

vim 终极实例

VIM 的功能太强大了，以至于小弟已经把它作为主力文本编辑器，现把自己整理的一些技巧与大家分享，大部分都来自于手册。

#####

h (左移) j (下行) k (上行) l (右移)

输入 **dw** 可以从光标处删除至一个单字/单词的末尾

输入 **d\$** 从当前光标删除到行末

删除命令 **d** 的格式如下：

[number] d object 或者 **d [number] object**

其意如下：

number - 代表执行命令的次数(可选项，缺省设置为 1)。

d - 代表删除。

object - 代表命令所要操作的对象(下面有相关介绍)。

一个简短的对象列表：

w - 从当前光标当前位置直到单字/单词末尾，包括空格。

e - 从当前光标当前位置直到单字/单词末尾，但是 ***不*** 包括空格。

\$ - 从当前光标当前位置直到当前行末。

欲删除整行，请输入：`dd`

输入 `p` 将最后一次删除的内容置入光标之后

输入 `r` 和一个字符替换光标所在位置的字符

要改变一个单字/单词的部分或者全部，请输入 `cw`

更改类指令的工作方式跟删除类命令是一致的。操作格式是：

`[number] c object` 或者 `c [number] object`

对象参数也是一样的，比如 `w` 代表单字/单词，`$`代表行末等等。

输入 `CTRL-g` 显示当前编辑文件中当前光标所在行位置以及文件状态信息。

输入 `SHIFT-G` 则直接跳转到文件中的某一指定行。

输入 `/` 以及尾随的字符串可以用以在当前文件中查找该字符串。要查找同上一轮的字符串，只需要按 `n` 键。要向相反方向查找同上一轮的字符串，请输入 `Shift-N` 即可。如果您想逆向查找字符串，请使用 `?` 代

替 / 进行。

按 % 可以查找配对的括号)、]、}。

输入 :s/old/new/g 可以替换 old 为 new。

要替换两行之间出现的每个匹配串，请输入 :#,#s/old/new/g (#,#代表的是两行的行号)。输入 :%s/old/new/g 则是替换整个文件中的每个匹配串。进行全文替换时询问用户确认每个替换需添加 c 选项，请输入 :%s/old/new/gc

输入 :! 然后紧随著输入一个外部命令可以执行该外部命令。

要将对文件的改动保存到文件中，请输入 :w FILENAME 。

要保存文件的部分内容，请输入 :#,# w FILENAME **

#,# 就是上面要求您记住的行号(顶端行号,底端行号)

要向当前文件中插入另外的文件的内容，请输入 :r FILENAME

特别提示：您所提取进来的文件将从光标所在位置处开始置入

输入 `o` 将在光标的下方打开新的一行并进入插入模式。输入大写的 `O` 可以在光标上方打开新的一行并将光标置于新开的行首，进入插入模式。

输入小写的 `a` 可以在光标所在位置之后插入文本。输入大写的 `A` 可以在光标所在行的行末之后插入文本。

输入大写的 `R` 可连续替换多个字符。******直至按 `<ESC>` 键退出替换模式而进入正常模式。

J: 把两行连起来

CTRL-R: redo

`"o"` 命令在光标下方建立一个新的空行，并把 **Vim** 切换到插入模式

`"O"` 命令(大写)在光标上方打开一个新行

w: 移动光标向前跳动一个词，移动到词首

b: 与 **w** 相反

`"e"` 命令可以移到下一个单词的词末，而 `"ge"` 则移动到前一个单词的末尾

`"$"` 命令把光标移动到当前行行尾

"^" 命令把光标移动到当前行的第一个非空字符

"0"（零） 命令则移到一行的第一个字符

f/F: 单字符查找命令，最有用的移动命令之一，"fx" 命令向前查找本行中的字符 x。"F" 命令则用于向左查找。

"tx" 命令与 "fx" 相似，但它只把光标移动到目标字符的前一个字符上。

提示："t" 表示 "To"。这个命令的反向版本是 "Tx"。

这四个命令可以通过 ";" 命令重复，"," 命令则用于反向重复。无论用哪个命令，光标永远都不会移出当前行，哪怕是这两行是连续的一个句子。

G: 移动到指定的行，"33G" 把你送到 33 行

"50%" 移动到文件的中间，而 "90%" 移到差不多结尾的位置。

H,M,L: 分别代表移到当前视野的 Home, Middle, Last 处

:set number 这会在每行的前面加上一个行号

有些 "操作符—动作" 命令由于经常被使用，所以被设置为单字符命令：

x 表示 dl （删除当前光标下的字符）

X 表示 dh （删除光标左边的字符）

D 表示 d\$ （删除到行尾）

C 表示 c\$ （修改到行尾）

s 表示 cl （修改一个字符）

S 表示 cc （修改一整行）

使用 "V" 命令来启动可视模式。左右移动不会有任何效果。而通过上下移动，你可以一次选择多行。如果你要处理一个矩形块内的文本，可以使用 CTRL-V 启动可视模式。

"daw" 的 "d" 是删除操作符。"aw" 是一个文本对象。提示："aw" 表示 "A Word"（一个单词），这样，"daw" 就是 "Delete A Word"（删除一个单词）。确切地说，该单词后的空格字符也被删除掉了。

"cis" 包括 "c"（change，修改）操作符和 "is" 文本对象。这表示 "Inner Sentence"（译者注：实在很难用中文表示这个意思了，各位还是记英文名吧）。还有一个文本对象是 "as"，区别是 "as" 包括句子后面的空白字符而 "is" 不包括。如果你要删除一个句子，而且你还想同时删除句子后面空白字符，就用 "das"；如果你想保留空白字符而替换一个句子，

则使用"cis"。

还有很多方法可以删除文本。这是一些经常用到的：

x 删除光标下的字符 ("dl"的缩写)

X 删除光标前的字符 ("dh"的缩写)

D 从当前位置删除到行尾 ("d\$"的缩写)

dw 从当前位置删除到下一个单词开头

db 从当前位置删除到前一个单词的开头

diw 删除光标上的单词 (不包括空白字符)

daw 删除光标上的单词 (包括空白字符)

dG 删除到文末

dgg 删除到文首

如果你用 "c" 代替 "d", 这会变成修改命令; 而改用 "y", 则变成拷贝命令, 等等等等。

还有一些常用的命令, 放在哪一章都不合适, 列在这里:

~ 修改光标下字符的大小写, 并移动到下一个字符。这不是一个操作符 (除非设置了 'tildeop'), 所以你不能连接一个动作命令。这个命令在可视模式下也有效, 它会改变被选中的所有文本的大小写。

I 移到当前行的第一个非空字符并启动插入模式

A 移动到行尾并启动插入模式

彩色的文字难以辨认

Vim 自动猜测你使用的背景色。如果是黑的（或者其它深色的色彩），

它会

用浅色作为前景色。如果是白的（或者其它浅色），它会使用深色作为

前景

色。如果 Vim 猜错了，文字就很难认了。要解决这个问题，设置一下

'background' 选项。对于深色：

```
:set background=dark
```

而对于浅色：

```
:set background=light
```

这两个命令必须在 `":syntax enable"` 命令前调用，否则不起作用。如果

要在

这之后设置背景，可以再调用一下 `":syntax reset"`。

```
:colorscheme evening
```

保留原始文件

如果你在编辑源程序，你可能想在修改之前保留一个备份。但备份文件会在你存盘的时候

被覆盖。这样它只能保留前一个版本，而不是最早的文件。

要让 Vim 保存一个原始的文件，可以设置 'patchmode' 选项。这个选项定义需要

改动文件的第一个备份文件的扩展名。通常可以这样设：

```
:set patchmode=.orig
```

这样，当你第一次编辑 data.txt，作了修改并执行存盘，Vim 会保留一个名为

"data.txt.orig" 的原始文件。

如果你接着修改这个文件，Vim 会发现这个原始文件已经存在，并不再覆盖它。进

一步的备份就存在 "data.txt~"（或者你设置的 'backupext' 指定的文件）中。

如果你让 'patchmode' 设为空（这是默认的情况），则原始文件不会被保

留。

文件间拷贝文本

本节解释如何在文件间拷贝文本。我们从一个简单的例子开始。编辑一个你要拷贝文本的

文件，把光标移到要拷贝的文本的开始处，用 "v" 命令启动可视模式，然后把光标移到

要拷贝文本的结尾处，输入 "y" 拷贝文本。

例如，要拷贝上面这段文字，你可以执行：

```
:edit thisfile
```

```
/本节解释
```

```
vjjj$y
```

现在编辑你要粘贴文本的文件。把光标移到你要插入文本的地方。用

"p" 命令把文本粘贴

到那里：

```
:edit otherfile
```

```
/There
```

```
p
```

当然，你可以用任何命令拷贝文本。例如，用 "V" 命令选中整行的内

容。或者用

CTRL-V 选择一个矩形区域。或者使用 "Y" 拷贝一个单行, "yaw" 拷贝一个单词等。

"p" 命令把文本粘贴到光标之后, "P" 命令则粘贴到光标之前。注意,

Vim

会记住你拷贝的是一整行还是一个矩形, 并用相同的方式把文本贴出来。

关闭窗口

以下命令用于关闭窗口:

:close

实际上, 任何退出编辑的命令都可以关闭窗口, 象 ":quit" 和 "ZZ" 等。

但 "close"可以避免你在剩下一个窗口的时候不小心退出 Vim 了。

关闭所有其它窗口

如果你已经打开了一整套窗口, 但现在只想编辑其中一个, 如下命令可以完成这个功能:

:only

这个命令关闭除当前窗口外的所有窗口。如果要关闭的窗口中有一个没有存盘，Vim 会

显示一个错误信息，并且那个窗口不会被关闭。

对所有窗口执行命令

你打开了几个窗口，现在你想退出 Vim，你可以分别关闭每一个窗口。

更快的方法是：

`:qall`

这表示 "quit all"（全部退出）。如果任何一个窗口没有存盘，Vim 都不会退出。同时光标会自动跳到那个窗口，你可以用 `":write"` 命令保存该文件或者 `":quit!"` 放弃修改。

如果你知道有窗口被改了，而你想全部保存，则执行如下命令：

`:wall`

这表示 "write all"（全部保存）。但实际上，它只会保存修改过的文件。

Vim 知道保

存一个没有修改过的文件是没有意义的。

另外，还有 `":qall"` 和 `"wall"` 的组合命令：

`:wqall`

这会保存所有修改过的文件并退出 Vim。

最后，下面的命令由于退出 Vim 并放弃所有修改：

`:qall!`

注意，这个命令是不能撤消的。

用 `vimdiff` 显示区别

有一种特殊的启动 Vim 的方法可以用来显示两个文件的区别。让我们打开一个 `"main.c"`

并插入一些字符。在设置了 `'backup'` 选项的情况下保存这个文件，以便产生 `"main.c~"`

备份文件。

在命令行中输入如下命令：（不是在 Vim 中）

`vimdiff main.c~ main.c`

Vim 会用垂直分割的方式打开两个文件。你只能看到你修改过的地方和上下几行的地方。

我 在 哪？

要知道当前文件在文件列表中的位置，可以注意一下文件的标题。那里应该显示类似

"(2 of 3)" 的字样。这表示你正在编辑三个文件中的第二个。

如果你要查看整个文件列表，使用如下命令：

```
:args
```

这是 "arguments"（参数）的缩写。其输出应该象下面这样：

```
one.c [two.c] three.c
```

这里列出所有你启动 Vim 时指定的文件。你正在编辑的那一个，例如，
"two.c"，被中括号括起来了。

移动到另一个参数

要回到前一个文件：

```
:previous
```

这个命令与 ":next" 相似，只不过它是向相反的方向移动。同样地，这个命令有一个

快捷版本用于 "保存再移动"：

```
:wprevious
```

要移动到列表中的最后一个文件：

```
:last
```

而要移动到列表中的第一个文件：

```
:first
```

不过，可没有 `":wlast"` 或者 `"wfirst"` 这样的命令了。

你可以在`":next"`和`":previous"`前面加次数前缀。例如要向后跳两个文件：

```
:2next
```

选项 `'fileformats'` 包含各种各样的格式，Vim 会在编辑一个新文件之初尝试该

选项定义得各种格式。例如，下面这个命令告诉 Vim 先尝试用 UNIX 格式，其次，尝试

MS-DOS 格式：

```
:set fileformats=unix,dos
```

`/joe/e`：设置光标到匹配"joe"的末尾

`/joe/e+1`：设置光标到匹配"joe"的末尾再后移一位

`/joe/s-2`：设置光标到匹配"joe"的开头再前移两位

`/joe/-2`：设置光标到匹配"joe"的行再向上移两行的开头

`/^joe.*fred.*bill/`：匹配以"joe"开头且"joe"到"fred"到"bill"之间没有字符或者有字符

`/^[A-J]\+/:` 搜索以'A'到'J'间的一个或者多个字母组合的开头

`/begin\_.*end`：多行匹配，`begin` 和 `end` 可以不在同一行，因为`\_.`包括`\n`

`/fred\s*joe/i`：可以是任何空白字符包括空格，`\t`，`\n` 等等

/fred\joe : 搜索 fred 或者 joe

/. *fred\&.*joe : 搜索同时包括 fred 跟 joe 的行。注意, \&表示“且”, 所以行中不一定要求 fred 在 joe 前

^<fred>/i : 搜索独立的单词 fred

^<d\d\d\d> : 搜索独立的 4 位数字

^D\d\d\d\dD : 搜索 6 位字符串, 中间 4 位数字, 首尾两位不能为数字

^<d\{4}> : 同^<d\d\d\d>

" 查找空行

/^n\{3} : 匹配 3 个连续的空行

" 使用正则表达式组查找

^(fred\).*(joe\).*\2.*\1

" 正则表达式重复

/^\([^,]*,\)\{8}

" visual searching

:vmap // y/<C-R>"<CR> : 可视模式下的键盘映射。把//映射成“查找当前选中的文本”, y 表示将当前选中内容拷贝到"寄存器中, 然后/开始查找,

查找内容是后面的输入，<C-R>表示 ctrl-r，<CR>是回车

:vmap <silent> // y/<C-R>=escape("@", '\. *\$^~[]')<CR><CR> : 包括空白字符，加上<silent>，是为了在屏幕最下面的命令行不显示该命令。

" \zs 和 \ze 匹配原 :h \zs

/<\zs[^>]*\ze> : 匹配尖括号中的内容

" 零宽度匹配 :h \@=

/<\@<=[^>]*>\@= : 匹配<>标签中的内容，而忽略<和>本身

/<\@<=[^>]*>\@= : 和上例相比，差别在于允许跨行

" 多行查找 _ 的意思是包括换行符

/<!--_p\{-}--> : 匹配<!--开始到-->结尾的所有内容

/fred_s*joe/i : 匹配 fred 开始到 joe，之间没有任何字符或者是有空白字符

/bugs\(_\.)*bunny : 匹配所有 bugs 到 bunny 的字符串

:h _ : help

" 查找函数声明，nmap 为 normal 模式下的键盘映射

:nmap gx yiw/^\(sub\<bar>function\)s\+<C-R>"<CR> : yiw 表示将当前所

在单词拷贝到寄存器"中，<bar>表示|，即 sub 或者 function

" 查找多个文件

:bufdo /searchstr/ : 在多个文件缓冲区里执行查找

" 更好的多文件查找定位方法

:bufdo %s/searchstr/&/gic : 在多个文件缓冲区里查找，按下 n 停止

" 怎样不使用 / 来查找网址

?<http://www.vim.org/> : 向上查找

" 查找指定字符以外的字符串

^c\v([[^]aeiou]&\a){4} : 查找 4 个辅音字母，\c 表示忽略大小写，\v 表示后面所有 ascii 字符，除了 0-9，a-z，A-Z 以及_，均有特殊含义，就是所谓非常 magic，而 \V 表示后面的内容中只有反斜杠\含有特殊含义，就是所谓的非常不 magic，(和)中间的内容，表示首先不能是 aeiou，但是必须是\a(字母)。

" 替换

:%s/fred/joe/igc : 普通替换命令

:%s/\r//g : 删除 DOS 的换行符 ^M

" 你的文本文件是否乱七八糟的排成一行? 使用如下命令

:%s/\r\r/g : 转换 DOS 回车符 ^M 为真正的回车符

:%s=*\$== : 删除行尾空白

:%s= \+\$== : 同上

:%s#\s*\r?\$## : 删除尾部空白和 dos 换行符

:%s#\s*\r*\$## : 同上

" 删除空行

:%s/^\n\{3}// : 删除连续 3 个空行

:%s/^\n\+/\r/ : 压缩空行, 多个替换为一个

%s#<[>]\+>##g : 删除 html 的 tag 部分

" 如果你只想学一招, 那么就是这个了

:'a,'bg/fred/s/dick/joe/igc : 非常有用

"a,"b 指定一个范围: mark a ~ mark b

g//用一个正则表达式指出了进行操作的行必须可以被 fred 匹配

看后面, g//是一个全局显示命令

s/dick/joe/igc 则对于这些满足条件的行进行替换

" 复制最后一个字段

:%s= [^]+\$=&&= : 复制最后一个字段

:%s= \f+\$=&&= : 一样

:%s= \S+\$=&& : 一样!

" 记忆（反向引用）

:s/(.*):(.*)/\2:\1/ : 将两个字段颠倒

:%s/^(.*)\n\1\$/\1/ : 删除重复行

" 非贪婪匹配 \{-}

:%s/^\{-}pdf/new.pdf/ : 将第一个 pdf 的名字换为 new.pdf

" 使用可选原子 \?

:%s#\<[zy]\?tbl_[a-z_]\+>#\L&#gc : 匹配 tbl 前面没有字母，或者 z 和 y 中的某一个字母，然后将所有的字母变为小写

" 跨越尽量多的行

:%s/<!--_{-}-->// : 删除多行注释

:help ^{-} : 查看非贪婪匹配的更多帮助

" 使用寄存器替换

:s/fred/<c-r>a/g : 将 fred 替换为寄存器 a 里的内容, <c-r>为按下 Ctrl 与 r, 然后输入 a 后, 寄存器 a 的内容会出现在命令行

:s/fred/<c-r>asome_text<c-r>s/g

:s/fred/^\=@a/g : 与第一条的作用相同, 但是更优雅一些, 因为不会在命令行显示寄存器的内容

" 在一行里写多种命令

:%s/\f+\.gif>/\r&\r/g | v/\.gif\$/d | %s/gif/jpg/ : 将所有带有.gif 的行, 前后均加入一个空行; 将不带有.gif 字样的行全部删除; 将所有行中的 gif 换成 jpg; 注意三条语句, 一旦某一条失败, 则不执行下面的语句

:%s/a/but|gie|:update|:next : 首先, 将当前文件中的所有 a 变为 but; 然后保存文件; 最后进入下一个文件缓存区。如果有多个文件需要如此处理, 可以考虑使用 @:来重复, @:的意义是: 执行上一次的命令行命令 n 次。

" 或运算

:%s/suck|buck/loopy/gc : 替换 suck 或者 buck 为 loopy (这里|不是管道)

" 调用 vim 函数

:s/___date___/\=strftime("%c")/ : 将___date___替换成当前日期, 使用 strftime 函数。注意\=表示后面是表达式, 结果可能是 2008-1-3 17:59:46

" 处理字段, 替换所有在第三个字段中的 str1 为 str2

:%s:\(\(\w\+\s\+\)\{2}\)str1:\1str2:

" 交换第一列跟第四列

:%s:\(\w\+\)\(.*s\+\)\(\w\+\)\$:\3\2\1:

" 过滤 form 中的内容放在寄存器里

:redir @*|sil exec 'g#<\(input\|select\|textarea\|/\=form\)\>#p'|redir END :

redir 是将执行的结果重定向到后面的*寄存器中, sil 是 silent

:nmap ,z :redir @*<Bar>sil exec

'g@<\(input<Bar>select<Bar>textarea<Bar>/\=form\)\>@p'<Bar>redir

END<CR> : 普通模式下, 敲入,z, 命令行执行 redir @*|sil exec

'g@<\(input\|select\|textarea\|/\=form\)\>@p'|redir END

" 一位以上的数字减 3 (带进位, 这个命令挺有趣)

:%s/\d\+/\=(submatch(0)-3)/

" 包含 loc 或者 functions 的行中的数字加 6

```
:g/loc\|function/s/d/\=submatch(0)+6/
```

" 比上面更好的方法

```
:%s#txtdev\zs\d#\=submatch(0)+1#g : \zs 表示是 s 要匹配的的内容的开头
```

```
:h /\zs 查看帮助
```

" 前缀为 gg 的数字加 6

```
:%s/\(gg\)\@<=\d\+\=submatch(0)+6/
```

```
:h zero-width 查看帮助
```

" 替换一个特定字符串为数字

```
:let i=10 | 'a,'bg/Abc/s/yy^\=i/ |let i=i+1 # 将 yy 转换成 10, 11, 12 等等
```

" 比上面的更精确

```
:let i=10 | 'a,'bg/Abc/s/xx\zsyy\ze^\=i/ |let i=i+1 # 将 xxyy 转换成 xx11(第一行), xx12 (第二行), xx13 (第三行)
```

" 发现要替换的文本, 放入内存, 然后使用 \zs 来简化替换

:%s/"([^\.]\+\.)*\zsxx\1/ : \zs 表示是 s 要匹配的的内容的开头

" 将光标下的单词，放入替换语句的 LHS 中

:nmap <leader>z :%s#\<<c-r>=expand("<cword>")<cr>\># : <leader>在没有定义 mapleader 时为\，定义后为 mapleader 的值。LHS 指的是 s 替换后面，被替换的内容（在左手边）。普通模式下，如果光标下为 abc，则输入\z 两个字符，结果命令行出现：%s#\<abc\>#，然后你就可以接着输入替换后的内容了。

" 将可视模式下的高亮文本，放入替换语句的 LHS 中

:vmap <leader>z :<C-U>%s/\<<c-r>*\\>/

" 以下功能相同，在替换语句内部进行替换

" 每行的一部分中，只执行多个单字符的替换

:%s,\(all/.*)\@<=/_,\g : 用_替换 all 后的所有/, all/中的/不变

" 功能同上

:s#all/\zs.*#\=substitute(submatch(0), '/', '_', 'g')#

" 通过将一行分开，然后重新组合，达到替换的效果

:s#all/#&^M#|s#/#_#g|-j! : 在 windows 下使用了 mswin 的模式，所以^M 得输入为\r，呵呵

" 在替换命令中使用替换

:%s/.*^='cp '.submatch(0).' all/'.substitute(submatch(0),'','_',g')/ : 不 work,
因为 all/后面的/和 s 所有的/有重复。所以我改为 :%s#.*#='cp
' '.submatch(0).' all/'.substitute(submatch(0),'','_',g')#

" 全局显示命令

:g/gladiolli/# : 查找并显示匹配的行号

:g/fred.*joe.*dick/ : 显示所有含有 fred,joe & dick 的行

:g/⟨fred⟩/ : 显示单一单词 fred

:g/^\s*\$/d : 删除所有空行

:g!/^dd/d : 删除不含字符串"dd"的行

:v/^dd/d : 同上

:g/fred/,joe/d : 删除所有的从 fred 到 joe

:g/-----/.-10,d : 以-----为标记删除之前的 10 行

:g/{/ ,/}/- s/\n\+/\r/g : 删除 {...}之间的空行

:v/^S/d : 删除空行

:v/./,./-j : 压缩空行

:g/^\$/./-j : 同上

:g/<input\|<form/p : 或运算

:g/^/put_ : 双倍行宽 (pu = put)

:g/^/m0 : 颠倒文件 (m = move)

:'a,'bg/^/m'b : 颠倒选中的 a 到 b

:g/^/t. : 重复行

:g/fred/t\$: 拷贝行从 fred 到结尾

:g/stage/t'a : 拷贝行从 stage 到 marker a (a 为标记的位置)

:g/^(^I[^I]*)\{80}/d : 删除最少包含 80 个 tab 的行, 注意: ^I 的输入方法是 ctrl-v, ctrl-i

" 隔行替换

:g/^/ if line('.')%2|s/^/zz /

" 查找标记 a 与 b 间所有包含"somestr"的行, 并全部复制到第一个包含"otherstr"的行后

:'a,'bg/somestr/co/otherstr/ : co(py) or mo(ve)

" 与上面相同，但是多做了一次替换

```
: 'a, 'bg/str1/s/str1/&&&/|mo/str2/
```

: %norm jdd : 隔行删除

" 增加数字 (键入 <c-a>) : 在 MS-Windows 中 <c-a> 已经被定义为全选

.., \$g/^d/exe "norm! \<c-a>": 增加从当前行首到结尾的数字

: 'a, 'bg/^d\+/norm! ^A : 增加数字

" 保存全局命令的结果 (注意必须使用添加模式) 你需要使用 qaq 清空寄存器 a.

" 将结果存储到寄存器 a 或者复制缓存区中

:g/fred/y A : 添加带有 fred 的行到寄存器到 a

:g/fred/y A | :let @*=@a : 除了同上外，还将寄存器 a 的内容，放入复制缓冲区

:let @a="|g/Barratt/y A |:let @*=@a

: 'a, 'b g/^Error/ . w >> errors.txt : 将查找内容放入一个文件 (文件必须存在)

" 复制每一行，然后在复制出来的每一行左侧加上一个 print '复制出来的内容'

```
:g/./yank|put|-1s/'"/g|s/.*/Print '&'/
```

" 用文件中的内容替换字符串，-d 表示删除上面的一行

```
:g/^MARK$/r tmp.ex | -d
```

" 优雅地显示

```
:g/<pattern>/z#.5 : 带有上下文一并显示
```

```
:g/<pattern>/z#.5|echo "===== " : 优雅地显示
```

" 将 g//和普通模式下的命令结合起来

```
:g//norm 2f|r* : 将第二个|替换为*号
```

" 将前面 g 命令的输出，发送到新窗口中

```
:nmap <F3> :redir @a<CR>:g//<CR>:redir END<CR>:new<CR>:put!  
a<CR><CR>
```

" 全局命令和替换命令联姻 (强大的编辑能力)

```
:'a,'bg/fred/s/joe/susan/gic : 可以使用反向引用来匹配
```

```
:g/fred/,/joe/s/fred/joe/gic : 非行模式
```

" 先找 fred, 然后找 joe

:/fred/;/joe/-2,/sid/+3s/sally/alley/gIC : 最后这一个 C, 手册中都没有, 估计是 c

" 为每一行生成一个文件, 文件名从 1.txt 开始, 依次为 1.txt,2.txt,3.txt 等等

:g/^/exe ".w ".line(".").".txt"

" 绝对精华

* # g* g# : 查找当前光标下的单词 (单个单词) (<cword>) (向前/向后)

% : 匹配括号 {}[]()

. : 重复上次操作

@ : 重复上次的命令

matchit.vim : 适%能匹配 <script> <?php 等标记

<C-N><C-P> : 插入模式下自动完成填词

<C-X><C-L> : 行自动完成 (超级有用)

/<C-R><C-W> : 把单个<cword>单词放入搜索或者命令行

/<C-R><C-A> : 把字符串中有的单词<cword>放入搜索或者命令行

:set ignorecase : 忽略大小写

:syntax on : 打开语法高亮 Perl,HTML,PHP 等等

:h regexp<C-D> : 按 ctrl+d 得到包含 regexp 的列表

(按 tab 自动补齐)

" 简单编辑更新 _vimrc 文件

:nmap ,s :source \$VIM/_vimrc : 普通模式下的键盘映射 ,s 映射成加载用户目录下的_vimrc 文件

:nmap ,v :e \$VIM/_vimrc : ,v 映射成打开_vimrc 文件

可视模式 (方便增加 HTML 标签)

:vmap sb "zdi<C-R>z<ESC> : 在可视模式下将选中的文本前后分别加上某些东西。"z 表示寄存器 z, d 是删除, 然后 i 表示插入, <C-R>指 ctrl-r, 然后加 z, 表示粘贴寄存器 z 的内容, 最后 escape 键。注意: 本例将删

除的内容帖了回去，所以结果是什么变化都没有。

:vmap st "zdi<?= <C-R>z ?><ESC>：在前后加上<?=和?>。

" 浏览

:Exp(lore)：浏览文件

:Sex(plore)：分割窗口浏览文件

:ls：显示缓冲区

:cd ..：设置当前目录位置

:args：查看当前打开的所有文件

:lcd %:p:h：改变路径到当前编辑的文件

:autocmd BufEnter * lcd %:p:h：放入.vimrc 自动完成上面的命令

" 缓冲区浏览(一直排名前 10 的 vim 脚本)

" 需要 bufexplorer.vim http://www.vim.org/script.php?script_id=42

\be：缓冲浏览器中查看缓冲列表

\bs：同上，但是分割窗口

" 转换大小写

guu：将整行的字母转换成小写

gUU：将整行的字母转换成大写

Vu：转换选中的行（小写）

VU：转换选中的行（大写）

g~~：反向转换

vEU：转换词大写，v 表示进入可视模式，E 表示覆盖到词的末尾，U 表示大写。

vE~：反向转换词

ggguG：将当前编辑文件内容全部转换成小写

" 可视模式下选择所有的字母及数字 (放入 .vimrc 文件中)

```
vmap ,c :s/^\<(\.)(\k*)\>/\u\1\L\2/g<CR>
```

" 大写所有句子的第一个字母

```
:%s/[.!?]\_s\+\a/U&\E/g
```

gf：打开当前光标下或后的文件

:nnoremap gF :view <cfile><cr>：打开当前光标下或光标后的文件，如果

不存在则创建

ga : 显示当前光标下单个字的 ascii 码, 十进制, 十六进制.....

ggVGg? : 将整个文件用 rot13 编码..... (谁看得懂啊~~hoho)

ggg?G : 同上 (针对大文件)

:8 | normal VGg? : 将第八行用 rot13 编码

:normal 10GVGg? : 同上

<C-A>,<C-X> : 增加, 减少当前光标下的数字, window 用户缺省是将 C-A 映射成选中全文, 所以需要重定义 CNTRL-A

<C-R>=5*5 : 插入 25 (小型计算器)

" 几个彩蛋, 有意思

:h 42 : also <http://www.google.com/search?q=42>

:h holy-grail

:h!

" 标记 & 移动

': 跳回最后编辑的行 (超有用)

`: 同上, 但是定位编辑点

`g` : 跳转到比较旧的编辑位置 (如果有的话) (vim6.3 后的新功能)

`g,` : 这个是较新的位置 (同上)

`:changes` : 打出改变列表

`:h changelist` : 查看“改变表跳转”的帮助

`<C-O>` : 依次沿着你的跳转记录向回跳 (从最近的一次开始)

`<C-I>` : 依次沿着你的跳转记录向前跳

`:ju(mps)` : 列出跳转轨迹

`:help jump-motions`

`:history` : 列出历史记录

`:his c` : 命令行历史

`:his s` : 搜索历史

`q/` : 搜索命令历史的窗口

`q:` : 命令行命令历史的窗口

`:<C-F>` : 历史窗口

" 缩写 & 映射

`:map <f7> :a,'bw! c:/aaa/x`

`:map <f8> :r c:/aaa/x`

`:map <f11> :.w! c:/aaa/xr<CR>`

:map <f12> :r c:/aaa/xr<CR>

:ab php : 查看以 php 开头的缩写

:map , : 列出所有的映射（以逗号开始的）

" 允许映射 F10 (win32)

set wak=no : :h winaltkeys

" 映射中常使用的表示

<CR> : 回车

<ESC> : Esc

<LEADER> : 右斜杠

<BAR> : 管道符号

<BACKSPACE> : 退格键

<SILENT> : 不回显

#显示自定义的 RGB 颜色显示当前光标下的字符串 例如 #445588

:nmap <leader>c :hi Normal guibg=#<c-r>=expand("<cword>")<cr><cr>

map <f2> /price only\\|versus/ :in a map need to backslash the \

" 简单的 PHP 调试将所有显示的变量放入寄存器 a

```
iab phpdb exit("<hr>Debug <C-R>a ");
```

" 使用寄存器来映射 (放入 .vimrc 文件自动加载)

```
:let @m=":'a,'bs/'
```

```
:let @s=":%!sort -u"
```

" 列出寄存器

:reg : 显示当前所有的寄存器

:reg a : 显示寄存器 a 中的内容

"1p... : 引用一个叫 1 的寄存器

:let @y='yy@" : pre-loading registers (put in .vimrc)

qqq : 清空寄存器 "q"

" 一些有用的诀窍

"ayy@a : 把当前行作为命令执行

yy@" : 上面的匿名寄存器

u@. : 只执行键入的命令

" 从其它命令处获得输入（需要外部命令）

:r!ls.exe：从 ls 获得输入插入到当前位置

!!date：从 date 获得输入（删除当前行）

" 使用外部 sort 排序

:%!sort -u：用 sort 排序整个文件（结果覆盖整个文件）

:'a,b'!sort -u：从 mark a 到 mark b 之间的内容进行排序

!1} sort -u：排序一个段落

:g/^\$/;,/^-\$/-1!sort：将每个块排序(注意这个关键的;)

" 多文件管理 (基本的)

:bn：跳转到下一个 buffer

:bp：跳转到前一个 buffer

:wn：保存当前 buffer 并跳转到下一个 buffer (超有用)

:wp：保存当前 buffer 并跳转到前一个 buffer

:bd：把当前文件从 buffer 移出 (超有用)

:bun：卸载当前 buffer (关闭这个窗口但是不移出)

:badd file.c : 添加 file.c 到 buffer 列表

:b 3 : 前往第三个 buffer

:b main : 前往含有 main 的 buffer 中 比如说 main.c

:sav php.html : 把当前文件存为 php.html 并打开

:sav! %<.bak : 换一个后缀名保存 (旧方法)

:sav! %:r.cfm : 同上

:sav %:s/fred/joe/ : 替换文件名

:sav %:s/fred/joe/:r.bak2 : 替换文件和后缀

:!mv % %:r.bak : 重命名当前文件

:e! : 打开未修改之前的文件

:w c:/aaa/% : 存储文件到指定位置

:e # : 编辑标记为#的文件在 buffer 中

:rew : 返回到第一个可编辑的文件

:brew : 回到第一个 buffer

:sp fred.txt : 分割窗口打开 fred.txt

:sball,:sb : 把所有的 buffers 分割显示在一个窗口中 (超有用)

:scrollbind : 让每个分离的窗口, 同步滚动

:map <F5> :ls<CR>:e # : 按 F5 显示所有 buffer, 并显示行号

:set hidden : 允许不保存当前 buffer 而进行切换

" 在分割窗口中快速切换

map <C-J> <C-W>j<C-W>_

map <C-K> <C-W>k<C-W>_

" 录制命令 (最好的技巧)

qq # 录制命令放入 q 寄存器

输入一些命令

q # 录制结束

@q :执行放入寄存器 q 中的内容

@@ : 重复

5@@ : 重复 5 次

" 编辑一个 寄存器/录制

"qp :显示寄存器 q 中的内容(普通模式下)

<ctrl-R>q :显示寄存器 q 中的内容 (插入模式下)

" 你现在可以看到记录内容，随便编辑

"qdd :删除，重新存入 q

@q :执行 录制/寄存器 q

" 在可视块中运行记录

1) 定义记录/寄存器

qq:s/ to/ from/g^Mq

2) 定义可视块

V}

3) 键入：将显示下面信息

:!<,'>

4)完成如下操作

:!<,'>norm @q

" 将一个录制和一个映射绑定 (在命令模式中结束)

nnoremap] @q:w!<bar>bd

" 可视化模式提供一种灵活易用的方法选择一块文本供操作符使用

" 记出

v: 进入可视化模式

V：进入可视化行选择模式

<C-V>：进入可视化块选择模式

gv：重新选择

o：选择的区域头尾移动

"*y：复制选择区域到 paste buffer

V%：选择一个匹配段

V}J：合并一个段落

V}gJ：合并一个段落，并保留空格

" 删除选中的 10 行的前两个字符（不过这里应该假设是紧凑的排版格式，不能包含空格、tab 等字符的，可是经实验应该是前 3 个字符才对啊？ ？ ）

0<c-v>10j2ld

" 如何用可视块拷贝几列

" 可视块(并非通常的 v 命令)

<C-V>，然后通过移动命令选择列 (win32 <C-Q>)

然后执行 c,d,y,r 等命令

" 用另外一个此类的块，覆盖可视块中的文本

选中第一块: `ctrl-v move "ay`

选中第二块: `ctrl-v move c ctrl-o "aP <esc>`

" `_vimrc` 基本设置

`:set incsearch` : 输入搜索命令时，立即显示目前输入的模式对应的匹配。
匹配的字符串被高亮。

`:set wildignore=*.o,*.obj,*.bak,*.exe` : `tab` 补全时忽略这些忽略这些

`:set shiftwidth=3` : 设置自动缩进为 3 个字符

`:set vb t_vb=".` : 安静模式，关闭响铃跟闪烁

`:set browsedir=buffer` : 设置文件浏览使用的目录

“注:

”`last` 使用文件浏览器最近访问相同的目录。

“`buffer` 使用相关缓冲区的目录。

”`current` 使用当前目录。

“`{path}` 使用指定目录。

" 启动 windows 中的 IE

```
:nmap ,f :update<CR>:silent !start c:\progra~1\intern~1\iexplore.exe  
file://%:p<CR>
```

```
:nmap ,i :update<CR>: !start c:\progra~1\intern~1\iexplore.exe  
<cWORD><CR>
```

" 在 vim 里打开 ftp

```
cmap ,r :Nread ftp://209.51.134.122/public_html/index.html
```

```
cmap ,w :Nwrite ftp://209.51.134.122/public_html/index.html
```

```
gvim ftp://www.somedomain.com/index.html # 使用 netrw.vim
```

" 向寄存器中添加内容 (使用相应寄存器名称的大写)

" 复制 5 行放入 a 寄存器，然后向下跳转 10 行再添加 5 行到 a 寄存器

```
"a5yy
```

```
10j
```

```
"A5yy
```

[I：显示当前行中字符的所有匹配(超级有用)

" 常规缩进

:'a,'b>> ： 将 mark a 到 mark b 之间的内容进行两次缩进

" 虚拟模式下缩进 (可重复)

:vnoremap < <gv

”这是一个虚拟模式下的键盘映射 < 映射为<gv

"< 意为向内缩进，gv 上面已有解释，为重复上次选区

“<gv 也就是先向?謁禔 缓笈僚≡窀詹诺难[]?

“这样就可以只按 < 实现重复缩进了

:vnoremap > >gv ： 向内缩进，原理同上

" 块缩进

>i{

>a{

" also

>% and <%

”自己试试看吧，涉及到用 { 的语言很有用，比如 c,c++等

" 重定向 & 粘贴到寄存器 * (*为寄存器名称)

:redir @*: 重定向命令到 paste 缓冲区

:redir END : 结束

:redir >> out.txt : 重定向到文件

" 操作粘贴缓冲区

"*yy : 复制到寄存器

"*p : 从寄存器中粘贴一行

" 复制到粘贴缓冲区 (扩展模式)

:'a,'by* : 复制一个范围到粘贴寄存器

:%y* : 复制一个括号匹配到粘贴缓冲区

:.y* : 复制当前行到粘贴缓冲区

" 从剪贴板上过滤非可打印字符

" 当从一些 GUI 程序粘贴时会有用处

```
:nmap <leader>p :let @* = substitute(@*,'^\[:print:]]','g')<cr>"*p
```

" 重新格式化文本

gq} : 合并一个段落

gqap : 当前段落

ggVGgq : 全部段落

Vgq : 当前行

" 在 70 列的时候换行

```
:s/\.{,69\};\s*\.\{,69\}\s\+/\&\r/g
```

" 命令使用于多个文件

:argdo %s/foo/bar/e : 在所有文件上操做 :args

:bufdo %s/foo/bar/e

:windo %s/foo/bar/e

:argdo exe '%!sort'|w! : 包含外部命令

" 命令行技巧

`gvim -h` : 显示帮助

`ls | gvim -` : 管道操作

`cat xx | gvim - -c "v/\^d\d\^[3-9]/d "` : 从管道出过滤内容

`gvim -o file1 file2` : 分割窗口显示两个文件

" 打开文件后执行一条命令

`gvim.exe -c "/main" joe.c` : 打开 `joe.c` & 跳转到 `"main"`

" 在打开一个文件时执行多条命令

`vim -c "%s/ABC/DEF/ge | update" file1.c`

" 在一组文件上执行多条命令

`vim -c "argdo %s/ABC/DEF/ge | update" *.c`

" 从一系列文件中删除一块区域

`vim -c "argdo /begin/+1,/end/-1g/\^d | update" *.c`

" 自动编辑文件 (编辑命令序列 `Ex commands` 已经包含在 `convert.vim` 中

了)

```
vim -s "convert.vim" file.c
```

#不加载.vimrc 跟任何 plugin(干净清新的 VIM^_^)

```
gvim -u NONE -U NONE -N
```

" 访问粘贴缓冲区中的内容 (放置到脚本/批处理文件中)

```
gvim -c 'normal ggdG"*p' c:/aaa/xp
```

" 将粘贴的内容送往默认的打印机

```
gvim -c 's/^\\=@*/|hardcopy!|q!'
```

" gvim 里的 grep (win32 or *nix)

:grep somestring *.php : 创建匹配的文件列表

" 使用 :cn(向后后) :cp(向前) 操纵列表

:h grep : 查看帮助

" gvim 的差异比较

```
gvim -d file1 file2 : vimdiff (比较不差异)
```

dp : 把光标处的不同放到另一个文件

do : 在光标处从另一个文件取得不同

" Vim traps

在正则表达式中 + | ({ 都要加上转义符(反斜杠)

/fred\+/: 匹配 fred/freddy 但不匹配 free

^(fred\){2,3}/: 注意你必须加上反斜杠

" \v , 或叫做 very magic (通常都是这么叫)可以取消转义符

/codes\(\\n\\s\)*where : 普通的正则表达式

^vcodes(\\n\\s\)*where : very magic

" 把对象送到命令行或者搜索行

<C-R><C-W> : 执行当前光标下的单个单词

<C-R><C-A> : 执行当前光标下尽可能多的单词

<C-R>- : 送至一个小型寄存器 (同样使用于插入模式)

<C-R>[0-9a-z] : 送至一个命名寄存器 (括弧同上)

<C-R>% : 送至文件名(#也行) (同上)

<C-R>=somevar : 送至一个变量 (例如 :let sray="ray[0-9]")

" 控制寄存器

:let @a=@_ : 清除寄存器 a

:let @a="" : 同上 a

:let @*=@a : 拷贝寄存器 a 到 paste buffer

:let @*=@: : 拷贝最后执行的命令到 paste buffer

:let @*=@/ : 拷贝最后执行的查找命令到 paste buffer

:let @*=@% : 拷贝当前文件到 paste buffer

map <f11> "qyy:let @q=@q."zzz"

" 帮助的帮助? (使用 TAB)

:h quickref : VIM 快速参考页

:h tips : Vim'自己的技巧帮助

:h visual<C-D><tab> : 虚拟模式的帮助列表

: 然后使用 tab 选择它们

:h ctrl<C-D> : 所有关于 ctrl 键的帮助列表

:helpg uganda : 过滤帮助文件 使用 :cn, :cp 查找下一个及后一个

:h :r : 关于 :ex 的命令帮助

:h CTRL-R : 普通模式相关

:h ^r : \r 是什么意思

:h \zs : 使用双反斜线查找关于 \zs 的帮助

:h i_CTRL-R : 在插入模式中 <C-R>的解释

:h c_CTRL-R : 在命令模式中 <C-R> 的解释

:h v_CTRL-V : 虚拟模式

:h tutor : VIM 快速指南

<C-[>, <C-T> : Move back & Forth in HELP History

gvim -h : VIM 命令行帮助

" 选项设置在那里

:scriptnames : 列出所有已经加载的 plugins, _vimrcs 文件

:verbose set history? :显示 history 的值并显示在那里定义的

:function : 列出所有函数

:func SearchCompl : 显示指定函数的细节

" 制作你自己的 VIM 帮助

:helptags /vim/vim64/doc : 重新编译所有 *.txt 的帮助文件在这个目录里

:help add-local-help : 如何添加本地帮助

" 用外部程序运行文件 (例如 php)

map <f9> :w<CR>:!c:/php/php.exe %<CR>

map <f2> :w<CR>:!perl -c %<CR>

" 在另一个 buffer 中, 捕捉当前脚本的输出

:new | r!perl # : 新建一个 buffer, 从另一个 buffer 中读入结果

:new! x.out | r!perl # : 同上, 并指定一个新文件名

:new+read!ls

" 创建一个新的缓冲区, 将寄存器 q 的内容粘贴进去, 然后对缓冲区内
容排序

```
:new +put q|%!sort
```

" 插入 DOS 换行符

```
:%s/$^\<C-V>\<C-M>&/g : <C-V>为 ctrl-v
```

```
:%s/$^\<C-Q>\<C-M>&/g : 对于 Win32 应该换成 ctrl-q
```

```
:%s/$^\^M&/g : 你看到的^M 是一个字符
```

" 自动删除行尾 Dos 回车符和空格

```
autocmd BufRead * silent! %s/[\r \t]\+$//
```

```
autocmd BufEnter *.php :%s/[\t\r]\+$//e
```

" 对指定文件或文件类型执行某个动作

```
autocmd VimEnter c:/intranet/note011.txt normal! ggVGg?
```

```
autocmd FileType *.pl exec('set fileformats=unix')
```

" 把最后一个命令贴到当前位置

i<c-r>:

" 把最后一个搜索指令贴到当前位置

i<c-r>/

" 更多的完成功能

<C-X><C-F>: 插入当前目录下的一个文件名到当前位置

在 insert 模式下使用, 然后用 Ctrl-P/Ctrl-N 可以翻页

" 替换一个 visual 区域

" 选择一个区域, 然后输入 :s/Emacs/Vim/ 等等, vim 会自动进入:模式

:<,>s/Emacs/Vim/g: 前面的'<.>' 是 vim 自动添加的

gv: 重新选择前一个可视区域 (高级!)

" 在文件中插入行号

:g/^/exec "s/^/".strpart(line(".")." ", 0, 4)

:%s/^\\=strpart(line(".")." ", 0, 5)

:%s/^\\=line('.'). ''

#用 VIM 的方式来编号行

:set number : 显示行号

:map <F12> :set number!<CR> : Show linenumbers flip-flop

:%s/^/\=strpart(line('.')." ",0,&ts)

#从任意行开始编号(需要 perl)

:'a,b!perl -pne 'BEGIN{\$a=223} substr(\$_,2,0)=\$a++'

#产生数字列表

#Type in number on line say 223 in an empty file

qqmnYP`n^Aq : in recording q repeat with @q

" 递增已存在数字到文件末

.,\$g/^d/exe "normal! \<c-a>"

" 高级递增, 参见:

http://vim.sourceforge.net/tip_view.php?tip_id=150

" 高级递增 (真的很有用)

" 把下面几句放到 `_vimrc`

```
let g:I=0
```

```
function! INC(increment)
```

```
let g:I =g:I + a:increment
```

```
return g:I
```

```
endfunction
```

" 例如从 mark a 到 mark b 递增，从 223 开始，步长为 5

```
:let I=223
```

```
:'a,'bs/^/=INC(5)/
```

" create a map for INC

```
cab viminc :let I=223 \| 'a,'bs/$/=INC(5)/
```

" 生成从 23-64 的数字列表

```
o23<ESC>qqYp<C-A>q40@q
```

" 在当前插入模式下编辑/移动 (真得很有用)

<C-U> : 删除全部

<C-W> : 删除最后一个单词

<HOME><END> : 移动到行首/行尾

<C-LEFTARROW><C-RIGHTARROW> : 向前/后移动一个单词

<C-X><C-E>,<C-X><C-Y> : 滚动, 只要在 insert 中保持 put

#加密(小心使用, 不要忘了密码)

:X : vim 会提示你输入密码

:h :X

" 模式行 (使文件只读等), 必须在前/后 5 行内

// vim:noai:ts=2:sw=4:readonly:

" vim:ft=html: : 使用 HTML 语法高亮

:h modeline

" 建立你自己的菜单项

```
amenu Modeline.Insert\ a\ VIM\ modeline <Esc><Esc>ggOvim:ff=unix
ts=4 ss=4<CR>vim60:fdm=marker<esc>gg
```

" 一个保存当前光标下的狭义字到一个文件的函数

```
function! SaveWord()
```

```
normal yiw
```

```
exe '!:echo '@0.' >> word.txt'
```

```
endfunction
```

```
map ,p :call SaveWord()
```

" 删除重复行的函数

```
function! Del()
```

```
if getline(".") == getline(line(".") - 1)
```

```
norm dd
```

```
endif
```

```
endfunction
```

:g/^/ call Del() #使用该函数的一个例子

" 双字节编码 (non alpha-numeric)

:digraphs : 显示编码表

:h dig : 帮助

i<C-K>e' : 输入 é

i<C-V>233 : 输入 é (Unix)

i<C-Q>233 : 输入 é (Win32)

ga : 查看字符的 hex 值

#删除非 ascii 字符

:%s/[<C-V>128-<C-V>255]//gi : where you have to type the Control-V

:%s/[€-ÿ]//gi : Should see a black square & a dotted y

:%s/[<C-V>128-<C-V>255<C-V>01-<C-V>31]//gi : All pesky non-asciis

:exec "norm /[\x00-\x1f\x80-\xff]/" : same thing

将非 ascii 字符，拉到搜索条上

yl/<C-R>" :

/[^a-zA-Z0-9_[:space:][:punct:]] : search for all non-ascii

" 文件名自动完成 (例如 main_c.c)

:e main_<tab> : tab 键完成

gf: 打开光标处广义字命名的文件 (normal 模式)

main_<C-X><C-F> : 文件名自动完成(insert 模式)

" Vim 复杂使用

" 交换两个单词

:%s/^\<\(on\|off\) \>/^=strpart("offon", 3 * ("off" == submatch(0)), 3)/g

" 交换两个单词

:vnoremap <C-X> <Esc>`.``gvP``P

" 把 text 文件转换成 html 文件(oh,ft)

:runtime! syntax/2html.vim : 转换 txt 成 html

:h 2html

" VIM 有一个内部自带的 grep 命令

:grep some_keyword *.c : 得到一个包含 some_keyword 的 c 文件名列表

:cn : 去下一个出现的位置

" 强制无扩展名的文件的语法着色方式

:set syntax=perl

" 取消语法着色 (很有用)

:set syntax off

" 改变色彩主题 (在~vim/vim??.colors 中的任何文件)

:colorscheme blue

" 通过使用模式行强迫使用 HTML 语法高亮

vim:ft=html:

" 强制自动语法加亮(非标准的文件扩展)

au BufRead,BufNewFile */Content.IE?/* setfiletype html

`:set noma (non modifiable)` : 防止修改

`:set ro (Read Only)` : 只读保护

" 会话 (打? 欢盐募?)

`gvim file1.c file2.c lib/lib.h lib/lib2.h` : 在"对话"中加载这些文件

`:mksession` : 生成一个 Session 文件 (默认是 Session.vim)

`:q`

`gvim -S Session.vim` : 重新加载所有文件

#标记(tags) (跳转到子程序/函数)

`taglist.vim` : 很流行的插件

`:Tlist` : 显示标记 (函数列表)

`<C-]>` : 跳转到光标处的函数

" 将 csv 文件分栏, 以便仅显示宽行的字符

`:let width = 20`

```
:let fill=' ' | while strlen(fill) < width | let fill=fill.fill | endwhile  
:%s/\([^;]*\);\\=\\=strpart(submatch(1).fill, 0, width)/ge  
:%s/\\s\\+$//ge
```

" 高亮显示特定的 csv 列 (放入.vimrc 文件)

```
function! CSVH(x)
```

```
execute 'match Keyword /^\\([\\^,]*\\)\\{'.a:x.'\\}zs[\\^,]*/'
```

```
execute 'normal ^'.a:x.'f,'
```

```
endfunction
```

```
command! -nargs=1 Csv :call CSVH(<args>)
```

" call with

:Csv 5 : 高亮显示第 5 列

" 折叠: 隐藏某些片断, 使查看更容易

zf} : 使用动作命令折叠一个段落

v}zf : 使用可视模式折叠一个段落

zf'a : 折叠到一个标记上

zo : 打开折叠

zc：重新关闭折叠

" 显示"不可见字符"

:set list

:h listchars

" 如何在不进入插入模式的情况下粘贴"普通模式的命令"

:norm qqy\$jq

" 处理文件名

:h filename-modifiers：帮助

:w %：写入当前文件

:w %:r.cfm：改变文件扩展名为 .cfm

:!echo %:p：显示完整路径和文件名

:!echo %:p:h：只显示完整路径

:!echo %:t：只显示文件名

:reg %：显示文件名

<C-R>% : 插入文件名 (插入模式)

"%p : 插入文件名 (普通模式)

/<C-R>% : 在文本中查找文件名

" 删除, 但不破坏 buffer 内容, "_为黑洞寄存器, 相当于 linux 上的 /dev/null

"_d : 你一直想要的东西

"_dw : 例如: 删除一个单词

" 送完整的路径名到剪贴板, 用于邮件附件等

:nnoremap <F2> :let @*=expand("%:p")<cr> :unix

:nnoremap <F2> :let @*=substitute(expand("%:p"), "/", "\\", "g")<cr> :win32

" 不用离开 Vim 就能修改文件名的简单 shell 脚本

\$ vim

```
:r! ls *.c
```

```
:%s/\(.*\).c/mv & \1.bla
```

```
:w !sh
```

```
:q!
```

" 在一个文本里计算单词数

```
g<C-G>
```

" 你自己设置高亮显示的例子

```
:syn match DoubleSpace " "
```

```
:hi def DoubleSpace guibg=#e0e0e0
```

" 重新逐字产生前面一行

```
:imap ] @@@<ESC>hhkyWjl?@@@<CR>P/@@@<CR>3s : 此种复杂的
```

映射，解释，会害人的:) 首先，输入@@@三个

然后 ESC 回到普通模式，hh 是往左移动两个字符，就是到了第一个@

上，然后 k 是到上一行。y 表示 yank，拷贝，W 是指拷贝到大单词的结

束。j 是往下一行，就是回到原来一行，l 表示往右移动一个字符，以便下面的?查找。?是向上查找，找的内容是@@@。<CR>回车执行后，P 是指将寄存器中的内容（即上一行的完全内容）拷贝到@@@之前，然后再用/查找@@@。3s 表示删除 3 个字符（就是@@@），并切换到插入模式。

```
:nmap ] i@@@<ESC>hhkyWjl?@@@<CR>P/@@@<CR>3s
```

" 根据文件类型映射快捷键

```
:autocmd bufenter *.tex map <F1> :!latex %<CR>
```

```
:autocmd bufenter *.tex map <F2> :!xdvi -hush %<.dvi&<CR>
```

" 读取 MS-Word 文档，需要有 antiword(一个免费的 word 文档阅读器)

```
:autocmd BufReadPre *.doc set ro
```

```
:autocmd BufReadPre *.doc set hlsearch!
```

```
:autocmd BufReadPost *.doc %!antiword "%"
```

" 折行的方法

" 第一行是设置选项，让系统认为<<<和>>>是折行标记，后面三行放

到正文中，就看到折叠效果了

```
:se filetype=help foldmethod=marker foldmarker=<<<,>>>
```

A really big section closed with a tag <<<

--- remember folds can be nested ---

Closing tag >>>

" 另一个 Vim 黑客的 JAVH

```
vim -c ":%s%s*%Cyrnfr)fcbafbe[Oenz(Zbbyranne%|:%s)[[()])-)Ig|norm
Vg?"
```

启动 vim 的时候执行了一个命令

先写入了 Please-sponsor-Bram-Moolenaar 的 rot13 编码，然后再解码