



下载APP



拓展5 | 其他插件和技巧：吴咏炜的箱底私藏

2020-09-07 吴咏炜

Vim 实用技巧必知必会

[进入课程 >](#)**讲述：吴咏炜**

时长 17:49 大小 16.32M



你好，我是吴咏炜。

上一讲我介绍了最重要的一些插件。今天这讲拓展，我们就算是查漏补缺，再分享一些我个人这些年压箱底的收藏。这些插件和技巧有新有旧，都非常好用，欢迎你挑选感兴趣的内容，纳入自己的个人收藏箱。

插件

Syntastic 和 ALE



说到代码检查插件，我最早是从 [Syntastic](#) 开始用的，然后慢慢转向了 [ALE](#)。不过，我因为主要写 C++ 和 Python，所以慢慢放弃了使用这两个插件，转而使用对这两种语言支持较好的 YCM（第 13 讲）和 Python-mode（拓展 3）。

虽然 YCM 和 Python-mode 集成的工具比较有限，比如 YCM 对 C++ 的代码检查仅限于 Clang 系列工具提供的支持，而不像 Syntastic/ALE 还可以选择很多其他的工具，但是，它们对我来讲还是够用了——何况对于 C 和 C++，要让 Syntastic 或 ALE 干活的话，大部分情况下需要配置头文件包含路径和编译选项，也是件麻烦事。

不过，对于其他语言的开发者，Syntastic/ALE 可能还是非常有用的。

先说 Syntastic。这是一个老牌的代码检查插件，其 1.0 版本发布在 2009 年。这些年来，这个插件里积累了好几十种语言的代码检查支持，既有常见的 C、C++、Python、Java、JavaScript 等语言，也有冷门一点的 ACPI、AppleScript、Julia、VHDL、z80 汇编等语言。对于每种语言，它能自动识别已经安装的代码检查器，并在你文件存盘时自动检查代码（也可以手工使用 `:SyntasticCheck` 命令来检查）。要检查当前文件 Syntastic 识别到了哪些代码检查器，可以使用 `:SyntasticInfo` 命令；而在 ALE 中没有等价的好用命令。

Syntastic 的主要问题是它的检查是同步的，代码检查时你不能同时进行编辑工作：不但不能即输即查，而且耗时长的检查还会中断正常的编辑流程。后起之秀 ALE 恰恰解决了这个问题，它会异步地检查你的代码，在编辑时即输即查，完全不会影响正常的编辑流程。从支持的语言上来说，ALE 虽然不及 Syntastic，但也已经覆盖到了大量的冷门语言。并且，它仍然处于积极开发之中：2020 年 7 月，Syntastic 一共有 3 次提交，而 ALE 有 48 次非合并提交和 13 次合并提交！

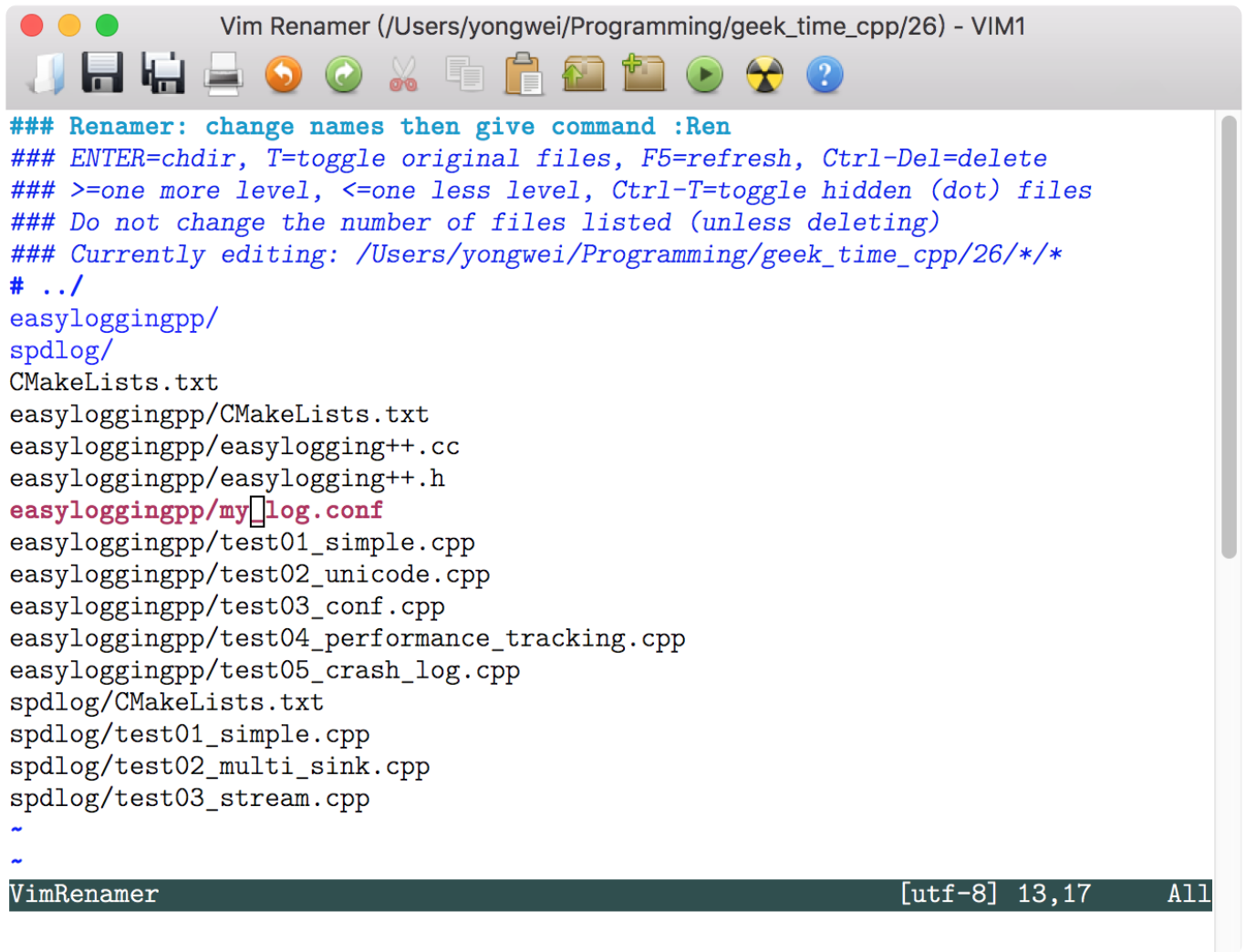
这两个插件的配置都略显复杂，通常需要你针对每种代码检查器进行配置。因此，总体来说，我对代码检查的推荐顺序是：

1. 使用 YCM、Python-mode、Vim-go 等有语言针对性的插件，如果你用的语言被支持，并且插件集成的代码检查功能够用的话
2. 使用 ALE，如果你的语言和代码检查插件它能够支持的话
3. 使用 Syntastic，如果其他选项不适用，或者你需要的检查执行够快的话

Renamer

在需要对文件进行批量更名时，我会使用 [@qpkorr/vim-renamer](#) 插件。它提供 `:Renamer` 命令，会打开当前目录下所有文件的列表。你随后就可以利用 Vim 强大的正则

表达式和编辑功能来调整这些名字了。在调整完成后，执行 `:Ren` 命令即可。



```

### Renamer: change names then give command :Ren
### ENTER=chdir, T=toggle original files, F5=refresh, Ctrl-Del=delete
### >=one more level, <=one less level, Ctrl-T=toggle hidden (dot) files
### Do not change the number of files listed (unless deleting)
### Currently editing: /Users/yongwei/Programming/geek_time_cpp/26/*/*
# ../
easyloggingpp/
spdlog/
CMakeLists.txt
easyloggingpp/CMakeLists.txt
easyloggingpp/easylogging++.cc
easyloggingpp/easylogging++.h
easyloggingpp/mylog.conf
easyloggingpp/test01_simple.cpp
easyloggingpp/test02_unicode.cpp
easyloggingpp/test03_conf.cpp
easyloggingpp/test04_performance_tracking.cpp
easyloggingpp/test05_crash_log.cpp
spdlog/CMakeLists.txt
spdlog/test01_simple.cpp
spdlog/test02_multi_sink.cpp
spdlog/test03_stream.cpp
~
~
VimRenamer [utf-8] 13,17 All

```

Renamer 的界面（额外展开了一层目录）

注意，图中的第二、三行提供了命令的说明，比如用 `>` 来多展开一层目录，等等。你应该只做行内的修改，而不去删除行或调整行的顺序，否则可能引致意外的后果。使用 `<C-Del>` 可以删除当前行的文件，这算是一种例外情况。

在图中，我只是手工修改了 “log.conf” 那行的文件名。更常见的情况是利用 Vim 的编辑功能做批量操作。比如，你需要把文件名变成小写，可以选中要修改的部分然后使用 `gu` 命令。又比如，如果你的文件里有大量的编号，你希望对编号进行增减的操作，也可以利用 Vim 的 `<C-A>` 和 `<C-X>` 来对编号进行加减操作。当然，这时你可能需要使用可视模式的列选择（`<C-V>`）。

你还有可能需要注意一下选项 `nrformats`，因为如果其中含有 `octal` 的话，Vim 会把 `0` 打头的数字序列当成八进制来处理。还好，如果你按照我目前给出的方式来设置 `vimrc` 配

置文件的话，缺省里面不含 octal，只会对 0b 和 0x 打头的数字做特别处理，这就不会跟普通的十进制数字编号有任何冲突了。

Undowarning

Vim 里有跨会话撤销修改的功能，这当然是它的强大的特色功能。不过，有时候也许你会发现，不小心多按了几下 u，你就退回到打开文件之前的版本去了。我想，这很有可能不是你想要的行为吧？如果你，像我一样，希望能够无限制地进行编辑撤销，同时还想在退回打开文件的状态之前能有一个提醒，那 undowarning.vim 可能就是你想要的。

这个插件不支持用包管理器自动安装。你需要自行下载 [@undowarning.vim](#)，并把它放到你的 Vim 配置目录的 plugin 子目录下。下图是一个运行中的示例：

```
if has('autocmd')
  " 为了可以重新执行 vimrc，开头先清除当前组的自动命令
  au!
endif

if has('gui_running')
  " 请在下面自行定制你需要的字体
  set guifont=DejaVu\ Sans\ Mono\ 10
  set guifontwide=Noto\ Sans\ Mono\ CJK\ SC\ 11

  " 不延迟加载菜单（需要放在下面的 source 语句之前）
  let do_syntax_sel_menu = 1
  let do_no_lazyload_menus = 1
endif

set enc=utf-8
source $VIMRUNTIME/vimrc_example.vim

set fileencodings=ucs-bom,utf-8,gb18030,latin1
set scrolloff=2
set nobackup
if has('persistent_undo')
```

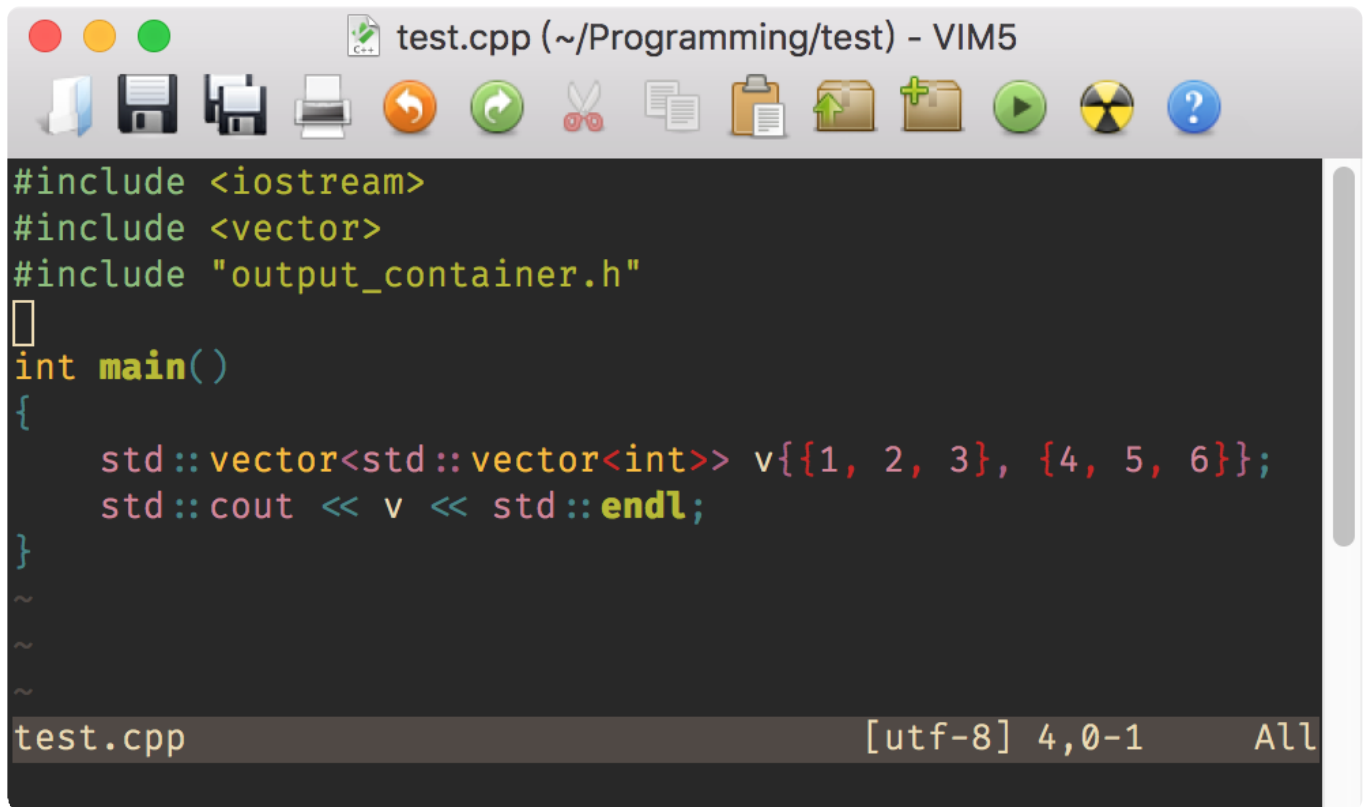
22,0-1

Top

Undowarning 的效果

Rainbow

代码中括号多了，有时候眼睛就有点看不过来，需要有个更好的颜色提示。因此，就有了很多彩虹效果的 Vim 插件。在这些插件中，我最喜欢的是 [@frazrepo/vim-rainbow](#)，它最妙的地方是，居然能把 C++ 代码中的尖括号也进行加亮，还能基本不会在出现小于、大于、流输入输出时进行错误的加亮。效果见下图：



```
#include <iostream>
#include <vector>
#include "output_container.h"

int main()
{
    std::vector<std::vector<int>> v{{1, 2, 3}, {4, 5, 6}};
    std::cout << v << std::endl;
}
```

彩虹括号的效果

效果默认不自动启用，可以用 `:RainbowToggle` 命令来切换，或用 `:RainbowLoad` 命令来加载。我觉得在括号多的时候按需启用挺好，推荐！

Auto-pairs


代码中永远有着大量成双成对的符号，输入一个，就自动出来另一个，会是一个非常有用的功能。但这样的功能，也需要处理一些特殊情况，比如，如果程序员输入了一对符号 `()`，结果千万不能是 `()()`。在很多现代的编辑器上，这已经是个标准功能了，但 Vim 一直没有类似的功能。

实际上，Vim 里已经有插件 [jiangmiao/auto-pairs](#) 支持了这个功能，并解决了大部分边角情况。我觉得可以推荐给大家。

这个插件，要不要推荐我还是犹豫了一下的。我一开始对它相当满意，但后来我发现仍然有一些边角情况处理不好，使用它就会导致无法编辑出我需要的效果。所以我又把它卸载了。但再后来，我又觉得，毕竟瑕不掩瑜，而且有问题时把它禁用不就得了！

所以，它的配置项我也只需要提一个，就是禁用的键映射。这个键映射由全局变量 `g:AutoPairsShortcutToggle` 控制，默认值是 `<M-p>`。如果你在 Mac 上，这个键多半就不工作了，除非你只在 Mac 终端里使用 Vim，并且在终端应用里配置了“将 Option

键用作 Meta 键”。对于大部分 Mac 用户，你需要进行类似下面的配置（因为 Mac 上按 Option-P 会产生 “π”）：

 复制代码


```
1 let g:AutoPairsShortcutToggle = 'π'
```

其他内容就请自行查看它的帮助文件了。

Largefile

如果你经常打开很大的日志文件，那 Vim 的一些自动功能可能不仅帮不了什么忙，反而会拖慢你的编辑速度。有一个 Vim 插件能在文件较大时自动关闭事件处理、撤销、语法加亮等功能，用来换取更快的处理速度和更短的响应时间。这个插件就是 [vim-scripts/LargeFile](#)。

这个插件的功能比较简单，唯一需要配置的就是多大算大。你只需要在 vimrc 配置文件中把你对大文件的阈值（以 MB 为单位）写到 g:LargeFile 变量里即可。比如，如果你认为超过 100MB 算大文件，那我们这样写就可以了：

 复制代码

```
1 let g:LargeFile = 100
```

Markdown Preview

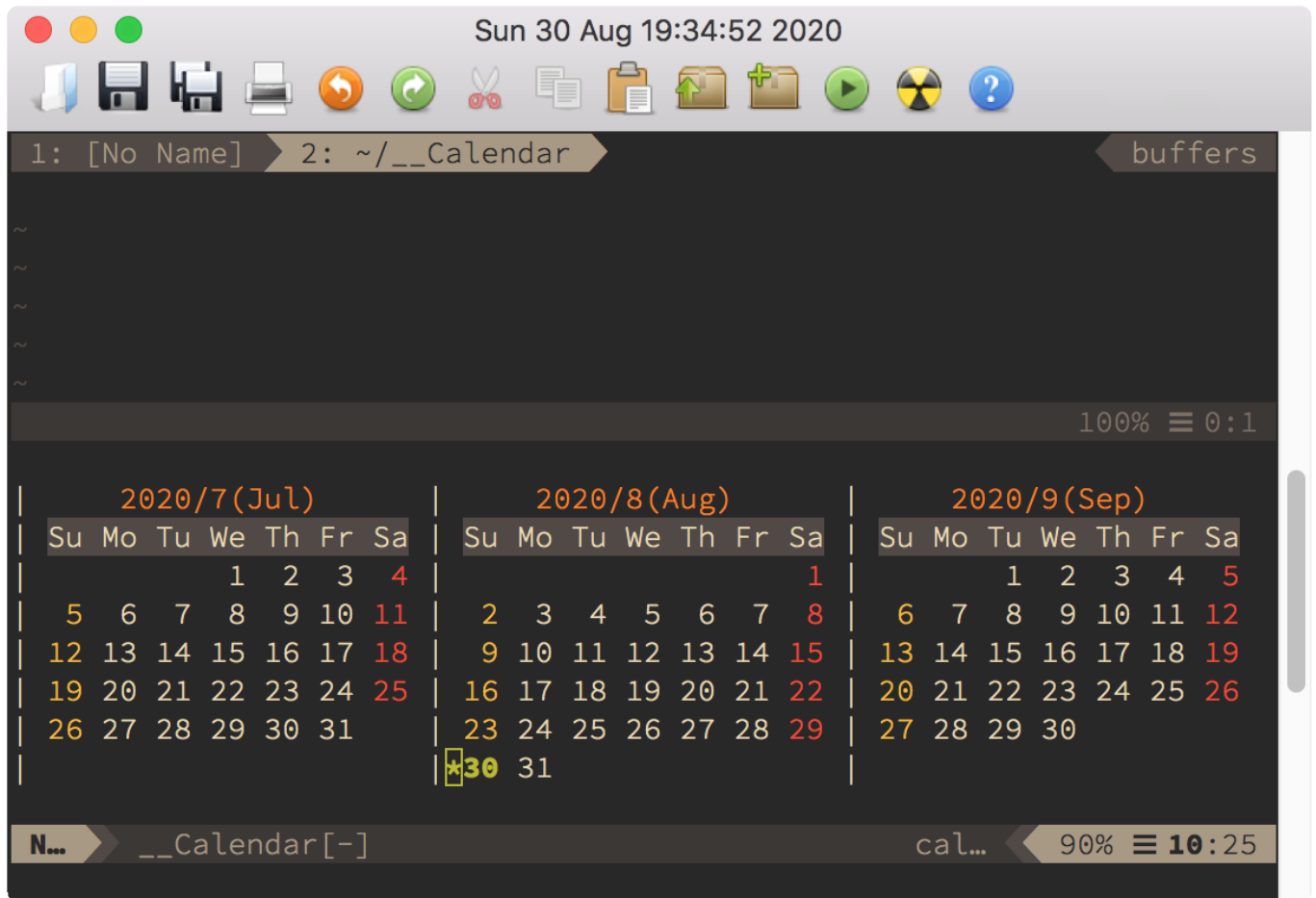
你如果像我一样常常写 Markdown 的话，你应该会喜欢 [Markdown Preview](#) 这个插件。Markdown 本来最适用的场景就是浏览器，纯文本的 Vim 只能编辑，没有好的预览终究是很不足的。Markdown Preview 解决了这个问题，让你在编辑的同时，可以在浏览器里看到实际的渲染效果。更令我吃惊的是，这个预览是完全实时、同步的，无需存盘，而且预览页面随着光标在 Vim 里移动而跟着滚动，效果相当酷。你可以直接到 Markdown Preview 的主页上看一下官方的示意图，我就不在这里放动图了。

这个插件唯一需要特别注意的是，你不能直接把 iamcco/markdown-preview.nvim 放到你的包管理器里了事。原因是它里面包含了需要编译的前端组件，需要下载或编译才行。在它的主页上描述了在不同包管理器里的安装方式，你只要跟着照做就行。

它的配置在主页上也有列表，但默认设置就已经完全可用了。如果有需求的话，你可以修改其中部分值，如 `g:mkdp_browser` 可以用来设定你希望打开页面的浏览器（我目前设的是 `'firefox'`）。

Calendar

🔗 **Calendar** 是一个很简单的显示日历的 Vim 插件，在包管理器里的名字是 `matttn/calendar-vim`。它的功能应该就不需要解释了，效果可以直接查看下图。



:CalendarH 产生的水平日历

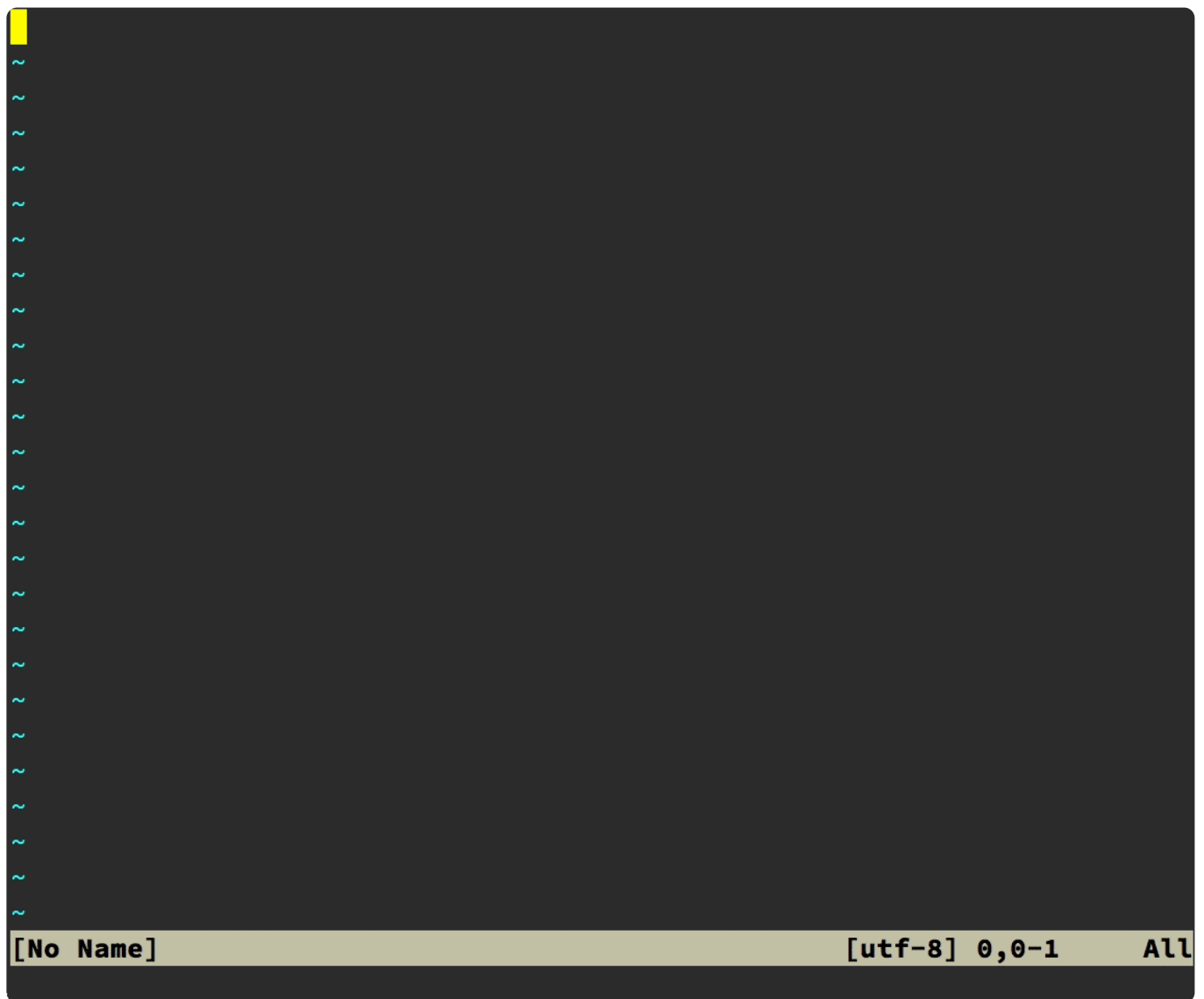
这个插件支持一些不同的样式和分割方式，上图中就是用 `:CalendarH`（或正常模式命令 `caL`）进行的横向分割，同时横向显示日历，你也可以用 `:Calendar`（或正常模式命令 `cal`）进行纵向分割和日历显示。此外，它还支持其他一些命令和组合。鉴于这个插件的帮助文件也不长，如果你对相关功能有兴趣的话，就请你自己去看一下了。

Matrix

上面介绍的插件，不管对你有没有用，都可以说是“有用”的。插件也不一定要做有用的事，我的机器一直装着下面这个“没用”的插件，🔗 [uguu-org/vim-matrix-](https://github.com/uguu-org/vim-matrix-)

screensaver。

它的效果不用解释，直接看下面的动图就好。



Matrix!

Killersheep

Vim 脚本不仅可以做没用的事情，还可以更进一步做娱乐的事情。比如，Vim 的作者 Bram 亲自操刀写了这个“愚蠢的游戏”，[🔗vim/killersheep](https://vim.kitware.com/vim/doc/killersheep.html)。

这当然不是一个真正非常好玩的游戏，不过我也玩通关了。你不妨也试试？小提示：屏幕拉高点，按键重复速度快点，重复前延迟短一点，这样更有助于你打好这个游戏。

当然，这个插件的本来意义就不是一个游戏，而是要演示 Vim 8.2 的下列新功能：

带颜色和掩码的弹出窗口

用于高亮文本的文本属性

声音

如果你想用这些功能的话，就可以去看看这个插件的源码。这也算一种寓教于乐吧？

技巧

行过滤

在编辑日志等类型的文本时，我们往往想过滤出我们感兴趣的内容。这时，我们可以用正则表达式，但使用 `:s` 命令并不是一种最高效的方式。如果你感兴趣的每一行都可以跟某个正则表达式模式匹配（如日期、某关键字等），最高效的命令应该是：

```
1 :v/匹配模式/d
```

[复制代码](#)

稍微解释一下，`:v` 命令（可以查看帮助 [🔗:help :v](#)）可以用来找出不符合匹配模式的行（对比一下 `grep -v` 命令），然后执行后面的动作。所以上面的命令就是找出不满足匹配模式的行，然后执行删除（`d`）。

顺便说一下，如果你需要找出符合匹配模式的行，需要的命令是 `:g`（可以查看帮助 [🔗:help :g](#)）。

自动关闭最后一个 quickfix 窗口

我们已经讨论到很多功能会用到 quickfix 窗口。如果打开 quickfix 窗口后，你关闭了编辑的主窗口，那 quickfix 窗口可能就成为这个 Vim 会话里剩下的唯一窗口了，而这个窗口多半你完全不再需要。你会希望，此时 Vim 应该自动关闭这个窗口直接退出。这当然是个可以用程序自动化的事情，所以我们也应该这样做，用下面的脚本放到 `vimrc` 配置文件里就可以做到：

```
1 aug QFclose
2   au!
```

[复制代码](#)

```
3   au WinEnter *   if winnr('$') == 1 && &buftype == "quickfix"|q|endif
4   aug END
```

你只要查一下 `winnr` 函数的帮助 ([@:help winnr\(\)](#)) 就很容易理解了，代码的意思还是非常清楚的：如果窗口的数量是 1 并且缓冲区类型是 `quickfix` 的话，那就退出 Vim。为了确保重复执行这段代码没有问题，它有一个自己的自动命令组，并会在清除这个自动命令组的所有自动命令后在进入窗口 (`WinEnter`) 这个事件中进行上面的检查。

(本技巧来自这个 [Stack Overflow 回答](#)。)


Home 键的行为

对于大部分现代的编辑器，Home 键的行为通常是：

当光标处于本行第一个非空白字符上时，跳转到行首

否则，跳转到本行第一个非空白字符上

虽然 Vim 的行为不是这样，但如果你希望配置出这样的行为，也不麻烦，把下面的代码加入到你的 `vimrc` 配置文件即可：

 复制代码

```
1 function! GoToFirstNonBlankOrFirstColumn()
2   let cur_col = col('.')
3   normal! ^
4   if cur_col != 1 && cur_col == col('.')
5     normal! 0
6   endif
7   return ''
8 endfunction
9
10 noremap <silent> <Home> :call GoToFirstNonBlankOrFirstColumn()<CR>
11 inoremap <silent> <Home> <C-R>=GoToFirstNonBlankOrFirstColumn()<CR>
```

在这个代码里，`col('.')` 用来获取光标所在的列号，然后我们跳转到第一个非空白字符处 (^)，随后检查是不是我们不在第 1 列并且列号没有变化。如果是的话，说明第一个非空白字符不在行首并且当前光标已经在第一个非空白字符处了，那我们就跳转到行首去 (0)。

另外要注意，我们这儿使用了 `normal!` 而不是 `normal`。这两者的区别是，用了 `!` 的 `normal` 命令会忽略键映射，否则 `normal` 就跟正常按键一样了。为了防止其他地方定义了键映射导致行为变化，一般推荐 `normal!` 命令而不是 `normal` 命令。

我们这样就修改了正常模式和插入模式下的 Home 键的行为。目前，在可视模式下这种方式不适用，你仍然只能手工选择合适的 `o` 或 `^` 命令。

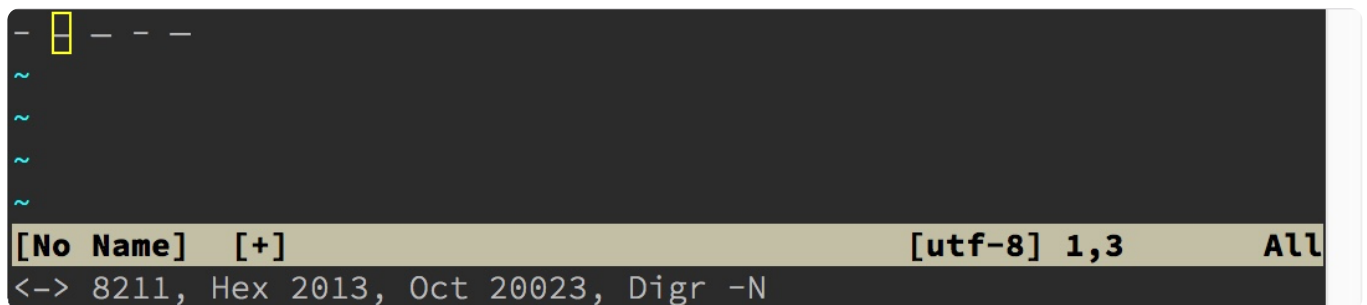
查看光标下字符的内码

有很多字符是很相似的，在 Vim 使用的等宽字体中尤其如此。比如，光看字形，你能区分下面这些字符分别是什么吗？

- - - - -

这些字符看起来虽然相似，但它们的意义是完全不同的。特别是在源代码中，如果你一不小心混入了一个相似的字符（字处理器有时候会自动替换一些 ASCII 字符，造成这种问题），代码运行就会出错了。这时，`ga` 命令就可以帮上忙。

下图展示了 `ga` 命令的一次执行结果：



执行 `ga` 命令的结果


我们可以看到这个字符实际上是 Unicode 字符 U+2013（十进制和八进制数值分别是 8211 和 20023），短破折号。这个字符在很多键盘上不能直接打出来，因而 Vim 提供了二合字母（digraph），可以用 `<C-K>-N` 的方式来输入。

为什么是 42？

好吧，最后这个不是技巧。你可能会奇怪：为什么在讲编程的很多代码里会出现 42 这个数字呢？那其实是因为一个科幻小说的“梗”。事实上，我在这个课程里也用到了 42。要想

获取进一步的信息，请在 Vim 里输入：

```
1 :help 42
```

 复制代码

内容小结

本讲的内容比较零散，我为你介绍了一些我认为值得介绍的插件和技巧，每一则篇幅都较短，我也就不再正式总结了。

我分享的这些内容大部分是有用的，但也有一部分可以认为基本是“无用”的。生活里如果要求每一件事情都有用，岂不是会变得非常无趣？对我来说，编程和编程工具也是如此。

课后练习

今天介绍的插件和技巧，虽说不能算必需，也是非常有意义的。强烈建议你挑至少三个来安装 / 实验一下。如有任何问题或意见，欢迎留言和我交流。

我是吴咏炜，我们下一讲再见！

提建议

更多课程推荐

程序员的数学基础课

在实战中重新理解数学

黄申

LinkedIn 资深数据科学家



涨价倒计时 🕒

今日秒杀 **¥79**, 9月11日涨价至 **¥129**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 拓展4 | 插件样例分析：自己动手改进插件

下一篇 结束语 | Vim 森林探秘，一切才刚刚开始

精选留言 (2)

写留言



我来也

2020-09-07

老师连压箱底的东西都拿出来了，我们赚到了哈。

Syntastic 和 ALE

前面这个插件已经被我屏蔽了，后面这个还保留着在。
当时是安装了coc.nvim后，觉得功能有些重复了。...

展开 ∨

作者回复: 最不可或缺的插件，你都会了，所以就得翻翻箱底啦。

Auto-pairs 我碰到的问题和你一样。也是 Vim 挑注释符号实在有点诡异啊。除了 Vim 脚本，基本上没遇到过奇怪问题。



2



湟水鱼儿

2020-09-08

没想到vim里还有搭车客中的彩蛋

展开

作者回复: 还可以 :help holy-grail

以前还曾经用 help! 能出来 Don't panic.

