目　录

致谢

当前文档 《Vim 实操教程（Learn Vim）》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-02-26。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/learn-vim

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

介绍

Vim 实操教程（Learn Vim）

以我个人学习 vim 的经验来看，通过看文档或看其他人操作其实是很难真正学会 vim 的，你必须在实际
应用中，进入真实场景才能逐渐熟悉并掌握相关命令。

因此，为了同时满足学习和操作的需求，项目中的文件都采用了 Markdown 格式，即可以当作说明文档来
阅读，也可以用 vim 打开文件进行实际操作（建议采用后者）。

? 现在也可以进入 **Telegram Group** 进行提问或讨论。

如何使用

1. 进入控制台
2. clone 项目到本地
3. 进入项目文件夹
4. 执行命令 `vim file-one.md`

排版规范

```
1. ## 大标题表示一大类
2.
```

```
### 小标题表示该大类下的小分类

没有任何格式的文本为正常描述，只有阅读功能。

> 嵌入到引用块中的文本为操作指示，你可以按照里面提到的内容进行操作
>
> 同时操作符或命令会包含在类似 `:w` 的符号中

命令中形如 f<X> 中的 < 和 > 不需要打出来，<X> 代表一个变量，即你可以打 fa 或 fb 亦或 fC

_注意：命令区分大小写（需要注意的事项会出现在当前行这样的符号中）_
```

导航

基础操作

1. 光标的移动
2. 打开文件、查找内容
3. 文档的修改与保存
4. 一些小技巧
5. 分屏与标签页
6. 块操作
7. vim 中的宏

附加内容

1. vim 插件
2. 插件推荐
    i. NERDTree
   ii. EasyAlign
  iii. Airline & Themes

Tips

- 教程中会有下一章或相关章节的导航，定位到文件名执行 `gf`（goto file）就可以打开相关文件
- 你可以随时打开相关章节查看，然后用 `:bp` 回到之前的文件（该命令会在第二章中讲到）
- 当你用 `:q` 或 `:qa` 退出教程时可能会收到文件未保存的错误提醒，试试在命令后面加上 `!`

TODO

- [ ] vimdiff
- [ ] more settings
- [ ] other mode
- [x] plugins

推荐几个 vim 配置方案

- dofy / **7th-vim**
- kepbod / **ivim**
- chxuan / **vimplus**
- SpaceVim / **SpaceVim**

推荐另外几个出色的 vim 教程

- 控制台运行 `vimtutor` 这是 vim 官方实操教程
- 简明 Vim 练级攻略 很不错的入门教程
- vim galore 更新频繁，vim 进阶必读
- 每日一Vim 共 30 篇，内容比较全

Cheatsheets

- http://www.nathael.org/Data/vi-vim-cheat-sheet.svg
- http://people.csail.mit.edu/vgod/vim/vim-cheat-sheet-en.png
- https://cdn.shopify.com/s/files/1/0165/4168/files/preview.png
- http://michael.peopleofhonoronly.com/vim/vim_cheat_sheet_for_programmers_screen.png

如果觉得本教程对你有帮助，可以给作者买杯咖啡鼓励一下 ☕

微信	支付宝	Bitcoin



再次感谢您的关注！如果爱，请分享。爱极客公园，爱 **VIM**！

来源

> *https://github.com/dofy/learn-vim*

光标的移动

光标的移动

欢迎进入第一章，这一章将学习简单的光标移动操作。

如果你已经有了一定基础，这部分可以略过，直接 `G` 到文档尾部按照操作进入下一章。

移动光标

单位级

- `h` 向左一字符
- `j` 下一行
- `k` 上一行
- `l` 向右一字符

单词级

- `w` or `W` 向右移动到下一单词开头
- `e` or `E` 向右移动到单词结尾
- `b` or `B` 向左移动到单词开头

注意：所有小写单词都是以分词符作为单词界限，大写字母以空格作为界限

> 在下面字符块中感受一下各种移动吧！

```
This project's GitHub url is https://github.com/dofy/learn-vim
Please clone it to your local folder and open the first file which is
named file-one.md via following command "vim file-one.md"
and welcome to http://geekpark.net :)
```

块级

- `gg` 到文档第一行
- `G` 到文档最后一行
- `0` 到行首（第 1 列）
- `^` 到第一个非空白字符
- `$` 到行尾
- `Ctrl-d` 向下移动半页
- `Ctrl-u` 向上移动半页
- `Ctrl-f` 向下移动一页
- `Ctrl-b` 向上移动一页
- `:<N>` or `<N>gg` 跳转到第 N 行
- `:+<N>` or `<N>j` 向下跳 N 行
- `:-<N>` or `<N>k` 向上跳 N 行

注意：所有命令前都可以加一个数字 *N*，表示对后面的命令执行 *N* 次，例如你想向下移动 *3* 行，除了
可以用 `:+3` 之外，还可以用 `3j` 来实现同样的效果。另外，上面实际上有两种命令：一种是键入后
立即执行的，比如 `gg`；还有一种是先输入 `:` 的（后面还会出现先按 `/` 的），这类命令需要在
输入完成后按回车执行，后面的教程中也是一样。

> 现在你可以在当前文件中畅游了，当你熟悉了各种移动操作后就可以通过 `G` 定位到当前文档到最后一行，按照提示进入下一章了。
>
> 将光标定位到后面文件名的任意位置上，直接敲键盘 `gf` 进入第二章。

打开文件、查找内容

打开文件、查找内容

打开文件

哈，现在你已经在无意间学会了一种在 vim 中打开文件的方式，虽然这种方式并不是最常用的，但却是最
直接的，尤其是当你的代码中 include 了某文件时，下面介绍另外两种常用的打开方式。

在 vim 中打开文件

- `:e <filename>` 打开名为 filename 的文件，若文件不存在则创建之
- `:Ex` 在 vim 中打开目录树，光标选中后回车打开对应文件（提示：`-` 进入上级目录）

查找

文档内查找

- `*` 向后查找光标当前所在单词
- `#` 向前查找光标当前所在单词
- `/<search>` 向后查找指定字符串
- `?<search>` 向前查找指定字符串
- `n` 继续查找下一个
- `N` 继续查找上一个

注意： `n` 和 `N` 是有方向性的，若你之前通过 `*` 查找，则 `n` 会继续向文档尾方向查找， `N`
向文档首方向；反之，若你通过 `#` 查找，则 `n` 指向文档首， `N` 指向文档尾

行内查找

- `f<X>` 当前行内向行尾方向查找并定位到字符 `X`
- `t<X>` 当前行内向行尾方向查找并定位到字符 `X` 之前
- `F<X>` 当前行内向行首方向查找并定位到字符 `X`
- `T<X>` 当前行内向行首方向查找并定位到字符 `X` 之后
- `;` 继续向当前方向查找下一个字符
- `,` 向当前相反方向查找下一个字符

> 当前文档中有几个 *“vim”* 单词，你可以尝试用 `*` 和 `#` 进行查找并感受 `n` 和 `N` 的方向性。
>
> 上面的 *“注意”* 中有几个字符 *“n”*，你可以在那试试行内查找并感受下 `;` 和 `,` 的方向性。

匹配查找

vim 中可以使用 `%` 对 `(` 和 `)`，`[` 和 `]`，`{` 和 `}` 进行匹配查找，当光标位于其中一个
符号上时，按下 `%`，光标会跳到与之匹配的另外一个符号上。

> 在下列文本中的 `(){[]}` 字符上按 `%` 看看效果，连续多按几次。

```
(function(win, doc) {
    var n = ((1 + 2) * (3 + 4)) / 7;
    var a = [1, 2, 3, 4, 5, 6, 7];
    var f = function(b) {
        if(b) {
            return false;
        } else {
            return true;
        }
    };
})(window, document);
```

下一章将介绍文档的修改，在这之前先简单介绍一下 vim 的 buffer，简单理解 buffer 就是当前 vim session 的文件历史记录。

> 现在你的 *buffer* 中应该已经有两个文件了，你可以用 `:buffers` 或 `:ls` 命令查看，看到 *buffer* 列表了吧，大概是这个样子的：

```
:ls
  1 #h   "file-one.md"                line 47
  2 %a   "file-two.md"                line 1
Press ENTER or type command to continue
```

> 接下来你可以尝试通过以下命令在文件缓存中进行跳转了

> - `:bn` 打开缓存中下一个文件
> - `:bp` 打开缓存中上一个文件
> - `:b<N>` 打开缓存中第 *N* 个文件
>
> 你也可以使用 `:bdelete<N>` 来删除所要关闭的缓冲区，缩写 `:bd<N>` 。
>
> 当然你也可以使用 `:Ex` 命令，选择 *file-three.md* 并打开，进入第三章。

文档的修改与保存

文档的修改与保存

修改文档

你现在已经学会了控制光标、打开文件、切换文件、并在文件中查找内容，这些操作都是在 vim 的 normal
模式下进行的。现在，是时候进入 vim 的另外一种模式 —— insert 模式，学习一下如何修改文件了。

插入

- `i` 当前字符前插入
- `a` 当前字符后插入
- `I` 行首插入
- `A` 行尾插入
- `o` 在下一行插入
- `O` 在上一行插入

注意：以上任何一个命令都会使 *vim* 进入 *insert* 模式，进入该模式后光标会发生变化，这时输入的
文字会直接出现在文档中，按 *Esc* 键或 *Ctrl-[* 或 *Ctrl-C* 退出 *insert* 模式。

删除(并保存到 vim 剪贴板)

- `x` 删除当前字符，相当于 insert 模式下的 `Delete`
- `X` 删除前一个字符，相当于 insert 模式下的 `Backspace`

- `dd` 删除当前行，并将删除的内容保存到 vim 剪贴板
- `d<X>` 删除指定内容并保存到 vim 剪贴板
- `c<X>` 删除指定内容并保存到 vim 剪贴板，同时进入 insert 模式

说明： `<X>` 部分是对操作内容的描述，如果要删除一个单词，就输入 `dw` 或者 `de`，要复制当前
位置到行尾的内容，就输入 `y$`，要删除后面 *3* 个字符并插入，就输入 `c3l` 诸如此类。

复制

- `yy` 复制当前行到 vim 剪贴板
- `y<X>` 复制指定内容到 vim 剪贴板

粘贴

- `p` 在当前位置后粘贴
- `P` 在当前位置前粘贴

合并

- `J` 将当前行与下一行合并

> 尝试在下面的文本中进行复制粘贴练习

```
1. 删除这一行
2. 粘贴到这一行下面
3. 剪切 ABC 并把它粘贴到 XYZ 前面，使这部分内容看起来像
4. 剪切 并把它粘贴到 ABC XYZ 前面。
```

替换

- `r<X>` 将当前字符替换为 X
- `gu<X>` 将指定的文本转换为小写
- `gU<X>` 将指定的文本转换为大写
- `:%s/<search>/<replace>/` 查找 search 内容并替换为 replace 内容

> 尝试修改下列文本的大小写

```
1. Change this line to UPPERCASE, THEN TO lowercase.
```

> 还有个更好玩的命令 `g~<X>`，先将光标定位到上面那行文本，执行 `0g~$` 看看发生了什么。

撤销、重做

- `u` 撤销
- `Ctrl-r` 重做

保存文件

- `:w` 保存当前文件
- `:wa` 保存全部文件
- `:wq` or `ZZ` 保存并退出
- `:q!` or `ZQ` 强制退出，不保存
- `:saveas <new filename>` 文件另存为
- `:w <new filename>` 文件另存一份名为 `<new filename>` 的副本并继续编辑原文件

> 你可以试着把当前（也许已经改得面目全非的）文件另存一份，然后继续下一章的学习。

一些小技巧

一些小技巧

简单设置 vim

“工欲善其事，必先利其器”。尽管 vim 非常强大，但默认配置的 vim 看起来还是比较朴素的，为了适合
我们的开发需求，要对 vim 进行一些简单的配置。

- `:set number` 显示行号
- `:set relativenumber` 显示相对行号（这个非常重要，慢慢体会）
- `:set hlsearch` 搜索结果高亮
- `:set autoindent` 自动缩进
- `:set smartindent` 智能缩进
- `:set tabstop=4` 设置 tab 制表符所占宽度为 4
- `:set softtabstop=4` 设置按 `tab` 时缩进的宽度为 4
- `:set shiftwidth=4` 设置自动缩进宽度为 4
- `:set expandtab` 缩进时将 tab 制表符转换为空格
- `:filetype on` 开启文件类型检测
- `:syntax on` 开启语法高亮

这里列出的是命令，你可以通过在 vim 中输入进行设置，但这种方式设置的参数只在本次关闭 vim 前生效，
如果你退出 vim 再打开，之前的设置就失效了。

若要永久生效，需要修改 vim 的一个自动配置文件，一般文件路径是
`/home/<user>/.vimrc`（Linux
系统）或 `/Users/<user>/.vimrc`（Mac OS 系统）

如果没有就新建一个，以 Mac OS 系统为例：

> 在控制台执行如下命令，每行结尾记得回车

```
1. cd ~
2. vim .vimrc
```

> 现在你已经在 *vim* 中打开了你的 *vim* 专属配置文件，将上面提到的配置复制到你的文件中，记得要删除每行开头的 `:`
>
> 修改完成执行 `:wq` 或者 `ZZ` 保存退出，再次进入 *vim* 时，你的这些配置就已经生效了
>
> 当然，机智如我也为你准备好了一份 *vimrc* 样本文件，你可以在控制台执行 `cp vimrc.vim ~/.vimrc` 直接使用，再次启动 *vim* 或在 *vim* 中执行 `:source ~/.vimrc` 你的配置就应该生效了。

[AD] 当然你也可以在我维护的另外一个项目 *7th-vim* 中找到一个更为完整的配置方案。

清除搜索高亮

前面提到的配置中，有一项是高亮全部搜索结果 `:set hlsearch` ，其作用是当你执行 `/` 、 `?` 、 `*` 或 `#` 搜索后高亮所有匹配结果。

> 如果你已经设置了这个选项，尝试执行 `/set`

看到效果了吧，搜索结果一目了然，但这有时候也是一种困扰，因为知道搜索结果后高亮就没用了，但高亮
本人并不这样认为，它会一直高亮下去，直到你用 `:set nohlsearch` 将其关闭。

但这样需要就打开，不需要就关闭也不是个办法，有没有更好的解决方案呢？当然！请看下面的终极答案：

> 再搜一个不存在的字符串

通常我用来清除搜索高亮的命令是 `/lfw` ，一是因为 `lfw` 这个组合一般不会出现（不适用于本文档…），二是这三个字母的组合按起来比较舒服，手指基本不需要怎么移动（你感受一下）。

重复上一次命令

vim 有一个特殊的命令 `.` ，你可以用它重复执行上一个命令。

> 按下面的说明进行操作

```
1. 按 dd 删除本行
2. 按 . 重复删除操作
3. 2. 再删除两行
4. 这行也没了
5. p 把刚才删掉的粘回来
6. 3. 又多出 6 行
```

缩进

- `>>` 向右缩进当前行
- `<<` 向左缩进当前行

> 在这一行上依次按 `3>>` ， `<<` 和 `<G` 看看效果
>
> 打酱油行

自动排版

- `==` 自动排版当前行
- `gg=G` 当前文档全文自动排版
- `<N>==` 对从当前行开始的 N 行进行自动排版
- `=<N>j` 对当前行以及向下 N 行进行自动排版
- `=<N>k` 对当前行以及向上 N 行进行自动排版

> 另外，还可以利用第二章中提到的匹配搜索对代码块进行批量排版，尝试用 `gf` 命令打开 *file-four-demo.js* 按照里面的说明进行操作

如果智能缩进设置生效了，执行后会看到如第二章中一样的排版效果。

下一章将介绍分屏和标签页。

分屏与标签页

分屏与标签页

窗口分屏

工作中经常会遇到这种情况，就是需要参照其他文档编辑当前文档（场景：翻译），或者从另外一个文档
copy 代码到当前文档（场景：复制 html 元素类名到 css 文档），这时候就是你最需要分屏的时候。

分屏方式

- `:split` 缩写 `:sp` or `Ctrl-w s` 上下分屏
- `:vsplit` 缩写 `:vs` or `Ctrl-w v` 左右分屏
- `:diffsplit` 缩写 `:diffs` diff 模式打开一个分屏，后面可以加上 {filename}

窗口跳转

- `Ctrl-w w` 激活下一个窗口
- `Ctrl-w j` 激活下方窗口
- `Ctrl-w k` 激活上方窗口
- `Ctrl-w h` 激活左侧窗口
- `Ctrl-w l` 激活右侧窗口

屏幕缩放

- `Ctrl-w =` 平均窗口尺寸
- `Ctrl-w +` 增加高度

- `Ctrl-w -` 缩减高度
- `Ctrl-w _` 最大高度
- `Ctrl-w >` 增加宽度
- `Ctrl-w <` 缩减宽度
- `Ctrl-w |` 最大宽度

> 实践！实践！实践！

标签页

第二章中提到过的 buffer 和刚刚讲到的分屏操作都很适合在少量文件之间进行切换，文件超过 3 个我觉得就不方便了，而标签页则更适合多文件之间的切换。

创建标签页

- `:tabnew` or `:tabedit` 缩写 `:tabe` 打开新标签页
- `Ctrl-w gf` 在新标签页中打开当前光标所在位置的文件名

注意：`:tabnew` 和 `:tabedit` 后面都可以跟一个 <空格><文件名> 用以在新标签页中打开指定文件，还可以在 `:` 后面加一个数字，指出新标签页在列表中的位置（从 *0* 开始）。

切换标签页

- `gt` or `:tabnext` 缩写 `:tabn` 下一个标签页（最后一个会循环到第一个）
- `gT` or `:tabprevious` 缩写 `:tabp` 上一个标签页（第一个会循环到最后一个）
- `:tabrewind` 缩写 `:tabr` or `:tabfirst` 缩写 `:tabfir` 到第一个
- `:tablast` 缩写 `:tabl` 到最后一个标签页

关闭标签页

- `:tabclose` 缩写 `:tabc` 关闭当前标签页
- `:-tabc` 关闭上一个标签页
- `:+tabc` 关闭下一个标签页
- `:tabonly` 缩写 `:tabo` 关闭其他标签页

下一章将介绍块操作。

块操作

块操作

我们经常会遇到这种情况：某处有一个多行文本，我们要把他复制到代码中用来初始化一个数组。 大部分
时候我们会这么做：

- 写好数组声明；
- 把内容复制到中括号内（大概长成下面那段文本的样子）
- 然后行首加 `'` 行尾加 `',`，重复直到最后一行（想象一下这段文本有50行）

> 有了 *vim* 块操作就不用这么麻烦了，按 `014gg`，然后跟着选中那一行的指示操作。

```
var myArray = [
Ctrl-v 进入块操作，$ 到行尾，j 到下一行（做！）。
按 j 到下一行
下面还好多行，干脆来个 4j 多跳几行
http://www.geekpark.net
http://www.geekpark.net
以后看好行号再跳！现在按 A 插入，然后输入 <单引号><逗号><Esc> 完成第一步。
// Oops... 跳多了，没事，按 k 回到上一行
];
```

> 现在已经完成了第一步，还需要补前面的 `'`，按 `14gg` 回到那一行，再操作一次，但是这次有三个
> 地方要变化一下：
>
> 1. 第一行按 `$` 时改按 `0`，因为这次要在行首插入；
> 2. 末行按 `A` 时改按 `I`，块操作中 `A` 是字符后插入， `I` 是字符前插入；
> 3. 最后按 `<单引号><Esc>`。
>
> 最后再做些收尾工作， `19gg$x` 删掉最后一行结尾处的 `,`，然后 `14gg7==` 把代码缩进一下。
>
> *Done!*

注意：选择行首行尾的操作也可以在选择完要处理的内容之后执行，即 `Ctrl-v jjj$',<Esc>`

接下来我们说说 vim 中的宏。

vim 中的宏

vim 中的宏

宏操作在 vim 中（甚至任何编辑器中）属于比较复杂的操作了，如果前面的内容都已经掌握了，那么你
已经可以算是一个 vim 高手了，所以，这位高手，我们不妨再来进阶一下吧。

还记得上一章中把文本转成数组的例子吧，我们还做同样的事，不过这次是用宏来操作。

> `12gg` 跳转到准备开始处理的起始行，按指示进行操作，先看效果后解释。

```
1. var myArray = [
2. 按 qa 开启宏录制，前方高能，连续按 I<单引号><Esc>A<单引号><逗号><Esc>jq7@a
3. 我也要
4. 我也要
5. 我也要
6. 我也要
7. 我也要
8. 我也要
9. 我也要
10. ];
```

OMG！发生了什么，有没有惊出一身冷汗，之前两次块操作的结果瞬间就完成了，最后再简单做些收尾工作，
去掉最后一行的逗号，集体缩进一下，搞定！

下面来解释一下刚才的操作：

- `q` 是开启录制宏，`a` 是给这次宏的录制过程一个存储位置，可以是 `0-9` 或 `a-z`；
- 然后 `I<单引号><Esc>A<单引号><逗号><Esc>j` 是你这次录制的整个宏的操作过程，意思就是行首插入单引号，行尾插入单引号和逗号，跳到下一行；
- 接下来的 `q` 是结束本次宏的录制；
- `@` 是唤起宏，`a` 是要唤起的宏的名字（存储位置），前面的 `7` 你应该明白吧，就是执行 7 次。

Tips: `@@` 再次唤起最后一次执行的宏。

日常工作中频繁用到的和不怎么用得上的在这七章中应该都已经涉及到了，如果 vim 中遇到了什么问题，

或者教程中遗漏了什么常规操作，欢迎在 `issues` 中提出来，我会尽我所能给予回答或完善到教程中。

再次感谢您的关注！如果爱，请分享。爱极客公园，爱 **VIM**！

vim 插件

vim 插件

虽然 vim 已经提供了非常强大的功能，但如果有几款好用的插件辅佐，更能让你的工作事半功倍。

安装插件

若要手动安装 vim 插件，需要完成如下步骤：

1. 创建 `.vim` 文件夹

```
cd ~
mkdir .vim
```

2. 在 `.vim` 文件夹中创建 `bundle` 文件夹

```
cd .vim
mkdir bundle
```

3. copy 或 clone 插件文件

```
cd bundle
git clone <repository-url>
```

注意：若插件没有 *git* 地址，也可以直接创建相关文件夹，并置一 `.vim` 文件于其中。

4. 修改 `runtimepath`

   通过修改 `runtimepath` 属性，可以让 vim 找到你要加载的插件，要查看 `runtimepath` 属性可用 `:set runtimepath` 命令

   启用新插件可在 `.vimrc` 中添加如下配置

```
set runtimepath^=~/.vim/bundle/<folder>/
" OR
set runtimepath^=~/.vim/bundle/<name>.vim
```

插件管理

当你的插件越来越多，就需要一个管理器来管理 vim 插件了，市面上比较流行的插件管理器有以下几款：

1. vim-plug
2. Vundle.vim

我个人比较喜欢 vim-plug，下面就简单介绍一下这款管理器。

vim-plug 介绍

项目地址

https://github.com/junegunn/vim-plug

安装

下载 plug.vim 文件，放入 `autoload` 文件夹中（一般该文件夹位于 `~/.vim/autoload/` ）。

Unix

```
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
    https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

Neovim

```
curl -fLo ~/.local/share/nvim/site/autoload/plug.vim --create-dirs \
    https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

Windows (PowerShell)

```
md ~\vimfiles\autoload
$uri = 'https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim'
(New-Object Net.WebClient).DownloadFile($uri,
$ExecutionContext.SessionState.Path.GetUnresolvedProviderPathFromPSPath("~\vimfiles\autoload\plug.
```

配置

添加 `vim-plug` 的配置到 `~/.vimrc` 中：

1. 配置以 `call plug#begin()` 开始
2. 插件列表，以 `Plug` 命令开头
3. 用 `call plug#end()` 结束，以初始化插件系统
    - 这将会自动开启 `filetype plugin indent on` 和 `syntax enable`，如果不希望这样，你可以在该配置后重置你的设置，例如：`filetype indent off`，`syntax off`

Example

```
" 指定插件保存位置
call plug#begin('~/.vim/bundle')

" 注意要使用单引号

" 一些插件列表

" 如果插件在 GitHub 的地址是 https://github.com/junegunn/vim-easy-align
" 可以缩写成下面这样
Plug 'junegunn/vim-easy-align'

" 或者直接给定插件 git 地址
Plug 'https://github.com/junegunn/vim-github-dashboard.git'

" 多个 `Plug` 命令可以写在一行，用 `|` 符号分割
Plug 'SirVer/ultisnips' | Plug 'honza/vim-snippets'

" 更多配置详情查看官网介绍

" 初始化插件系统
call plug#end()
```

> 重启 *vim* 或重载 `.vimrc` 文件并执行 `:PlugInstall` 安装配置好的插件
>
> 重载命令是 `:source ~/.vimrc`

常用命令

命令	说明
`PlugInstall [name ...] [#threads]`	安装插件
`PlugUpdate [name ...] [#threads]`	安装或升级插件
`PlugClean`	清理插件
`PlugUpgrade`	升级 vim-plug
`PlugStatus`	查看已安装插件的状态

注意：更多命令查看官网介绍

去哪里找插件

1. GitHub 上有很丰富的 vim 插件资源，可以通过 `vim plug` 关键字进行搜索查看所有相关资源
2. Vim Scripts vim 官方提供的脚本集合，注意里面除了插件，还有很多 vim 脚本
3. Vim Awesome vim 插件推荐
4. :best of Vim 常用插件推荐
5. 本教程也将在 插件推荐 中不定期更新一些优秀插件以及使用说明

插件推荐

插件推荐

- NERDTree

  NERDTree 是 vim 中的文件浏览插件，可以通过命令打开/关闭目录树并浏览/打开文件。

- EasyAlign

  EasyAlign 是一款用来做字符对齐的插件，简直是强迫症患者的福音。

- Airline & Themes

  Airline 可以在 vim 中显示更友好的状态栏，再配以漂亮的配色，工作更开心。

NERDTree

NERDTree

NERDTree 是 vim 中的文件浏览插件，可以通过命令打开/关闭目录树并浏览/打开文件。

项目地址

https://github.com/scrooloose/nerdtree

安装

> 修改 `.vimrc` 配置，在 `call plug#begin()` 后面添加如下配置

```
1. Plug 'scrooloose/nerdtree', { 'on': 'NERDTreeToggle' }
```

注意：*vim* 插件管理器的使用请参考*plugin.md*

> 运行 *vim* 并执行命令 `:PlugInstall`，可能会得到如下提示：

```
1. 1 Updated. Elapsed time: 6.008607 sec.
2. 2 [===]
3. 3
4. 4 - Finishing ... Done!
5. 5 - vim-github-dashboard: Already installed
6. 6 - vim-easy-align: Already installed
7. 7 - nerdtree: Resolving deltas: 100% (158/158), done.
```

使用

看到上面的提示说明插件安装成功，可以执行下面的命令来控制目录树了。

命令	说明
`:NERDTree`	打开目录树
`:NERDTreeClose`	关闭目录树
`:NERDTreeToggle`	打开/关闭目录树
`:NERDTreeFind`	打开目录树并定位到当前文件

Tips

绑定快捷键

在 `.vimrc` 中添加如下配置：

```
" NERDTree
map <C-n> :NERDTreeToggle<CR>
" map 是快捷键映射命令
" <C-n> 定义了快捷键，表示 Ctrl-n
" 后面是对应的命令以及回车键 <CR>
```

重载 `.vimrc` 文件后，就可以用 `<Ctrl-n>` 来打开或关闭目录树了。

显示隐藏文件

NERDTree 插件默认是不显示隐藏文件的，有两种方式可以查看隐藏文件：

1. 打开目录树后按 `shift-i` 显示隐藏文件，再次按下，关闭显示隐藏文件
2. 在 `.vimrc` 中添加设置 `let NERDTreeShowHidden=1` 可在打开时默认显示隐藏文件

EasyAlign

EasyAlign

EasyAlign 是一款用来做字符对齐的插件，简直是强迫症患者的福音。

项目地址

https://github.com/junegunn/vim-easy-align

安装

> 修改 `.vimrc` 配置，在 `call plug#begin()` 后面添加如下配置

```
Plug 'junegunn/vim-easy-align'
```

注意：*vim* 插件管理器的使用请参考*plugin.md*

> 运行 *vim* 并执行命令 `:PlugInstall`，可能会得到如下提示：

```
1 Updated. Elapsed time: 6.008607 sec.
2 [===]
3
4 - Finishing ... Done!
5 - vim-github-dashboard: Already installed
6 - nerdtree: Already installed
7 - vim-easy-align: Resolving deltas: 100% (136/136), done.
```

使用

配置

> 添加 `.vimrc` 配置如下:

```
" EasyAlign
xmap ga <Plug>(EasyAlign) " Visual 模式下快捷键
nmap ga <Plug>(EasyAlign) " Normal 模式下快捷键

" 配置一些自定义符号
let g:easy_align_delimiters = {
\ '>': { 'pattern': '>>\|=>\|>'  },
\ '/': {
\     'pattern':         '//\+\|/\*\|\*/',
\     'delimiter_align': 'l',
\     'ignore_groups':   ['!Comment'] },
\ ']': {
\     'pattern':       '[[\]]',
\     'left_margin':   0,
\     'right_margin':  0,
\     'stick_to_left': 0
\   },
\ ')': {
\     'pattern':       '[()]',
\     'left_margin':   0,
\     'right_margin':  0,
\     'stick_to_left': 0
\   },
\ 'd': {
\     'pattern':      ' \(\S\+\s*[;=]\)\@=',
\     'left_margin':  0,
\     'right_margin': 0
\   }
\ }
```

实验

> 在下面的代码中尝试如下操作:

```
let a=1;// one
let  bcd=test=2; // two
let      longword=others= 'some content';// string
```

> 首先定位光标到上面代码中的任意一句，按 `gaip<Space>` ，应该会得到

> 如下结果

```
let a=1;// one
let bcd=test=2; // two
let longword=others= 'some content';// string
```

> 保持光标不动，按 `gaip=` ，应该会得到如下结果

```
let a        = 1;// one
let bcd      = test=2; // two
let longword = others= 'some content';// string
```

> 保持光标不动，按 `u` ，再按 `gaip*=` ，应该会得到如下结果

```
let a        = 1;// one
let bcd      = test   = 2; // two
let longword = others = 'some content';// string
```

下面解释一下按键的意思：

- `gaip`
  - `ga` 是开启 EasyAlign 的快捷键，我们在 `.vimrc` 配置文件中定义的
  - `ip` 是定义操作区域，可以用任意选择操作命令完成，`ip` 是选择当前段落，如果光标在代码第一行，则可以用 `2j` 代替 `ip`
  - 此时命令区域会出现 `:EasyAlign (_)` 字样，表示等待匹配输入
  - 按下 `<Space>` 是要将第一个空格前后对齐
- `gaip=`
  - 意义同上，`=` 即将`<等号>`前后对齐
- `gaip*=`
  - 为了看到更明显的效果，先用 `u` 撤销了前面的操作
  - `=` 前面的 `*` 是一个描述符，可以是数字，代表第几个等号，也可以是负数，代表倒数第几个，也可以是星号，代表所有。

> 如果确定修改好了上面提到的配置，你还可以尝试 `gaip/` 来对齐注释部分，执行后看上去会像下面这样

```
let a        = 1;                       // one
let bcd      = test   = 2;              // two
let longword = others = 'some content'; // string
```

匹配符说明

上面提到的 `<Space>` `=` 都是 EasyAlign 中定义好的特殊符号，用来表示某一类特征符

，具体可以参考下表：

按键	使用场景说明
`<Space>`	匹配空白符
`=`	包含等号的操作符（`=`，`==`，`!=`，`+=`，…）
`:`	应用于 JSON 或 YAML 格式
`.`	应用于多行点语法调用
`,`	应用于多行参数列表
`&`	对 LaTeX table 进行格式化，匹配 `&` 和 `\\`
`#`	应用于对 Ruby/Python 的注释的对齐
`<Bar>`	Markdown 表格

注意：上表中提到的 `<Bar>` 键即 `|` 键

> 尝试在上表中按 `gaip*|` 对 *markdown* 表格进行对齐

Tips

对齐方式

在等待输入匹配符，命令区出现 `:EasyAlign (_)` 字样时，可以按 `<Enter>` 键选择对齐方向，按一次切换为右对齐，显示 `:EasyAlign[R] (_)`，再按一次切换为居中对齐，显示为 `:EasyAlign[C] (_)`，再按切换回默认的左对齐，显示也恢复原样。

> 在上面的 *markdown* 表格中尝试一下 `gaip<Enter><Enter>*|`

正则匹配

在等待输入匹配符时，按快捷键 `<Ctrl-x>` 进入匹配模式，此时输入你需要的匹配特征正则即可匹配特殊组合。参考下面的文字：

```
1. Lorem<-ipsum
2. dolor  <--sit
3. amet<=  consectetur <- adipiscing
4. elit<~~ sed <~ do
5. eiusmod<-= tempor<=- incididunt
6. ut <== labore
```

> 尝试输入 `gaip*<Ctrl-x>`，此时进入正则匹配模式，继续输入 `<[-=~]*<Enter>` 会得到如下所示结果。

```
1. Lorem    <- ipsum
2. dolor   <-- sit
3. amet     <= consectetur  <- adipiscing
4. elit    <~~ sed          <~ do
5. eiusmod <-= tempor      <=- incididunt
6. ut      <== labore
```

更多参考

更多内容请参考项目页的介绍

Airline & Themes

Airline & Themes

Airline 可以在 vim 中显示更友好的状态栏，再配以漂亮的配色，工作更开心。

项目地址

- Airline https://github.com/vim-airline/vim-airline
- Airline Themes https://github.com/vim-airline/vim-airline-themes

安装

> 添加 `.vimrc` 配置

```
" 同时安装两个插件
Plug 'vim-airline/vim-airline' | Plug 'vim-airline/vim-airline-themes'
```

> *vim* 中执行 `:source ~/.vimrc` 重载配置文件
>
> 执行 `:PlugInstall` 安装 *Airline* 和 *Aairline Themes* 插件
>
> 在 `.vimrc` 中添加如下设置使 *Airline* 生效

```
set laststatus=2    " 始终显示状态栏

" Airline
let g:airline#extensions#tabline#enabled=1    " 开启 tab 栏
```

> 重载配置文件使之生效，如果没有效果可以尝试重启 *vim*

现在你应该会看到 vim 下方显示出了一条状态栏，显示信息一般会包含 当前模式，
当前文件，文件类型，文件编码以及当前行列信息。

因为我们同时安装了 Airline Themes 插件，所以可以通过如下方式设置 Airline 的样式。

> 执行 `:AirlineTheme simple` 应用 `simple` 样式，或者修改 `.vimrc` 文件，添加如
> 下配置：

```
let g:airline_theme='simple'
```

Airline Themes 包含了非常多的样式，具体可以参考该项目的
样式文件夹
其中所列的样式都可以应用。

更多细节和效果图可以参考项目地址。