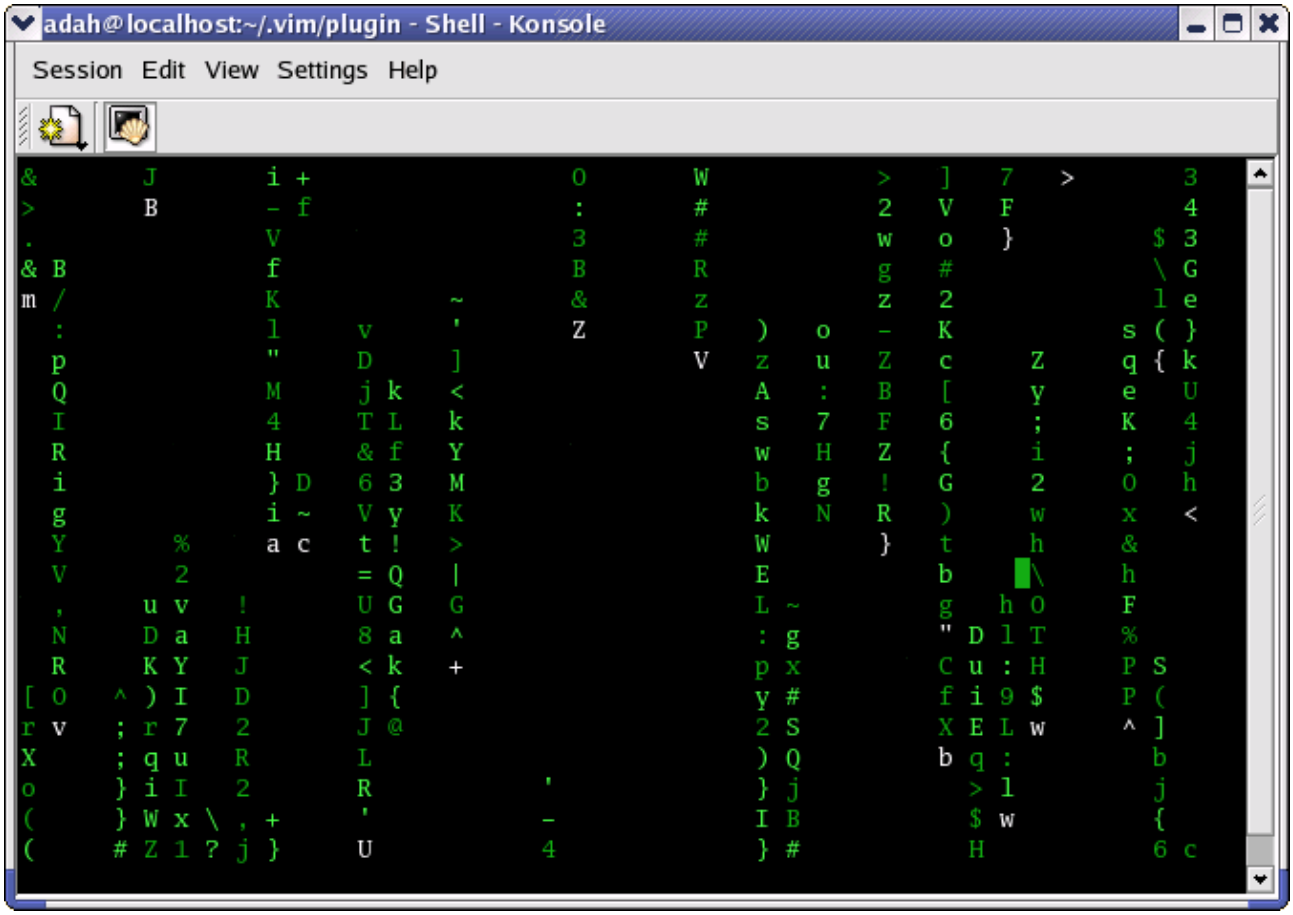

图 15

3. 定制 Vim

前面两章讲的都是如何使用现有的 Vim 系统，本章则会通过实例来讲如何**定制** Vim 的行为。良好的定制可以让使用 Vim 变得更为得心应手；同时，在掌握了基本的定制之后，也许你就会想进一步写一些自己的 Vim 脚本，从而真正地成为一个 Vim 专家……

Happy Vimming!

3.1 Vim 脚本基础

在 .vimrc 文件中，和在第二章提到的插件和语法文件中，使用的语言就是 Vim 脚本语言。这种脚本语言语法有点像 BASIC，表达式有点像 C，还是比较容易理解的。本章中并不打算对其作很系统的介绍（要完整了解的话，请参见“`:help usr_41.txt`”），而只是介绍一些基本知识，特别是，了解定制 .vimrc 所需要的基本知识。

Vim 脚本相当于可直接在命令模式下执行的命令，只是不需要输入前面的冒号（如果用了冒号也不会出错）。因此，像设置选项、创建键盘映射这样的命令是直接可用的。当然，作为一种脚本语言，除了普通键盘上会输入的命令外，我们还需要一些更复杂的功能，特别是：变量，表达式，条件和循环语句，函数。

3.1.1 变量

Vim 中使用如下的语法对变量进行赋值（创建变量）：

```
let 变量名 = 数值
```

变量类型有两种，整数和字符串，在第一次赋值之前都不能使用。变量名除了可使用常规的字母、下划线和数字外，还可以使用几种特殊的前缀：

- “`b:`”——只对当前缓冲区（buffer）有效的变量；
- “`w:`”——只对当前编辑窗口（window）有效的变量。
- “`g:`”——全局变量（在函数中访问全局变量必须使用该前缀，不加前缀的话则认为是函数内的局部变量）；
- “`s:`”——变量名只在当前脚本中有效；
- “`a:`”——函数的参数；
- “`v:`”——Vim 内部预定义的特殊变量（参见“`:help vim-variable`”）。

下面三个前缀用来访问特殊的数值，由于行为和变量较为相似（可以读取和修改），也放在这儿一起讲：

- “`$`”——访问环境变量；
- “`&`”——访问 Vim 选项；
- “`@`”——访问寄存器。

当变量不再使用时，可以使用“`unlet` *变量名*”删除变量。

3.1.2 表达式

和 C 非常类似，可以使用变量和常量，可以使用括号，可以调用函数（“*函数名*`(...)`”），支持加法（“`+`”）、减法（“`-`”）、乘法（“`*`”）、除法（“`/`”）和取模（“`%`”），支持逻辑操作（“`&&`”、“`||`”和“`!`”），支持三元条件表达式（“`a ? b : c`”）。字符串操作方面当然比 C 要强，可以使用“`.`”进行字符串拼接；可使用“`==`”、“`<=`”等进行字符串大小比较，可使用“`=~`”和“`!~`”进行正则表达式匹配，而且可以在比较操作符后面添加“`#`”或“`?`”来强制进行大小写敏感或不敏感的比较（缺省受 Vim 选项 `ignorecase` 影响）。显示一个表达式的结果，可以使用“`:echo` *表达式*”显示到状态栏上，或者在插入模式下使用“*Ctrl*-`R`=*表达式*”插入到缓冲区的文本中。

和其它很多在 Unix 下成长起来的语言一样，Vim 的字符串常量有双引号和单引号两种方式。使用单引

号的话，单引号间的任何字符都是字符串的一部分，其中不能再包含单引号。使用双引号的话，则可以使用“\”产生换码序列（具体可参考“:help expr-quote”），如“\n”代表换行符，“\"”代表双引号，“\\”代表反斜杠本身，等等。

需要注意的话，双引号除了可以表示字符串常量外，还可以表示注释。行首的“"”，以及表达式中出现的成单的“"”，都表示“"”后面的部分全部是注释。

3.1.3 条件和循环语句

条件语句形式如下：

```
if 表达式
  语句
endif
```

或

```
if 表达式
  语句
else
  语句
endif
```

或

```
if 表达式
  语句
elseif 表达式
  语句
endif
```

循环语句形式如下：

```
while 表达式
  语句
endwhile
```

条件和循环语句都可以嵌套。这些比较简单，就不多加说明了。

3.1.4 函数

在表达式中使用函数时，就跟 C 里面的方式类似，直接使用函数名加括号，括号里写上参数（可选）。在不需要返回值的情况下调用函数时，稍稍有些不同，要使用“call”命令，后面跟函数名和括号（括号里面写上可能有的参数）。

定义函数使用下面的语法：

```
function 函数名称(参数列表)
  语句
endfunction
```

如果已有同名函数存在，Vim 会报错，除非在“function”后面加上一个“!”。

如果参数中不包含“...”，那么参数的数量是固定的，函数的调用者必须提供跟定义同样多的参数（在函数定义中使用参数名之前加上“a:”进行访问）。如果参数中包含“...”，那么参数的数量不固定，除了可以使用参数名称访问传递过来的参数外，还可以使用“a:0”知道额外传递的参数数量，使用“a:1”、“a:2”等访问这些额外传递的参数。

要在函数的中间返回，或者要返回数值的话，可以使用“return”语句。

Vim 内部定义了一百多个函数，详细列表请参见“:help function-list”。

3.2 我的 .vimrc

作为一个 Vim 脚本的一个具体示例，我将讲解一下最实用的情况，我的 .vimrc 文件。文件 .vimrc.html 是我的 .vimrc 文件通过以下步骤生成的 HTML 文件：

1. 在 Vim 中打开 .vimrc 文件；
2. 执行命令“`:colorscheme koehler`”（缺省配色可能在浏览器中效果不佳）；
3. 执行命令“`:%!nl -w4 -s' '`”（1.11 节）；
4. 执行命令“`:TOhtml`”（1.13 节）；
5. 执行命令“`:w`”。

可以把浏览器中的文本内容粘贴到 Vim 中，然后使用下面这个替换命令“`:%s/^ \+[0-9]\+ //`”删除前面的行号，来恢复出最初的 .vimrc 文件。

下面逐行进行讲解，并包含理解其内容所需的资料的链接。建议大家直接阅读 .vimrc 文件的内容，并在有疑问时查阅下面的解释。

第 1 行：注释（3.1.2 节末段），其中包含一个模式行（1.4 节和 1.5 节）。

第 2 行：首先判断系统是否具有“自动命令”（*autocmd*）的支持，有的话才执行第 3 到第六行的内容（1.1 节、“`:help has`”和“`:help feature-list`”）。

第 3 行：纯注释（后面我将跳过注释行不再说明）。

第 4 行：清除所有的自动命令（“`:help autocmd-remove`”），以方便调试，可以使用“`source ~/.vimrc`”查看一些修改后的效果（“`:help source`”）。

第 6 行：对于后缀为“.asm”的文件，认为其是微软的 Macro Assembler 格式（“`:help masm-syntax`”）。

第 7 行：与第 2 行的 `if` 语句配对。

第 8–10 行：当使用了图形界面时（“`:help feature-list`”），确保所有的文件类型会在菜单“语法”（“Syntax”）下出现，而不是出现一个菜单项“Show filetypes in menu”。缺省行为可以让 Vim 启动得更快一点点。

第 11–13 行：当使用了图形界面，并且环境变量 `LANG` 中不含“.”（即没有规定编码）时，把 Vim 的内部编码设为 UTF-8。

第 14 行：不需要保持和 vi 非常兼容（“`:help 'compatible'`”）。

第 15 行：执行 Vim 缺省提供的 .vimrc 文件的示例，包含了打开语法加亮显示等最常用的功能。

第 16 行：打开自动缩进（1.4 节）。

第 17 行：缺省不产生备份文件（“`:help 'backup'`”）。

第 18 行：在输入括号时光标会短暂地跳到与之相匹配的括号处，不影响输入（“`:help 'showmatch'`”）。

第 19 行：正确地处理中文字符的折行和拼接（1.12 节）。

第 20 行：可自动识别的文件类型为带 BOM 字符的 Unicode 文件、UTF-8 编码的文件和 GBK 编码的文件。

第 21 行：设置状态行，使其能额外显示文件的编码信息，如图 2 中的“gbk”和“big5”（“`:help 'statusline'`”）。

第 22–24 行：如果该 Vim 支持鼠标，则启用鼠标支持（1.3 节）。

第 25–29 行：判断 Vim 是否包含多字节语言支持（*multi_byte* 特性），并且版本号（“`:help v:version`”）大于 6.1（包含 `ambiwidth` 选项）。

第 26–28 行：如果 Vim 的语言（“`:help v:lang`”；受环境变量 `LANG` 影响）是中文（`zh`）、日文（`ja`）或韩文（`ko`）的话，将模糊宽度的 Unicode 字符的宽度（`ambiwidth` 选项，1.2 节）设为双宽度（`double`）。

第 31–36 行：改变上、下方向键行为方式：通常情况下这些键的作用范围是逻辑行，所以如果行很长的话光标的移动可能会不太方便；这些键盘映射把这些键的作用范围改成屏幕行（“`help gk`”），还为习惯使用“`j`”、“`k`”的人增加了映射“*Ctrl*-`j`”和“*Ctrl*-`k`”作用于屏幕行。前面四个映射使用的命令是“`noremap`”，作用于正常模式、可视模式和命令执行时；后面两个映射使用的命令是“`inoremap`”，仅作用于插入模式，其中使用“*Ctrl*-`o`”临时执行一个普通模式的命令（“`:help`

i_CTRL-O”）。

第 38–41 行：在 Vim 中的插入模式中可以使用“*Ctrl*-R=”计算整数表达式的数值，但 Vim 本身没有计算浮点表达式的能力。这四个映射提供了浮点表达式的计算能力：使用“\ma”（假设 Leader 字符为缺省的“\”，参见“:help <Leader>”）可将计算的结果放到下一行上（待计算的表达式为当前行或在可视模式选中的内容），使用“\mr”则用计算的结果替换待计算的表达式（同样为当前行或在可视模式选中的内容）。这些映射假设有一个命令“calcu”可用来计算一个表达式的内容。该命令可用下面的 shell 脚本简单实现：

```
#! /bin/sh
echo "$*" | \
sed -e $'s/\r$//' -e 's/sin *(/s(/g' -e 's/cos *(/c(/g' \
    -e 's/atan *(/a(/g' -e 's/log *(/l(/g' -e 's/exp *(/e(/g' | \
bc -l
```

该脚本把表达式转换成 bc [27] 能接受的形式（把“sin(x)”转换成“s(x)”，等等），并通过标准输出送到 bc 的标准输入。

该映射较为复杂，此处不详加解释了——其中心思想都是选取待计算的表达式，放到无名寄存器中，然后使用“*Ctrl*-R"”粘贴到命令行上，使用 calcu 进行计算，再把结果粘贴回正在编辑的缓冲区中；最后一个最复杂，因为为了替换原先的表达式，还需要记住原先被选中的内容的起始和结束位置，你可能希望看一下“:help gv”、“:help v_o”、“:help m”、“:help `”，并复习节 1.11。可以注意一下，在映射中使用了“<silent>”（“:help map-<silent>”），这会防止命令行上回显执行的内容。

第 43–44 行：允许用户使用 *F2* 来取消搜索/替换的加亮显示。此处一个映射用于正常模式（nmap），一个用于插入模式（imap）。上面已经提过一次，“*Ctrl*-O”可以在插入模式中执行一个正常模式的命令。

第 46–47 行：这两个映射用于 taglist 插件（2.6 节），使用 *F9* 直接打开（或关闭）taglist 的窗口。

第 49–50 行：方便快速修订窗口（1.10 节）的使用，可使用 *F11*（和 *F12*）查看下一个（上一个）错误（或 grep 项等）。

第 52–65 行：一些适用于文本模式运行的 Vim 的设定；详见下面的具体说明。

第 54–56 行：将变量 Tlist_Inc_Winwidth 的值设为 0，防止 taglist 插件改变终端窗口的大小（有些情况下会引起系统不稳定）。使用“has('eval')”是让该语句仅在功能较为完整、至少支持表达式的 Vim 版本中运行。

第 58–64 行：在系统支持 *wildmenu* 特性（“:help 'wildmenu'”）启用文本模式的菜单。

第 59 行：打开 wildmenu 选项，启动具有菜单项提示的命令行自动完成。

第 60 行：确保字符序列“<C-Z>”被理解为 *Ctrl*-Z 而不是分开的五个字符（“:help 'cpoptions'”）。

第 61 行：设置使用 *Ctrl*-Z 激活自动完成提示。

第 62–63 行：把正常模式和插入模式下的 *F10* 映射成执行菜单项，并自动提示菜单内容。注意缺省菜单仍不会自动载入，我使用该特性的主要目的是在文本模式的 Vim 中使用 CVS 菜单。图 16 是按 *F10* 键后再按 *Tab* 键的结果。

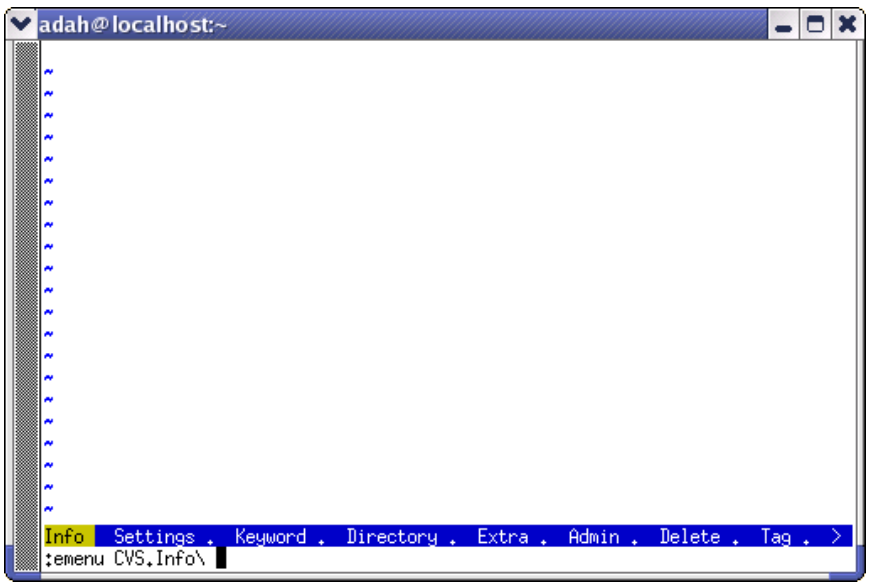

图 16

第 66–161 行：使用自动命令（*autocmd*）特性的设置。使用“has”来防止该部分内容在不支持自动命令的 Vim 版本中运行。

第 67–129 行：定义了若干个下面的自动命令会用到的函数，具体在下面的自动命令中讲。请注意在每个“function”之后都用

了一个“!”（“:help E122”）：这也是为了方便调试，让“source ~/.vimrc”能正确运行而不会报告函数已定义的错误。

第 131–133 行：只要没有将环境变量 VIM_HATE_SPACE_ERRORS 的值设为零，则把变量 c_space_errors 的值设为 1——效果是在 C/C++ 代码中“不正确”的空白字符（行尾的空白字符和紧接在制表符之前的空格字符）将会被高亮显示。图 17 所示的代码中，第 3 行的行尾多了两个空格，第 5 行

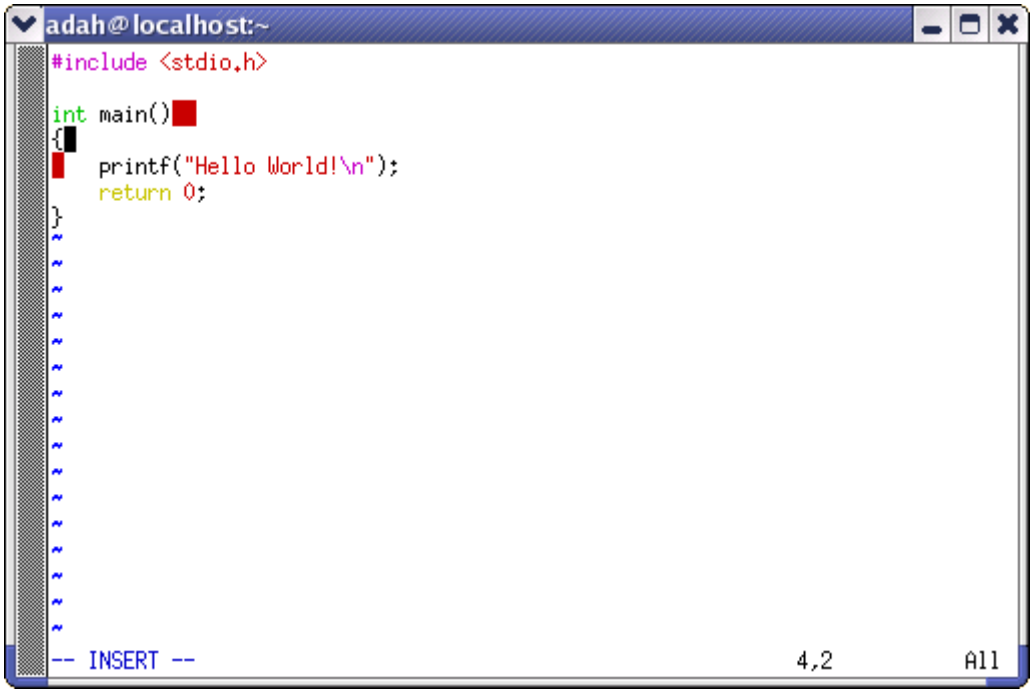

图 17

的第一个制表符之前多了个空格。Vim 提示#935 里有一些额外的说明。同时请参看对第 160 行的说明。

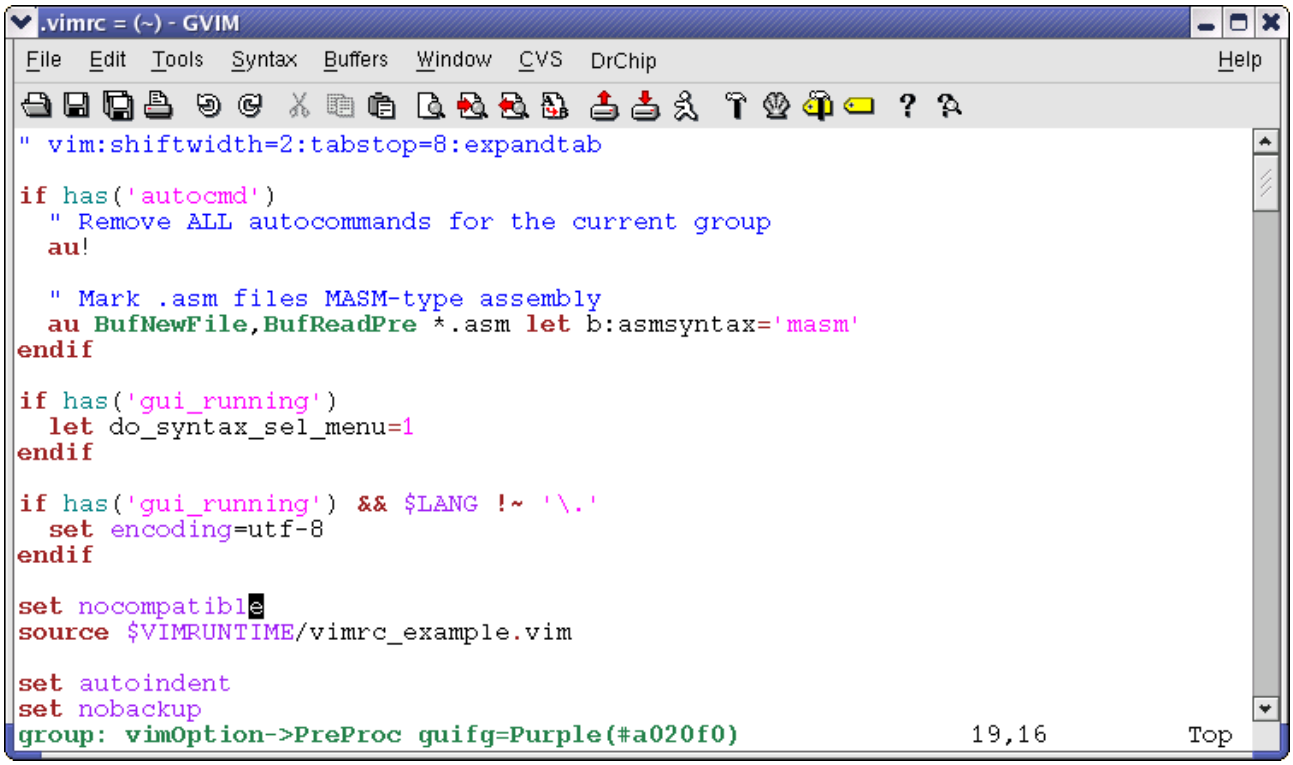

图 18

第 135 行：使用的英文拼写变体为加拿大风格，即：使用拼写“abridgement”（而不是“abridgment”）、“colour”（而不是“color”）、“realize”（而不是“realise”）、“theatre”（而不是“theater”）等，比较符合中国人一般的英语教科书中的拼写方式，也比较适合于写“国际”英语。

第 138 行：使用键盘映射“\a”来查看光标下字符的属性，主要用于调试 Vim 的语法文件。图 18 显示了光标下的字符所属的语法“组”为 vimOption，使用配色方案中的 PreProc（预处理符号）项，前景色为紫色（RGB：#a020f0）。有兴趣可查看 Vim 脚本#383 的具体内容。

第 140 行：在函数找不到时（“:help FuncUndefined”），自动在运行环境（Linux 下一般为 ~/.vim）的 autoload 目录下读入与函数名同名的 .vim 文件。这是脚本#383 的建议安装方式（SyntaxAttr.vim 文件放在 autoload 目录下，仅在执行时载入）。

第 142 行：设置适用于 C/C++ 文件的选项（1.4 节）。

第 143 行：把补丁文件的缩进和制表符宽度设定设成和 C/C++ 文件相同（1.4 节）。

第 144 行：取消 Vim 对 HTML 标记自动产生的缩进，但打开自动缩进选项（1.4 节）。

第 145 行：对于变更日志类型的文件，设置行宽为 76 个字符（1.12 节）。

第 147 行：当文件后缀为“.gb”时，认为这是一个 GBK 编码的文件，在读入文件之前（“:help BufReadPre”）调用函数 SetFileEncodings 把原先的 fileencodings 选项的内容保存在本缓冲区的一个变量中（3.1.1 节），然后把 fileencodings 设成 gbk，即只尝试对文件内容作为 GBK 字符序列来解释。

第 148 行：类似于上面把“.big5”后缀的文件当作 Big5 编码的文件，在读入文件之前把 fileencodings 设成 big5，只尝试对文件内容作为 Big5 字符序列来解释。

第 149 行：类似于上面把“.nfo”后缀的文件当作 CP437 编码（即英文 DOS 的 OEM 字符集编码）的文件。效果可参看图 19。

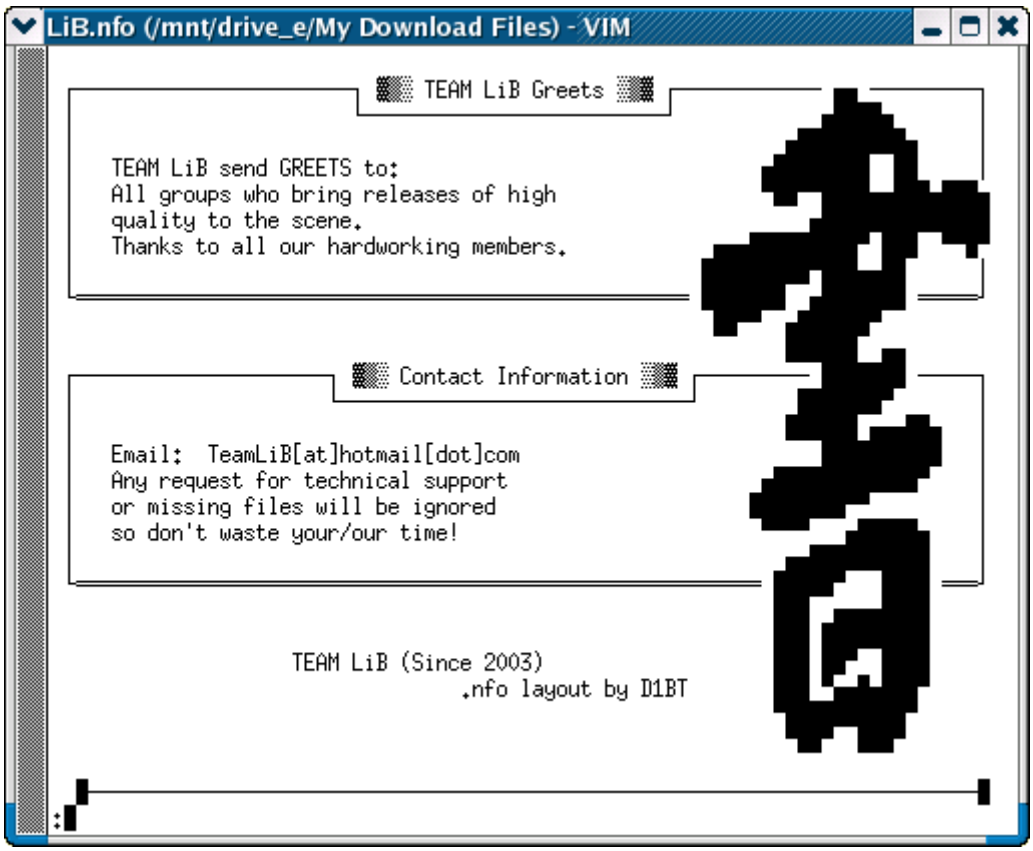

图 19

第 150 行：在读入 .gb、.big5 或 .nfo 文件之后（“:help BufReadPost”），调用函数 RestoreFileEncodings 恢复保存起来的 fileencodings 原数值。

第 151 行：对于 .txt 后缀的文件，在显示文件时（“:help BufWinEnter”，确保在模式行被执行之后）调用函数 CheckFileEncoding 检查文件是否已修改并且 fileencoding 设有数值。条件满足的话说明该文件在模式行中修改了 fileencoding，因而使用该编码（“:help ++enc”）重新强制（“!”）读入该文件以保证文件被正确解码。Vim 提示#911 里有一些额外的说明。

第 153 行：在遇到 HTML 文件时，如果 Vim 判断出的编码类型和 HTML 代码中使用“<meta http-equiv="Content-Type" content="text/html; charset=*编码*">”规定的编码不一致，将使用网页

中规定的编码重新读入该文件。函数 `ConvertHtmlEncoding` 会把一些网页中使用的编码名称转换成 Vim 能够正确处理的编码名称；函数 `DetectHtmlEncoding` 在判断文件类型确实是 HTML 之后，会记下当前的光标位置，并搜索上面这样的 HTML 代码行，找出字符集编码后，在编码不等于当前文件编码（`fileencoding`）时且当前文件编码为空或等于系统判断出的文件编码时，使用该编码强制重新读入文件，忽略任何错误（“`silent!`”）。该自动命令写成是可嵌套执行的（“`:help autocmd-nested`”），目的是保证语法高亮显示有效，且上次打开文件的光标位置能够正确保持。Vim 提示#1074 里有一些额外的说明。

第 155–156 行：确保把 /usr/include/c++ 和 /usr/include/g++-3 目录下的所有文件都当成 C++ 类型的文件，不管 Vim 原先认定这些文件类型是什么（“`:help BufEnter`”）。C++ 的很多标准头文件（如“algorithm”）没有文件后缀，缺省情况下不会被 Vim 当作 C++ 文件。

第 158 行：第 142 行把 C/C++ 文件的制表符宽度设成了 4（个人设置），但系统的源代码一般使用 GNU 编码规范，制表符宽度为 8。该行设置所有 /usr 目录下的文件都使用 GNU 编码规范（1.4 节）。

第 160 行：在写文件之前（“`:help BufWritePre`”），调用函数 `RemoveTrailingSpace`：只要没有将环境变量 `VIM_HATE_SPACE_ERRORS` 的值设为零，则对于文件类型为 C、C++、Vim 脚本类型的文件，自动悄悄清除所有的行尾空白字符；“`normal m`”记忆当前的光标位置，“`normal ```”恢复记忆下来的光标位置。

至此为此，我已经介绍了 Vim 的基本知识、很多实用技巧和一些最常用的 Vim 插件，并通过定制 .vimrc 文件介绍了脚本的基本知识。如果有需要进一步深入学习 Vim 或是想提什么关于 Vim 的特定问题的话，不妨从 Vim 的网站上参加 Vim 的邮件讨论列表，应该会获益良多。而作者也希望本文至此也已经完成了引导读者学习、了解 Vim 的高级特性的任务。

参考资料

[1] Vim Online: http://www.vim.org/

[2] 中国国家标准 GB2312-1980；参考网页: http://www.answers.com/GB2312

[3] 中国国家标准 GB18030-2000；参考网页: http://www.answers.com/GB18030

[4] 国标编码扩展；参考网页: http://gollum.easycp.de/gollum/gollum.php?wl=zh&q=gbk

[5] ISO/IEC 8859-1: http://www.answers.com/ISO_8859-1

[6] UTF-8: http://www.answers.com/UTF-8#Wikipedia

[7] Unicode Home Page: http://www.unicode.org/

[8] FAQ—UTF & BOM: http://www.unicode.org/faq/utf_bom.html#22

[9] East Asian Width: http://www.unicode.org/reports/tr11/

[10] CXTERM’s Unofficial Homepage: http://cxterm.sourceforge.net/

[11] PuTTY: A Free Telnet/SSH Client: http://www.chiark.greenend.org.uk/~sgtatham/putty/

[12] XTERM—Terminal Emulator for the X Window System: http://dickey.his.com/xterm/

[13] GNOME Terminal: http://www.gnomefiles.org/app.php?soft_id=113

[14] Konsole Homepage: http://konsole.kde.org/

[15] Exuberant Ctags: http://ctags.sourceforge.net/

[16] Languages Supported by Exuberant Ctags: http://ctags.sourceforge.net/languages.html

[17] make: http://www.answers.com/make#Wikipedia

[18] grep: http://www.answers.com/grep

[19] Text File: IEEE Std 1003.1, 2004 Edition, Section 3.392, http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap03.html

[20] The gzip home page: http://www.gzip.org/

[21] bzip2: http://www.bzip.org/

[22] Compress for UNIX: http://www.answers.com/compress#Wikipedia

[23] agrep: http://www.answers.com/agrep

[24] CVS—Concurrent Versions System: http://www.nongnu.org/cvs/

[25] Wu Yongwei's Programming Page: http://wyw.dcweb.cn/

[26] Doxygen: http://www.stack.nl/~dimitri/doxygen/

[27] bc: http://www.answers.com/bc%20(Unix)