

# 程序员的自我修养

Leo Hui

Published  
with GitBook



# 目錄

---

1. 介紹
2. 程序员基础知识
  - i. 字符编码
  - ii. 技术名词
  - iii. 语义化版本
  - iv. 命名规范
  - v. 书写文档
  - vi. 开源协议
  - vii. 目录结构
3. 数据结构与算法
  - i. XML和JSON
  - ii. 算法学习之路
4. 代码架构
  - i. 设计模式
    - i. 常用的Javascript设计模式
  - ii. 面向对象编程
    - i. 继承
    - ii. 多态
    - iii. 封装
  - iii. 面向接口编程
5. 代码评审
  - i. 六种量化你代码的方式
  - ii. 程序员必备的代码审查 (Code Review) 清单
6. 网络知识
7. 懂点设计
  - i. 佳作赏析
  - ii. 无缝平铺
  - iii. Sketch学习
  - iv. 设计与实现的平衡
8. 写点东西
  - i. 使用gitbook
9. 团队合作
  - i. 关于招聘
  - ii. 培训新人
  - iii. 领导能力
10. PM
  - i. 如何做一位與工程團隊合作良好的產品經理
11. 获取知识
  - i. MOOC
  - ii. 英语学习
  - iii. 设计学习
  - iv. 前端学习
  - v. iOS学习
  - vi. 游戏开发
12. 提升效率
  - i. 学会提问
  - ii. 善用搜索
  - iii. 学会写作
  - iv. 如何翻墙

- v. 时间管理
  - vi. 知识管理
  - vii. 文件管理
  - viii. 密码管理
  - ix. 健康生活
  - x. 制作视频
  - xi. 制作PPT
  - xii. 论音乐对效率的影响
13. SOHO
14. Hacker
- i. 保护隐私
15. 常用软件
- i. Windows
    - i. 硬件配置
    - ii. 系统安装
    - iii. 常用软件
  - ii. Mac
    - i. 通用设置
    - ii. 常用软件
      - i. iTerm2
      - ii. brew
      - iii. zsh
      - iv. 1Password
      - v. TextExpander
      - vi. 欧陆词典
      - vii. popClip
      - viii. manico
      - ix. 窗口管理器
      - x. BetterTouchTool
    - iii. 开发环境
    - iv. 快捷键设置
    - v. 常用终端命令
    - vi. dotfiles
  - iii. Android
    - i. 常用软件
    - ii. 如何登录美国区GooglePlay
  - iv. 开发工具
    - i. git
    - ii. EditorConfig
    - iii. node
    - iv. shadowsocks
    - v. ST3--Windows篇
    - vi. ST3--Mac篇
    - vii. gulp
    - viii. 字体的选择
    - ix. Emacs
  - v. Sketch
    - i. Sketch中文学习资料
  - vi. Trello
    - i. 使用Trello管理项目的经验
  - vii. git进阶
    - i. 15分钟学会使用Git和远程代码库
    - ii. GitHub秘籍

16. 附录

- i. 计算机科学与技术
- ii. 网站
- iii. 书籍

# A Programmer Prepares

---

Being a programmer...

## 程序员基础知识

---

想要成为一个合格的程序员，扎实的基础是必不可少的。

想要成为一个优秀的程序员，对计算机的发展需要有深入浅出的了解。

那么不如我们说说其中某些方面的前世今生。

## 字符编码

---

为什么要先说说字符，因为计算机只认识0和1。所以字符的重要性不言而喻。

## 参考资料

---

- [字符编码常识及问题解析](#)
- [人机交互之字符编码](#)

## 技术名词

---

技术上有很多的名词，有的是缩写，有的是创新的东西。了解他们，正确的说出来，会让别人觉得你很专业。

## 参考资料

---

- [IT 圈里有哪些经常被读错的词？](#)
- [关于那些开发语言中的常用名词，大家读的都正确吗？](#)



# 语义化版本

---

## 摘要

---

版本格式：主版本号.次版本号.修订号，版本号递增规则如下：

- 主版本号：当你做了不兼容的API 修改。
- 次版本号：当你做了向下兼容的功能性新增。
- 修订号：当你做了向下兼容的问题修正。

先行版本号及版本编译信息可以加到“主版本号.次版本号.修订号”的后面，作为延伸。

## 简介

---

在软件管理的领域里存在着被称作“依赖地狱”的死亡之谷，系统规模越大，加入的套件越多，你就越有可能在未来的某一天发现自己已深陷绝望之中。

在依赖高的系统中发布新版本套件可能很快会成为恶梦。如果依赖关系过高，可能面临版本控制被锁死的风险（必须对每一个相依套件改版才能完成某次升级）。而如果依赖关系过于松散，又将无法避免版本的混乱（假设兼容于未来的多个版本已超出了合理数量）。当你專案的进展因为版本相依被锁死或版本混乱变得不够简便和可靠，就意味着你正处于依赖地狱之中。

作为这个问题的解决方案之一，我提议用一组简单的规则及条件来约束版本号的配置和增长。这些规则是根据（但不局限于）已经被各种封闭、开放源代码软件所广泛使用的惯例所设计。为了让这套理论运作，你必须先有定义好的公共API。这可以透过文件定义或代码强制要求来实现。无论如何，这套API 的清楚了是十分重要的。一旦你定义了公共API，你就可以透过修改相应的版本号来向大家说明你的修改。考虑使用这样的版本号格式：XYZ（主版本号.次版本号.修订号）修复问题但不影响API 时，递增修订号；API 保持向下兼容的新增及修改时，递增次版本号；进行不向下兼容的修改时，递增主版本号。

我称这套系统为“语义化的版本控制”，在这套约定下，版本号及其更新方式包含了相邻版本间的底层代码和修改内容的信息。

## 语义化版本控制规范（SemVer）

---

以下关键词MUST、MUST NOT、REQUIRED、SHALL、SHALL NOT、SHOULD、SHOULD NOT、RECOMMENDED、MAY、OPTIONAL 依照RFC 2119 的叙述解读。（译注：为了保持语句顺畅，以下文件遇到的关键词将依照整句语义进行翻译，在此先不进行个别翻译。）

1. 使用语义化版本控制的软件“必须MUST”定义公共API。该API可以在代码中被定义或出现于严谨的文件内。无论何种形式都应该力求精确且完整。
2. 标准的版本号“必须MUST”采用XYZ的格式，其中X、Y和Z为非负的整数，且“禁止MUST NOT”在数字前方补零。X是主版本号、Y是次版本号、而Z为修订号。每个元素“必须MUST”以数值来递增。例如：1.9.1 -> 1.10.0 -> 1.11.0。
3. 标记版本号的软件发行后，“禁止MUST NOT”改变该版本软件的内容。任何修改都“必须MUST”以新版本发行。
4. 主版本号为零（0.yz）的软件处于开发初始阶段，一切都可能随时被改变。这样的公共API 不应该被视为稳定版。
5. 1.0.0 的版本号用于界定公共API 的形成。这一版本之后所有的版本号更新都基于公共API 及其修改内容。
6. 修订号Z（xyZ | x > 0）“必须MUST”在只做了向下兼容的修正时才递增。这里的修正指的是针对不正确结果而进行的内部修改。
7. 次版本号Y（xYz | x > 0）“必须MUST”在有向下兼容的新功能出现时递增。在任何公共API 的功能被标记为弃用时也“必须MUST”递增。也“可以MAY”在内部程序有大量新功能或改进被加入时递增，其中“可以MAY”包括修订级别的改变。每

- 当次版本号递增时，修订号“必须MUST”归零。
- 主版本号X (Xyz | X > 0) “必须MUST”在有任何不兼容的修改被加入公共API时递增。其中“可以MAY”包括次版本号及修订级别的改变。每当主版本号递增时，次版本号和修订号“必须MUST”归零。
  - 先行版本号“可以MAY”被标注在修订版之后，先加上一个连接号再加上一连串以句点分隔的标识符号来修饰。标识符号“必须MUST”由ASCII码的英数字和连接号[0-9A-Za-z-]组成，且“禁止MUST NOT”留白。数字型的标识符号“禁止MUST NOT”在前方补零。先行版的优先级低于相关联的标准版本。被标上先行版本号则表示这个版本并非稳定而且可能无法达到兼容的需求。范例：1.0.0-alpha、1.0.0-alpha.1、1.0.0-0.3.7、1.0.0-x.7.z.92。
  - 版本编译信息“可以MAY”被标注在修订版或先行版本号之后，先加上一个加号再加上一连串以句点分隔的标识符号来修饰。标识符号“必须MUST”由ASCII的英数字和连接号[0-9A-Za-z-]组成，且“禁止MUST NOT”留白。当判断版本的优先级时，版本编译信息“可SHOULD”被忽略。因此当两个版本只有在版本编译信息有差别时，属于相同的优先层级。范例：1.0.0-alpha+001、1.0.0+20130313144700、1.0.0-beta+exp.sha.5114f85。
  - 版本的优先层级指的是不同版本在排序时如何比较。判断优先层级时，“必须MUST”把版本依序拆分为主版本号、次版本号、修订号及先行版本号后进行比较（版本编译信息不在这份比较的列表中）。由左到右依序比较每个标识符号，第一个差异值用来决定优先层级：主版本号、次版本号及修订号以数值比较，例如1.0.0 < 2.0.0 < 2.1.0 < 2.1.1。当主版本号、次版本号及修订号都相同时，改以优先层级比较低的先行版本号决定。例如：1.0.0-alpha < 1.0.0。有相同主版本号、次版本号及修订号的两个先行版本号，其优先层级“必须MUST”透过由左到右的每个被句点分隔的标识符号来比较，直到找到一个差异值后决定：只有数字的标识符号以数值高低比较，有字母或连接号时则逐字以ASCII的排序来比较。数字的标识符号比非数字的标识符号优先层级低。若开头的标识符号都相同时，栏位比较多的先行版本号优先层级比较高。范例：1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0。

## 为什么要使用语义化的版本控制？

这并不是一个新的或者革命性的想法。实际上，你可能已经在做一些近似的事情了。问题在于只是“近似”还不够。如果没有某个正式的规范可循，版本号对于依赖的管理并无实质意义。将上述的想法命名并给予清楚的定义，让你对软件使用者传达意向变得容易。一旦这些意向变得清楚，弹性（但又不会太弹性）的依赖规范就能达成。

举个简单的例子就可以展示语义化的版本控制如何让依赖地狱成为过去。假设有个名为“救火车”的函式库，它需要另一个名为“梯子”并已经有使用语义化版本控制的套件。当救火车创建时，梯子的版本号为3.1.0。因为救火车使用了一些版本3.1.0所新增的功能，你可以放心地指定相依赖于梯子的版本号大等于3.1.0但小于4.0.0。这样，当梯子版本3.1.1和3.2.0发布时，你可以将直接它们纳入你的套件管理系统，因为它们能与原有相依的软件兼容。

作为一位负责任的开发者，你应当确保每次套件升级的运作与版本号的表述一致。现实世界是复杂的，我们除了提高警觉外能做的不多。你所能做的就是让语义化的版本控制为你提供一个健全的方式来发行以及升级套件，而无需推出新的相依套件，节省你的时间及烦恼。

如果你对此认同，希望立即开始使用语义化版本控制，你只需声明你的函式库正在使用它并遵循这些规则就可以了。请在你的README文件中保留此页连结，让别人也知道这些规则并从中受益。

## 参考资料

- [Semantic Versioning 2.0.0](#)

## 命名规范

---

说到命名规范，个人认为包含了目录，文件以及变量的命名。

混乱或错误的命名不仅让我们对代码难以理解，更糟糕的是，会误导我们的思维，导致对代码的理解完全错误。相反，良好的命名，则可以让我们的代码非常容易读懂，也能向读者正确表达事物以及逻辑的本质，从而使得代码的可维护性就大大增强，读命名好的文章是非常流畅的，会有一种享受的感觉。

良好的命名，以及良好的命名习惯，由于我们总是对每个概念的名称要求非常苛刻，我们会思考这个名称所表达的概念是否正确，该名称是否正确表达了事物的本质或正确反映了某个行为的逻辑。所以，这种对命名的良好思考习惯，可以反过来帮助我们纠正之前的一些错误设计和代码实现；比如，你之前有一个地方可能命名不太准确，然后你发现后面有另一个地方需要用这个名字，且更合理。所以你会发现这个名字对前面的地方就不适合了，从而你会去思考前面的地方可能需要用其他的名字，或者你会发现前面的地方的设计本身就是有问题的。这种就是名字可以促使你思考你的设计是否正确的例子。

## 目录

---

## 文件

---

## 变量

---

## 参考资料

---

- [对代码命名的一点思考和理解](#)

## 书写文档

---

一个程序员，最基本的是需要写代码，代码之中其实就包含了注释。如果是一个系统或者库文件，其实还需要书写文档以配合。本文就教大家如何书写文档。

## github如何管理文档

---

讨论一下我们如何使用 [Github Page](#) 服务运行 [Github 帮助文档](#)（目前每月有上百万的访问量）的。

## 以前的流程

---

- 用于托管维护网站、管理网站所用资源和文档搜索增强的 Rails 应用程序
- 用户托管由一大堆 Markdown 文件组成的网站具体内容

我们正常的撰写流程可能是这个样子的：

- 当有新特征开发出来的时候文档团队首先编写好文档内容
- 创建一个新的 issue 去追踪这个特征
- 当文档更新完毕一切就绪之后，我们会发起一个 pull request 去迭代更新文档内容。
- PR 发起成功后，我们会使用 @ 方式提醒团队（比如 @github/docs）并会让队友们审查一下我们的内容。
- 当这个特征开发完毕已经上线的时候，我们会合并之前创建的 PR。使用 webhook 能够帮助我们在内容仓库快速激活我们部署的 Rails 应用程序。webhook 承担了负责更新数据库的任务。

## 新的流程

---

当 Jekyll 2.0 发布的时候，我们看到了曙光，是时候该把我们这套该死的流程换成纯静态站了！特别是新增加的 Collections 文档类型能让你定义你需要的文件结构。另外，Jekyll 2.0 还增加了 Sass 和 CoffeeScript 的支持，这将使得编写前端代码变的更为简单便捷。

## 参考资料

---

- [Github 是如何用 Github 撰写 Github 文档的](#)
- [markdown + git 最适合程序员的wiki系统](#)
- [gollum: github的文档系统。](#)

## 开源协议

---

## 参考资料

---

- [choosealicense](#)

## 目录结构

---

## 参考资料

---

- [Baidu EFE team](#)
- [Baidu EFE team: spec](#)

## 数据结构与算法

---

### 参考资料

---

- [How to: What are the lesser known but useful data structures?](#)

## XML和JSON

---

### 参考资料

---

- [XML和JSON](#)



# 算法学习之路

---

## 关于

---

严格来说，本文题目应该是我的数据结构和算法学习之路，但这个写法实在太绕口——况且CS中的算法往往暗指数据结构和算法（例如算法导论指的实际上是数据结构和算法导论），所以我认为本文题目是合理的。

## 这篇文章讲了什么？

---

- 我这些年学习数据结构和算法的总结。
- 一些不错的算法书籍和教程。
- 算法的重要性。

## 初学

---

第一次接触数据结构是在大二下学期的数据结构课程。然而这门课程并没有让我入门——当时自己正忙于倒卖各种MP3和耳机，对于这些课程根本就不屑一顾——反正最后考试划个重点也能过，于是这门整个计算机专业本科最重要的课程就被傻逼的我直接忽略过去了。

直到大三我才反应过来以后还要找工作——而且大二的折腾证明了我并没有什么商业才能，以后还是得靠码代码混饭吃，我当时惊恐的发现自己对编程几乎一无所知，于是我给自己制订了一个类似于建国初期五年计划的读书成长计划，其中包括C语言基础、数据结构以及计算机网络等方面的书籍。

读书计划的第一步是选择书籍，我曾向当时我觉得很牛的“学长”和“大神”请教应该读哪些算法书籍，“学长”们均推荐《算法导论》，还有几个“大神”推荐计算机程序设计艺术（现在我疑心他们是否翻过这些书），草草的翻了下这两本书发现实在看不懂，但幸运的是我在无意中发现了豆瓣这个神奇的网站，里面有很多质量不错的书评，于是我就把评价很高而且看上去不那么吓人的计算机书籍都买了下来——事实证明豆瓣要比这些“学长”或是“大神”靠谱的多得多。

[数据结构与算法分析——C语言描述](#)是我学习数据结构的第一本书：当时有很多地方看不懂，于是做记号反复看；代码看不明白，于是抄到本子上反复研读；一些算法想不通，就把它所有的中间状态全画出来然后反复推演。事实证明尽管这种学习方法看起来傻逼而且效率很低，但对于当时同样傻逼的我却效果不错——傻人用傻办法嘛，而且这本书的课后题大多都是经典的面试题，以至于日后我看到[编程之美](#)的第一反应就是这货的题目不全是抄别人的么。

至今记得，这本书为了说明算法是多么重要，在开篇就拿最大子序列和作为例子，一路把复杂度从 $O(N^3)$ 杀到 $O(N^2)$ 再到 $O(N \lg N)$ 最后到 $O(N)$ ，当时内心真的是景仰之情=如滔滔江水连绵不绝，尼玛为何可以这么屌，

此外，我当时还把这本书里图算法之前的数据结构全手打了一遍，后来找实习还颇为自得的把这件事放到简历里，现在想想真是傻逼无极限。

凭借这个读书成长计划中学到的知识，我总算比较顺利的找到了一份实习工作，这是后话。

## 入门

---

我的实习并没有用到什么算法（现在看来就是不停的堆砌已有的API，编写一堆自己都不知道对不对的代码而已），在发现身边的人工作了几年却还在和我做同样的事情之后，我开始越来越不安。尽管当时我对自己没什么规划，但我清楚这绝壁不是我想做的工作。

在这个摇摆不定的时刻，[微软的梦工场](#)成了压倒骆驼的最后一支稻草，这本书对微软亚洲研究院的描写让我下定了“找工作就要这样的公司”的决心，然而我又悲观的发现无论是以我当时的能力还是文凭，都无法达到微软亚研院的要求，矛盾之下，我

彻底推翻了自己“毕业就工作”的想法，辞掉实习，准备考研。

考研的细节无需赘述，但至今仍清楚的记得自己在复试时惊奇且激动的发现北航宿舍对面就是微软西格玛大厦，那种离理想又进了一步的感觉简直爽到爆。

## 算法设计与分析

我的研究生生涯绝对是一个反面典型——翘课，实习，写水论文，做水研究，但有一点我颇为自得——从头到尾认真听了韩军教授的算法设计与分析课程。

韩军给我印象最深的有两点：课堂休息时跑到外面和几个学生借火抽烟；讲解算法时的犀利和毫不含糊。尽管韩军从来没有主动提及，但我敢肯定**算法设计与分析基础**就是他算法课程事实上的（de-facto）教材，因为他的课程结构几乎和这本书的组织结构一模一样。

如果数据结构与算法分析——C语言描述是我的数据结构启蒙，那么韩军的课程和**算法设计与分析基础**就是我的算法启蒙，结合课程和书籍，我一一理解并掌握了复杂度分析、分治、减治、变治、动态规划和回溯这些简单但强大的算法工具。

**算法引论**是我这时无意中读到的另一本算法书，和普通的算法书不同，这本书从创造性的角度出发——如果说算法导论讲的是有哪些算法，那么**算法引论**讲的就是如何创造算法。结合前面的**算法设计与分析基础**，这本书把我能解决的算法问题数量扩大了一个数量级。

之后，在机缘巧合下，我进入微软亚洲工程院实习，离理想又近了一步，自我感觉无限牛逼。

## 巩固

在微软工程院的实习是我研究生阶段的一个非常非常非常重要的转折点：

- 做出了一个还说的过去的小项目。
- 期间百度实习面试受挫，痛定思痛之下阅读了大量的程序设计书。
- 微软的实习经历成为了我之后简历上为数不多的亮点之一（本屌一没成绩，二没论文，三没ACM）。

这里就不说1和3了（和本文题目不搭边），重点说说2。

由于当时组内没有特别多的项目，我负责的那一小块又提前搞定了，mentor便很慷慨的扔给我一个Kinect和一部Windows Phone让我研究，研究嘛，自然就没有什么deadline，于是我就很鸡贼的把时间三七开：七分倒腾Windows Phone，三分看书&经典论文。

然而一件事打断了这段安逸的生活--百度实习面试。

基友在人人发百度实习内推贴，当时自我感觉牛逼闪闪放光芒，于是就抱着看看国内IT环境+虐虐面试官的变态心理投了简历，结果在第一面就自己的师兄爆出翔：他让我写一个stof（字符串转浮点数），我磨磨唧唧半天也没写出完整实现，之后回到宿舍赶快写了一个版本发到师兄的邮箱，结果对方压根没鸟我。

这件事对我产生了很大的震动：

- 原来自己连百度实习面试都过不去。
- 原来自己还是一个编程弱逼。
- 原来自己还是一个算法菜逼。

痛定思痛，我开始了第二个“五年计划”，三七开的时间分配变成了七三开：七分看书，三分WP。而这一阶段的重点从原理（Principle）变成了实现（Implementation）——Talk is cheap, show me the code.

## 进阶的书籍

由于一直觉得名字里带“Elements of”的都是酷炫叨炸天的书，所以我几乎是毫不犹豫的买了这本[Elements of Programming](#)，事实上这本书里的代码（或者说STL的代码）确实是：快，狠，准，古龙高手三要素全齐。

百度面试被爆出翔的经历让我意识到另一个问题，绝大多数公司面试时都需要在纸上写C代码，而我自己却很少用C（多数情况用C#），考虑到自己还没牛逼到能让公司改变面试流程的地步，我需要提升自己编写C代码的能力（哪怕只是为了面试）。一顿Google之后，我锁定了[C Interfaces and Implementation](#)——另一本关于如何写出狂炫酷帅叨炸天的C代码的奇书，这里套用下Amazon的评论：Probably the best advanced C book in existance.

严格来说上面两本书都不是传统的算法书，因为它们侧重的都不是算法，而是经典算法的具体实现（Implementation），然而这正是我所需要的：因为算法的原理我能说明白，但要给出优雅正确简练的实现我就傻逼了，哪怕是stof这种简单到爆的“算法”。

依然是以前的傻逼学习方法：反复研读+一遍又一遍的把代码抄写到本子上，艰难的完成了这两本书后，又读了相当数量的编程实践（Programming Practice）书籍，自我感觉编程能力又大幅提升，此外获得新技能——纸上编码。这也成为了我之后找工作面试的三板斧之一。

## 应用

说老实话，自从本科实习之后，我就一直觉得算法除了面试时能有用，其它基本用不上，甚至还写了一篇当时颇为自得现在读起来极为傻逼的文章来黑那些动不动就“基础”或“内功”的所谓“大牛”们，这里摘取一段现在看起来很傻逼但当时却觉得是真理的文字：所以那些动则就扯什么算法啊基础啊内功啊所谓的大牛们，请闭上你的嘴，条条大道通罗马。算法并不是编程的前提条件，数学也不会阻碍一个人成为优秀的程序员。至少在我看来，什么算法基础内功都是唬人的玩意，多编点能用的实用的程序才是王道，当然如果你是一个pure theorist的话就当我说什么都没说好了。

然而有意思的是，写了这篇文章没多久，鼓吹算法无用论的我自己做的几个大大小小的项目全部用到了算法——我疑心是上天在有意抽我的脸。

## LL(k)

我在微软实习的第一个项目做的是代码覆盖率分析——计算T-SQL存储过程的代码覆盖率。

简单的看了下SQL Server相关的文档，我很快发现SQL Reporting Service可以记录T-SQL的执行语句及行号，于是行覆盖（line coverage）搞定，但老大说行覆盖太naive，我们需要更实际的块覆盖（block coverage）。

阅读了块覆盖的定义后，我发现我需要要对T-SQL进行语法分析，在没有找到一个好用的T-SQL Parser的情况下，只能自己动手搞一个。比较奇葩的是，做这个项目时当时我刚把ANTLR作者的[Language Implementation Patterns](#)（中文）看了一半，什么LL(k)啊Packrat啊AST Walker的概念啊正热乎着呢。

于是，自己自己就照着T-SQL的官方EBNF，三下五除二撸了一个T-SQL存储过程的LL(k) Parser，把代码转换成AST，然后用一个External AST Walker生成代码块覆盖的HTML报表，全部过程一周不到。

老大自然是很满意——我疑心他的原计划是花两三个月来完成这个项目，因为这个项目之后的两个月我都没什么活干，天天悠哉游哉。

## 拼音索引

拼音索引是我接的一个手机应用私活里的小模块，用户期待在手机文本框可以根据输入给出智能提示：

比如说输入中国：给出可能的关键词，同样，输入拼音也应给出提示。中文匹配这个简单，但拼音匹配就得花时间想想了——懒得造轮子的我第一时间找到了微软的拼音库，但接下来我就发现微软这个鸟库在手机上跑不动，研究了下发现WP7对Dictionary的items数量有限制，貌似是7000还是8000个item就会崩盘，而标准汉字则有两万多个，尼玛。

痛骂MS坑爹+汉字坑爹之余，还是得自己撸一个库出来：

- 首先把那两万个汉字搞了出来，排序，然后弄成一个超长的字符串。
- 接下来用Int16索引了汉字所有的拼音（貌似500多个）。
- 再接下来用Int64建立汉字和拼音的关联——汉字有多音字，所以需要把多个拼音pack到一个Int64里，这个简单，位操作就搞定。
- 最后用二分+位移Unpack，直接做到从汉字到拼音的检索。
- 后来小测了下性能，速度是MS原来那个库的五十倍有余，而代码量只有336行。

用户很happy——因为我捎带把他没想到的多音字都搞定了，而且流畅的一逼。

我也很happy，因为没想到自己写的库居然比MS的还要快几十倍，同时小十几倍。

从这个事情之后我变得特别理解那些造轮子的人——你要想想，如果你需要一个飞机轮子但市场上只有自行车轮子而且老板还催着你交工，你能怎么搞。

## 快速字符串匹配

前面提到在微软实习时老大扔给我一个Windows Phone让我研究下，我当时玩了玩就觉着不太对劲，找联系人太麻烦。

比如说找“张晓明”，WP只支持定位到Z分类下——这意味着我需要在Z分类下的七十多个联系人（姓张的姓赵的姓钟的等等）里面线性寻找，每次我都需要滑动四五秒才能找到这个张姓少年。

这TMD也太傻逼了，本屌三年前的老破NOKIA都支持首字母定位，996->ZXM->张晓明，直接搞定，尼玛一个新时代Windows Phone居然会弱到这个程度。

搜了一下发现没有好用的拨号程序，于是本屌就直接撸了一个支持首字母匹配的拨号程序出来扔到WP论坛里。

结果马上就有各种问题出现——最主要的反映是速度太慢，一些用户甚至反馈按键有时要半秒才有反应。本屌问了下他的通讯录大小：大概3000多人。吐槽怎么会有这么奇葩的通讯录之余，我意识到自己的字符串匹配算法存在严重的性能问题：读取所有人的姓名计算出拼音，然后一个个的匹配——结果如果联系人数量太多的话，速度必然抽汁。

于是我就开始苦思冥想有没有一个能够同时搜索多个字符串的高端算法，以至于那两天坐地铁都在嘟囔怎么才能把这个应用搞的快一些。

最终还是在[Algorithms on Strings, Trees and Sequences](#)里找到了答案——确实有能够同时搜索多个字符串的方法：Tries，而且这本书还用足足一章来讲怎么弄Multiple string comparison，看得我当时高潮迭起，直呼过瘾。

具体细节不多说，总之换了算法之后，匹配速度快了大约九十多倍，而且代码还短了几十行。哪怕是有10000个联系人，也能在0.1秒内搞定，速度瓶颈就这样愉快的被算法搞定。

## Writing Efficient Programs

之后又做了若干个项目，多多少少都用到了“自制”的算法或数据结构，最奇葩的一次是写一个电子书阅读器里的分页，我照着模拟退火（Simulated Annealing）的原理写了一个快速分页算法，事实上这个算法确实很快——但问题是我也不知道为啥它会这么快。

总之，算法是一种将有限计算资源发挥到极致的武器，当计算资源很富余时算法确实没大用，但一旦到了效率瓶颈算法绝对是开山第一刀（因为算法不要钱嘛！要不还得换CPU买SSD升级RAM，肉疼啊！！）。一些人会认为这种说法是有问题，因为编写新算法的人力成本有时比增加硬件的成本还要高——但别忘了增加硬件提升效率也是建立在算法是Scalable的基础上——说白了还是得撸算法。

说到优化这里顺便提一下[Writing Efficient Programs](#)——很难找到一本讲代码优化的书（我疑心是自从Knuth说了过早优化是万恶之源之后没人敢写，万恶之源嘛，写它干毛），注意这本书讲的是代码优化——在不改变架构、算法以及硬件的前提下进行的优化。尽管书中的一些诸如变量复用或是循环展开的trick已经过时，但总体仍不失为一本好书。

## 提高

---

实习实习着就到了研二暑假，接下来就是求职季。

求职季时我有一种莫名的复仇感——尼玛之前百度实习面试老子被你们黑的漫天飞翔，这回求职老子要把你们一个个黑回来，尼玛。

现在回想当时的心理实属傻逼+幼稚，但这种黑暗心理也起了一定的积极作用：我丝毫不敢有任何怠慢，以至于在5月份底我就开始准备求职笔试面试，比身边的同学早了两个月不止。

我没有像身边的同学那般刷题——而是继续看书抄代码学算法，因为我认为那些难得离谱的题面试官也不会问——事实上也是如此。

### Algorithm Design Manual

因为很多Coding Interview的论坛都提到这本红皮书，我也跟风搞了一本。事实证明，仅仅是关于Backtrack Template那部分的描述就足以值回书价，更不用说它的Heuristics和课后题。

### 编程珠玑&更多的编程珠玑

这两本书就不用多介绍，[编程珠玑](#)和[更多的编程珠玑](#)，没听说过这两本书请自行面壁。前者偏算法理论，后者偏算法轶事，前者提升能力，后者增长谈资，都值得一读。

### The Science of Programming

读到编程珠玑里面关于Binary Search的正确性证明时我大呼过瘾，原来程序的正确性也是可以推导的，然后我就在那一章的引用里发现David Gries的The Science of Programming。看名字就觉得很厉害，直接搞了一本开撸。

不愧为编程珠玑引用的书籍，撸完The Science of Programming之后，本屌获得了证明简单代码段的正确性这个技能——求职面试三板斧之二。

证明简单代码段的正确性是一个很神奇的技能——因为面试时大多数公司都会要求在纸上写一段代码，然后面试官检查这段代码，如果你能够自己证明自己写的代码是正确的，面试官还能挑剔什么呢？

之后就是各种面试，详情见之前的博客，总之就是项目经历、纸上代码加正确性证明这三板斧，摧枯拉朽。

## 进化

---

求职毕业季之后就是各种Happy，Happy过后本屌发现即将面临另一个问题：算法能力不足。

因为据说以后的同事大多是ACM选手，而本屌从来没搞过算法竞赛，而且知道的算法和数据结构都极为基础：像那些元胞自动机、斐波那契堆或是线段树这些高端数据结构压根只是能把它们的英文名称拼写出来，连用都没用过，所以心理忐忑的一逼。

为了不至于到时入职被鄙视的太惨烈，加上自己一贯的算法自卑症，本屌强制自己再次学习算法。

### Algorithms 4th

Algorithms是我重温算法的第一本书，尽管它实际就是一本数据结构入门书，但它确实适合当时已经把算法忘光的本屌——不为学习，只为重温。

这本书最大的亮点在于它把Visualization和Formatting做到了极致——也许它不是最好的数据结构入门书，但它绝壁是我读过的排版最好的书，阅读体验爽的一逼；当然这本书的内容也不错，尤其是红黑树那一部分，我想不会有什么书会比此书讲的

更明白。

## 6.851 Advanced Data Structures

[Advanced Data Structures](#)是MIT的高级数据结构教程，为什么会找到这个教程呢？因为Google Advanced Data Structures第一个出来的就是这货。

这门课包含各种让本屌世界观崩坏的奇诡数据结构和算法，它们包括但不限于：

- 带“记忆”的数据结构（Data Structure with Persistence）。
- van Emde Boas（逆天的插入，删除，前驱和后继时间复杂度）。
- $O(1)$ 时间复杂度的LCA、RMQ和LA解法。
- 奇幻的 $O(n)$ 时间复杂度的Suffix Tree构建方法。
- $O(\lg \lg n)$ 的BST。
- ...

总之高潮迭起，分分高能，唯一的不足就是没有把它们实现一圈。以后本屌一定找时间把它们一个个撸一遍。

## 总结

---

从接触算法到现在，大概七年：初学时推崇算法牛逼论，实习后鼓吹算法无用论，读研后再被现实打回算法牛逼论。

怎么这么像辩证法里的肯定到否定再到否定之否定。

现在来看，相当数量的鼓吹算法牛逼论的人其实不懂算法的重要性——如果你连用算法解决实际问题的经历都没有，那你如何可以证明算法很有用？而绝大多数鼓吹算法无用论的人不过是低水平码农的无病呻吟——他们从未碰到过需要用算法解决的难题，自然不知道算法有多重要。

Peter Norvig曾经写过一篇非常精彩的SICP书评，我认为这里把SICP换成算法依然适用：To use an analogy, if algorithms were about automobiles, it would be for the person who wants to know how cars work, how they are built, and how one might design fuel-efficient, safe, reliable vehicles for the 21st century. The people who hate algorithms are the ones who just want to know how to drive their car on the highway, just like everyone else.

MIT教授Erik Demaine则更为直接：If you want to become a good programmer, you can spend 10 years programming, or spend 2 years programming and learning algorithms.

总而言之，如果你想成为一个码农或是熟练工（Code Monkey），你大可以不学算法，因为算法对你确实没有用；但如果你想成为一个优秀的开发者（Developer），扎实的算法必不可少，因为你会不断的掉进一些只能借助算法才能爬出去的坑里。

## 参考资料

---

- [我的算法学习之路](#)

## 面向对象编程

---

OOP.

## 设计模式

---

设计模式。

## 参考资料

---

- [蔡学镛架构设计方法](#)



## 常用的Javascript设计模式

---

### 参考资料

---

- [常用的Javascript设计模式](#)



## 继承

---

继承从代码复用的角度来说，特别好用，也特别容易被滥用和被错用。不恰当地使用继承导致的最大的一个特征就是高耦合。在这里我要补充一点，耦合是一个特征，虽然大部分情况是缺陷的特征，但是当耦合成为需求的时候，耦合就不是缺陷了。耦合成为需求的例子在后面会提到。

## 总结

---

可见，代码复用也是分类别的，如果当初只是出于代码复用的目的而不区分类别和场景，就采用继承是不恰当的。我们应当考虑以上3点要素看是否符合，才能决定是否使用继承。就目前大多数的开发任务来看，继承出现的场景不多，主要还是代码复用的场景比较多，然而通过组合去进行代码复用显得要比继承麻烦一些，因为组合要求你有更强的抽象能力，继承则比较符合直觉。然而从未来可能产生的需求变化和维护成本来看，使用组合其实是很值得的。另外，当你发现你的继承超过2层的时候，你就要好好考虑是否这个继承的方案了，第三层继承正是滥用的开端。确定有必要之后，再进行更多层次的继承。

## 参考资料

---

- [跳出面向对象思想\(一\) 继承](#)

## 多态

---

### 总结

---

多态在面向对象程序中的应用相当广泛，只要有继承的地方，或多或少都会用到多态。然而多态比起继承来，更容易被不明不白地使用，一切看起来都那么顺其自然。在客户程序员这边，一般是只要多态是可行方案的一种，到最后大部分都会采用多态的方案来解决问题。

然而多态正如它名字中所暗示的，它有非常大的潜在可能引入不属于对象初衷的逻辑，巨大的灵活性也导致客户程序员在遇到问题的时候不太愿意采用其他相对更优的方案，比如IOP。在决定是否采用多态时，我们要有一个清晰的角色概念，做好角色细分，不要角色混乱。该是拦截器的，就给他制定一个拦截器接口，由另一个对象（逻辑上的另一个对象，当然也可以是自己）去实现接口里的方法集。不要让一个对象在逻辑上既是拦截器又是业务模块。这样才方便未来的维护。另外也要注意被覆重方法的作用，如果只是单纯为了提供父类所需要的中间数据的，一律都用IOP，这是比直接采用多态更优的方案。

IOP能够带来的好处当然不止文中写到的这些，它在其他场合也有非常好的应用，它最主要的好处就在于分离了定义和实现，并且能够带来更高的灵活性，灵活到既可以对语言过高的自由度有一个限制，也可以灵活到允许同一接口的不同实现能够合理地组合。在架构设计方面是个非常重要的思想。

### 参考资料

---

- [跳出面向对象思想\(二\) 多态](#)

## 封装

---

## 面向接口编程

---

IOP。

## 代码评审

---

### 参考资料

---

- [6 Ways to Quantify Your Code—and Why You Need to Do It](#)

## 六种量化你代码的方式

本文为译文，译者为Leo Hui(我自己！)。

Businesspeople dig numbers. They don't necessarily want to hear that you got something done; they want to hear how much

商人关注的是量化，他们想从你哪里知道你做了什么，带来了什么价值，而不是你做了多少。

Some professionals have it easy when it comes to quantifying their job performance. Salespeople can measure their achie

教授可以轻松量化出他们的工作，销售人员可以计算出他们的收益。其他的领域也可以通过一些方式算出他们的贡献。

For software developers and some other technology-based roles, however, quantifying your work can be a struggle without

但是从事软件开发以及技术相关的人员，量化工作确实一个困难的事情。量化这件事情，不是在求职，更是一个技术人员生涯的一部分。绩效评估，有效的了解沟通，高效的和非技术人员合作，确保你再团队或组织中的价值。

So how do you measure the value of the applications you build, scale, monitor, test, and otherwise support? Here are so

但是我们要如何量化工作中的价值呢？这里有一些New Relic推荐的做法：

"I like to see work accomplishments described in terms of situation, action and results," says Merilee Krebs, a technic

New Relic的技术招聘人员这样说：“我喜欢看到用情况，行动和结果去描述工作成果，技术人员需要解决的问题是什么？采用什么样的行动去解决和提升这个问题。”

What does that look like in the real world? Try asking yourself some pointed questions: Did your monitoring and testing

现实世界中是怎样的呢？你试着问自己一些关键的问题：你有在更新你的代码的时候去监控和测试...这就是量化的目标。如果你在日程表前六周就完成了了一个app，你肯定会去炫耀一下。但是你有考虑公司的战略目标吗，如果没有，请思考一下。...

If this exercise feels unnatural to you, you're not alone—many programmers often aren't born sales and marketing pros.

你是不是感受到一些不同的感觉，这不是你一个人的问题，技术人员的通病。如果技术人员做好了量化这一块，那么他们也许就去从事销售了。所以，让我们考虑六种去量化你代码以及工作的方式：

### Think in percentages



**Get involved with open source projects**

---

**Measure progress, not just products**

---

**Keep a work journal**

---

**Communicate in two languages**

---

**Collect recommendations**

---

**参考资料**

---

- [6 Ways to Quantify Your Code—and Why You Need to Do It](#)

## 程序员必备的代码审查（Code Review）清单

---

在我们关于高效代码审查的博文中，我们建议使用一个检查清单。在代码审查中，检查清单是一个非常好的工具——它们保证了审查可以在你的团队中始终如一地进行。它们也是一种保证常见问题能够被发现并被解决的便利方式。

软件工程学院的研究表明，程序员们会犯15-20种常见的错误。所以，通过把这些错误加入到检查清单当中，你可以确保不论什么时候，只要这些错误发生了，你就能发现它们，并且可以帮助你杜绝这些错误。

为了帮助你开始创建一个清单，这里列出了一些典型的内容：代码审查清单。

### 常规项

---

- 代码能够工作么？它有没有实现预期的功能，逻辑是否正确等。
- 所有的代码是否简单易懂？
- 代码符合你所遵循的编程规范么？这通常包括大括号的位置，变量名和函数名，行的长度，缩进，格式和注释。
- 是否存在多余的或是重复的代码？
- 代码是否尽可能的模块化了？
- 是否有可以被替换的全局变量？
- 是否有被注释掉的代码？
- 循环是否设置了长度和正确的终止条件？
- 是否有可以被库函数替代的代码？
- 是否有可以删除的日志或调试代码？

### 安全

---

- 所有的数据输入是否都进行了检查（检测正确的类型，长度，格式和范围）并且进行了编码？
- 在哪里使用了第三方工具，返回的错误是否被捕获？
- 输出的值是否进行了检查并且编码？
- 无效的数值是否能够处理？

### 文档

---

- 是否有注释，并且描述了代码的意图？
- 所有的函数都有注释吗？
- 对非常规行为和边界情况处理是否有描述？
- 第三方库的使用和函数是否有文档？
- 数据结构和计量单位是否进行了解释？
- 是否有未完成的代码？如果是的话，是不是应该移除，或者用合适的标记进行标记比如'TODO'？

### 测试

---

- 代码是否可以测试？比如，不要添加太多的或是隐藏的依赖关系，不能够初始化对象，测试框架可以使用方法等。
- 是否存在测试，它们是否可以被理解？比如，至少达到你满意的代码覆盖(code coverage)。
- 单元测试是否真正的测试了代码是否可以完成预期的功能？
- 是否检查了数组的“越界”错误？
- 是否有可以被已经存在的API所替代的测试代码？

### 总结

你同样需要把特定语言中有可能引起错误的问题添加到清单中。

这个清单故意没有详尽的列出所有可能会发生的错误。你不希望你的清单是这样的，太长了以至于从来没人会去用它。仅仅包含常见的问题会比较好。

## 优化你的清单

---

把使用清单作为你的起点，针对特定的使用案例，你需要对其进行优化。一个比较棒的方式就是让你的团队记录下那些在代码审查过程中临时发现的问题，有了这些数据，你就能够确定你的团队常犯的错误，然后你就可以量身定制一个审查清单。确保你删除了那些没有出现过的错误。（你也可以保留那些出现概率很小，但是非常关键的项目，比如安全相关的问题）。

## 得到认可并且保持更新

---

基本规则是，清单上的任何条目都必须明确，而且，如果可能的话，对于一些条目你可以对其进行二元判定。这样可以防止判断的不一致。和你的团队分享这份清单并且让他们认同你清单的内容是个好主意。同样的，要定期检查你的清单，以确保各条目仍然是有意义的。

有了一个好的清单，可以提高你在代码审查过程中发现的缺陷个数。这可以帮助你提高代码标准，避免质量参差不齐的代码审查。

## 参考资料

---

- [程序员必备的代码审查（Code Review）清单](#)

## 懂点设计

---

## 佳作欣赏

---

### 设计相关

---

- [Tuts+](#): 应有尽有。
- [smashingmagazine](#): 海纳百川。
- [themeforest](#): Website Templates and Themes.
- [inspiretrends](#)
- [dribbble](#)
- [canva](#): design school
- [beautifulpixels](#)

### 前端效果

---

- [codepen](#)
- [css deck](#)
- [TheCodePlayer](#): Learn HTML5, CSS3, Javascript and more...
- [codrops](#): 很多效果的代码示例。
- [css winner](#): 各种站点欣赏。

### 工具

---

- [InVision](#): 设计作品标注，分享站点。

## 无缝平铺

---

电脑的桌面背景设置中就有一个平铺的选项，它能使得图片无限重复，充满整个屏幕。CSS的背景设置也可以如此。那么这个是怎么制作的呢？

## 参考资料

---

- [这个节点的背景纹理图片是怎么做出来的？](#)
- [subtlepatterns](#)

## Sketch学习

---

### 参考资料

---

- [Sketch 3视频教程专辑](#)
- [优酷上的专辑](#)
- [Sketch完全自学资源合集](#)

## 设计与实现的平衡

---

dribbble上又一堆视觉设计高大上的东西，但可能大部分人都无法实现，可见设计与实现还是有一定的距离，如何缩小这一部分呢？

## 参考资料

---

- [视觉设计师是怎样让前端工程师 100% 实现设计效果的？](#)



## 写点东西

---

程序员能把逻辑梳理出来，是另一大技能！

## 使用gitbook

---

之前整理的习惯是使用Evernote，其实主要做的是收录（输入）的功能，东西多了，就难以消化掉（处理），以至于没有总结（输出）。Markdown格式是我很喜欢的一种书写格式，我觉得特别适合程序员，配合gitbook，就能达到写书的效果。

### Version 1.x.x

---

gitbook刚推出的时候，我就在使用，这个时候的版本都是1.x系列。安装方式是：

```
npm install -g gitbook
```

目录结构上，只要有 README.md 和 SUMMARY.md 即可。

使用方式：

```
gitbook build
```

即可完成编译。

### Version 2.x.x

---

最近再看gitbook的时候，发现它已经推出了2.x系列，安装方式也变化了。需要先删除之前1.x系列版本，再安装：

```
npm install -g gitbook-cli
```

## editor

---

关于editor，其实有两种选择：

- gitbook站点上edit，线上编辑
- 下载editor，本地编辑

### 线上编辑

先说说线上编辑，毕竟我刚刚使用了下，就是web版本的应用。有一点特别的好处是可以绑定github repo，编辑直接保存在github上。省去了本地编辑，再push的过程。

但是我也发现了一点点不方便的地方，就是无法上传整个目录文件，必须一个一个上传，且无法设置目录。。。很蛋疼。

### 本地编辑

就是一个本地软件，可以本地编辑，应该是通过图形化性质整合命令行，包含了build功能。1.x版本刚推出的时候，在window平台下体验过，会有再次打开丢失项目的情况，每次都要重新打开项目，很麻烦。但是新版应该有所改善。

本地编辑还有另外一个方式，就是我常用的手段：使用Sublime Text进行编辑，结合Markdown插件，产生预览。最后通过git提交到远程。

## 目录结构

---

官方的格式说明，[Format](#)。必须的文件是 `README.md` 和 `SUMMARY.md`。

## 使用方法

---

### init

`gitbook init`，根据 `SUMMARY.md` 生成目录。

### build

`gitbook build`，生成 `_books` 目录。

### serve

`gitbook serve`，开启本地预览。

## 参考资料

---

- [Gitbook主页](#)
- [Gitbook Help](#)
- [Version 2.0说明](#)
- [github: GitbookIO](#)
- [gitbook editor](#)

## 团队合作

---

除了独立开发者，大部分技术人员还是要进行团队合作的。

可惜程序员大多数都喜欢单打独斗，不管难易如何，全部都要自己拿下。

一个好的程序员，不单单关注的是程序本身，还应该关注到整个团队，帮助大家协同合作，减轻自己的压力，也提高了团队的效率。

## 关于招聘

---

搞技术的，不缺千里马，却的是伯乐。

我们要强大自己的团队，就需要招聘到对的人：技术和思想都一致的人。

## 招聘原则

---

### STAR原则

所谓STAR原则，即Situation（情景）、Task（任务）、Action（行动）和Result（结果）四个英文单词的首字母组合。STAR原则是结构化面试当中非常重要的一个理论。

S指的是situation,中文含义是情景，也就是在面谈中我们要求应聘者描述他在所从事岗位期间曾经做过的某件重要的且可以当作我们考评标准的事件的所发生的背景状况。

T指的是task，中文含义为任务，即是要考察应聘者在其背景环境中所执行的任务与角色，从而考察该应聘者是否做过其描述的职位及其是否具备该岗位的相应能力。

A指的是action,中文含义是行动，是考察应聘者在其所描述的任务当中所担任的角色是如何操作与执行任务的。

R指的是result,中文含义为结果，即该项任务在行动后所达到的效果，通常应聘者求职材料上写的都是一些结果，描述自己做过什么，成绩怎样，比较简单和宽泛。

而我们在面试的时候，则需要了解应聘者如何做出这样的业绩，做出这样的业绩都使用了一些什么样的方法，采取了什么样的手段，通过这些过程，我们可以全面了解该应聘者的知识、经验、技能的掌握程度以及他的工作风格、性格特点等与工作有关的方面。而STAR原则正是帮我们解决上述问题的。

## 参考资料

---

- [STAR原则](#)



## 领导能力

---

招聘，培训，下面就是领导力了。正巧，我最近也做一段时间的leader，负责培训。

### 责任

---

作为一个leader，自己也是有上级的。所以应该做到几点：

1. 合理的分解，分配工作。
2. 保持和上面的良好沟通。
3. 团结成员，不要偏心。
4. 能帮团队成员说话，顶得出去。
5. 在业务上能帮助成员成长。
6. 有胆放手让别人去做。

对于一个小公司来说，虽然一开始可能必须得靠管理者亲力亲为，但是当业务逐渐成型、走上正规之后，一定要把团队成员的能力都锻炼出来。否则公司的发展可能会受到限制，因为管理者的精力有限，被日常琐事缠绕就无法集中精力思考发展方向，或者是谋求外部的合作机会。所以要学会彻底的放权，让团队的成员都忙起来。在可以承受的范围内让大家去犯错，去成长，去做一线的决策，最终成为能独当一面的人。

## PM

---

项目经理/产品经理，我觉得程序员在做任何一个项目的时候，理应把自己设想为一个PM,对自己的项目负责。技术人员本身需要对产品有深入的理解。除此之外，产品能力在做开源软件、共享软件、接单单和创业时都会让你受益匪浅，实乃居家必备。

## 参考资料

---

- [专为程序员设计的产品课程「技术人的产品修炼之路」全集](#)



## 如何做一位與工程團隊合作良好的產品經理

產品經理(Product Manager)是任何新創產品的關鍵角色，他定義了產品的需求規格，找出產品的價值所在。而一個成功的產品代表結合了好的產品需求規格，以及能夠依據這個需求規格實作出來的工程團隊。我們說產品經理負責 Building The Right Product，而工程團隊負責 Building The Product Right，兩者缺一不可。由此可知產品經理與工程團隊之間的關係有多麼重要。

### 定義清楚這個產品

這是最基本也是重要的，要講三遍：「定義產品」、「定義產品」、「定義產品」，為什麼這麼根本的事情還作不好呢？很多時候是組織的問題，沒有人專職負責，例如：

- 產品經理就是老闆。但是老闆太忙了，只定義了「high-level」需求，而沒有定義細節規格。於是工程師就得自己腦補，最後做錯了被罵又重做。
- 行銷客服來兼職產品經理。行銷很了解如何告訴世界這個產品有多棒，但是卻不一定能夠發現需求並定義清楚這個產品，這兩者的技能是不一樣的。並且很容易發生傳聲筒現象，直接把單一客戶的需求告訴工程師，而非經過一致的產品思考規劃以及使用者體驗設計。

我們會在之後的文章繼續探討什麼是好的規格文件。它代表了使用者應該要有什麼體驗、產品有哪些功能和行為，以及功能的開發優先順序。除了基本的 User Story，也應包含 Wireframe 介面流程和關鍵的測試案例。

### 建立良好的溝通管道

就算規格寫的再清楚，開始實作之後就會有一堆問題冒出來。如果沒有人可以問或是等不到產品經理的回覆，開發人員就會開始腦補這個功能，硬著頭皮繼續做下去(然後又做錯了被罵，需要重做)。或是就停下來，找其他低優先的事情來做。無論怎樣都是很大的浪費。

建立一個良好的溝通管道，讓工程師可以很快地跟產品經理問到答案。最好是同一個辦公室，越是 Remote 的工作環境，那麼規格文件就需要越清楚。

### 不要負責專案管理

如果沒有專職的專案經理(Project Manager)，就請 Lead Engineer 來負責專案進度。不要讓產品經理去兼專案管理。專案管理的重點在於執行及發佈產品、協調開發、測試與營運團隊。讓產品經理來做不只是力有未逮，也會造成兩方關係的緊繃，壓榨工程師的故事不斷真實上演，就是因為由不了解軟體工程的人負責專案管理。

在知名的 Scrum 敏捷軟體開發方法中就提到：產品經理決定作什麼，讓工程團隊決定如何做及可以做多少。

### 不要告訴工程團隊怎麼做，告訴他們要做什麼

很多產品經理喜歡把需求描述成解決方案，指定了一個不可行或者是成本效益很低的作法，而非告訴工程師他們想要達到的目標，或是想要解決的問題根本是什麼。

F-16 是史上最成功的戰鬥機之一，在一開始設計的要求是需要超過兩馬赫的速度，首席設計師問了美國空軍為什麼飛行速度這麼重要，答覆是「飛機必須可以從戰鬥中逃脫」。最後的設計並沒有超過兩馬赫，但是卻可以讓飛機很敏捷的逃脫，透過了無框的坐艙、抬頭顯示器，讓飛行員有更好的視野。可傾斜的座位降低了重力影響。飛行控制桿是安裝在右手邊上，而非傳統的在兩腿之間，用來輔助在高G值時候轉彎。這些設計不但也能達成設計目標，更降低了生產成本。

這個問題其實不只產品經理，工程師本身也常犯這個錯誤，常常可以在討論區看到有人問「請問這個指令 X 怎麼用」，然後

底下的討論怎樣都無法完全解決他的問題，直到有人問「為什麼你需要這個 X？」，他回答「因為他要做 Y 功能」，大家恍然大悟用 X 來解決根本是錯的，應該用指令 A 就可以很簡單的解決了。

在大多數的情況下，工程師比產品經理更清楚知道哪個實作方案最適合。

## 相信工程師專業

---

產品經理負責 Building The Right Product，而工程團隊負責 Building The Product Right。相信工程師的專業，就是讓工程團隊可以實行任何他們認為建構一個有品質產品需要的作法，例如撰寫自動化測試、Pair Programming、建構 CI 持續整合環境，Code Review 程式碼複審等等。這些工作雖然不直接與產品需求相關，卻對工程品質有很大的影響。這些事情現在不做，在專案開發幾個月之後，很快就會吞噬整個專案進度和品質。

## 总结

---

以上五點可以幫助產品經理和工程團隊合作的更愉快，但是請記得無論工程團隊再優秀，如果產品經理無法定義出一個值得建造的產品，一切也是枉然。

## 参考资料

---

- [如何做一位與工程團隊合作良好的產品經理](#)

## 获取知识

---

程序员是一个活到老，学到老，还有三分学不到的职业。这就无形的要求我们要不断进取，不断学习。

既然学习是一个必然的过程，那么我们就将这个过程变得高效和有趣。

关于书籍的选择，技术方面还是推荐英文资料，因为翻译质量的问题。你看中文可能10分钟的内容，理解却要1小时。而英文你看需要30分钟，理解只需要10分钟。

## 资源

---

这里特别提及一下技术学习的一些资源。我个人比较喜欢书本的理论知识结合视频介绍的实践。自己通过项目去学习。

提高技术的方式：

## 论坛

---

- [stackoverflow](#)
- [github](#)
- [reddit:Programming](#)

## 视频教程

---

- [tuts+](#): 首推这家，资源多，质量好，不管视频还有文字教程。
- [Lynda](#): 同tuts+，看过几门课，很有条理。
- [Course Hero](#)

## MOOC

---

我一直比较关注MOOC的发展。个人认为互联网是肯定会改变人们学习的方式的。但是这个过程需要更多时间去完善。

### 发展现状

---

#### 国内

网易的[网易公开课](#)应该是最早发起的在线教育。而[果壳网的MOOC学院](#)是一个较大的MOOC课程社区。

其他的一些：[慕课网](#)和[极客学院](#)多为带有培训性质的教育网站。且传授内容以计算机为主。

#### 国外

目前来说的三大平台为：[Coursera](#), [edX](#)和[Udacity](#)。

Coursera和edX的教育资源大都是顶级大学，Udacity里的课有很多业界大牛。从表面上来讲，Coursera和edX的课更加偏理论，而Udacity则更加偏重技能传授。

### 参考资料

---

- [MOOC的核心是什么？它与普通的大学公开课、视频公开课有什么本质区别？](#)
- [M三大 MOOC 网站：Coursera 与 Udacity 和 edX 比较，哪个更适合中国人？你有何经验分享？](#)

## 英语学习

---

现代的这个社会，不管你做什么，学好英语都是有好处的。

## 学习方法

---

### 音标

英式音标是20个元音，但是美式好像是24个。网上有一个很好的教材，[海伦教你学音标](#)。

美式音标（KK音标）与英式音标的区别：[国际音标&KK音标对照表详细打印版](#)。

### 单词

所有的文章是由句子组成，句子是由单词组成。了解单词的含义是学习的一个重要过程。我喜欢查看英英解释，在线的词典有这么几个：

- [Dictionary](#)
- [海词](#)
- [同义词查询](#)

软件的话，全部设备推荐欧陆词典，原因是可扩展词库。关于词典的选择，参考[市面上最常见的牛津高阶英汉双解词典](#)，[朗文当代高级英语辞典](#)和[柯林斯COBUILD高阶英汉双解学习词典](#)有何特色？

### 语法

语法的话，赖世雄有本书好像不错。

### 运用

当学习到了一定的阶段，应该进行大量的输入过程，并持久化。输入的材料可以是TED或者初级的外文书籍。

- [网易公开课:TED](#)

## 学习资料

---

教材方面，我有看过新概念和赖世雄的教程。推荐赖世雄的教程，从发音到初级，中级，高级，循序渐进。

- [奶爸的英语教室](#)
- [漏屋：外语学习的真实方法及误区分析](#)

## 设计学习

---

能将自己的想法融入到作品之中，去影响，帮助到他人。是我一直的梦想。

这里摘录一份设计师自学指南：

本文译自国外高质量问答社区Quora，原文作者Karen X. Cheng，原是微软Excel的项目经理，后通过自学转型成为设计师。她讲述的自学过程详实细致且条理有序，读完会发现与想象中的大不一样，对于想自学设计但迷茫不知道从何入手的童鞋来说，可以遵循她的步骤去学习，除了规划得当，还能对设计有一个全局的了解：)

## 设计师完全自学指南

---

我不是毕业于设计学系，但我得到了一份设计师的工作。

我想换工作，成为一位设计师，但我没有四年和十万元美金让我回到学校上课，所以我决定自修。起初，我相当怀疑一个人到底能不能靠着自修来找到相关的工作。

答案是：可以

我边上班边自学了六个月的设计。虽然我觉得还没准备好，但我还是参加了许多公司的面试，最后我成功在一间很棒的新创公司 Exec 找到工作。

我知道我不比许多专业科系毕业的设计学生还要优秀，但我的能力已足以将我的工作做好。我会设计许多东西，例如网页、iPhone 应用程序、电子邮件、社群媒体和印刷品等。

### 一、学会观察

---

新手最容易犯的错误是直接上手Photoshop，要清楚学会Photoshop并不能让你成为设计师，就像你买了一套高质量的画笔也不会成为艺术家一样，从真正的基础开始吧。

#### 学一点绘画

1. 你不需要挤在一个画室里，跟一群艺术家一起画裸女
2. 你甚至不需要画得非常好，只要掌握一点基础，就能画一幅素描画
3. 你只有一件事，去学习如何画画，我看了很多关于绘画的书，这里推荐一本最好的：如何在30天内学会画画（小编找不到中文版，倒是有英文版PDF下载），然后每天花半个小时去练习画画，坚持一个月，会有惊人的效果。

#### 学习平面设计理论

1. 从《Picture This》这本书开始学起。本书虽以童话《小红帽》为例，却能教会你一切平面设计的基础。
2. 学会运用颜色、排版，以及栅格。若附近有开班教平面设计，就去报名吧。

#### 学习使用者经验的基础

坊间有许多谈论使用者经验的书，我推荐从这两本开始：

- 《The Design of Everyday Things》→ 中文版（设计心理学）PDF下载
- 《Don't Make Me Think》→ 中文版《不要让我思考》PDF下载

#### 学习如何写作

不要用废话来充版面。身为一个设计师，你的工作不只是画出美美的图片而已，你还必须是个优秀的沟通者。想想你过去的

一切经验，并且慎选每一个用字遣词。记得要「说人话」，而不是像学校写报告时拿来充字数的连篇鬼话。

1. 读《Made to Stick》→ 中文版《粘住》PDF下载。这是我这辈子最爱的书之一，它将教会你如何获得你专属的死忠读者。
2. 「Voice and Tone」这个网站充满了很多很棒的例子，教你如何与使用者沟通。

## 学习放弃自己的作品

这是最困难的部份。要作好心理准备，随时有可能必须像扼杀自己的孩子般销毁自己的作品。若能越快做好这样的心理准备，工作就能越快上手，因此若觉得成品不够好时，就随时砍掉重练吧。

找到公正的另一双眼：向懂设计的人询问看完你作品后的意见。若身边没有这类朋友，就去参加设计师聚会或相关活动上认识几个吧！

也要问问不懂设计的人，对你的作品有什么看法。让你将来的使用者试用看看你做的网页或应用程序。不要害怕问陌生人的意见，有一次班机误点，我就利用这个机会问问航厦中的其他乘客对于我设计中的应用程序的看法。他们大都乐意协助，我也获得了许多很棒的意见。

## 聆听

确实地聆听别人的意见，而且不要辩解。当你问别人意见，而对方愿意花时间和精力回答你时，不要用辩解来回报他们。相反地，你可以感谢他们，并且问他们问题，然后再自行考虑是否采纳他们的意见。

## 二、学会使用 Photoshop 和 Illustrator

---

耶！现在你在视觉设计与用户体验上已经有相当坚实的基础，可以朝下一步迈进了。事实上，我建议从 Illustrator 开始学，接着再去碰 Photoshop。设计师通常使用 Illustrator 来制作商标或图示，而 InDesign 则在制作传单或名片等印刷品时非常好用。

### 学习使用 Illustrator

有无数本书跟网站都在教怎么用 Illustrator，你可以挑自己喜欢的，以下是我自己很喜欢的：

- 《Adobe Illustrator Classroom in a Book》：这本书很无聊，但却非常实用。
- 《Vector Basic Training》：这本书会教你如何用 Illustrator 做出真的还蛮好看的东西。

好玩的来了，到以下的免费学习网站看看你学会了多少吧！这是我最喜欢的两个：logo 和 scenic landscape。

### 学习使用 Photoshop

网络上有成千上万的教学网站，但很多都很烂。幸好，还是有不少高质量的教学网站，例如 PSDTuts 就是其一。想学做 iPhone 应用程序的话，这里有很棒的教学；想学做网页的话，则看这个网站。

若每天挤出一两个小时来复习这些教学课程，你进步的速度将会连你自己都感到难以置信。

## 三、学会专业技能

---

你想设计的是应用程序、网页、还是信息图表？我建议在全部分尝试过后，选择其中你比较有兴趣的领域去钻研学习。

### 学习设计商标

1. 想学会设计商标，建议读《Logo Design Love》→ 中文版《超越LOGO设计》PDF下载。

2. 若连网站和名片都想一起学会，就建议读《Designing Brand Identity》。

## 学习设计应用程序

1. 从这个教学课程开始学习应用程序的视觉设计。
2. 读这本关于 iPhone 的书《Tapworthy》→ 中文版《触动人心：设计优秀的 iPhone 应用》PDF 下载，它能教你如何作出美观又实用的应用程序。
3. 好好研究你手机中的应用程序。你觉得哪边很棒，哪边又很糟？

## 学习设计网页

1. 读《Don't Make Me Think》，学习如何做出好读的网页。
2. 想做出好看的网页，就去读《The Principles of Beautiful Web Design》。
3. 列出你觉得很漂亮的网页，并找出他们的共同点。SiteInspire 上有很多不错的例子。难题来了：身为设计师，一定要懂 HTML 或 CSS 吗？我的答案是，依工作性质而定，但若懂这些肯定对工作加分的。网络上有非常多学习 HTML 和 CSS 的资源：
4. 我最喜欢的免费网站是 Web Design Tuts。
5. 我最喜欢的付费网站则是 Treehouse（每个月只要 25 美元）。假如你是从头开始学，而且希望有人为你详细解说一切，就去上 Treehouse 的教学课程吧。

## 四、建立自己的作品集

想成为设计师，你不需要去过学校上课，但你一定需要一个作品集。话虽如此，又刚开始学设计又没读过相关学系，要从哪里生出作品集来？告诉你一个好消息：你不需要真的处理过企划项目才能建立一份作品集，你可以列出以下成品：

1. 你为 T 恤所做的疯狂设计。
2. 替一个很丑的网站重新设计新风貌。
3. 设计一个 iPhone 应用程序。
4. 参加设计比赛
5. 寻找当地的非营利组织，提供免费设计服务。

另外记得：

1. 不要把每个作品都放进作品集，只要把最好的一些作品放进来即可。
2. 找灵感：先不必担心原创的问题，就像刚学新乐器时，一定是先学如何演奏别人的歌曲，最后才学做自己的歌曲。

## 五、找到一份设计师的工作

当我刚开始学设计时，我曾去过一个设计师工作坊，里面充满了想找工作的资深设计师，有好几位有过 5、10、甚至 15 年的工作经验了。想到必需跟他们竞争，我感到十分胆怯。然而一年后，我就成功找到了一份设计师的工作。我认为我比其他设计师多了一项关键优势：我懂得如何与软件开发人员合作。

去学一些交互式设计，或是基本的 HTML 和 CSS 语法吧！科技业的设计师（交互式装置、网页和应用程序设计师等）炙手可热且薪水很高。假如没有跟开发人员合作过的经验，可以参加 Startup Weekend、Hackathons。

最后，告诉你身边所有人你想成为一位设计师吧！谁都有可能成为帮你找到工作的贵人。

### 找到工作后也要持续学习

我已经在 Exec 工作了一年了，也从这份工作中学习到许多。我会去向比我资深的设计师学习、去找其他设计课程，像 TutsPlus 都是不错的在线课程，甚至翻遍书店中设计类的书籍。还有许多东西是我可以学习以及改善的。记得要不断磨练你的技能，并且不停学习。





## 前端学习

---

## ios开发

---

移动设备的开发，ios比较规范，设备的尺寸和性能都比较直接。因此想要开发移动设备，首推还是ios。

## Objective-C

---

OC是开发ios的必备基础。但只要你有过C/C++，以及一些面向对象的基础，学习起来没有那么难。只是语法和新特性上需要花些时间去了解。

### 在线教程：

- [官方文档:Programming With Objective-C](#)
- [stanford CS 139P](#)
- [Try Objective-C](#)
- [Cocoa:Learn Objective-c](#)
- [Learn Objective-C](#)

### 书籍：

- [Objective-C编程](#)
- [Programming in Objective-C, 4th Edition](#)
- [Objective-C基础教程](#)

### Reference:

- [Foundation Framework Reference](#)
- [Introduction to String Programming Guide for Cocoa: Objective-C字符串的使用。](#)

看苹果文档不要从参考(Reference)开始看，从指南(Guide)开始看。一般你看到一个具体类的时候，如果有相关的指南都会有链接直接跳过去的。

当对OC有了了解之后，请直接找一个iOS项目开始吧！实践检验真理！

## 入门ios开发

---

### 书籍

- [马上着手开发 iOS 应用程序 \(Start Developing iOS Apps Today\)](#)：官方的Guide。
- [iOS7初学者入门](#)：作者王寒，国内一位iOS游戏开发者，自己总结的，感觉不错，基础入门。
- [iOS Programming](#)
- [ios开发从新手到App Store上架](#)

### 在线教程

- [斯坦福大学公开课：iOS 7应用开发\(网易公开课\)](#)
- [standford: developing ios 7 apps](#)
- <http://www.raywenderlich.com/>：很多step by step的教程，适合掌握一定知识之后跟着教程做着玩。
- [code4app代码库](#)

## 论坛

- [v2ex](#)
- [Cocoa China BBS](#)

## 游戏开发

---

在ios下开发游戏，使用原生API操作的情况较少，多数会借助于其他框架。比如Cocos2d和Unity3D。

### Cocos2d

---

cocos2d最早是专门为oc设计的，用于ios的开发。但随着android设备的崛起，现在的cocos2d-x更加流行。cocos2d-x是国内触控科技主导的，用c++编写，在世界都具有很大的影响力，且cocos2d的原作者也已经加入cocos2d-x的开发之中。

### Cocos2d-Swift

这其实是最早的版本，之前名称叫做 `cocos2d-iphone`。使用OC编写，适用于开发ios游戏。配合 `SpriteBuilder`，可以方便快捷的开发游戏。

关于cocos2d-swift的教程不多，起码国内的我没找到什么。我的建议也是先去官方看[Getting Start](#)和[Cocos2D & SpriteBuilder Developer Guide](#)。

但是！我看完Getting Start也没发现我能干啥，我只能参考[How To Make A Simple iPhone Game with Cocos2D 3.0 Tutorial](#)的教程开始我的cocos2d之旅。

## 参考资料

---

- [Cocos2d-swift](#)
- [Cocos2d-x](#)
- <http://www.raywenderlich.com/>：很多step by step的教程，适合掌握一定知识之后跟着教程做着玩。
- [Learn to make iPhone games!](#): Cocos2d官方文档里推荐的一个网站，通过例子学习开发。

## 提升效率

---

我是个很在意效率的人，虽然可能执行力不那么强。但是能省的绝对不会浪费精力去做。

这里的效率也包含了如何获取信息，如何与人交流，总之是对个人有益的总结。

## 学会提问

---

这是我遇到的第一个提升效率的方式。

刚接触网络和论坛的时候，我常常去问一些浪费别人时间去回答的问题，比如：“如何XXX”，其实这类问题如果再花几分钟，或者再思考下都是很容易得出答案的问题。但是互联网给人的就是浮躁，很多情况下我们只学会了伸手去索取。

一个好的答案需要一个好的问题。学会提问，你会得到更多的信息。网络上有那篇文章：[学会如何提问](#)。我是没有读过，但是我理解它的意思。当你被别人问了几次基础或者无脑的问题，你就知道你希望得到的提问是什么样的。那么你其实也学会了如何提问。

## 参考资料

---

- [技术问答社区中回答的艺术？](#)

## 善用搜索

---

学会了如何提问，其实已经能把握住问题的中心，也就可以先使用搜索去尝试解决问题。

## 搜索引擎选择

---

通常我也会根据具体的搜索内容决定使用什么引擎，比如我要搜技术类的问题，肯定优先考虑google。搜索下载资源，优先考虑相应的下载资源网站。做到缩小搜索范围，才能让搜索结果的质量得到提高。

其实配合上 `site` 的用法，在google上也能很快锁定搜索结果。

常用的几个站点：

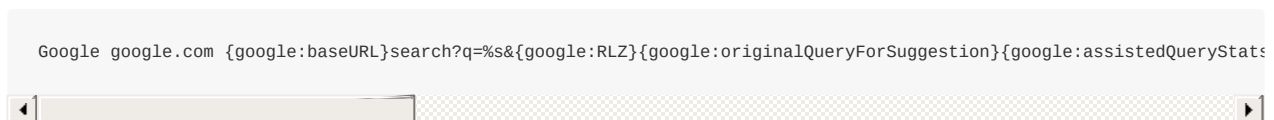
- [google](#)
- [baidu](#)
- [Duck Duck Go](#)
- [海盗湾](#)
- [天天美剧](#)
- [豆瓣](#)

## 设置Chrome的搜索

---

Chrome是我最常用的浏览器，大部分搜索我都是在它之中完成的，那么如何节约时间呢。首先了解一个快捷键：`command+l`，快速定位到地址栏。

其次借助Chrome的地址栏搜索功能，快速搜索。中文版默认的可能是百度搜索，我比较喜欢设置google搜索，打开 `Setting->Search->Manage search engines`，修改default search为：

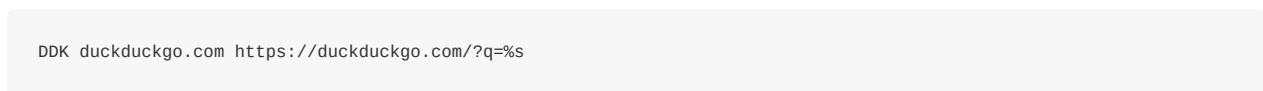


如果设置了代理的，可以先访问[google.com/ncr](http://google.com/ncr)选择为通用版本。确保访问到页面是通用版本，在地址栏里搜索试试。如果还会自动转向到代理的国家，Chrome也会提示你继续使用还是切换为google.com。同时需要设置 `Setting->Currently showing search results in` 为English, 中文(简体)。

如果只选择结果为English,那么挂上日本的代理，出现的结果优先会是日文。

## 其他搜索设置

在Chrome的地址栏中，也可以设置快捷键触发其他搜索，在 `Setting->Search->Manage search engines`，添加：



类似的规则，在地址栏中输入duckduckgo然后按一下tab，就可以选择自定义的引擎进行搜索。

## 搜索技巧

---



直接给一张图解释:

## 从谷歌搜获更多

Get More Out of Google

一些提示和技巧  
教你用上没用的谷歌的

对于学生时代的你来说，每天从海量的搜索结果中筛选出最相关的，让人头疼的不仅仅是搜索本身，更重要的是如何从海量的搜索结果中，快速准确地找到你想要的信息。谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

以上就是我们为你准备的一些谷歌搜索技巧，帮助你更高效地使用谷歌搜索。

---

### 如何谷歌

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 你想搜什么

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 怎样谷歌呢

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

---

### 谷歌学术搜索

谷歌学术搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 你想搜什么

谷歌学术搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 怎样谷歌呢

谷歌学术搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

---

### 其它谷歌技巧

#### 字词定义

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 数学计算

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 单位换算

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

---

### 快捷键

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 在本页查找

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 放大/缩小

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 选中浏览器地址栏

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 切换标签页和程序

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

---

### 进一步提示

#### 善用你的图书馆网站

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 不要引用维基百科

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

#### 找参考文献

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

---

### 中文版译制

谷歌搜索可以帮助你快速找到你想要的信息，但如果你不知道如何正确使用谷歌搜索，那么搜索结果可能会让你感到困惑。

HackCollege.com

References

## 参考资料

---

- [Google语法详解](#)
- [搜索引擎有哪些常用技巧？](#)
- [如何用好 Google 等搜索引擎](#)

## 如何写作

---

关于写作，我们为什么要写作，这是一个前提。

## 为什么写作

---

无论是否是技术人员，我觉得都应该坚持写作。写作带给你的是思维的总结，因为有些事情你只是去想，貌似是很简单。当你去深入思考，其实又是另一个境界。我觉得写作能帮助到我最大的一点是，理清思路。

关于其他，我也比较推荐刘未鹏的两篇博客：

- [为什么你应该（从现在开始就）写博客](#)
- [书写是为了更好的思考](#)

程序员是理科出生，写文章是文科的特征。但是程序员养成写作的习惯，对编程也是很用处的，比如：

- 写文章和写代码最核心的共同之处在于它们都需要清晰思考的能力。好代码与好文章一样都需要言简意赅，不然就会浪费 CPU 资源或人的精力。
- 软件工程师应该写作因为现在开发协作变得越来越重要。不论是 GitHub 评论、代码注释，还是技术文档都需要清晰准确的文字。良好的写作能力方便了人们之间的交流，使得项目更好地运行下去。
- 即使没人读你的文章，写作的过程也是有益无害。它帮助你理清思路，明确对于某个问题的看法，加强或削弱你的某种观点。把凌乱的想法汇聚成精简的文字非常有价值。

## 用什么写作

---

Markdown是我极力推崇的一种通用格式。

但是目前存在多种解释器，导致语法有些混乱，但是熟悉标准的语法，依然能够写好文章。

说几点常用的：

- 删除线: 需要删除的内容前后添加 `~`。
- 强制断行：正常，回车后形成的断行，不会被 Markdown 解析为断行，需要在断行前面段落后加两个空格或者写入 `<br/>` 标签。
- 水平分割线: 三个以上的星号，减号，或者下划线，除了空格，不允许出现其它符号。 `-----`。

## 在哪里写作

---

小时候我们常常使用的是笔和纸，形式多为日记。其实日记并没有什么不好，但是大多数人应该和我应该，会把日记写成流水账，没有太大的意义。

现在是互联网的时代，手机和电脑可以替代我们完成输入，并且记录的形式也多样化，除了文字还可以记录声音，图片和视频。

如果是个人的记录，我比较喜欢使用云笔记，比如印象笔记或者为知笔记。设置一个访问密码，以防无意被他人阅读。

技术性的文章，我更喜欢以博客的形式去发布，一方面会有知识积累的成就感，另一方面也可以与他人交流。博客可以自己搭建，比如github+jekyll,或者使用简书这样的web服务。

## 参考文档

- [Markdown 标准格式](#)
- [为什么软件工程师应该养成写作的习惯？](#)
- [You Should Write Blogs](#)
- [技术文章的写作技巧](#)

## 如何翻墙

---

既然想学点东西，就不能被网络束缚住。国内的网络环境，并不利于我们成长。

## 什么是墙

---

[great firewall](#), 中国特有的。就是国家对网络的封锁。想要看看外面的世界，就得翻墙。

## 如何翻墙

---

翻墙的服务一般都是收费的，免费的除了goAgent，其他都不要尝试了，不然只是在浪费生命。我比较提倡用钱能解决的问题，尽量还是花些钱吧。

目前来说，我了解的方式主要有两种：VPN和Shadowssocks。

## VPN

---

在公用网络上建立专用网络，进行加密通讯。原理是在一个国外网络无阻的机器上与本地机器通信。

优点是方便设置，网络速度稳定。

缺点是稳定的服务价格不便宜，且本地流量全部走的是代理。

### 配合VPN实现国内外分流访问

“分流”，这个词不知道说的准不准确，意思就是实现对不需要翻墙的网站直接连接，需要翻墙的网站走VPN代理。这样国内的网站访问不受影响，又能正常访问国外站点。

实现方式就是修改系统的路由表，网络上有这样的开源项目，专门收集国内被“墙”的网站。利用这些数据，让vpn客户端在进行连接的时候自动执行。

通过这些路由脚本，可以让用户在使用vpn作为默认网络网关的时候，不使用vpn进行对中国国内ip的访问，从而减轻vpn的负担，和增加访问国内网站的速度。

#### Mac设置：

1. 下载附件mac.zip, 并将其中的ip-up, ip-down两个文件放入到 /etc/ppp目录中。
2. 开终端，执行命令 `cd /private/etc/ppp`, 进入/etc/ppp目录下。
3. 在该目录下执行`sudo chmod a+x ip-up ip-down`
4. 好了，可以连接VPN试试了。

#### Windows设置：

1. 下载附件windows.zip, 并将其中的vpnup.bat, ivpndown.bat两个文件解压到任意目录。
2. 右击vpnup.bat, 选择以管理员权限运行，然后会弹出cmd窗口，等待运行完毕，窗口会自动关闭。（Windows需要每次开机时执行一次）
3. 好了，可以连接VPN试试了。

测试方式:连接VPN后，可以访问ip.cn查看当前的ip,如果显示南京就对了，然后再打开youtube.com看看能否访问。

一点说明: ip-down和vpndown.bat是恢复路由表的脚本。

## Shadowsocks

---

shadowsocks的出现,我觉得真是一大利好。它解决了网络流量分配的问题,借助代理插件,可以方便的实现只针对需要翻墙的网站都代理访问。且价格便宜。

一般情况下,我使用Chrome的插件: [Proxy SwitchySharp](#)。

## chnroutes

---

[chnroutes](#), 此项目通过修改路由表,解决通过VPN链接网络时的流量转向问题。达到不需要翻墙的申请走正常网络,需要翻墙的请求走VPN。

## ChinaDNS

---

[ChinaDNS](#), 此项目解决的是DNS污染问题,我还没搞明白。。。

## 参考资料

---

- [Chnorutes](#)生成的脚本下载地址: 需要翻墙。

## 时间管理

---

时间就是金钱，如果能把握时间，你就掌握了自己的命运吧。

### 感受时间

---

大学的浑浑噩噩，上班之后的混水摸鱼。我们只觉得到了时间过的好快，但我们真的感受过时间吗？所以在谈管理时间之前，我想先感受下时间。

你可以停掉手中的工作，远离电子设备，远离浮躁的互联网，静静的待上三分钟。我喜欢用打坐的方式去感受时间，放空自己，让自己清醒。我觉得只有大脑清晰，你才能高效，你才能把握时间。

有人说互联网的信息垃圾很多，没有营养，我们就是为了吃饱，才会去吃更多的垃圾。所以何不换个方式，试着去体会呼吸的节奏，让自己多吸收些营养。

### GTD

---

GTD是英文“Getting Things Done”的缩写。是一种高效的管理时间的方式。个人感受就是划分任务，把重要的先去完成，专注一定的时间去处理一些事情。然后再休息，再继续专注，周而复始。

通常每一天，最好的方式是在早晨花上一定的时间规划一天的安排。

专注时间的方式，比较有效的是[番茄工作法](#)。

### 音乐

---

这里提到音乐，是因为有时候我喜欢带上耳机，隔绝外界的环境。

音乐类型我比较推荐白噪音，大自然的声音或者雨声。会很容易让人安静，专注起来。如果你喜欢音乐，我觉得也可以单曲循环一首歌曲。记住，听歌不是为了分神，而是为了更加专注！



## 知识管理

---

当你有时间，有计划，有效率的去工作生活的时候。你会涉及和吸取很多的知识。我不觉得有人能记住所有的信息，所以对知识也应该做一些管理。

对于知识，我也比较推荐"IPO"的形式，就是input->process->output。摄入的知识并不是你的收获，而只有output出来的内容才是自己的。

### Input

---

在网络时代，我对知识的吸收主要通过web。所以我会通过微博，博客，新闻的形式收集信息。

#### 微博

可以建立不同的分组，里面关注对于的账号。这样每天差翻阅翻阅，就能了解大致的内容。

#### RSS

我喜欢这个已经被google淘汰的技术，可以方便我去订阅自己感兴趣的网站，以及个人博客。目前我使用的客户端是feedly。

#### 网络书签

在网络上看到好的内容，我们需要去收藏。一般情况下只是对书签进行保存。

保存的方式可以放在Chrome的书签中，使用google账号同步。我个人喜欢第三方工具，比如pinboard，它可以对书签打上标签和描述，方便搜索。

### process

---

### output

---

## 文件管理

---

需要管理的文件，通常包含了一些用来同步的配置，有用的软件和照片等。

### 工具

---

#### BitTorrent Sync

BitTorrent Sync，是一个可以自己搭建p2p服务的同步软件。主要的意思就是可以自定义网盘，且文件大小不受限制。目前我发现的一个缺点是无法设置忽略文件。这导致很多隐藏文件也会备份。。。

#### 百度网盘

这个主要是网盘大，速度还不错。

#### 金山快盘

同步盘，主要同步一些配置文件。

#### Dropbox

最好的网盘肯定是dropbox,之前看到有人这么评价过：

试了一圈国内的网盘，发自内心感叹Dropbox的牛X。Dropbox不仅可以在服务器上通过客户端同步，同时也可以选择创建应用同步，Dropbox在创建应用时就可以为自己生成access > token。

网盘	无法支持的原因
酷盘 <a href="#">文档</a>	无增量接口
百度PCS <a href="#">文档</a>	有增量接口，但申请一周后仍未有进展
快盘 <a href="#">文档</a>	无增量接口
新浪微盘 <a href="#">文档</a>	有增量接口，但应用的沙箱的文件不出现的用户网盘中。

## 照片

---

### 文件命名

之前在一个podcast中听到的方式，文件先按照设备去划分，二级目录使用日期去划分。这样的好处是保留原片。如果需要share的东西，可以拷贝原片再按照类别划分。

### 文件保存

通常instagram的照片我通过IFTTT保存到了dropbox中。

手机的照片我尽量定期拷贝到电脑中。

单反的照片，每次使用完都拷贝到电脑中，然后格式化相机的存储卡，因为相机的存储空间还是不太大的，不适合长期保留文件。

原片存储好之后，一般会进行一次筛选，删除一些废片。这其实是一项巨大的工作，至今我2014上半年的照片还没有整理

完。。。



## 健康生活

---

好的身体，好的体力才能支撑你的行为，才会更加有效率。

### 作息睡眠

---

高质量的睡眠能带来好的体力，这才是效率的保证。

晚上9-11点为免疫系统（淋巴）排毒时间，此段时间应安静或听音乐

晚间11-凌晨1点，肝的排毒，需在熟睡中进行。

凌晨1-3点，胆的排毒，亦同。

凌晨3-5点，肺的排毒。此即为何咳嗽的人在这段时间咳得最剧烈，

因排毒动作已走到肺；不应用止咳药，以免抑制废积物的排除。

凌晨5-7点，大肠的排毒，应上厕所排便。

早上7-9点，小肠大量吸收营养的时段，应吃早餐。

疗病者最好早吃，在6点半前，养生者在7点半前，

不吃早餐者应改变习惯，即使拖到9、10点吃都比不吃好。

半夜至凌晨4点为脊椎造血时段，必须熟睡，不宜熬夜。

### 为什么失眠

我睡不着的多数情况是觉得有事情没有干完，比如刷新闻，比如想明天干嘛等等。

其实想要入睡，我们需要放开那些电子产品，需要告诉大脑：现在是休息的时间。想的再多，耽误的还是自己的睡觉，形成的就是恶心的循环。

### 如何入眠

学会控制自己，放松自己。这是我从廖阅鹏的《每天用一点神奇催眠术》中学到的方法。

当身体得到休息的信号，放空大脑，让身体的各个部位方式，直至大脑。最近我看到的一种说法是眼睛最耗费我们的精力，放松眼睛，身体也能得到很好的休息。

### 枕头的选择

既然是为了改善睡眠，睡个好觉，那么首先就要看你到底是什么情况：

1. 睡眠质量不错：可以试试带回弹的类型。
2. 打鼾，睡醒头疼/头晕：颈椎可能有问题，试试乳胶B型枕，或者慢回弹B型枕吧。

品牌的话，我推荐[眠趣](#)。

### 睡眠时间

---

不同年龄段，所需的睡眠时间也不尽相同。美国国家睡眠基金会推荐的睡眠时间如下图。对照一下，你的睡眠时间达标了

么？

- 0~2个月：12-18h
- 3~11个月：14-15h
- 1~3岁：12-14h
- 3~5岁：11~13h
- 5~10岁：10~11h
- 10~17岁：8.5~9.25h
- 成年人：7~9h

## 睡眠质量

---

此外，也不能追求睡眠时间的“数量”，而忽视睡眠质量。高质量睡眠的一般特征是：1.半小时内入睡；2.整夜基本不醒。

最佳睡眠时间：晚**10**点至早**6**点。这一点对于IT业的和创业初期的朋友来说，可能比较困难。

## 强身健体

---

## 参考资料

---

- [体质极差的人如何从头开始恢复身体素质？](#)
- [凌晨3点半不想睡但又必须睡怎么办？](#)
- [知乎周刊: 真的睡不着](#)
- [熬夜的南京人，你们还能活多久？太恐怖了！](#)
- [详解熬夜对人体的各种危害](#)
- [如何提高睡眠质量？](#)

## 制作视频

---

既然我这么喜欢分享，图文是一种方式，视频不更直观嘛。周末抽个时间学习一下如何制作视频，特别是能吸引人的视频。

## 与众不同

---

授人以鱼不如授人以渔，如果要教育好别人，是替他人开发心智，让他人学会自主学习，不断提升。前几天我也看到 @iBuick 说到，“我对 OS X 图书的看法，早年间，特别热衷各种技巧介绍，各种 defaults write 觉得了不起。后来觉得重要的在应用层面，各种 App 走起。现在觉得，写操作系统本身的书最重要，你把这东西弄明白了，给读者讲明白了，他们看完书以后会自己动手解决问题了，才是最重要的。”

我个人也很看好在线教育的发展，在线教育要的是研究，优化，抽离，信息化教学习惯和流程。

保持课程的与时俱进，让你的课程与众不同。

## 课程大纲

---

确定了录制的方向，就要确定内容大纲。关于大纲，我觉得是做任何事都需要的一个思维习惯，很有效率的一件事。

### 大纲的要素

- 谁适合学
- 学什么内容
- 学完了可以做什么
- 老师是谁
- 如何学
- 学习周期与频率
- 具体课程与章节

课程以一个向导系列的课程，可以将很多内容拆分为小课程，每个章节对应的课程时间应该也把握在25min以内。利用好零碎的时间，现在的人都比较浮躁。

麦子网的备课录课的比例，大约是3:1。所以录制出一小时的视频，其实是需要花费3-5小时的。

## 视频录制的规范

---

### 视频分类

视频可分为三大类：

- 软件操作的课程，主要是电脑的界面录制
- 硬件操作的课程，主要录制的是操作台
- 非技术类的课程，主要是录制人，以及后期的剪辑

### 操作系统的分类

- windows
- MacOS
- Linux

## 视频分辨率

早起为1024x768,现在主流为1280x720, 1280x800。

## 录制环境

- 电脑的选择
- 麦克风：主要测试一下噪音等问题
- 分贝增强的设置：最好不要设置，防止出现电流声
- 录制的背景音：最好在按键的环境下，避免嘈杂的背景音。
- 准备手写板：电子白板。
- 录屏软件：不同的操作系统软件不同。

## 录制的时长

互联网的时代是碎片时间，最好是5-25min，方便他人利用零碎时间观看。

## 视频之间的起承转合

---

- 视频开始的时候要再说一下subject
- 视频结束要布置homework
- 课程资料以及demo等需要和视频一一对应

## 备课工具和形式

---

- PPT是一种方式，但是需要和演示的内容经常切换。
- 云笔记之类的随时敲打也是一种方式。
- 思维导图：这是一种很好的方式，能够清晰的梳理好我们的知识点和思路。包括够花一些草图也很方便。Mindjet是一款很好的思维导图软件，各平台在都有。操作确实很漂亮，但是收费也很高。
- 绘画笔或者电子白板：直接在屏幕上进行一些标注。Mac的电子白板不是很多，选择的时候需要注意。

## 录屏软件的使用

---

### Windows下

### Mac OS下

推荐使用ScreenFlow。这个软件的使用过程，需要注意的是音量的设置，以及背景环境音。录制完成即可预览。

这个软件录制完成，会自动将音频和视频分开。

压缩软件的话，推荐使用HandBrake。记得勾选web optimized选项。

### Linux下

## 后期的处理

---

- 如何删除内容，将录制出错的地方删掉。
- 裁剪区域：选择合适的播放内容范围。比如可以截取除去顶部菜单，或者底部内容的区域。
- 音频的控制，调制音频的大小，使得清晰。



- 导出的格式：适合web的，比如H.264的MP4格式。
- 视频压缩：投放到网页或者与人分享的时候，文件较小一些便于传输。

## 录课的技巧

---

- 需要注意的是，录课和现实的讲课是有区别的。现实的课堂是有学员的，录制课程可以假象有一堆学员，保持积极的态度。
- 形成自己的风格，养成自己的习惯。有清晰的思路，有合适的Homework，方便考察。
- 线上的标注，使用电子笔进行标注可以更清晰。
- 养成课程的竞争优势，讲解出与众不同，有特点，更专业的内容。
- 音频的深入，可以使用专业的录制设备。
- 可以使用专业的剪辑和后期。

## 参考资料

---

- [麦子学院：如何录制好在线视频](#)



## 论音乐对效率的影响

---

我相信很多人都有在学习或者工作的时候听音乐的习惯。那么音乐对我们有帮助吗？应该是有的，起码我知道胎教的时候应该多听听古典音乐。

### 白噪音

---

最初的时候，我是杂食，很喜欢在学习的时候听流行音乐。不自觉的就会被曲子，或者歌词带入到另外一个世界里。工作之后，有时候办公环境很嘈杂，也会把你的思绪打乱。所以我又必要找一种能让你迅速进入状态的音乐，其实也就是培养一种习惯，条件反射而已。

应该是今年（2014）上半年的时候，我看到了一些国外的应用，有一些专门利用环境音乐的app，比如雨声，雷声，鸟鸣声。后来搜索了下，原来这叫做白噪音。

## 巴洛克超级学习音乐和罗扎夫记忆音乐

---

内心里我觉得这一类肯定是假的，寄托于音乐帮助你学习是不可能的。但是能让你专注的音乐，我觉得是可以尝试的。

### 参考资料

---

- [白噪音](#)
- [阿尔法脑电波](#)
- [巴洛克音乐](#)
- [巴洛克超级学习音乐和罗扎夫记忆音乐是一种安慰剂效应](#)

## SOHO

---

Small office/home office (or single office/home office; SOHO) refers to the category of business or cottage industry th

我觉得SOHO的工作形式，是未来的发展的一种趋势，但不是所有的人都适合的形式。在我能遇见的时间内，我觉得这是Geek生存的一种方式。但这也需要很大的自制力和执行力。

## 参考资料

---

- [awesome-remote-job](#): A curated list of awesome remote jobs and resources.

## Hacker

---

### 参考资料

---

- [知道创宇：余弦](#)

## 保护隐私

---

### 参考资料

---

- [养成哪些上网习惯可以避免泄露重要的个人隐私？](#)

## 常用软件

---

"工欲善其事，必先利其器"。

在这样的章节中，我想说说如何高效的利用好各个系统。

## Windows篇

---

通常来说，windows应该是最先接触到的系统，也是除了Mac笔记本外最常见的预装系统。



## 硬件配置

---

好马配好鞍，操作系统也应该在一定性能的机器上才能发挥更大的作用。

一般来说，如果别人让我推荐电脑，我会按照对方的需求去推荐。

### 办公型

---

就是简单的处理文档，浏览网页，逛逛淘宝的。

这类的我比较推荐轻薄的，便携和续航较强的电脑。我觉得chrome book就挺好。。。

### 游戏型

---

Dota，魔兽这类的，内存和显卡有个保障就行。

大型游戏，最好还是显卡强劲一点。

### 工作型

---

比如程序员这种类型，这里还可以对前后端划分一下。

前端开发主要是以浏览器为主，偶尔开虚拟机测试。能够书写代码就行，脚本语言都不会设计到什么编译，性能不必太高。

后端开发设计语言的编译，数据库的运行，本地服务器环境的搭配。最好还是性能强劲一些。

### 什么叫性能强劲

---

建议大家去中关村之类能搜索电脑价位的网站，普通电脑价位倒叙搜索看一看。CPU差不多就那几款，CPU的性能提升个人感觉远不如内存和SSD带来的提升大。

所以有钱的话，建议投入内存和SSD之中，触屏的笔记本目前来看没有必要。

### 关于外设

---

我推荐投入：

- 显示器：最为重要，直接和效率挂钩，推荐24寸以上。戴尔不错。
- 鼠标：最好有多功能，无线Mini接收器。罗技不错。
- 键盘：只要手感舒适就行，机械键盘我用过茶轴（介于青轴与黑轴之间），准备尝试红轴。
- 音响：耳机也行，能隔绝外界的环境，沉浸于属于你自己的世界。

## 系统安装

---

通常我不是很喜欢预装的系统，并且预装的机器C盘都比较大，浪费空间。所以还是有必要重新安装下系统(我安装的是盗版，我鄙视自己)。

## 系统下载

---

系统我比较喜欢下载纯净的版本，一般我会从这里:[MSDN我告诉你](#)中下载需要的版本。

我自己也有存储一些镜像文件到网盘之中：[windows系统镜像](#)。

## 如何安装

---

### 硬盘安装

此种方法适合能正常开机的机器，无需借助U盘就可以安装新系统。使用的是NT6这个软件，异次元有一篇文章讲解的很清楚：[NT6 HDD Installer 使用教程 - 在没有光驱U盘情况下直接通过本机硬盘重装系统 \(支持Win8/Win7等\)](#)。

需要准备的东西就两样：

- 系统镜像文件
- NT6安装文件

然后找一个非系统盘，格式化并将系统镜像文件和NT6安装到此盘符的根目录下。通过NT6重启进入安装。安装过程我一般会格式化C盘，待安装完成之后再格式化其他盘符。

### 引导盘启动

此方法适合无法进入系统的情况，一般需要一个U盘制作为启动盘。制作启动盘的方式有：

- [WinSetupFromUSB](#)
- [大白菜](#)

由于我已经完了这种方式如何使用，特别是PE工具格式化等操作。请大家自行搜索。

## 硬盘分区

---

我也忘记了:D，等我用到了再说吧。

## windows软件

---

记录是给自己的回顾和总结，也是给别人的一种分享。所以我想说说使用Windows的经验。

为了方便自己系统安装，我会把一些安装文件定期更新到网盘中，而常用的一些软件以及配置文件放在可同步的网盘中。

## 常用软件

---

常用软件就是装机必备啦，说说我常用的软件：（待添加下载地址）

- **Chrome**: 主力浏览器，使用google账号登陆并且同步。
- **Office**系列: 其实我用WPS多一些，或者google文档。
- **MSE**: 微软官方的杀毒软件。
- **CCleaner**：小巧方便的清理软件。
- **腾讯电脑管家**: 有时候我就是用腾讯电脑管家替代杀毒软件加清理软件。
- **输入法**：我就用原生的，不折腾，不弹框。
- **PotPlayer**：影音播放器。
- **FastStone Image Viewer**：图片查看工具。
- **有道词典**：方便取词，高端一点我就用欧陆词典。
- **欧陆词典**: 无需插件取词，无广告，可自定义扩充词库，替换有道词典。
- **福昕阅读器**：最好的PDF阅读器。
- **foxmail**：邮件客户端。
- **迅雷**：下载工具。
- **BitTorrent Sync**: p2p同步服务，可实现多设备网盘同步功能。
- **7-zip**：压缩/解压缩工具。
- **Flash**: Flash播放器，最好下插件，以及独立播放器。
- **f.lux**: 根据日出日落去调节色温，保护视力。
- **HexChat**: IRC Client, 沟通工具，程序员用的多。

有了这些，差不多使用windows就没问题了。下面我们再谈一谈如何更好的使用windows。

## 提高效率的工具

---

- **launchy**: 快速开启工具。我还设置了快速web搜索。
- **strokesplus**：全局鼠标手势工具，占用内存小，且支持LUA编程。
- **Wox**: 暂时未使用，和launchy类似。
- **MasterSeeker**: 全盘文件搜索工具，搜索可选择项比Everything多。
- **Everything**: 全盘文件搜索工具。
- **Listary**: 目录搜索，操作工具。
- **ditto**: 历史剪切板工具，可设置快捷键为 alt+v。
- **VistaSwitcher**: 程序切换软件，特别方便的是提供了`alt+`的切换，类似mac下的操作。使用方式就是按住alt+tab切换，也可按住alt+tab后，松开tab按数字键快速切换。
- **PicPick**: 截图，标尺工具。
- **Clover 3**: 资源管理器扩展工具，使得类似chrome，带书签功能。
- **Q-Dir**: 由于Clover在windows8上经常崩溃，我又不喜欢TotalCommander，发现这个也不错。
- **NetSetMan**: 方便切换ip/dns, 适合经常切换Home/Work环境。
- **AutoHotkey**: 可自定义操作的脚本，还未使用，据说功能强大。
- **teracopy**: 据说复制大文件时候速度更快。
- **chocolatey**: 类似Unix下的 apt-get 命令，安装软件。
- **joytokey**: 可以使用手柄模拟鼠标的软件。

- **XMind**: 脑图工具。

## 开发者必备工具

---

这里可以单独开一章节了，但是我先试着在这列一下吧：

### 开发环境

开发环境下，我会配置好常用的语言，python, ruby, nodejs等等。并且最好安装上Virtual Studio，很多需要编译的环境依赖其中的VC++。

- **gow**: 扩展一些常用的\*unix命令。
- **ConEmu**: 可代替cmd的工具，界面舒服，内置Clink。
- **Clink**: 支持命令行下粘贴复制。
- **tdm-gcc**: windows下的C/C++编译器。
- **git**: 版本管理工具，安装后可命令行使用。
- **sourcetree**: Git的图形管理工具。
- **svn**: 一般公司用的比较多，也是版本管理工具。
- **Koala**: 前端预处理语言编译器。

## 文件编辑工具

---

- **SublimeText 3**: 最方便的编辑器，插件也多。
- **Beyond Compare**: 对比文件工具。

## 开发辅助工具

---

- **charles**: 代理工具。
- **texturepacker**: 处理动画图片工具。
- **FileZilla**: FTP传输工具。
- **Xshell 4**: 远程连接服务器工具。
- **TeamViewer**: 远程操作电脑工具。

### 设计

- **亿图**: 除了制作思维导图，还提供流程图，UML等。
- **licecap**: gif录屏软件。
- **camstudio**: 录屏软件。

## Bat文件

---

如果有一些命名是自己常用的，不妨整理到一个 `bin` 目录中，并添加到PATH中。将一个命名包装为bat文件的形式如下：

```
@echo off
%-dp0\nant-0.92\bin\NAnt.exe
```

`%-dp0` 可指代当前目录。

## 参考资料

---

- [Windows 下有什么软件能够极大地提高工作效率？](#)
- [2014 年 Windows 平台软件推荐：这些工具都很有用](#)
- [我最喜欢的软件windows版](#)

## Mac篇

---

终于换了Mac了。

除了按键不习惯，其他都很好！貌似把键盘的ctrl修改为command会比较好。

## 通用设置

---

想要一个系统顺手，还是要做一些适合自己的设置。

## App Store

---

应用商店最大的问题有两个：

- 下载失败，提示"使用已购页面再试一次"。
- 下载速度慢。

这两个问题常见的解决方式就是设置DNS为114.114.114.114。但是貌似这个下载应该还和apple的服务器解析有关，有时候还是会抽风。

## 快捷键

---

- 屏幕切换，command+数字键。
- 截图：shift+command+4。
- afred2: ~~alt+s~~，已换成 CapsLock 。

## 显示器设置

---

设置显示器屏幕不出现菜单栏，取消 System Preferences -> Mission control -> Display have separate Spaces 即可，需要重新登录。

## 触控板设置

---

添加按住ctrl加滚轮缩放屏幕： System Preferences -> Accessibillity -> Zoom 。

增加触控板的灵敏度和双击拖拽功能： System Preferences -> Accessibillity -> Mouse & Trackpad -> Trackpad Options 。

## 参考资料

---

- [mac-setup](#): 外国人写的，介绍mac的使用。
- [Mac技巧索引](#): 《TacTalk人生元编程》作者整理。
- [Mac攻略](#)
- [Hacker's Guide to Setting up Your Mac](#)
- [Mac开发环境设置](#)
- [Mac OS X Setup Guide](#)

## Mac软件

---

换了大Mac之后，我首先想到的还是装一些常用软件。

mac的软件安装不同于window，一般直接从app store里搜索下载。或者去软件的官网下载dmg格式安装，或者app直接拖到application再安装。

## 常用软件

---

- **Chrome**: 官网直接下载。
- 搜狗输入法: 联想能力比较出众。
- **欧陆词典**: 不知道为什么，官方和app-store里的版本居然不一致，官网下载的也是新版本，只要能买到注册码，一样激活。
- 金山快盘: 国内的限制，被迫放弃dropbox。
- **dropbox**: 为了同步1password, 主要用于手机端内容同步。需要翻墙。
- **QQ**: 对windows的版本对比，真心简洁。
- 迅雷: 我发现到了mac下都变得简洁了。
- **MPlayerX**: 播放器，据说解码能力强。
- **shadowsocks**: shadowsocks-ios版本默认支持自动代理模式。
- **iStat Menus**: 查看系统状态，磁盘，CPU，温度等状态的工具。
- **BitTorrent Sync**: p2p同步服务，可实现多设备网盘同步功能。
- **Homebrew**: mac下用于安装命令行下工具的apt-get。
- **Homebrew cask**: mac下用于安装应用的apt-get。
- **为知笔记**: 个人觉得最好的云笔记。
- **Mou**: 最好的markdown编辑器，但是我更习惯ST3编辑。
- **CleanMyMac 2**: 电脑垃圾清理软件。用的盗版，对其他软件有损伤，初次用来除去系统多语言还是不错的，但是话又说回来，系统语言又不占多少大小。。。
- **Keka**: 正版购入，方便的压缩工具。
- **Spillo**: pinboard书签服务客户端，MAS购入，68RMB。
- **ReadKit**: 一站式阅读工具，MAS购入，68RMB。
- [Day One]: 书写类工具，日记软件，支持Markdown。
- **Fantastical**: 日历软件。MAS购入，68RMB。

## 系统相关

---

- **\*\*Startupizer 2**: 管理自启动项工具，可根据日期或标签设置不同启动。MAS购入，68RMB。
- **MacUpdate Desktop 6**: app安装更新工具。我觉得可以替换brew cask了吧，起码能看什么软件有更新。下载资源包括了MAS和直接下载。
- **AppCleaner**: app卸载工具。
- **DaisyDisk**: 显示磁盘状态的工具。
- **OptimApps**: 系统优化工具，包含三个功能。
- **F.lux**: 根据日出日落调整屏幕色温，保护视力。个人喜欢设置4000-5500。
- **caffeine**: App Store下载，免费。取消自动休眠的功能。
- **Yolink**: 临时存储文件或内容的工具，感觉不是很必要。
- **Timing**: 付费软件，统计 Mac 使用习惯，每天做了什么。
- **fliqlo**: 时钟屏保。
- **CheatSheet**: 显示快捷键操作。

## 提高效率的工具

---



- **Hider 2**: 隐藏系统文件的工具。
- **xtraFinder**: Finder的插件。
- **alfred 2**: 效率神器/快速启动。一定要购买powerpack配合使用哦！
- **1Password**: 最佳密码管理工具。
- **TextExpander**: 最佳输入辅助工具。
- **Manico**: 付费软件, 通过option快速切换应用程序。
- **Karabiner**: 修改按键映射。
- **Seil**: 配合karabiner,修改CapsLock映射。
- **AutoKeyboard**: MAS免费购入。
- **BetterTouchTool**: 自定义手势操作, 以及鼠标, 按键等。最明显的帮助是让我的鼠标中间左右切换起作用了。
- **ShortCat**: 快速移动鼠标软件, 默认 `shift+command+space` 查找, 按住 `control` +对应字母进行快速切换。
- [Moom]: 付费软件,68元,其实sizeup也是付费的。但我更看好moom的拖放, 和自定义窗口大小。
- **Bartender**: 官网下载, 付费软件。管理右上角menu bar图标的软件。
- **SynergyKM**: 可以使多台设备共用一套键鼠。
- **popClip**: 付费软件, 30元, 文本选择辅助工具, 高效。
- **ClipMenu**: 付费软件, 6元, 剪切板历史管理工具。不如windows下的ditto好用。
- **XMind**: 思维导图软件, 先用这个免费版本, 熟悉熟悉。
- **jitouch2**: 触控板辅助工具。删除的原因是全屏的手势比较容易误操作, 且不一定所有程序都支持。
- **sizeup**: 窗口管理工具, 类似的还有moom, divvy。这软件开发的公司其他作品也都很高效。

## 设计相关

---

- **Sketch**: Evernote出品, 截屏、标注工具。
- **LilyView**: 图片查看工具。
- **1000 Open Type Fonts**: 字体相关软件。
- [SnapRuler]: MAS购入, 68RMB, 标尺以及截图工具。

## 开发工具

---

- **xCode**: mac下开发必备吧。
- **iTerm2**: 据说是最好的终端。
- **SublimeText3**: 最好的文本编辑器。
- **Dash**: 超全文档查看工具。
- **SourceTree**: git GUI工具。
- **cotnerstone**: SVN客户端。
- **Abode**系列: 其实也就是用用Photoshop。
- **MAMP**: 本地服务器。
- **CodeRunner 2**: 代码直接编辑预览工具。

## brew安装的

---

- git
- fasd
- htop
- zsh
- brew-cask
- joe/gitignore
- nvm
- ruby

## 想要购买的

---

- [iThoughtX](#): 脑图工具,
- [BOOM 2](#): 音效增强工具。
- [iStat Menus](#): 查看系统状态。
- [mac-app-blocker](#): 给软件加密。
- [Multimon](#): 多屏工作最佳伴侣。
- [Snagit](#): 截屏录屏编辑一条龙。
- [commandQ](#): 防止误按 `command+Q` , 售价9.99\$。

## 参考资料

---

- [Best-App](#)
- [Mac 软件](#)

## iTerm2

早就听说这个终端工具了，可以完全替代terminal。

## 设置

### 配色

[Solarized](#), 是目前最完整的 Terminal/Editor/IDE 配色项目，几乎覆盖所有主流操作系统（Mac OS X, Linux, Windows）、编辑器和 IDE（Vim, Emacs, Xcode, TextMate, NetBeans, Visual Studio 等），终端（iTerm2, Terminal.app, Putty 等）。类似的项目还有 [Tomorrow Theme](#)。

拿tomorrow-theme举例，下载Tomorrow-Night-Eighties.itermcolors文件，双击自动导入到iTerm2中，在Preferences->Profiles->Colors->Load Presets中可以看到对应的配色。修改即可。

### 中文乱码问题

确保Preferences->Profiles->Terminal->Terminal Emulation中的字符编码为UTF-8。

中文乱码的问题需要设置一下locale，在对于的shell配置文件中，比如bash对应的就是 `~/.bashrc`，zsh对应的就是 `~/.zshrc`，这里以zsh为例，打开.zshrc文件，修改其中 `# You may need to manually set your language environment export LANG=en_US.UTF-8`：

```
# You may need to manually set your language environment
export LANG=en_US.UTF-8
```

接着重启一下终端，或者输入：`source ~/.zshrc`。

### 一些快捷操作

- `command+方向键`：切换tab。
- `command+enter`：全屏模式。
- `command+f`：搜索，支持正则表达式。
- `command+d`：垂直分屏。
- `command+shift+d`：水平分屏。
- `command+[ 或 command +]`：在最近使用的分屏直接切换。
- `command+t`：打开新标签。
- `command+w`：关闭新标签。
- `command+;`：自动补全历史命令。
- `command+r`：清除屏幕，相当与clear。
- `command+p/n`：上一条/下一条命令，相当于方向键上和下。
- `ctrl+r`：搜索命令历史。

### 编辑操作

基本的Emacs移动光标方式。还有一些很好的操作方式，我觉得都可以借鉴配置到SublimeText中。

- `ctrl+d` : 删除当前字符。
- `ctrl+h` : 删除之前的字符。
- `ctrl+u` : 删除整行。
- `ctrl+k` : 删除当前到文本末尾的字符。
- `ctrl+w` : 删除光标前的单词。
- `ctrl+t` : 交换当前光标和前一个位置, 互换。

## 参考资料

---

- [官方文档](#)
- [我在用的mac软件\(1\)--终端环境之iTerm2](#)

# brew

## homebrew

[homebrew](#)，是mac下类似apt-get的软件管理工具。

通常情况下brew安装的软件都会在 `brewprefix` 返回的目录中，不会在额外创建文件。

### 安装

没啥说的，直接安装官方提供的方式，终端下运行：`ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

### 使用

安装完brew之后，执行 `brew update` 和 `brew doctor`，然后按照提示稍微设置下，需要我们将 `/usr/local/bin` 添加到PATH路径的最前面，保证系统优先调用到的是brew下载的。在命名行下输入：`echo export PATH='/usr/local/bin:$PATH' >>`

`~/.bash_profile`。

一般来说，我会用brew安装git, nvm, zsh。V2EX有人统计了下常用的安装软件：

```
wget=26
pkg-config=17
automake=17
autoconf=17
readline=15
git=14
gettext=14
libtool=14
xz=14
pcre=13
imagemagick=12
zsh=12
jpeg=12
gdbm=12
openssl=11
mongodb=11
cmake=11
coreutils=11
libevent=11
tree=11
freetype=11
sqlite=11
libpng=10
macvim=10
mtr=10
libyaml=10
mysql=9
node=9
mercurial=9
tig=9
nginx=9
gnutls=8
redis=8
libpg-error=8
ack=8
gmp=8
tmux=8
libtiff=8
glib=7
go=7
libsba=7
nettle=7
```

```

p11-kit=7
unrar=7
phantomjs=7
python=7
libffi=7
gawk=7
libtasn1=7
lua=7
apple-gcc42=7
libxslt=7
dos2unix=6
fontconfig=6
python3=6
htop-osx=6
pixman=6
ctags=6
gd=5
intltool=5
git-extras=5
swig=5
curl=5
neon=5
jasper=5
curl-ca-bundle=5
icu4c=5
p7zip=5
postgresql=5
zlib=5
libxml2=5

```

## 删除

这个说一下吧，因为公司的电脑是别人之前使用的，暂时有些资料需要保存，我无法删除，我新建了个用户继续使用。

之前的brew可能是其他用户安装的，导致我的新用户能使用brew命令，但是无法安装，更新（应该是权限读取的问题）。所以我想卸载brew重新安装。官方提供的一个脚本好像是几年前的，[uninstall\\_homebrew.sh](https://uninstall_homebrew.sh)。而我找到一个说明，感觉比较简单明了：

### Uninstall

WARNING: Before copying and pasting these commands on your shell, make sure the first one ( `brew --prefix` ) returns the path where homebrew was installed properly. If not, you might ending up removing stuff from your computer you did not intend to remove.

```

cd `brew --prefix`
rm -rf Cellar
brew prune
rm -rf Library .git .gitignore bin/brew README.md share/man/man1/brew
rm -rf ~/Library/Caches/Homebrew
http://superuser.com/questions/203707/how-to-uninstall-homebrew-osx-packet-> manager

```

### Reinstall

```

ruby -e "$(curl -fsSL https://raw.githubusercontent.com/mxcl/homebrew/go)"
https://github.com/mxcl/homebrew/wiki/Installation

```

也许第一次删除安装还是会出问题，那就结合官方提供的shell里的路径，全部删除，再次重启。

## 特别说明

我这里删除再安装之后，可以通过 `brew install` 安装软件，比如git。但是执行 `brew doctor` 时，会报告 link 的错误：`You have unlinked kegs in your Cellar`，执行 `brew link git` 之后，会提示 `could not symlink opt is not writable`。

此问题是 `/usr/local` 目录缺少权限，执行 `sudo chmod -R g+w /usr/local`，再次 `brew link git` 即可。

## homebrew-cask

---

`cask`，它是brew的一个扩展，提供命令行下安装软件的功能。它所安装的所有软件都在 `/opt/homebrew-cask/Caskroom` 目录下，自动完成了软连接到 `Application`。

### 安装

```
brew install caskroom/cask/brew-cask，一键安装。
```

如果已经安装的用户，可以升级到最新版本：

```
brew update && brew upgrade brew-cask && brew cleanup && brew cask cleanup
```

还有要说明的一点：默认通过brew cask安装的软件，是软连接到 `~/Applications` 目录下，这是可以通过设置修改的，具体的使用可以通过 `man brew-cask` 查看。

我是直接 `echo export PATH='/usr/local/bin:$PATH' >> ~/.zshrc`，写入到配置文件中。

### 运行

第一次安装过后，执行 `brew cask --help` 查看下说明（其实只要是第一次运行），需要你提供root权限，方便cask软连接到 `Application`中。

安装其他软件的话，可以先使用 `brew cask search <name>` 查看是否有匹配的。再使用 `brew cask install <name>` 进行安装。成功之后，就可以直接在 `Application`中找到刚刚安装的软件。

## 关联Alfred2

---

安装完应用之后，如果你是Alfred2用户，还需要设置一下link，使得Alfred可以搜索通过brew cask安装的应用，操作方法是：

```
# 查看状态
brew cask alfred status
# 设置连接
brew cask alfred link
```

## 使用brew cask安装的软件

---

- Dropbox
- QQ
- TextExpander

## LaunchRocket

---

LaunchRocket是一个管理brew安装的service的工具，安装之后可以看所有的service的运行状态。

## 参考资料

---

- [homebrew-FAQ](#)
- [使用brew cask来安装Mac应用](#)
- [LaunchRocket](#)



## zsh

zsh是shell语言类型，兼容bash，提供强大的命令行功能，比如tab补全，自动纠错功能等。缺点就是配置太麻烦，好在有一个叫做 `oh-my-zsh` 的开源项目，很好的弥补了这一缺陷，只需要修修改改配置文件，就能很顺手。

## 安装zsh

安装方式我使用：`brew install zsh`。

替换bash的方式：`chsh -s /bin/zsh`。关闭终端，再次打开即为zsh。

注意：之前我们使用bash，我们为了使用brew安装的软件，修改了 `~/.bash_profile` 文件，新的zsh自己也有配置文件，是 `~/.zshrc`，需要将配置拷贝到 `~/.zshrc` 中。

或者在安装完oh-my-zsh后，执行 `echo export PATH='/usr/local/bin:$PATH' >> ~/.zshrc`。

## oh-my-zsh

由于zsh的配置是很复杂的，所以有这个一个开源项目`oh-my-zsh`，帮助我们简化zsh的配置。

官网有自动安装的方法，我选择的是：`curl -L http://install.ohmyz.sh | sh`。

安装完成后，重启终端就能看到界面的变化了。zsh的配置文件是 `~/.zshrc`，配色对应的是 `ZSH_THEME`。

## zshrc

zshrc是zsh的配置文件，我会在此添加一些alias设置。比如：

```
alias st='open -a "Sublime Text"'
```

## oh-my-zsh插件

oh-my-zsh的强大之处还在于提供了完善的插件系统。相关的文件存储在 `~/.oh-my-zsh/plugins` 中，默认提供了100多种。。。

默认提供的插件是git,需要添加的话，修改 `~/.zshrc` 中 `plugins=(git autojump)` 即可。

## 自动跳转

`z`和`autojump`。是两个可以实现自动跳转的插件，都是可以通过brew下载的。

我目前使用的是 `autojump`，通过 `brew install autojump` 下载，并且在 `~/.zshrc` 中修改 `plugins=(git autojump)`。重启终端。

使用就可以使用j来代替cd命令了，并可以添加自定义目录，具体使用说明参考autojump的文档或者 `autojump --help`。

## fasd

`fasd`，功能上和z, autojump差不多，功能和速度上更优。它会按照访问的频率记录下文件，帮助用户快速访问。

安装还是通过brew：`brew install fasd`，安装之后，我安装官网的步骤，执行了：`eval "$(fasd --init auto)"`，且在zshrc中

开启对应的配置：`plugins=(git fasd)`，重启终端即可使用。

asd comes with some useful aliases by default:

```
alias a='fasd -a'      # any
alias s='fasd -si'    # show / search / select
alias d='fasd -d'     # directory
alias f='fasd -f'     # file
alias sd='fasd -sid'  # interactive directory selection
alias sf='fasd -sif'  # interactive file selection
alias z='fasd_cd -d'  # cd, same functionality as j in autojump
alias zz='fasd_cd -d -i' # cd with interactive selection
```

Example:

```
f foo      # list frecent files matching foo
a foo bar  # list frecent files and directories matching foo and bar
f js$     # list frecent files that ends in js
f -e vim foo # run vim on the most frecent file matching foo
mplayer `f bar` # run mplayer on the most frecent file matching bar
z foo     # cd into the most frecent directory matching foo
open `sf pdf` # interactively select a file matching pdf and launch `open`
```

## 参考资料

---

- [分享了下自己的终端环境，iTerm2,zsh,z,tmux。](#)
- [终极 Shell——ZSH](#)

# 1Password

---

## 参考资料

---

- [是时候在 iPhone 上忘掉密码了：1Password 5.0 for iOS 上手完全指南](#)

# TextExpander

---

[Textexpander](#),是一款绝佳的输入辅助工具，完成最简单的扩展功能，强大的自定义，利用好之后，效率绝对提升百倍。

## 下载

---

首先这是一款收费的软件，而且价格不便宜。单用户授权的价格是34.9美元，大约220人民币。我每验证是否单授权可以用于多台机器，理论上不太多应该都可以。

## 设置

---

安装完成之后，首先是一段设置向导，里面有简单的介绍。默认用法就是直接输入就帮助你扩展了。我在测试的使用，如果使用搜狗输入法的英文状态是不可行的，这算是一个Bug吧。

## Group

软件左侧的目录可以设置Group，方便管理类别，TextExpander官方提供了一些Group方便下载使用，在添加中就可查看。注意别添加重复了。

额外推荐一个Markdown的扩展，[TextExpander-Snippets](#)。

## 备份

---

它默认会自动备份到本地，也可以选择Dropbox同步，但是它会直接在Dropbox根目录下创建一个TextExpander目录。(强迫症比较反感，应该放在App目录下才对嘛！)

## 参考资料

---

- [TextExpander：深度解读 – 到底值不值得买？](#)
- [利用 TextExpander 提高在 Mac 上撰写 Markdown 的效率](#)
- [TextExpander 使用技巧第一弹](#)

## 欧陆词典

---

欧陆被成为最好的Mac词典，比较老牌了，Mac下做的很好，反倒在window下感觉不适。

## 软件安装

---

免费版的话，可以在app store中下载。付费版本app store中也有，但是售价也高。可以参与一些团购活动，我就是通过这种方式购买的。一个激活码，可以激活三台设备。

## 软件设置

---

通用设置中，选择开机自动启动，关闭新闻相关提示。LightPeek好像就是一个常驻menubar的搜索框，我也关闭了。

### 快捷键

取词的话，我设置了command按下取词。划词功能也被我关闭了，因为我配合popClip实现划词翻译的功能。

所有的快捷键中，我只设置了ctrl+commander+x开启窗口的功能，方便详细的查询。

### 词典设置

关于词典的选择，参考[市面上最常见的牛津高阶英汉双解词典](#)，[朗文当代高级英语辞典](#)和[柯林斯COBUILD高阶英汉双解学习词典有何特色？](#)

## popClip

---

popClip是一个划词扩展插件，可以方便的实现划词复制，粘贴等功能。比较有特点的是它支持扩展，可以结合很多应用，比如欧陆词典。

## 插件

---

### Eudic

欧陆词典的插件，可以通过欧陆查词。

### Wikipedia

查询维基百科，可选查询语言。

### Dash

可以调用dash查询。

### PDF Highlight

可以在Preview和Skim中标注高亮。

## 参考资料

---

- [扩展下载](#)

## manico

---

manico是国人开发的一款快速启动辅助工具，特点是按住 `option+<other>`，`other`可为任意按键，启动应用。默认是数字结合qwerty一次排序。

## 设置

---

首先我的Mac下安装了 Alfred2，一般的app可以快速启动。购买manico主要的用处是指定几个专属的启动快捷键。这一点让我回想起了黑莓手机，它的全键盘能够快速启动任何应用。

### 设置原则

我按照左手操作，尽量不借助右手的原则，分配了数字键: `1 2 3 4` ,字母: `q w e r a d`。没有包含 `s` 是因为个人将 `option+s` 分配给了 `alfred2`。注意这些按键，正好是键盘的三排。每一排我按照功能进行不同分配：

- 第一排: finder, chrome, evernote
- 第二排: Dash, SublimeText, iTerm2
- 第三排: Spillo, Alfred2, Doit.im

除了这三排按键的设置，我还按照了应用的功能分配了一些其他按键，比如 `option+m`，我启动的一个Music相关的应用，比如iTunes。

这样设置的话，可以通过左手去快速切换我常用的任何应用。读者不需要按照这样去设置，这里举例的都是个人喜好，原则是希望告诉大家去设置一下常用的应用，节约切换app的时间（不要小看这个时间）。

## 窗口管理器

---

目前个人选择的是Moom, 原因是用的人多, 好评多。但是经常在论坛中看到有人对窗口管理器进行讨论, 也记录一下最近看到的。

### 软件列表

---

- [spectacle](#): Spectacle 的 `resize` 和 `move` 功能基本齐全, 还支持使用快捷键放大和缩小窗口, 本身也免费、开源。可是唯独缺了移动窗口到其他 workspace 的功能。
- [Moom](#)
- [Ddivy](#)
- [sizeup](#): 想要的 `move` 功能还有 `resize` 都包含在内 (就差 Spectacle 的放大缩小窗口了), 键盘党的福音。
- [BetterSnapTool](#): BetterSnapTool 也是挺有自己特色的, 尤其是给窗口边框专门定制了一些功能。价格 1.99 美元, 也算是几个收费应用里最便宜的。
- [slate](#):, 号称可以替换 Divvy, SizeUp, ShiftIt (这货头次听说)。和 Spectacle 一样是开源的。可定制性强, 但配置麻烦, 配置文件很长。
- [Mjolnir](#): Mjolnir 比 Slate 更像 AwesomeWM 的缩微移植版, 连配置文件都是用 lua 写的。
- [Layouts](#): Layouts 是纯粹用 Alfred Workflow 实现的窗口管理器, 如果你是个 Alfred 小狂人, 不妨试上一试。

### 使用感受

---

之前一直在用 Spectacle, Spectacle 的 `resize` 和 `move` 功能基本齐全, 还支持使用快捷键放大和缩小窗口, 本身也免费、开源。可是唯独缺了移动窗口到其他 workspace 的功能。

因为工作需要开很多窗口, 快速移动窗口是必须的功能, 所以只能放弃 Spectacle 啦。于是考察了 Moom, Divvy, BetterSnapTool 等等。

Moom, 看起来是不错, 不过我是键盘党, 所以好大一块卖点我用不到, 而且也不支持 workspace 间的移动。放弃。

Divvy, 和 Moom 一样的问题。放弃。

BetterSnapTool, 没找到试用版。用过朋友的分享一下使用感受吧。

我最后买了.....SizeUp, 想要的 `move` 功能还有 `resize` 都包含在内 (就差 Spectacle 的放大缩小窗口了), 键盘党的福音。虽然原价比其他贵一点, 不过随便搜搜就能找到 30% off 的 coupon, 算下来还是挺便宜的。



## BetterTouchTool

---

**BetterTouchTool** (简称 BTT) ,是一款完全免费的 Mac 辅助应用, 可以用来代替默认的系统操作方式 (组合键、修饰键、手势等), 其目的是方便用户创造出更适合自身习惯的操作行为, 是 Mac 上非常强大的触摸板辅助工具。

### 参考资料

---

- [Mac 触摸板增强神器:BetterTouchTool 上手指南](#)
- [Mac 触摸板增强神器 : BetterTouchTool 进阶指南](#)

## 开发环境

---

在PC下，我一般都会安装好各种开发语言，设置好命令行工具，搭配好开发的编辑器或IDE。

## 开发语言

---

mac系统上自带了gcc, g++, ruby, python的环境。Objective-C的开发当安装上xCode之后也配置好了。

我一般还会安装上nodejs, git ,java。

安装的方式我会选择brew，类似linux系统下的apt-get。但有些命令比如git在xCode安装的时候就已经绑定了，这时需要我们将 `/usr/local/bin` 添加到PATH路径的最前面，保证系统优先调用到的是brew下载的。在命名行下输入：`echo export PATH='/usr/local/bin:$PATH' >> ~/.bash_profile` 。

这一步也可以执行 `brew doctor` 来检测。

## 终端

---

### Finder中打开

通常Finder会搭配上XtraFinder插件，可以在目录中直接打开终端，且可指定终端为iTerm2。

### iTerm2

在我还没有使用mac的时候，我就常常看见别人推荐iTerm 2这个强大的终端软件，用来替代原生的终端。

目前我设置过的就是新建窗口的大小，默认是80x25我觉得太小了，改为了120x30。

下一步打算修改一下配色，以及设置一下全局开启的快捷键。

### zsh

shell是终端与系统交互的一种语言，默认的是bash，但是最好的是zsh。安装方式我使用：`brew install zsh` 。

替换bash的方式：`chsh -s /bin/zsh`。关闭终端，再次打开即为zsh。

注意：之前我们使用bash，我们为了使用brew安装的软件，修改了 `~/.bash_profile` 文件，新的zsh自己也有配置文件，是 `~/.zshrc`，需要将配置拷贝到 `~/.zshrc` 中。

或者在安装完oh-my-zsh后，执行 `echo export PATH='/usr/local/bin:$PATH' >> ~/.zshrc` 。

### oh-my-zsh

由于zsh的配置是很复杂的，所以有这个一个开源项目`oh-my-zsh`，帮助我们简化zsh的配置。

官网有自动安装的方法，我选择的是：`curl -L http://install.ohmyz.sh | sh` 。

安装完成后，重启终端就能看到界面的变化了。zsh的配置文件是 `~/.zshrc`，配色对应的是 `ZSH_THEME` 。

### oh-my-zsh插件

oh-my-zsh的强大之处还在于提供了完善的插件系统。相关的文件存储在 `~/.oh-my-zsh/plugins` 中，默认提供了100多种。。。

默认提供的插件是git,需要添加的话，修改 `~/.zshrc` 中 `plugins=(git autojump)` 即可。

## 自动跳转

`z`和`autojump`。是两个可以实现自动跳转的插件，都是可以通过brew下载的。

我目前使用的是 `autojump`，通过 `brew install autojump` 下载，并且在 `~/.zshrc` 中修改 `plugins=(git autojump)`。重启终端。

使用就可以使用`j`来代替`cd`命令了，并可以添加自定义目录，具体使用说明参考`autojump`的文档或者 `autojump --help`。

## 参考资料

---

- [分享了下自己的终端环境，iTerm2,zsh,z,tmux。](#)
- [终极 Shell——ZSH](#)

## 快捷键设置

---

Mac与PC上手最大的不同，肯定就是按键问题了。我喜欢将普通键盘上的win与alt映射为mac下的option和command键。这样键盘的布局与标准的苹果键盘相似。

通常PC下很多ctrl的组合操作，都能对于为command的组合操作。

通过一段时间的使用，我发现我切换程序多使用的是 `alfred2 + Manico + 快捷键` 方式。

`option` 按键主要用于切换程序，`shift+command+其他` 主要用于程序的功能。

## 全局快捷键

---

先提一点，我喜欢把Mac键盘最上方的那一排保持F1~F12的功能，快捷功能通过fn的组合键形式实现，实现这一点请勾选：System Preference -> Keyboard -> Use all f1, f2, etc keys as standard function keys.

修改快捷键在: System Preference -> Keyboard -> Shortcuts中。

我的主要修改如下：

### Lunchpad & Dock:

- Turn Dock Hiding On/Off: 取消设置。
- Show Launchpad: `F4` .

### Display:

默认。

### Mission Control :

- Mission Control: `Ctrl+top` .
- Show Desktop: `F11` .
- Move left a space: `Ctrl+left` .
- Move right a space: `Ctrl+right` .
- Switch to Desktop: `Command+num` . 我个人是创建了四个桌面。

### Keyboard:

默认。

### Input Sources:

- Select the previous input source: `Command+Space` .
- Select next source in Input menu: 取消设置。

### Screen Shots:

默认。

### Services:

默认。

### Spotlight:

- Show Spotlight search field: 取消设置。
- Show Spotlight window: 取消设置。

### Accessibility:

默认。

### App Shortcuts:

这里的内容，是设置全局的快捷键，也可以指定某个软件内的快捷键，但是需要设置对应的菜单项名称才OK。比如我设置Finder中的 `New Terminal Here`，（此功能是通过Extra Finder插件实现的）。则添加：

- Finder.app: `New Terminal Here`，设置 `ctrl+command+t`。

## Karabiner

---

[Karabiner](#)，原因叫做 `KeyRemap4MacBook` 是一款很出色的修改键盘映射的工具，我目前还没有开发出它的潜能，只是用来替换了左下角的fn与control。

此软件可以设置几个配置方案，比如我建立了 `Default` 和 `Coding` 两种方案。

## Seil

---

配合Karabiner,修改CapsLock按键作用。并且通过Karabiner,设置了双击shift切换CapsLock.

## SublimeText 3

---

单独开贴介绍了。

## Alfred 2

---

`alt+s`，弹出窗口。

配合Karabiner+Seil, 映射到CapsLock上。

## Manico

---

默认，`alt+num` 选择程序。

## Moom

---

### Mouse:

- 开启Snap to Edges and Corners: 实现拖拽到边缘放大的功能。
- Delay设置了:0.1s。

**Custom**, 添加三种自定义的方式:

- Move & Zoom: 全屏, `shift+command+1`。
- Resize: 大小100x600, `shift+command+2`。
- Move & Zoom: 屏幕上方全屏, `shift+command+3`。

## ClipMenu

---

设置 `shift+command+v`。弹出窗口。

## ExtraFinder

---

Add items to Finder menus:

- Copy Path: Default: Path.
- Show Hidden Items: `shift+h` .
- New Terminal Here: `iTerm` .
- New File.: 勾上.

## EuDic

---

通用:

- 启动时最小化欧陆词典主窗口: 勾上.
- LightPeek 快捷搜索: 关闭.

取词:

- 开启鼠标自动取词功能: `command`键按下时启动.
- 开启划词搜索: 关闭.

快捷键:

- 显示/隐藏窗口: `shift+command+x` .
- 其余: 关闭.

## 1Password

---

被誉为最好的密码管理工具。名不虚传。默认没有直接打开界面的功能，所以我通过 `Manico` 绑定了 `option+x` 的快捷方式。

默认提供的快捷键修改为：

- Lock: ~~`shift+command+l`~~，取消原因是和sublime有冲突。
- Show Mini: `shift+command+\`
- autofill: `command+\`

## SnapRuler

---

这是一个测量工具，也提供了截图的功能。索性我就用它来替换系统的截图工具吧。

从keyboard->shortcuts中取消系统的截图快捷键。设置SnapRuler的快捷键为 `shift+command+4`。

图片保存路径为 `Pictures->SnapRuler`。

快捷键设置

## 参考资料

---

- [OS X : 键盘快捷键](#)
- [Karabiner](#)

## 常用终端命令

---

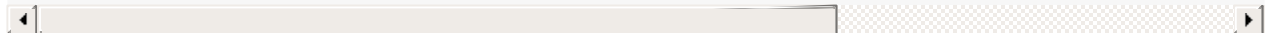
介绍一些终端下常用的命令。首先说明一下，本人的环境为：iTerm2+zsh。

### 常用的

---

先插一句，说一个好玩的，可以查看最近经常使用的命令，在终端执行：

```
history | awk '{CMD[$2]++;count++;} END { for (a in CMD )print CMD[ a ] " " CMD[ a ]/count*100 "% " a }' | grep -v "./"
```



- clear: 清除屏幕。
- pwd: 查看当前路径。
- cd: 进入目录，常常配合一些字符使用。比如：
  - .. : 返回上级目录。
  - - : 返回用户主目录。
  - - : 返回上次操作目录。
  - / : 返回系统根目录。
- ls: 列出目录信息。
- mkdir: 创建目录。
- touch: 创建文件。
- cp: 拷贝文件。
- mv: 移动文件，或者重命名文件。
- rm : 删除文件。
- cat: 查看文件内容。
- grep: 查找文本信息。
- man: 手册命名，查看各个命名的帮助。

### 系统操作

---

- open: 可以打开文件，目录和程序。通过man open查看具体内容，我常用来在终端下打开Finder，比如 open .。Windows下对应的使用 explorer 命令。或者打开Applications下的程序，使用 open "/Applications/Sublime Text.app" test.md。



## dotfiles

---

dotfiles就是软件的配置文件。一般用于软件设置，可以通过备份dotfiles的方式，同步软件设置。

## 同步原理

---

主要是应用了 `ln` 软连接的功能，命名格式如下：

```
ln [参数][源文件或目录][目标文件或目录]
```

在我们备份dotfiles中常用的参数有`ln -s`软链接，s是代号symbolic的意思，所谓软链接，她只会在你选定的位置上生成一个镜像，而不会占用磁盘空间，而如果使用`ln`不带参数的话，则就是硬链接，会在选定的位置上生成一个和源文件大小相同的文件，占用磁盘空间。注意在创建软连接之前，保证目标文件是不存在的。

## 如何同步

---

可以使用git/dropbox管理dotfiles。之前使用过dropbox，受限于国内网速问题，多台设备常常会出现版本冲突。

以后索性使用git管理吧，创建一个dotfiles目录，将所有的文件放置到此目录下。

```
cd ~
mv .zshrc ~/Users/l/dotfiles/zshrc
ln -s ~/Users/l/dotfiles/zshrc .zshrc
```

这里举例的是zsh的配置文件，其他原理同此。最后把git push到服务器端既可。

## 如何恢复

---

首先更新下对应的dotfiles目录，然后删除掉恢复的配置文件，再次使用软连接恢复。

```
git clone xxxx
rm -rf .zshrc
ln -s dotfiles/zshrc .zshrc
```

关于恢复，还可以通过脚本文件实现自动化。以后补充。

## 参考资料

---

- [dotfiles](#)
- [dotfiles.github.io](#)
- [mackup](#)

## Android篇

---

Android是google推出的一款操作系统，主要使用在手机端。

优点是开源免费，缺点是版本过多，设备尺寸碎片化。导致很多应用无法全部适配。

## 安卓应用

---

应用列表在豆瓣中建立了一个豆列: [我的安卓应用](#)。

## 常用必备

---

- **Chrome**: 最常用的浏览器。配合google账号, 可以同步书签。
- 知趣天气: 喜欢里面的多天气源和桌面的小插件。
- 触宝: 联系人和拨号辅助工具, 号码来源比较智能。
- 搜狗输入法/百度输入法: 比较喜欢百度的简洁。
- 怪物闹钟: 提供多种闹钟类型, 比较喜欢“面包机”。
- **Morning Routine**: 一个游戏公司开发的闹钟应用, 关闭闹钟可以显示天气, 并设置跳转到指定应用。
- 豌豆荚: 国内的andorid应用市场。
- **google服务**: 包括邮件, 日历, 联系人, play应用市场等功能。
- **My Day**: 日期倒计时工具, 记录一些重要的日期, 可以按照年月日, 或者天数查看。

## 系统辅助

---

- **Switchr**: 通过边缘滑动触发, 切换最近的应用。
- **SuperSU**: 提供root权限, 不必每次确认是否使用。
- **Adblock Plus**: 智能拦截应用中的广告。
- 钛备份: 备份应用到SD卡中, 安装但没使用过。
- 绿色守护: 防止一些应用在后台自动运行。
- **SetDNS**: 方便切换DNS。
- 猎豹清理大师: 清理系统垃圾。
- **AirDroid**: 有对应的客户端, 功能比Pushbullet强大, 就是“重”了点。
- **Pushbullet**: 推送服务, 可以和电脑互相推送网页, 图片。
- **Link Bubble**: 付费软件, 帮助节省页面加载的等待时间。
- 蓝色光波过滤: 长期阅读文字时, 配色比较不刺眼。
- 薄暮微光: 根据日出日落控制光波, 保护视力, 安卓上的f.lux。
- 智能应用保护: 锁定指定应用, 防止别人打开。
- **Lockdown Pro**: 锁定指定应用, 和上面的差不多, 这两个在我锤子都安装不了, 解析数据包错误。
- **LastPass**: 密码保存工具。
- **1Password**: 全平台密码管理工具。
- 影梭: shadowsocks翻墙服务。
- **Wifi万能钥匙**: 链接别人提供的无线网络。
- **BitTorrent Sync**: p2p同步服务, 可实现多设备网盘同步功能。

## 个人效率管理

---

- **SolMail**: 我喜欢这家公司的产品风格, 这个是邮件客户端。
- **Doit.im**: GTD类工具, 付费。
- 为知笔记/印象笔记: 笔记软件。
- **Pinboard**: 书签保存与搜索软件。
- **Sunrise**: 日历软件。
- 挖财: 理财软件。
- **Trello**: 团队项目管理。

## 社交类

---

- 微信：足以替代短信，何况还有朋友圈和订阅号。
- 微博：订阅一些账号，分组查看。当了解新闻。
- **Instagram**：图片社交软件，保存自己一些手机摄影图片。

## 阅读类

---

- 搜狐新闻：新闻类软件。
- 知乎，知乎日报：知乎日报每日必读。
- **Press**：付费软件，RSS订阅服务，RSS源使用feedly。
- **Pocket**：稍后阅读，支持离线阅读。
- 多看阅读：电子书软件。
- 最美应用：提供发现好的应用途径。
- 什么值得买：发现优惠信息。

## 图片处理

---

- 快图浏览：浏览手机照片，足以替代手机默认相册。
- **Snapseed**：处理照片软件。
- 网易云相册：同步照片到网易云相册中。

## 生活相关

---

- 大众点评：很好的解决一个吃货的选择问题。
- 团800：各种团购的综合搜索。
- 快的打车：解决出行打车问题。
- 猫眼电影：解决电影订座问题。
- 高德地图：解决路痴问题，开车的情况下使用较好。
- 百度地图：公交功能实用，不开车的导航情况下比高德好。
- 支付宝钱包：方便网购，转账。
- 招商银行：查看银行账户信息。
- 联通手机营业厅：方便查看剩余话费和流量。

## 影音视频

---

- 蜻蜓FM：电台软件，收音机。
- 荔枝FM：有不少优质的节目。
- 优酷视频：优酷的纪录片频道还可以。
- 搜狐视频：订阅美剧。
- 风云直播：查看电视直播。

## 运动健康

---

- 家庭用药：对症下药，不要乱投医。
- **Nike Running**：跑步软件。
- **runtastic** 系列：这个系列的软件没一个都值得拥有。

## 学习类

---

- 欧陆词典：优点是可自定义扩充词库。
- 扇贝单词：背单词软件。
- 网易公开课：看看TED。

## 个人兴趣

---

- 爱尚吉他：教学与吉他谱资源较多。

## GooglePlay登录美国区的方式

双十一的时候购入了锤子手机，整体感觉很优雅。由于是Android系统，所以第一件事情我还是安装Google的相关服务。然后下载一个Google Play市场，下一些正品的应用。

### Google服务框架

自从使用智能机器以来，我一直都是用的Android，小米、魅族、锤子，这三个优秀的Android系统我都体验过了。总体来说，还是很喜欢锤子的系统，喜欢这件事情，是一个很主观的事情，必须要自己去体验一下。

Android系统，不用Google服务，总感觉缺失了一些安全感。作为一个程序员，有着良好的科学上网方式，我是必须装上Google服务的。锤子这一点我觉得做的就比其他系统好很多，它的内置应用商店中就有个 `Google服务下载器`，并且当你要下载使用Google服务相关的应用时候，它的软件描述中还会告知用户请先下载Google服务，很贴心。

下载并安装完成之后，可能要先翻墙一下，然后登陆Google账号。然后选择你要同步的内容即可。我不推荐同步联系人和人脉，google的联系人中还包含了你发送邮件的地址，显得很混乱。不如使用 `QQ同步助手` 这样专门用于联系人同步的软件。一般我只用来同步日历。

### 设置翻墙

选择一个翻墙方式，VPN或者shadowsocks。一定要记住必须是美国的ip,可以在百度中搜索IP查看验证。

### 电脑端设置

首先清空或者选择Chrome的隐私模式。首先登陆 `Google Wallet`，不是Google Play。

设置下付款地址，由于我之前绑定了信用卡，在左侧的导航栏里，选择 `Payment Method`，保险起见，建议右侧的Setting里面的Home Address也一并修改了。修改地址为：

```
Leo Hui
1 World Way
Los Angeles World Airports
Los Angeles CA 90045 US
```

修改完成之后，登陆 `Google Play`，理论上看到的就是美区的内容了，特点是左侧导航里会有图书，音乐等选项。

选择一本免费得图书，点击购买，选择使用兑换码购买，输入一个使用过的代码，比如 `2M7J2LPCU7K62QK6U54G` 即可。点击下一步，会告知此兑换码已经使用过，这就达到我们的目的了。就是为了记录一次购买经历。然后就可以关闭当前付款窗口。重新打开购买，就可以免费购买图书了。

这一步，其实已经完成了账号绑定到美国区的过程。

### 手机端设置

同样的先选择一个能翻墙到美国区的方式。

建议先从软件设置中清空Google Play的数据。等于重新打开。切换区域可能存在一定的延时，所以可以多试几次（清空数据再登录）。

如果一切正常，那么你看到的也应该是美国区的Google Play。里面的应用可以说是应用尽有。

Enjoy it!

## 参考资料

---

- [喜大普奔!Google Play永久锁定美国区教程!-断尾的Zekrom](#)

## 开发工具

---

本章节介绍一些开发过程中经常使用到的工具。



# Git

---

git是一种分布式版本控制系统，是目前项目管理使用较多的一种工具。

## 下载安装

---

### windows

[官网](#)，下载安装包，直接运行之后命令行就可以运行了。

### Mac

如果是安装了xcode的话，会自带一个版本的git。

如果想要安装新版本的git，推荐使用[Homebrew](#): mac下的apt-get。

安装brew，它下载的命令是存在放 `/usr/local/bin` 中的，所以要想正常工作，还需要在PATH中添加这个路径，在命名行下输入：`echo export PATH='/usr/local/bin:$PATH' >> ~/.bash_profile`。

### Linux

Linux下可以使用 `apt-get install git` 来安装。

## 学习教程

---

- [最好的中文教程](#)
- [Git Magic](#): 网上较火的一套教程。
- [GotGithub](#): 一本中文的介绍github使用的书。
- [Git Tutorials](#): bitbucket的教程。
- [Try Git](#) : codeschool教程。
- [Git Immersion](#): 答题的形式学习git.
- [Learn Version Control with Git](#)
- [github](#): Git your game on!
- [git-game](#): terminal game to test git skills.
- [Learn Git Branching](#): 形象直观的图形化练习网站。

## 使用技巧

---

### 资料

- [GitHub秘籍](#)
- [Set up Git](#): bitbucket教程。
- [Github Help](#)
- [使用git和github进行协同开发流程](#)

### 配置文件

一般都在 `/user` 下，window对应的目录是 `c:\Users\username`，mac和linux对于的就是用户主目录 `/Users/username`。配置文件名称为 `.gitconfig`。

通常我们使用git的时候最先会去配置username和email:

```
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "MY_NAME@example.com"
```

一般而言, 我会使用 `git add . -> git commit -m "xxxx" -> git push` 的方式提交, 第一次git操作的时候, git会给我如下的提示:

```
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)
```

其实就是告诉我设置下push.default,执行 `git config --global push.default simple` 即可。

## 存储密码

### https协议

如果使用的是https协议访问git仓库, 与服务器端同步的时候每一次都会提示输入密码。解决这个问题的方式就是能让密码保存起来。

在windows下, 我使用的是[git-credential-winstore](#)。下载安装即可使用。

在Mac下, 我使用的是[git-credential-osxkeychain](#)。首先在终端中检测是否已经有git-credential-osxkeychain, `git credential-osxkeychain`。提示 `usage: git credential-osxkeychain <get|store|erase>`, 代表有。

如果没有, 下载这个工具, 执行:

```
Move the file to the /usr/local/bin directory.
$ sudo mv git-credential-osxkeychain /usr/local/bin/

Make the file an executable:
$ chmod u+x /usr/local/bin/git-credential-osxkeychain
Configure git to use the helper.

$ git config --global credential.helper osxkeychain
# Set git to use the osxkeychain credential helper
```

如果存在的话, 直接执行 `git config --global credential.helper osxkeychain`。

在Linux下, 执行 `git config --global credential.helper store`。

操作完之后，检查下 `.gitconfig` 文件，看是否添加了 `[credential]` 字段。

如果需要取消设置，执行 `git config --unset --global credential.helper`。

## ssh协议

需要设置ssh的私钥和公钥。

## 切换协议

查看当前remote: `git remote -v`。

更新remote: `git remote set-url origin https://git.oschina.net/username/YourRepo`

# EditorConfig

---

## EditorConfig:

EditorConfig helps developers define and maintain consistent coding styles between different editors and IDEs. The Edit



简单的说，就是一个代码缩减格式化辅助工具，需要编辑器或者IDE插件支持。

## 配置示例

---

```
# EditorConfig helps developers define and maintain consistent
# coding styles between different editors and IDEs
# editorconfig.org

root = true

[*]

# change these settings to your own preference
indent_style = space
indent_size = 4

# we recommend you to keep these unchanged
end_of_line = lf
charset = utf-8
trim_trailing_whitespace = true
insert_final_newline = true

[* .md]
trim_trailing_whitespace = false

[.{package,bower}.json]
indent_style = space
indent_size = 4
```

## 参考资料

---

- [EditorConfig](#)
- [SublimeText Plug: editorconfig](#)

## node

---

node由于各个版本特性不同，很多项目需要使用不同版本的node，所以推荐使用nvm(Node Version Manager)进行管理。

## Mac下安装

---

### brew方式

如果机器没有安装过node，那么首先 `brew install nvm` 安装nvm。

其次需要在shell的配置文件(`~/.bashrc`, `~/.profile`, or `~/.zshrc`)中添加如下内容：

```
# For NVM
export NVM_DIR=~/.nvm
source $(brew --prefix nvm)/nvm.sh
```

注意配置的顺序，以防开启新终端，node出现找不到的情况。

重启终端，命令行下即可使用nvm，使用 `nvm install <version>` 进行对应的node版本安装，写这篇文章时，我使用的是 `nvm install 0.10`，安装的版本是v0.10.32。使用 `nvm use <version>` 使用，再通过 `nvm alias default <version>` 确保有默认版本。最后使用 `nvm ls` 查看。

### brew方式补充

如果之前通过'brew install node'方式安装过node，那么需要先删除系统中存在的node：

```
brew remove --force node
sudo rm -r /usr/local/lib/node_modules

brew prune
sudo rm -r /usr/local/include/node

# 检查brew是否正常
brew doctor
```

### nvm安装方式

`curl https://raw.githubusercontent.com/creationix/nvm/v0.17.2/install.sh | bash` 进行安装，安装完成后，运行 `nvm` 测试命令是否正确，如果不正确，参考官网提供的说明，也是需要在shell的配置文件中加入相应的配置。

如果安装正确，同样使用 `nvm install <version>` 安装对应版本node，使用 `nvm use <version>` 使用，再通过 `nvm alias default <version>` 确保有默认版本。最后使用 `nvm ls` 查看。

设置完nvm之后，node的路径其实是 `/Users/#{username}/.nvm/#{nodeVersion}/bin/node`，一些sublimeText插件默认的路径是 `/usr/local/bin/node`。个人建议创建一个软连接：

```
ln -s /Users/#{username}/.nvm/#{nodeVersion}/bin/node /usr/local/bin/node
```

## Windows下安装

---

window下我之前都是直接node官网下载mis文件安装。后续尝试使用类nvm工具安装管理。

## nvm-windows方式

睡觉前看了一眼，简直不能再便捷了!!! 项目地址: [nvm-windows](#)

下载安装包，不管之前系统安装过node与否，安装过会接管。就能直接使用nvm命令。

## npm的管理

---

通过nvm安装的node，每个版本都有一个对应的npm。每次切换，可以使用 `npm update -g` 进行一次升级，安装程序的话，需要使用sudo权限。

有一点疑问，如何同步之前安装的所有-g模块。。??

## 参考资料

---

- [node包教不包会](#)

## shadowsocks

换上Mac后,发现shadowsocks居然可以设置全局和自动代理,真实比Windows下方面多了!之前我一直购买的是东哥的服务,一年才50RMB,速度也还可以,720p无压力。自己另有一台DigitalOcean VPS,不用也浪费,搭建一个好了。

## Ubuntu下安装

首先,建议使用root用户登录,或者使用sudo命名,我这里以root用户为例把。请按顺序执行下面操作哦!

### 安装shadowsocks

第一次装的时候乱七八糟,也记不起来顺序了,反正用到的就是python版本,通过python-pip下载安装的 shadowsocks :

```
apt-get install python-pip
pip install shadowsocks
```

### 配置shadowsocks

配置文件需要自行创建:

```
vim /etc/shadowsocks.json
```

写入:

```
{
  "server": "0.0.0.0", # replace your server IP
  "server_port": 4762,
  "local_port": 1080,
  "password": "8d779a1ee2db776db8e20adffaa12d0c",
  "timeout": 300,
  "method": "aes-256-cfb"
}
```

### 安装Supervisor

```
apt-get update
apt-get install python-pip python-m2crypto supervisor
```

### 配置Supervisor

编辑或创建:

```
vim /etc/supervisor/conf.d/shadowsocks.conf
```

如果端口 < 1024,把上面的 user=nobody 改成 user=root。

### 优化

在 `/etc/default/supervisor` 最后加一行：

```
ulimit -n 51200
```

## 启动shadowsocks服务

使用Supervisor后台运行shadowsocks:

```
service supervisor start  
supervisorctl reload
```

## 查看服务状态

运行状态：`supervisorctl status`

如果遇到问题，可以检查日志：

```
supervisorctl tail -f shadowsocks stderr
```

## 重启服务

如果修改了 shadowsocks 配置 `/etc/shadowsocks.json`，可以重启 shadowsocks：`supervisorctl restart shadowsocks`

如果修改了 Supervisor 的配置文件 `/etc/supervisor/*`，可以更新 supervisor 配置：`supervisorctl update`

## 可能出现的异常情况

在我第一次安装的时候，出现过 `unable to resolve host` 的情况。解决方法就是将其指向 `127.0.0.1`，编辑 `/etc/hosts` 文件，在 `127.0.0.1` 后面，添加上自己主机的名称。

## 参考资料

- [Shadowsocks 使用说明](#)
- [用 Supervisor 运行 Shadowsocks](#)
- [在 Linode 上快速搭建 Shadowsocks](#)
- [sudo 出现unable to resolve host 解决方法](#)



## Sublime Text 3--Windows版

选择一个好的编辑器，可以极大的提高你的开发效率。我使用过Vim、Emacs和SublimeText。个人还是比较推荐现代化的sublimeText编辑器配合vim,emacs的操作方式。

更多内容，可以查看官方推荐的非官方文档->[文档地址](#)。

### 特色功能

- ctrl+p, 搜索。这个搜索可以左侧的Folders里所以文件，而且是模糊搜索，不需要完整的文件名。配合#, @, :可以搜索变量，函数，行数。
- 多行编辑。按住ctrl加左击，可以出现多个光标位置。
- 多重选择， ctrl+d可以多重选择，结合光标键，可以批量修改。
- 多屏编辑， alt+shift+数字键。
- Projects， 通过View->Side Bar->show Side Bar左侧文件结构管理。
- snippet, 不同格式的文件，可以设置不同的snippet,就是简写，通过tab扩展成相应的内容。
- 各种插件支持
- 正则表达式搜索,比如我要删除所有的空行，可以使用 `^\[\s]+\n` 来选择所有空行。可以使用 `(?<=<h2>).+(?=</h2>)` 来匹配h2标签内的内容。
- ctrl+shift+p, 功能菜单。只有你想不到，没有做不到的事情。

### 下载安装

ST3虽然没有提供稳定版本，但是相比ST2，速度提升还是很明显的。缺点就是插件不够完善，以及插件的编写全部采用Python3.x版本。这里给出ST3[下载地址](#)。

个人最喜欢的一点新特性是：新增了跳转到函数定义处功能，在大菜单Goto中可以查看到。

首次使用，建议先打开侧栏，方便管理文件结构。打开方式: View->Side Bar->Show Side Bar 。

### 插件安装

插件通过[Package Control](#)来管理。

#### 安装Package Control

进入Package Control页面，选择对应版本的代码进行复制，比如ST3如下：

```
import urllib.request,os,hashlib; h = '7183a2d3e96f11eeadd761d77e62404' + 'e330c659d4bb41d3bdf022e94cab3cd0'; pf = 'Pa
```

使用 `view->Show Console` 打开控制台，粘贴复制的代码，敲击回车运行。待程序右下角提示 `success` 后，重启ST。

再次进入ST后，可以通过菜单 `Preferences->Package Control` 或者按键 `ctrl+shift+p` 查找 `install package`。输入对应的插件名称，即可安装插件。

更多插件，可以通过Package Control中的[search](#)查找。

## 配置修改

配置包括Preferences->Settings-Default和Key Bindings-Default。

修改配置文件时，以上两个默认文件最好不要修改，自行讲需要设置的参数写入到Settings-User和Key Bindings-User里，它们会自动覆盖Default相同属性。

### 备份配置

配置文件的路径，点击Preferences->Browse Packages打开目录，找到User目录，这里的文件就是自己的配置文件，最好备份，方便下次替换。

## 插件推荐

### 主题配色和代码配色

配色其实分为主题配色和代码配色。主题配色就是程序的外形设置，代码配色则是打开文件高亮显示的配置。

代码配色我是选择的自己备份的主题 Peacock (SL).tmTheme ,放置在了 Packages/User/theme/ 目录下，主要是我针对 markdown 语法进行了设置，其他可选择的推荐 Dayle Rees Color Schemes 插件。

主题配色我使用的是 Theme-Phoenix 插件，插播一句，编程的字体应该选择等宽类型的。在windows下强烈推荐使用 yaheiconsolashybrid。

安装完了插件，可以在 Preferences->Color Scheme 中查看修改。也可以通过配置文件修改：

```
{
  "caret_style": "phase",
  "color_scheme": "Packages/User/theme/Peacock (SL).tmTheme",
  "default_line_ending": "unix",
  "font_face": "Monaco",
  "font_size": 18.0,
  "highlight_line": true,
  "hot_exit": false,
  "highlight_modified_tabs": true,
  "show_encoding": true,
  "ignored_packages":
  [
    "Vintage"
  ],
  "original_color_scheme": "Packages/User/theme/Peacock (SL).tmTheme",
  "phoenix_color_green": true,
  "phoenix_dirty_bottom_bar_red": true,
  "phoenix_eighties": true,
  "phoenix_highlight_current_tab": true,
  "phoenix_sidebar_tree_large": true,
  "phoenix_solid_current_tab": true,
  "phoenix_tabs_medium": true,
  "rulers":
  [
    80,
    100,
    120
  ],
  "soda_folder_icons": false,
  "tab_size": 4,
  "theme": "Phoenix Dark.sublime-theme",
  "translate_tabs_to_spaces": true,
  "word_separators": ".\\(\\)\\'\";,:;<-!@#%&*|+=[\\]{}`~?\"",
  "word_wrap": true,
  "wrap_width": 0
}
```

这里我列出的是我的全部配置文件，可以看到相关的主题配色、代码配色和字体设置。

## ST辅助类

- **SideBarEnhancements** 提升右侧导航栏功能
- **Sublimerge Pro** 文件对比功能
- **Markdown Preview** 书写markdown格式文本，预览等功能。绑定了快捷键 `ctrl+m`。
- **Terminal** 直接在对应文件所在目录打开terminal功能。绑定了快捷键 `ctrl+alt+t`。
- **IMESupport** 使得输入法能跟随光标位置，mac下无此问题。

## 代码显示辅助类

- **BracketHighlighter** 高亮显示匹配括号，会在左侧的行号标识处显示对应的括号位置和范围。
- **HTML-CSS-JS Prettify** 格式化代码工具，默认快捷键 `ctrl+shift+h`。
- **CSScomb** 按照一定规律格式化CSS的属性顺序。

## 代码书写辅助类

- **Emmet** 必装插件，辅助书写HTML, CSS。
- **AutoFileName** 书写代码时，自动提示补充文件路径。
- **DocBlockr** 辅助书写注释
- **JSHint Gutter** 利用 `jslint` 检测js代码是否规范的插件。
- **LiveStyle** 配合对应的chrome插件，可以达到修改文件后，自动刷新页面的效果。但目前对 `less`, `sass` 之类预编译语言支持不够好。

## 使用技巧

---

### 快捷键操作

默认的快捷操作，可以查看 `Preferences->Key Binding`，或者文档:[Keyboard Shortcuts-Windows/Linux](#)和[Keyboard Shortcuts-OSX](#)。

个人常用的快捷键设置如下：

```
[
/*===== Emacs Style =====*/
{ "keys": ["ctrl+b"], "command": "move", "args": {"by": "characters", "forward": false} },
{ "keys": ["ctrl+f"], "command": "move", "args": {"by": "characters", "forward": true} },
{ "keys": ["ctrl+p"], "command": "move", "args": {"by": "lines", "forward":
false} },
{ "keys": ["ctrl+n"], "command": "move", "args": {"by": "lines", "forward":
true} },
{ "keys": ["ctrl+a"], "command": "move_to", "args": {"to": "bol", "extend": false} },
{ "keys": ["ctrl+e"], "command": "move_to", "args": {"to": "eol", "extend": false} },
{ "keys": ["ctrl+l"], "command": "show_at_center" },
/*===== End Emacs Style =====*/
```

```

/*===== switch tabs =====*/
{ "keys": ["ctrl+1"], "command": "select_by_index", "args": { "index": 0 } },
{ "keys": ["ctrl+2"], "command": "select_by_index", "args": { "index": 1 } },
{ "keys": ["ctrl+3"], "command": "select_by_index", "args": { "index": 2 } },
{ "keys": ["ctrl+4"], "command": "select_by_index", "args": { "index": 3 } },
{ "keys": ["ctrl+5"], "command": "select_by_index", "args": { "index": 4 } },
{ "keys": ["ctrl+6"], "command": "select_by_index", "args": { "index": 5 } },
{ "keys": ["ctrl+7"], "command": "select_by_index", "args": { "index": 6 } },
{ "keys": ["ctrl+8"], "command": "select_by_index", "args": { "index": 7 } },
{ "keys": ["ctrl+9"], "command": "select_by_index", "args": { "index": 8 } },
{ "keys": ["ctrl+shift+t"], "command": "reopen_last_file" },
/*===== End switch tabs =====*/

/*===== Modify Default key-mapping =====*/
{ "keys": ["alt+a"], "command": "select_all" },
{ "keys": ["ctrl+t"], "command": "new_file" },
{ "keys": ["f5"], "command": "open_in_browser" },
// autocomplete
{ "keys": ["alt+/", "command": "auto_complete" },
// paste
{ "keys": ["ctrl+v"], "command": "paste_and_indent" },
{ "keys": ["ctrl+shift+v"], "command": "paste" },
// reindex
{ "keys": ["ctrl+i"], "command": "reindent" },
// find and goto
{ "keys": ["alt+f"], "command": "show_panel", "args": {"panel": "find"} },
{ "keys": ["ctrl+g"], "command": "find_all_under" },
{ "keys": ["alt+p"], "command": "show_overlay", "args": {"overlay": "goto", "show_files": true} },
{ "keys": ["alt+r"], "command": "show_overlay", "args": {"overlay": "goto", "text": "@"} },
{ "keys": ["alt+l"], "command": "show_overlay", "args": {"overlay": "goto", "text": ":"} },
{ "keys": ["alt+;"], "command": "show_overlay", "args": {"overlay": "goto", "text": "#"} },
{ "keys": ["alt+d"], "command": "goto_definition" },
{ "keys": ["alt+-"], "command": "jump_back" },
{ "keys": ["alt+="], "command": "jump_forward" },
/*===== End Modify Default key-mapping =====*/

/*===== Plugin =====*/
// Emmet expand
{ "keys": ["alt+e"], "args": {"action": "expand_abbreviation"}, "command": "run_emmet_action", "context": [{"key": "emmet"}]
// js Hint Grunt
{ "keys": ["alt+j"], "command": "jshint"},
// markdown preview
{ "keys": ["ctrl+m"], "command": "markdown_preview", "args": {"target": "browser", "parser": "markdown"} },
// terminal
{ "keys": ["ctrl+alt+t"], "command": "open_terminal" },
{ "keys": ["ctrl+shift+alt+t"], "command": "open_terminal_project_folder" }
/*===== End Plugin =====*/
]

```

其中涉及到了emacs移动光标，多标签切换，以及快速查找等方式。

## snippet

snippet是代码片段，可以方便的自动补全。创建方式通过 `Tools->New Snippet` 完成。

默认的文件如下：

```

<snippet>
  <content><![CDATA[
Hello, ${1:this} is a ${2:snippet}.
]]></content>
  <!-- Optional: Set a tabTrigger to define how to trigger the snippet -->
  <!-- <tabTrigger>hello</tabTrigger> -->
  <!-- Optional: Set a scope to limit where the snippet will trigger -->
  <!-- <scope>source.python</scope> -->
</snippet>

```

代码段写在 `CDATA[]` 中，`${}` 为占位字符。

`tabTrigger` 为自动补全需要的字符, `scope` 设置的是文件格式。

创建完成之后, 个人建议保存在 `User->snippet` 目录下, `snippet` 需要自行创建, 方便管理。

## build命令和Macro命令

这些命令的使用请参考文档->[Reference](#)。

## 参考文档

---

- [sublimeText官网](#)
- [非官方手册](#)
- [Package Control](#)

## Sublime Text 3--Mac版

Mac篇其实应该和Windows差不多，主要区别是一些按键的设置和插件的配置。

### 特色功能

- super+p, 搜索。这个搜索可以左侧的Folders里所以文件，而且是模糊搜索，不需要完整的文件名。配合#, @, :可以搜索变量，函数，行数。
- 多行编辑。按住super加左击，可以出现多个光标位置。
- 多重选择， super+d可以多重选择，结合光标键，可以批量修改。
- 多屏编辑， super+alt+数字键。
- Projects, 通过View->Side Bar->show Side Bar左侧文件结构管理。
- snippet, 不同格式的文件，可以设置不同的snippet,就是简写，通过tab扩展成相应的内容。
- 各种插件支持
- 正则表达式搜索,比如我要删除所有的空行，可以使用 `^\s*\n` 来选择所有空行。可以使用 `(?<=<h2>).+(?=</h2>)` 来匹配h2标签内的内容。
- super+shift+p, 功能菜单。只有你想不到，没有做不到的事情。

### 下载安装

ST3虽然没有提供稳定版本，但是相比ST2，速度提升还是很明显的。缺点就是插件不够完善，以及插件的编写全部采用Python3.x版本。这里给出ST3[下载地址](#)。

个人最喜欢的一点新特性是：新增了跳转到函数定义处功能，在大菜单Goto中可以查看到。

首次使用，建议先打开侧栏，方便管理文件结构。打开方式: `View->Side Bar->Show Side Bar`。

### 插件安装

插件通过Package Control来管理。

#### 安装Package Control

进入Package Control页面，选择对应版本的代码进行复制，比如ST3如下：

```
import urllib.request,os,hashlib; h = '7183a2d3e96f11eeadd761d777e62404' + 'e330c659d4bb41d3bdf022e94cab3cd0'; pf = 'Pa
```

使用 `View->Show Console` 打开控制台，粘贴复制的代码，敲击回车运行。待程序右下角提示 success 后，重启ST。

再次进入ST后，可以通过菜单 `Preferences->Package Control` 或者按键 `ctrl+shift+p` 查找 `install package`。输入对应的插件名称，即可安装插件。

更多插件，可以通过Package Control中的search查找。

### 配置修改

配置包括Preferences->Settings-Default和Key Bindings-Default。

修改配置文件时，以上两个默认文件最好不要修改，自行讲需要设置的参数写入到Settings-User和Key Bindings-User里，它们会自动覆盖Default相同属性。

## 备份配置

配置文件的路径，点击Preferences->Browse Packages打开目录，找到User目录，这里的文件就是自己的配置文件，最好备份，方便下次替换。

## 插件推荐

### 已安装的插件

```
{
  "in_process_packages":
  [
  ],
  "installed_dependencies":
  [
    "_package_control_loader",
    "bz2"
  ],
  "installed_packages":
  [
    "AlignTab",
    "All Autocomplete",
    "AutoFileName",
    "BracketHighlighter",
    "CSScomb",
    "DocBlockr",
    "EditorConfig",
    "Emmet",
    "Git",
    "HTML-CSS-JS Prettify",
    "JSHint Gutter",
    "LiveStyle",
    "Markdown Preview",
    "Modific",
    "Package Control",
    "SideBarEnhancements",
    "Sublimerge Pro",
    "Terminal",
    "Theme - Phoenix"
  ]
}
```

### 主题配色和代码配色

配色其实分为主题配色和代码配色。主题配色就是程序的外形设置，代码配色则是打开文件高亮显示的配置。

代码配色我是选择的自己备份的主题 Peacock (SL).tmTheme ,放置在了 Packages/User/theme/ 目录下，主要是我针对 markdown 语法进行了设置，其他可选择的推荐 Dayle Rees Color Schemes 插件。

主题配色我使用的是 Theme-Phoenix 插件，插播一句，编程的字体应该选择等宽类型的，所以Mac下选择的是 Monaco 字体。

安装完了插件，可以在 Preferences->Color Scheme 中查看修改。也可以通过配置文件修改：

```
{
  "caret_style": "phase",
  "color_scheme": "Packages/User/theme/Peacock (SL).tmTheme",
  "default_line_ending": "unix",
  "font_face": "Monaco",
  "font_size": 18.0,
```

```

    "highlight_line": true,
    "hot_exit": false,
    "highlight_modified_tabs": true,
    "show_encoding": true,
    "ignored_packages":
    [
        "Vintage"
    ],
    "original_color_scheme": "Packages/User/theme/Peacock (SL).tmTheme",
    "phoenix_color_green": true,
    "phoenix_dirty_bottom_bar_red": true,
    "phoenix_eighties": true,
    "phoenix_highlight_current_tab": true,
    "phoenix_sidebar_tree_large": true,
    "phoenix_solid_current_tab": true,
    "phoenix_tabs_medium": true,
    "rulers":
    [
        80,
        100,
        120
    ],
    "soda_folder_icons": false,
    "tab_size": 4,
    "theme": "Phoenix Dark.sublime-theme",
    "translate_tabs_to_spaces": true,
    "word_separators": ".\\(\\)\\'\":,.;<-!@#$$%^&*|+=[]{}`~`?",
    "word_wrap": true,
    "wrap_width": 0
}

```

这里我列出的是我的全部配置文件，可以看到相关的主题配色、代码配色和字体设置。

## ST辅助类

- **SideBarEnhancements** 提升右侧导航栏功能
- **Sublimerge Pro** 文件对比功能
- **Markdown Preview** 书写markdown格式文本，预览等功能。
- **Terminal** 直接在对应文件所在目录打开terminal功能。

## 代码显示辅助类

- **BracketHighlighter** 高亮显示匹配括号，会在左侧的行号标识处显示对应的括号位置和范围。
- **HTML-CSS-JS Prettify** 格式化代码工具，默认快捷键 `ctrl+shift+h`。
- **CSScomb** 按照一定规律格式化CSS的属性顺序。

## 代码书写辅助类

- **Emmet** 必装插件，辅助书写HTML, CSS。
- **AutoFileName** 书写代码时，自动提示补充文件路径。
- **DocBlockr** 辅助书写注释。
- **JSHint Gutter** 利用 `jslint` 检测js代码是否规范的插件。
- **LiveStyle** 配合对应的chrome插件，可以达到修改文件后，自动刷新页面的效果。但目前对 `less`, `sass` 之类预编译语言支持不够好。



- **AlignTab** 变量竖向对齐工具。
- **All Autocomplete** 代码补全插件。
- **EditorConfig** 代码编码规范。

## 使用技巧

### 快捷键操作

默认的快捷操作，可以查看 [Preferences->Key Binding](#)，或者文档：[Keyboard Shortcuts-Windows/Linux](#)和[Keyboard Shortcuts-OSX](#)。

个人常用的快捷键设置如下：

```
[
/*===== Emacs Style =====*/
{ "keys": ["ctrl+b"], "command": "move", "args": {"by": "characters", "forward": false} },
{ "keys": ["ctrl+f"], "command": "move", "args": {"by": "characters", "forward": true} },
{ "keys": ["ctrl+p"], "command": "move", "args": {"by": "lines", "forward":
false} },
{ "keys": ["ctrl+n"], "command": "move", "args": {"by": "lines", "forward":
true} },
{ "keys": ["ctrl+a"], "command": "move_to", "args": {"to": "bol", "extend": false} },
{ "keys": ["ctrl+e"], "command": "move_to", "args": {"to": "eol", "extend": false} },
{ "keys": ["ctrl+l"], "command": "show_at_center" },
/*===== End Emacs Style =====*/

/*===== switch tabs =====*/
{ "keys": ["ctrl+1"], "command": "select_by_index", "args": { "index": 0 } },
{ "keys": ["ctrl+2"], "command": "select_by_index", "args": { "index": 1 } },
{ "keys": ["ctrl+3"], "command": "select_by_index", "args": { "index": 2 } },
{ "keys": ["ctrl+4"], "command": "select_by_index", "args": { "index": 3 } },
{ "keys": ["ctrl+5"], "command": "select_by_index", "args": { "index": 4 } },
{ "keys": ["ctrl+6"], "command": "select_by_index", "args": { "index": 5 } },
{ "keys": ["ctrl+7"], "command": "select_by_index", "args": { "index": 6 } },
{ "keys": ["ctrl+8"], "command": "select_by_index", "args": { "index": 7 } },
{ "keys": ["ctrl+9"], "command": "select_by_index", "args": { "index": 8 } },
{ "keys": ["super+shift+t"], "command": "reopen_last_file" },
/*===== End switch tabs =====*/

/*===== Modify Default key-mapping =====*/
{ "keys": ["super+a"], "command": "select_all" },
{ "keys": ["super+t"], "command": "new_file" },
{ "keys": ["f5"], "command": "open_in_browser" },
// autocomplete
{ "keys": ["ctrl+/"], "command": "auto_complete" },
// paste
{ "keys": ["super+v"], "command": "paste_and_indent" },
{ "keys": ["super+shift+v"], "command": "paste" },
// reindex
{ "keys": ["ctrl+i"], "command": "reindent" },
// find and goto
{ "keys": ["super+f"], "command": "show_panel", "args": {"panel": "find"} },
{ "keys": ["super+p"], "command": "show_overlay", "args": {"overlay": "goto", "show_files": true} },
{ "keys": ["super+r"], "command": "show_overlay", "args": {"overlay": "goto", "text": "@"} },
{ "keys": ["super+l"], "command": "show_overlay", "args": {"overlay": "goto", "text": "."} },
{ "keys": ["super+,"], "command": "show_overlay", "args": {"overlay": "goto", "text": "#"} },
{ "keys": ["super+d"], "command": "goto_definition" },
{ "keys": ["super+-"], "command": "jump_back" },
{ "keys": ["super+="], "command": "jump_forward" },
// keep the shortcut as Windows
{ "keys": ["ctrl+d"], "command": "find_under_expand" },
{ "keys": ["ctrl+g"], "command": "find_all_under" },
{ "keys": ["ctrl+j"], "command": "join_lines" },
{ "keys": ["ctrl+shift+j"], "command": "expand_selection", "args": {"to": "indentation"} },
{ "keys": ["ctrl+shift+k"], "command": "run_macro_file", "args": {"file": "res://Packages/Default/Delete Line.subli
/*===== End Modify Default key-mapping =====*/
```

```

/*===== Plugin =====*/
// Emmet expand
{"keys": ["super+e"], "args": {"action": "expand_abbreviation"}, "command": "run_emmet_action", "context": [{"key":
// js Hint Grunt
{"keys": ["super+j"], "command": "jshint"},
// markdown preview
{ "keys": ["super+m"], "command": "markdown_preview", "args": {"target": "browser", "parser": "markdown"} },
// terminal
{ "keys": ["ctrl+super+t"], "command": "open_terminal" },
{ "keys": ["ctrl+shift+super+t"], "command": "open_terminal_project_folder" }
/*===== End Plugin =====*/
]

```

其中涉及到了emacs移动光标，多标签切换，以及快速查找等方式。

## snippet

snippet是代码片段，可以方便的自动补全。创建方式通过 `Tools->New Snippet` 完成。

默认的文件如下：

```

<snippet>
  <content><![CDATA[
Hello, ${1:this} is a ${2:snippet}.
]]></content>
  <!-- Optional: Set a tabTrigger to define how to trigger the snippet -->
  <!-- <tabTrigger>hello</tabTrigger> -->
  <!-- Optional: Set a scope to limit where the snippet will trigger -->
  <!-- <scope>source.python</scope> -->
</snippet>

```

代码段写在 `CDATA[]` 中，`${}` 为占位字符。

`tabTrigger` 为自动补全需要的字符，`scope` 设置的是文件格式。

创建完成之后，个人建议保存在 `User->snippet` 目录下，`snippet` 需要自行创建，方便管理。

## build命令和Macro命令

这些命令的使用请参考文档->[Reference](#)。

## 参考文档

- [sublimeText官网](#)
- [非官方手册](#)
- [Package Control](#)
- [推荐！Sublime Text 最佳插件列表](#)
- [Gif多图：我常用的 16 个 Sublime Text 快捷键](#)

# Gulp

前端自动化流程管理。在JavaScript的世界里，Grunt.js是基于Node.js的自动化任务运行器。2013年02月18日，Grunt v0.4.0发布。Fractal公司积极参与了数个流行Node.js模块的开发，它去年发布了一个新的构建系统Gulp，希望能够取其精华，并取代Grunt，成为最流行的JavaScript任务运行器。

## Gulp和Grunt的异同点

- 易于使用：采用代码优于配置策略，Gulp让简单的事情继续简单，复杂的任务变得可管理。
- 高效：通过利用Node.js强大的流，不需要往磁盘写中间文件，可以更快地完成构建。
- 高质量：Gulp严格的插件指导方针，确保插件简单并且按你期望的方式工作。
- 易于学习：通过把API降到最少，你能在很短的时间内学会Gulp。构建工作就像你设想的一样：是一系列流管道。

## Gulp特点

- 易用：Gulp相比Grunt更简洁，而且遵循代码优于配置策略，维护Gulp更像是写代码。
- 高效：Gulp相比Grunt更有设计感，核心设计基于Unix流的概念，通过管道连接，不需要写中间文件。
- 高质量：Gulp的每个插件只完成一个功能，这也是Unix的设计原则之一，各个功能通过流进行整合并完成复杂的任务。例如：Grunt的imagemin插件不仅压缩图片，同时还包括缓存功能。他表示，在Gulp中，缓存是另一个插件，可以被别的插件使用，这样就促进了插件的可重用性。目前官方列出的有673个插件。

## Gulp示例

```
var gulp = require('gulp');
var jshint = require('gulp-jshint');
var concat = require('gulp-concat');
var rename = require('gulp-rename');
var uglify = require('gulp-uglify');

// Lint JS
gulp.task('lint', function() {
  return gulp.src('src/*.js')
    .pipe(jshint())
    .pipe(jshint.reporter('default'));
});

// Concat & Minify JS
gulp.task('minify', function(){
  return gulp.src('src/*.js')
    .pipe(concat('all.js'))
    .pipe(gulp.dest('dist'))
    .pipe(rename('all.min.js'))
    .pipe(uglify())
    .pipe(gulp.dest('dist'));
});

// Watch Our Files
gulp.task('watch', function() {
  gulp.watch('src/*.js', ['lint', 'minify']);
});

// Default
gulp.task('default', ['lint', 'minify', 'watch']);
```

## 参考资料

- [gulp](#)
- [gulp fiction](#): 可视化配置gulp工作流程。
- [前端工程的构建工具对比 Gulp vs Grunt](#)

## 字体的选择

---

编程的字体，最好选择的是等宽，这样代码看起来比较规整。

## 参考资料

---

- [最佳编程字体：M+](#)

## Emacs

---

### 参考资料

---

- [一年成为Emacs高手\(像神一样使用编辑器\)](#)

## Sketch

---

本项目派生自 [github.com/diessica/awesome-sketch](https://github.com/diessica/awesome-sketch) 并进行中文化，同时适当增加了一些内容。欢迎派生并提交合并请求来完善此文档。谢谢！

# awesome Sketch

---

你是否知道 [Sketch 3](#) 正在成为最好的 UI/UX 设计工具？好吧... 那你知道多少？

这是一份为想学 Sketch 的设计师、前端工程师们准备的不完全列表，包含了 Sketch 视频、文章、手册等。

- 视频
  - [Sketch 3 教程](#) (19 惊赞 课程)
  - [LearnSketch 频道](#)
  - [一步一步学 Sketch 3 for iOS 程序设计](#)
- 手册
  - [Sketch 快捷键英文版 / 中文版](#)
  - [官方文档 / 中文版](#)
- 文章
  - [Fireworks vs. Sketch](#)
  - [Supercharge your Workflow in Sketch](#)
  - [Sketch for Beginners: Design a Login Form Interface](#)
  - [The A to Z of Sketch](#)
  - [7 Tips for Sketch Users](#)
  - [9 Sketch Features You Should be Using](#)
  - [Mastering the Bézier Curve in Sketch](#)
  - [10 Tips & Tricks for Sketch](#)
  - [Typography in Sketch 3: Linked Text Styles](#)
  - [11 tips for prototyping with Sketch](#)
- 社区
  - [TeamSketch](#), 基于 Slack 的 Sketch 交流社区
  - [Sketch 中文用户讨论组](#), 基于 Slack 的交流社区
  - [Google+ 群组](#)
  - [Facebook 群组](#)
  - [位于 Reddit](#)
  - [SketchTalk](#), 非官方论坛
  - [Sketch 中文网](#)
- 其它
  - (邮件列表) [Sketch 官方邮件列表](#)
  - (邮件列表) [Sketch tricks](#)
  - (Screencast) [SketchCasts](#)
  - (资源) [SketchApp Resources](#)
  - (资源) [SketchLand](#)
  - (资源) [SketchResources](#)
  - [SketchTips](#), 一个关于 Sketch 的博客
  - [Medium 上的 sketch-tricks](#)
- 插件

插件管理器 [Sketch Toolbox](#) 强烈推荐.



- 必不可少的插件
  - [Content Generator](#) : 内容自动生成工具
  - [Renamelt](#) : 改名工具
  - [Sketch Measure](#) : 测量工具
  - [Style Inventory](#): 样式清单
  - [Dynamic Button](#) : 可调整按钮
  - [Page Switch](#) : 页面切换
- 哪里可以得到更多
  - [SketchApp Resources: Plugins for Sketch](#)
  - [Sketch Plugin Directory](#)
  - [SketchPlugins mailing list](#)
  - [Awesome Sketch Plugins](#)

## Trello

---

Trello由Joel Spolsky创建的Fog Creek公司开发，是一种在线的看板式管理应用程序，从创建以来一直不断改进，已经有多家公司开始使用它来管理敏捷项目。

## 使用Trello管理项目的经验

---

### 参考资料

---

- [使用Trello实现敏捷项目管理](#)
- [用 Trello 做项目管理](#)

## git进阶

---

## 15分钟学会使用Git和远程代码库

---

### 工作步骤

---

- 创建一个远程的空代码库（在BitBucket上）
- 在本地代码库添加一个项目
- 在分支上开发新功能
- a) 保留新功能 或者 b) 丢弃它们
- 也许，回到某个早先的时间点
- 将本地代码库推送到远程代码库
- 在另一台机器上取得远程代码库

### 参考资料

---

- [15分钟学会使用Git和远程代码库](#)

# GitHub秘籍

---

本秘籍收录了一些Git和Github非常酷同时又少有人知的功能。灵感来自于Zach Holman在2012年Aloha Ruby Conference和2013年WDCNZ上所做的演讲：[Git and GitHub Secrets\(slides\)](#)和[More Git and GitHub Secrets\(slides\)](#)。

Read this in other languages: [English](#), [한국어](#), [日本語](#), [简体中文](#).

## 目录

---

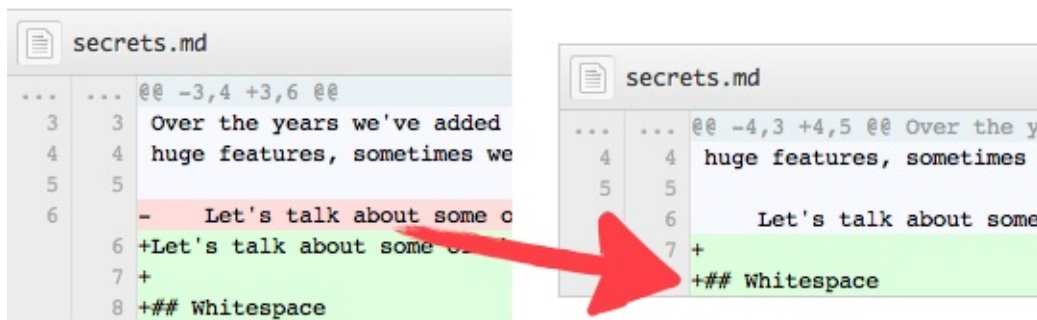
- [GitHub](#)
  - [忽略空白字符变化](#)
  - [调整Tab字符所代表的空格数](#)
  - [查看某个用户的Commit历史](#)
  - [克隆某个仓库](#)
  - [分支](#)
    - [将某个分支与其他所有分支进行对比](#)
    - [比较分支](#)
    - [比较不同派生库的分支](#)
  - [Gists](#)
  - [Git.io](#)
  - [键盘快捷键](#)
  - [整行高亮](#)
  - [用commit信息关闭Issue](#)
  - [链接其他仓库的Issue](#)
  - [设置CI对每条Pull Request都进行构建](#)
  - [Markdown文件高亮语法](#)
  - [表情符](#)
  - [静态与动态图片](#)
    - [在GitHub Wiki中嵌入图片](#)
  - [快速引用](#)
  - [快速添加许可证](#)
  - [任务列表](#)
    - [Markdown文件中的任务列表](#)
  - [相对链接](#)
  - [GitHub Pages的元数据与插件支持](#)
  - [查看YAML格式的元数据](#)
  - [渲染表格数据](#)
  - [撤销Pull Request](#)
  - [Diffs](#)
    - [可渲染文档的Diffs](#)
    - [可变化地图](#)
    - [在diff中折叠与扩展代码](#)
    - [查看Pull Request的diff和patch](#)
    - [渲染图像发生的变动](#)
  - [Hub](#)
  - [贡献者指南](#)
  - [GitHub资源](#)
    - [GitHub讨论](#)
- [Git](#)
  - [前一个分支](#)

- Stripspace命令
- 检出Pull Requests
- 提交空改动 :trollface:
- 更直观的Git Status
- 更直观的Git Log
- Git查询
- 合并分支
- 使用网页查看本地仓库
- Git配置
  - Git命令自定义别名
  - 自动更正
  - 带颜色输出
- Git资源
  - Git参考书籍

## GitHub

### 忽略空白字符变化

在任意diff页面的URL后加上 `?w=1`，可以去掉那些只是空白字符的变化，使你能更专注于代码的变化。



详见 [GitHub secrets](#).

### 调整Tab字符所代表的空格数

在diff或者file页面的URL后面加上 `?ts=4`，这样当显示tab字符的长度时就会是4个空格的长度，不再是默认的8个空格。ts后面的数字还可以根据你个人的偏好进行修改。不过，这个小诀窍在Gists页面和raw file页面不起作用。

下面是我们在Go语言的source file页面URL后加 `?ts=4` 前的例子：

```
file 69 lines (57 sloc) 1.86 kb
Open Edit Raw Blame History Delete
1 package flint
2
3 import (
4     "path/filepath"
5 )
6
7 type lintError struct {
8     Level int
9     Message string
10 }
11
```

然后是我们添加 `?ts=4` 后的例子：

```

file 69 lines (57 sloc) 1.86 kb
Open Edit Raw Blame History Delete
1 package flint
2
3 import (
4     "path/filepath"
5 )
6
7 type lintError struct {
8     Level int
9     Message string
10 }
    
```




## 查看某个用户的Commit历史

查看某个用户的所有提交历史，只需在commits页面URL后加上 `?author=username`。





```
https://github.com/rails/rails/commits/master?author=dhh
```

branch: master rails / Commits

Apr 08, 2014

-  **Dont abbreviate that which needs no abbreviation**  
dhh authored 8 days ago 304d2f19c8 [Browse code](#)
-  **Dont encourage aliases now that we have variants**  
dhh authored 8 days ago 10570cfd5b [Browse code](#)
-  **Use short-form for the scaffold render calls and drop the needless test**  
dhh authored 8 days ago 4b0c8a9467 [Browse code](#)

Mar 21, 2014

-  **Update test helper to use latest Digester API**  
dhh authored a month ago 9d44b3f886 [Browse code](#)
-  **Digester should just rely on the finder to know about the format and ...**  
dhh authored a month ago 637bb726ca [Browse code](#)
-  **Log the full path, including variant, that the digester is trying to ...**  
dhh authored a month ago 4bca34750d [Browse code](#)
-  **Fix for digester to consider variants for partials -- this still need...**  
dhh authored a month ago 06b4f01fca [Browse code](#)

[深入了解提交视图之间的区别](#)

## 克隆某个仓库

当我们克隆某一资源时，可以不要那个 `.git` 后缀。

```
$ git clone https://github.com/tiimgreen/github-cheat-sheet
```

[更多对 Git clone 命令的介绍.](#)

## 分支

### 将某个分支与其他所有分支进行对比

当你点击某个仓库的分支 (Branches) 选项卡时



```
https://github.com/{user}/{repo}/branches
```

你会看到一个包含所有未合并的分支的列表。

你可以在这里查看比较（Compare）页面或点击删除某个分支。

**Branches** Recently Active Stale

Showing 3 branches not merged into master. [View merged branches.](#)

Branch	Status	Commits	Action
<b>master</b>	Base branch	✓ Last updated 3 days ago by mzogol.	
<b>1.x-master</b>	547 ahead 743 behind	✓ Last updated 14 hours ago by mzogol.	<a href="#">Compare</a>
<b>delegation</b>	1 ahead 110 behind	✓ Last updated 4 months ago by timmywil.	<a href="#">Compare</a>
<b>1.9-stable</b>	106 ahead 743 behind	Last updated a year ago by tomfuertes.	<a href="#">Compare</a>

有的时候我们需要将多个分支与一个非主分支（master）进行对比，此时可以通过在URL后加入要比较的分支名来实现：

```
https://github.com/{user}/{repo}/branches/{branch}
```

**Branches** Recently Active Stale

Showing 2 branches not merged into 1.x-master. [View merged branches.](#)

Branch	Status	Commits	Action
<b>1.x-master</b>	Base branch	✓ Last updated 14 hours ago by mzogol.	
<b>master</b>	743 ahead 547 behind	✓ Last updated 3 days ago by mzogol.	<a href="#">Compare</a>
<b>delegation</b>	634 ahead 547 behind	✓ Last updated 4 months ago by timmywil.	<a href="#">Compare</a>

可以在URL后加上 `?merged=1` 来查看已经合并了的分支。

**Branches** Recently Active Stale

Showing 1 branch merged into 1.x-master. [View unmerged branches.](#)

Branch	Status	Commits	Action
<b>1.x-master</b>	Base branch	✓ Last updated 14 hours ago by mzogol.	
<b>1.9-stable</b>	0 ahead 441 behind	Last updated a year ago by tomfuertes.	<a href="#">Compare</a>

你可以使用这个界面来替代命令行直接删除分支。

## 比较分支

如果我们想要比较两个分支，可以像下面一样修改URL：

```
https://github.com/user/repo/compare/{range}
```

其中 {range} = master...4-1-stable

例如：

```
https://github.com/rails/rails/compare/master...4-1-stable
```

🔗 master ... 4-1-stable Edit

Please review the [guidelines for contributing](#) to this repository.

Create Pull Request Open a Pull Request for this comparison to discuss and review your changes with others. ?

🔗 247 commits    📄 273 files changed    💬 5 comments    👤 36 contributors

Commits    Files changed    Commit comments

Feb 18, 2014		
	Update versions for 4.1.0.rc1	✖ 211ec1f
	Revert "Update versions for 4.1.0.rc1" -- old format for versions! ...	ccddc40
	Update versions for 4.1.0.rc1 (using new format)	✔ 78ba185
	Pointing README links to 4-1-stable [ci skip]	51bd49b
	`rails new --edge` should use the '4-1-stable' branch ...	✔ 24e1fff
	Merge pull request #14100 from chancancode/rails_new_edge ...	✔ e27b6fe

{range} 还可以使用下面的形式:

```
https://github.com/rails/rails/compare/master@{1.day.ago}...master  
https://github.com/rails/rails/compare/master@{2014-10-04}...master
```

日期格式 YYYY-DD-MM

🔗 master@{2014-10-04} ... master Edit

🔗 130 commits    📄 123 files changed    💬 5 comments    👤 39 contributors

Commits    Files changed    Commit comments

Aug 07, 2013		
	Add tests to ActiveSupport:XmlMini to_tag method	✔ 05d7cde
Nov 17, 2013		
	Fix insertion of records for hmt association with scope, fix #3548	✖ ec09280
Jan 07, 2014		
	Auto-generate stable fixture UUIDs on PostgreSQL. ...	✔ 9330631

...这样你就能查看master分支上一段时间或者指定日期内的改动。

[了解更多关于比较跨时间段的提交记录.](#)

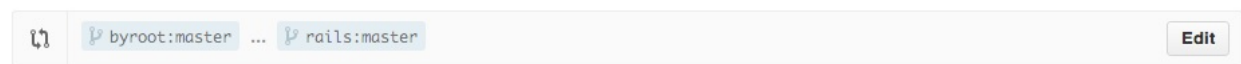
## 比较不同派生库的分支

想要对派生仓库 (Forked Repository) 之间的分支进行比较, 可以像下面这样修改URL实现 :

```
https://github.com/user/repo/compare/{foreign-user}:{branch}...{own-branch}
```

例如 :

```
https://github.com/rails/rails/compare/byroot:master...master
```



Please review the [guidelines for contributing](#) to this repository.

[Create Pull Request](#) Open a Pull Request for this comparison to discuss and review your changes with others. ?

↩ 6,802 commits      📄 309 files changed      💬 14 comments      👤 64 contributors

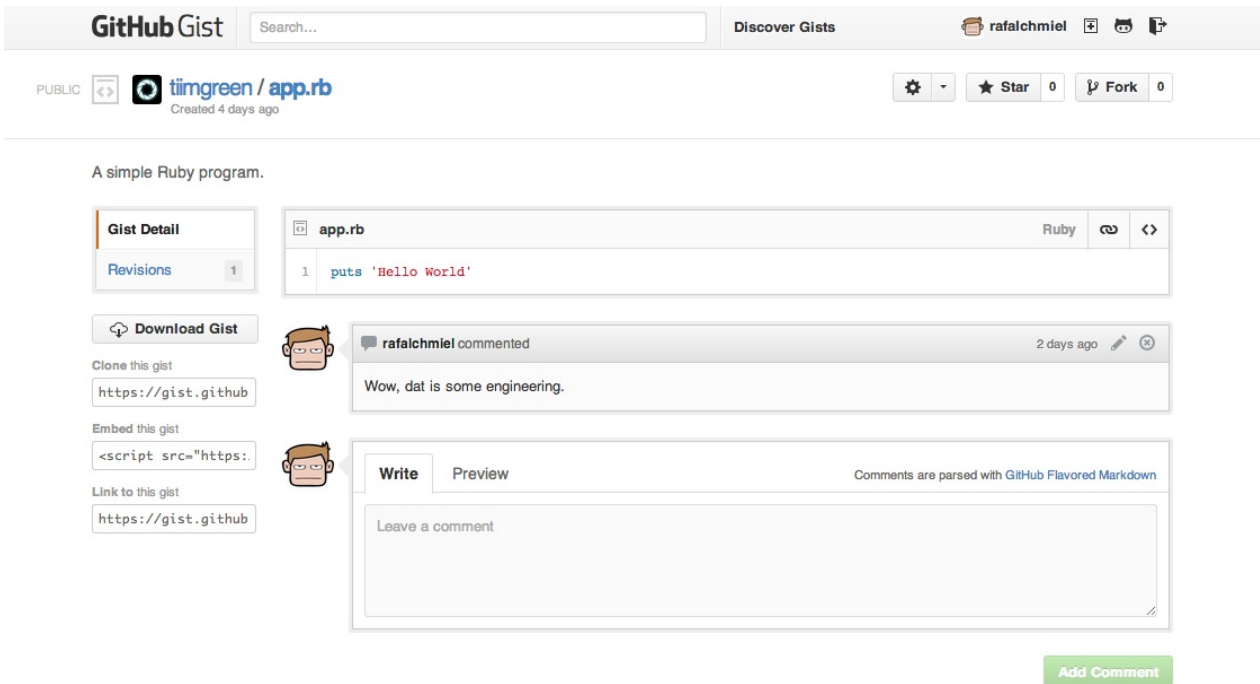
Commits    Files changed    Commit comments

**This comparison is big! We're only showing the most recent 250 commits**

Apr 02, 2014		
	PostgreSQL, Support for materialized views. [Dave Lee & Yves Senn] ...	✓ def6071
	We can conditional define the tests depending on the adapter or ...	✗ ee36af1
	Merge pull request #14565 from rajcybage/conditional_test_cases ...	✓ c82483a
	DRY AS::SafeBuffer a bit using existing helper	✓ 8482895
	Fixed small documentation typo ...	✗ 8ae3f24
	Merge pull request #14568 from alex88/patch-1 ...	✓ 3bcc51a

## Gists

[Gists](#) 给我们提供了一种不需要创建一个完整的仓库, 使小段代码也可以工作的简单方式。



Gist的URL后加上 `.pibb`，可以得到更适合嵌入到其他网站的HTML版本。

Gists还可以像任何标准仓库一样被克隆。

```
$ git clone https://gist.github.com/tiimgreen/10545817
```



这意味着你可以像 Github 仓库一样修改和更新 Gists：

```
$ git commit
$ Username for 'https://gist.github.com':
```

```
$ Password for 'https://tiimgreen@gist.github.com':
```

[进一步了解如何创建 gists.](#)

## Git.io

Git.io是Github的短网址服务。



你可以通过Curl命令以普通HTTP协议使用它：

```
$ curl -i http://git.io -F "url=https://github.com/..."
HTTP/1.1 201 Created
Location: http://git.io/abc123

$ curl -i http://git.io/abc123
HTTP/1.1 302 Found
Location: https://github.com/...
```

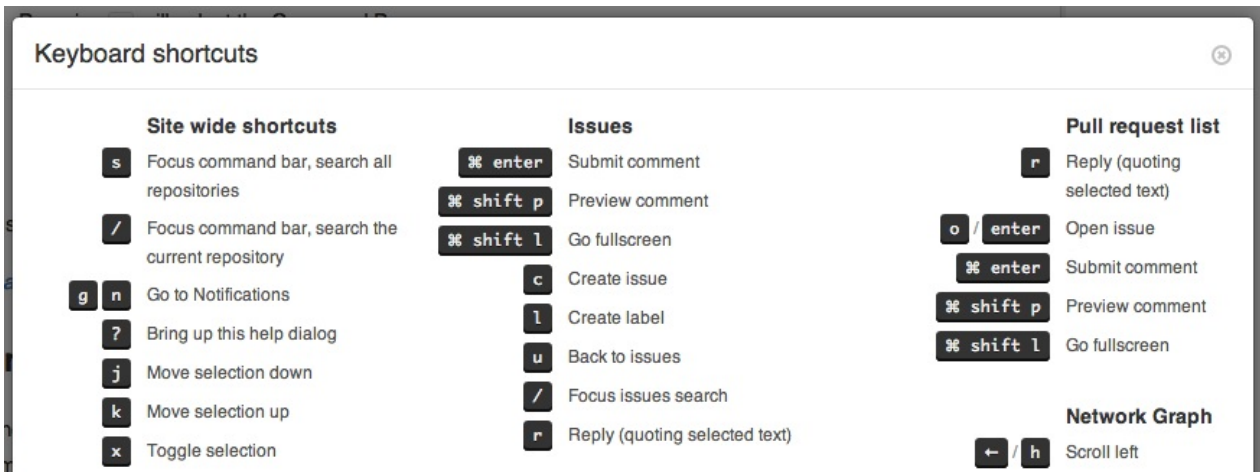
[进一步了解 Git.io.](#)

## 键盘快捷键

在仓库主页上提供了快捷键方便快速导航。

- 按 `t` 键会打开一个文件浏览器。
- 按 `w` 键会打开分支选择菜单。
- 按 `s` 键会激活顶端的命令栏 (Command Bar)。
- 按 `l` 键编辑Issue列表页的标签。
- 查看文件内容时（如：`https://github.com/tiimgreen/github-cheat-sheet/blob/master/README.md`），按 `y` 键将会冻结这个页面，这样就算代码被修改了也不会影响你当前看到的。

按 `?` 查看当前页面支持的快捷键列表：



进一步了解如何使用 [Command Bar](#).

## 整行高亮

在代码文件地址后加上 `#L52` 或者单击行号52都会将第52行代码高亮显示。

多行高亮也可以，比如用 `#L53-L60` 选择范围，或者按住 `shift` 键，然后再点击选择的两行。

```
https://github.com/rails/rails/blob/master/activemodel/lib/active_model.rb#L53-L60
```

```

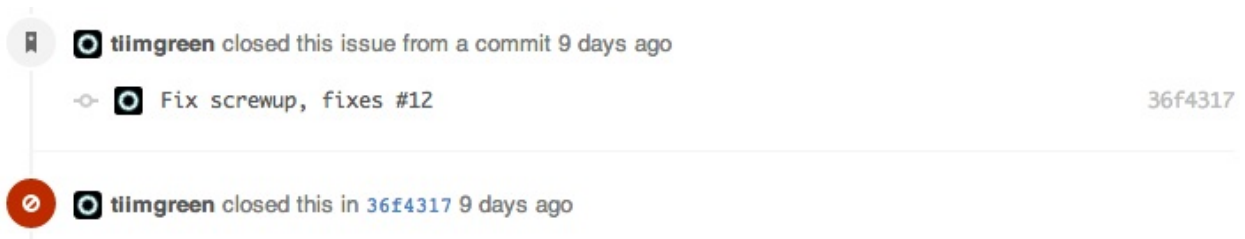
43   autoload :Serialization
44   autoload :TestCase
45   autoload :Translation
46   autoload :Validations
47   autoload :Validator
48
49   eager_autoload do
50     autoload :Errors
51   end
52
53   module Serializers
54     extend ActiveSupport::Autoload
55
56     eager_autoload do
57       autoload :JSON
58       autoload :Xml
59     end
60   end
61
62   def self.eager_load!
63     super
64     ActiveSupport::Serializers.eager_load!
65   end
66 end
67
68 ActiveSupport.on_load(:i18n) do
69   I18n.load_path << File.dirname(__FILE__) + '/active_model/locale/en.yml'
70 end
    
```

## 用commit信息关闭Issue

如果某个提交修复了一个Issue，当提交到master分支时，提交信息里可以使用 `fix/fixes/fixed`，`close/closes/closed` 或者 `resolve/resolves/resolved` 等关键词，后面再跟上Issue号，这样就会关闭这个Issue。

```
$ git commit -m "Fix screwup, fixes #12"
```

这将会关闭Issue #12, 并且在Issue讨论列表里关联引用这次提交。

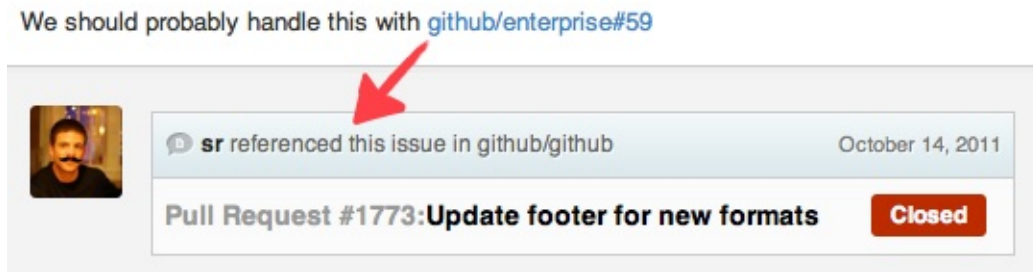


进一步了解通过提交信息关闭Issue.

## 链接其他仓库的Issue

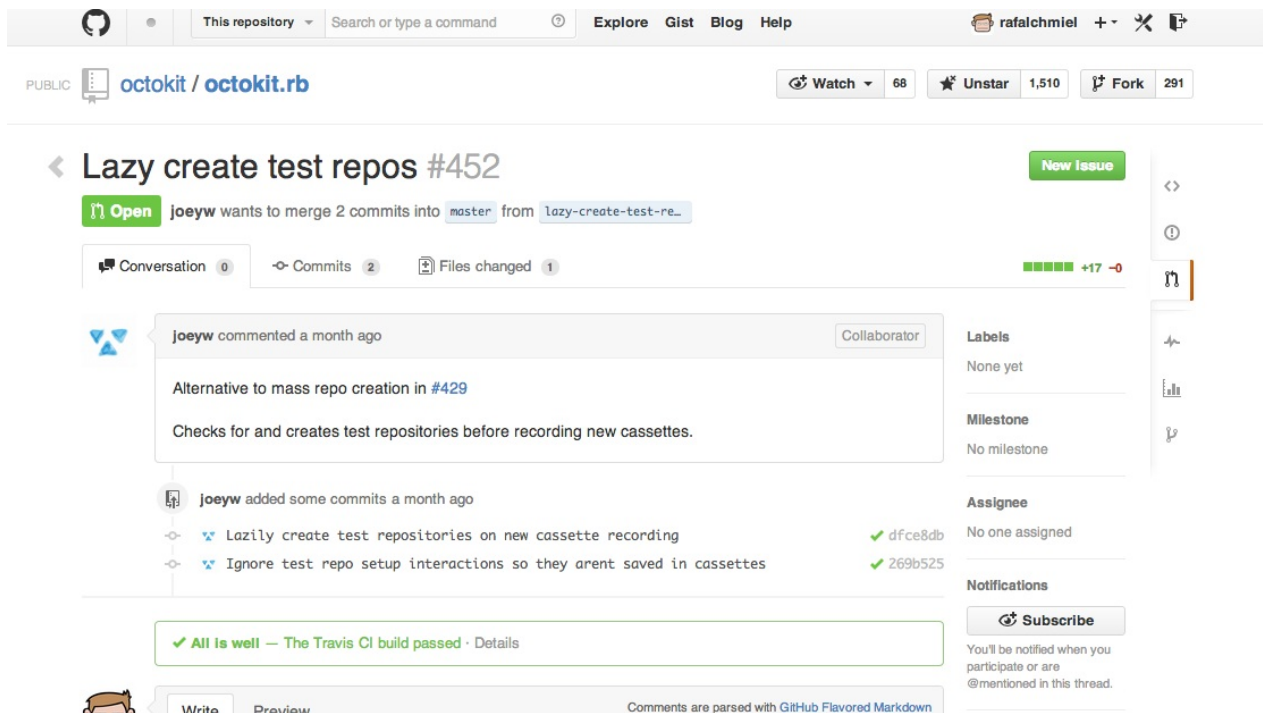
如果你想引用到同一个仓库中的一个Issue, 只需使用井号 # 加上Issue号, 这样就会自动创建到此Issue的链接。

要链接到其他仓库的Issue, 就使用 `user_name/repo_name#ISSUE_NUMBER` 的方式, 例如 `tiimgreen/toc#12`。



## 设置CI对每条Pull Request都进行构建

如果配置正确, Travis CI会为每个你收到的Pull Request执行构建, 就像每次提交也会触发构建一样。想了解更多关于Travis CI的信息, 请看 [Travis CI入门](#)。



进一步了解 [Commit status API](#).

## Markdown文件高亮语法

例如，可以像下面这样在你的Markdown文件里为Ruby代码添加语法高亮：

```
```ruby
require 'tabbit'
table = Tabbit.new('Name', 'Email')
table.add_row('Tim Green', 'tiimgreen@gmail.com')
puts table.to_s
```
```

效果像下面这样：

```
require 'tabbit'
table = Tabbit.new('Name', 'Email')
table.add_row('Tim Green', 'tiimgreen@gmail.com')
puts table.to_s
```

Github使用 [Linguist](#) 做语言识别和语法高亮。你可以仔细阅读 [languages YAML file](#)，了解有哪些可用的关键字。

[进一步了解 GitHub Flavored Markdown.](#)

## 表情符

可以在Pull Requests, Issues, 提交消息, Markdown文件里加入表情符。使用方法 `:name_of_emoji:`

```
:smile:
```

将输出一个笑脸：

```
:smile:
```

Github支持的完整表情符号列表详见[emoji-cheat-sheet.com](#) 或 [scotch-io/All-Github-Emoji-Icons](#)。

Github上使用最多的5个表情符号是：

1. `:shipit:` - `:shipit:`
2. `:sparkles:` - `:sparkles:`
3. `:-1:` - `:-1:`
4. `:+1:` - `:+1:`
5. `:clap:` - `:clap:`

## 静态与动态图片

注释和README等文件里也可以使用图片和GIF动画：

```
![Alt Text](http://www.sheawong.com/wp-content/uploads/2013/08/keephatin.gif)
```

仓库中的原始图片可以被直接调用：

```
![Alt Text](https://github.com/(user)/(repo)/raw/master/path/to/image.gif)
```





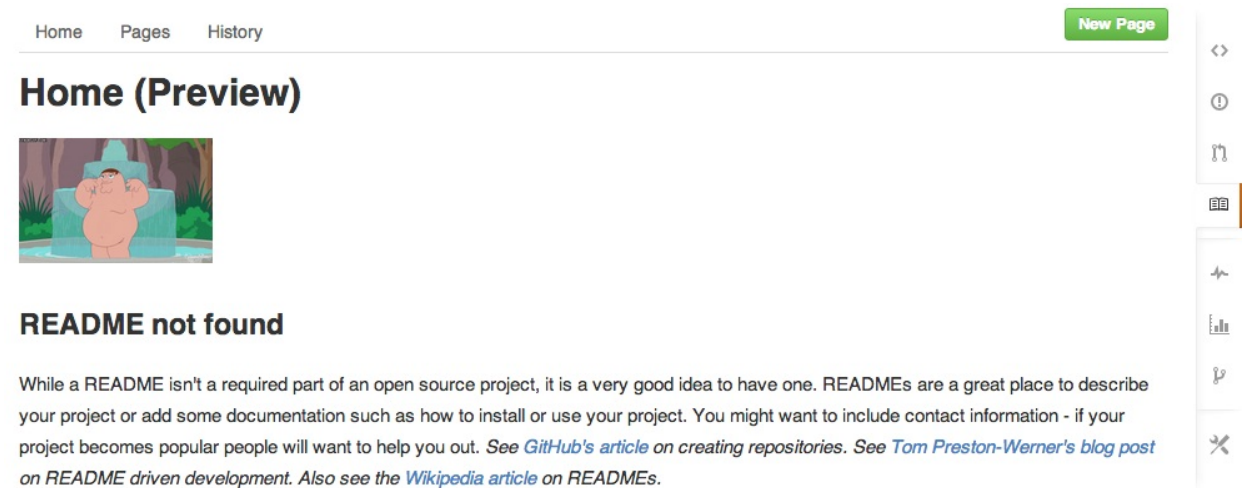
所有图片都缓存在Github，不用担心你的站点不能访问时就看不到图片了。

## 在GitHub Wiki中嵌入图片

有多种方法可以在Wiki页面里嵌入图片。既可以像上一条里那样使用标准的Markdown语法，也可以像下面这样指定图片的高度或宽度：

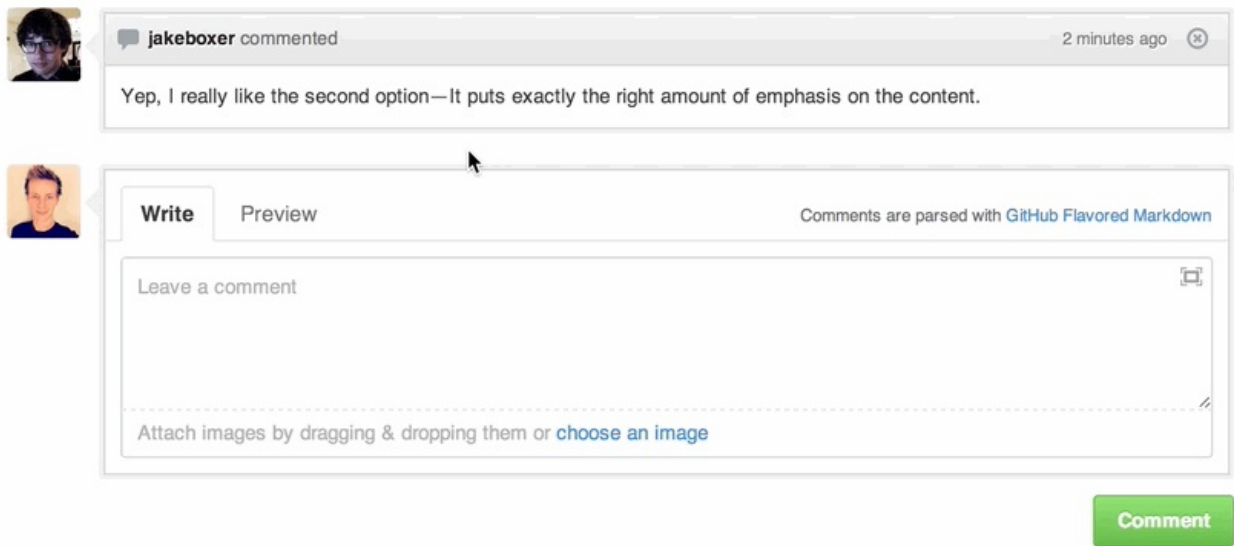
```
[[ http://www.sheawong.com/wp-content/uploads/2013/08/keephatin.gif | height = 100px ]]
```

结果：



## 快速引用

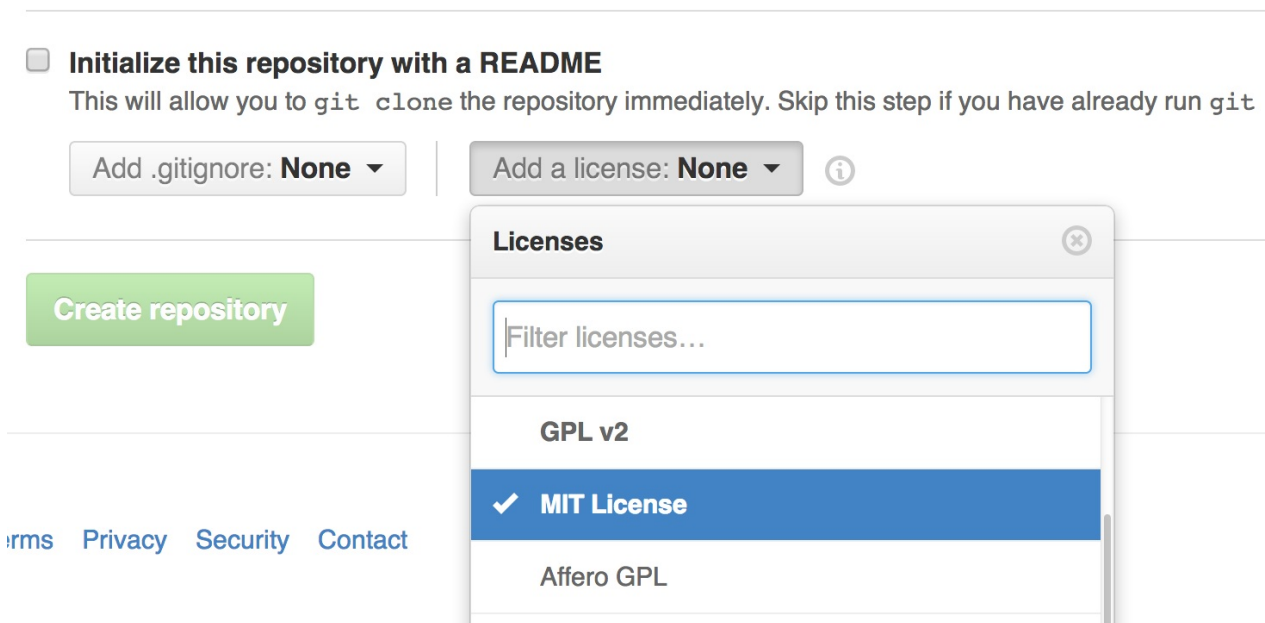
在主题评论中引用之前某个人所说的，只需选中文本，然后按 **r** 键，想要的就会以引用的形式复制到你的输入框里。



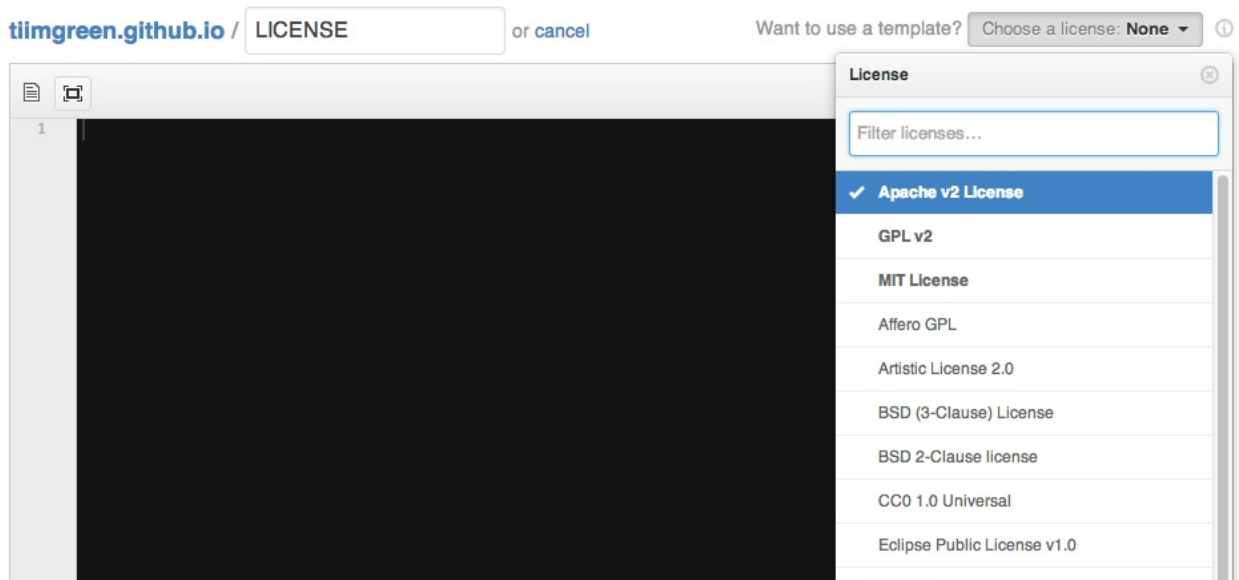
[进一步了解快速引用.](#)

## 快速添加许可证

创建一个仓库时，Github会为你提供一个预置的软件许可列表：



对于已有的仓库，可以通过web界面创建文件来添加软件许可。输入 `LICENSE` 作为文件名后，同样可以从预置的列表中选择 一个作为模板。



这个技巧也适用于 `.gitignore` 文件。

[进一步了解 open source licensing.](#)

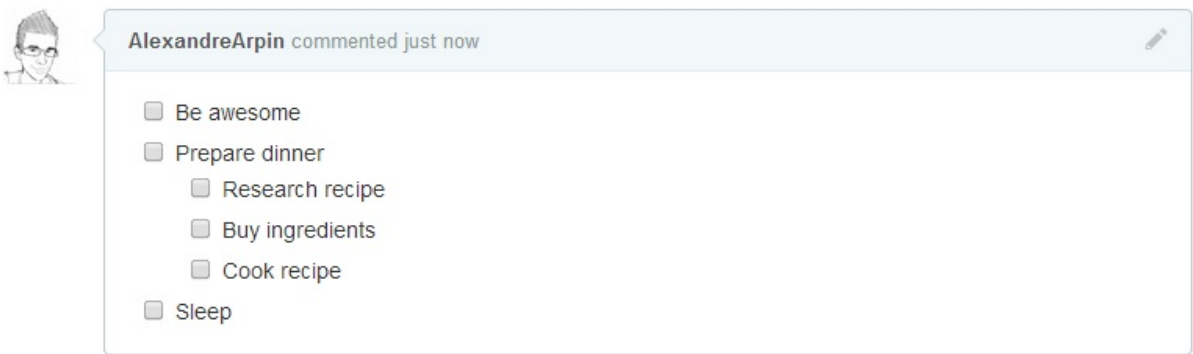
## 任务列表

Issues和Pull requests里可以添加复选框，语法如下（注意空白符）：

```
- [ ] Be awesome
- [ ] Prepare dinner
  - [ ] Research recipe
  - [ ] Buy ingredients
  - [ ] Cook recipe
- [ ] Sleep
```

## < TODO #1

Open AlexandreArpin opened this issue just now · 0 comments



当项目被选中时，它对应的Markdown源码也被更新了：

```
- [x] Be awesome
- [ ] Prepare dinner
  - [x] Research recipe
  - [x] Buy ingredients
  - [ ] Cook recipe
```

```
- [ ] Sleep
```

[进一步了解任务列表.](#)

## Markdown文件中的任务列表

在完全适配Markdown语法的文件中可以使用以下语法加入一个只读的任务列表

```
- [ ] Mercury
- [x] Venus
- [x] Earth
  - [x] Moon
- [x] Mars
  - [ ] Deimos
  - [ ] Phobos
```

- [ ] Mercury
- [x] Venus
- [x] Earth
  - [x] Moon
- [x] Mars
  - [ ] Deimos
  - [ ] Phobos

[进一步了解Markdown文件中的任务列表](#)

## 相对链接

Markdown文件里链接到内部内容时推荐使用相对链接。

```
[Link to a header](#awesome-section)
[Link to a file](docs/readme)
```

绝对链接会在URL改变时（例如重命名仓库、用户名改变，建立分支项目）被更新。使用相对链接能够保证你的文档不受此影响。

[进一步了解相对链接.](#)

## GitHub Pages的元数据与插件支持

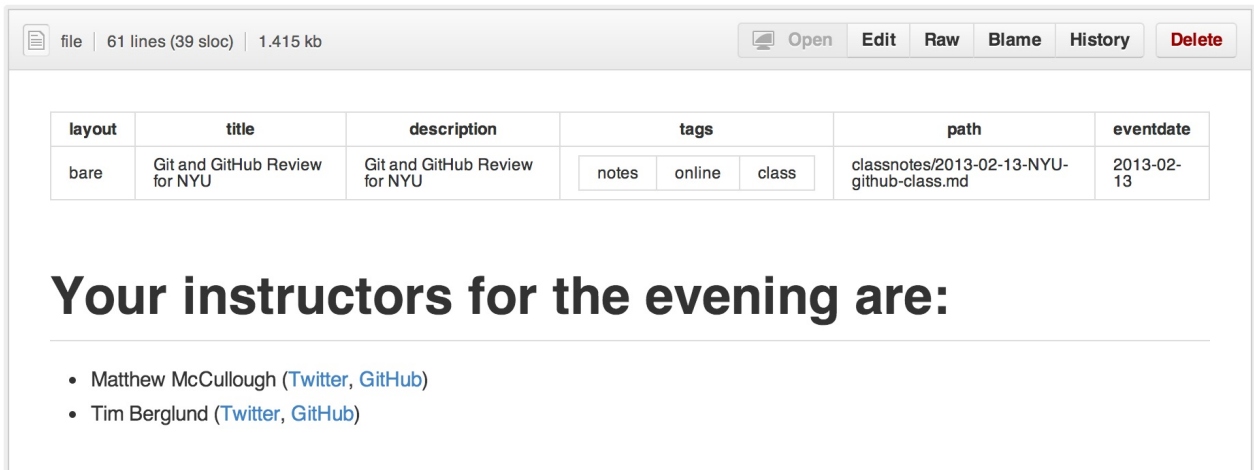
在Jekyll页面和文章里，仓库信息可在 `site.github` 命名空间下找到，也可以显示出来，例如，使用 ``显示项目标题。

Jemoji和jekyll-mentions插件为你的Jekyll文章和页面增加了emoji和@mentions功能。

[了解更多 GitHub Pages的元数据和插件支持.](#)

## 查看YAML格式的元数据

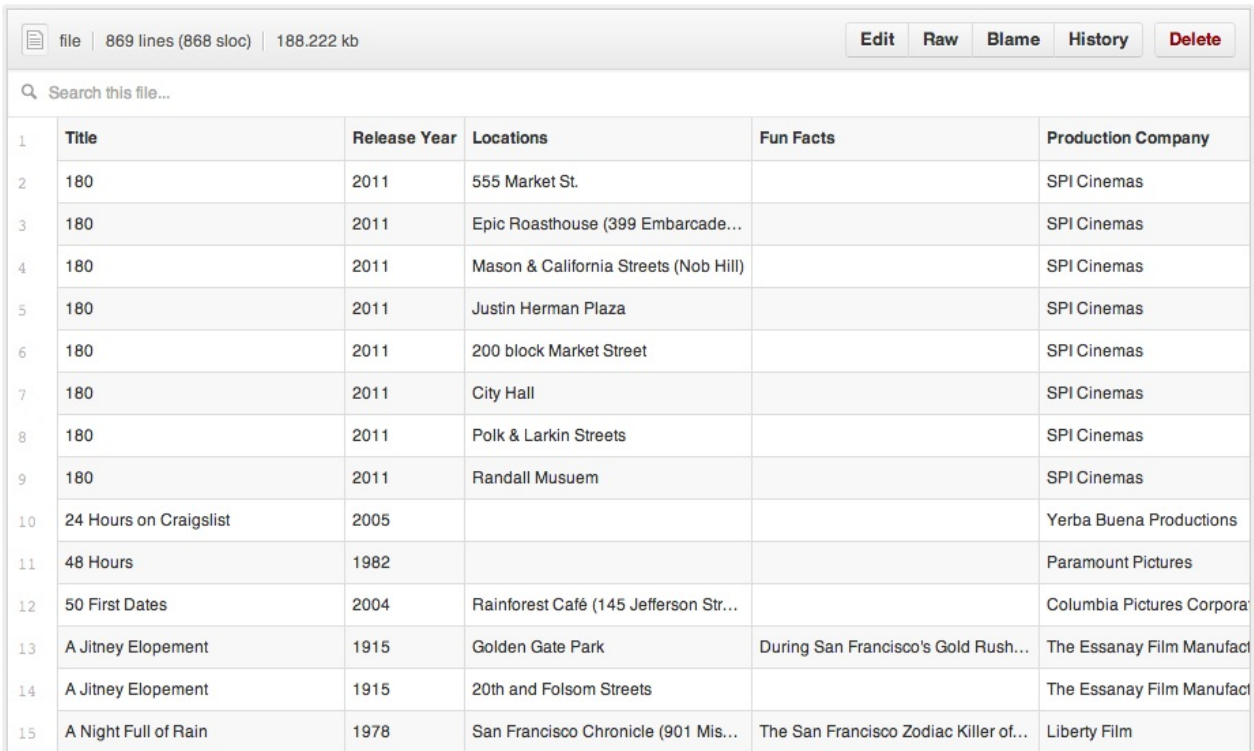
许多博客站点，比如基于Jekyll的GitHub Pages，都依赖于一些文章头部的YAML格式的元数据。Github会将其渲染成一个水平表格，方便阅读。



进一步了解 在文档里查看YAML元数据.

## 渲染表格数据

GitHub支持将 `.csv` (逗号分隔)和 `.tsv` (制表符分隔)格式的文件渲染成表格数据。



进一步了解渲染表格数据.

## 撤销Pull Request

可以通过Pull Request中的Revert按钮来撤销一个已合并的Pull Request中的commit。按下按钮后会自动生成一个进行逆向操作的Pull Request。

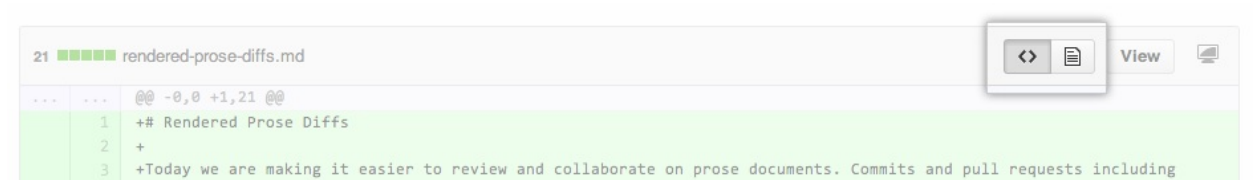


\*进一步了解“撤销”按钮

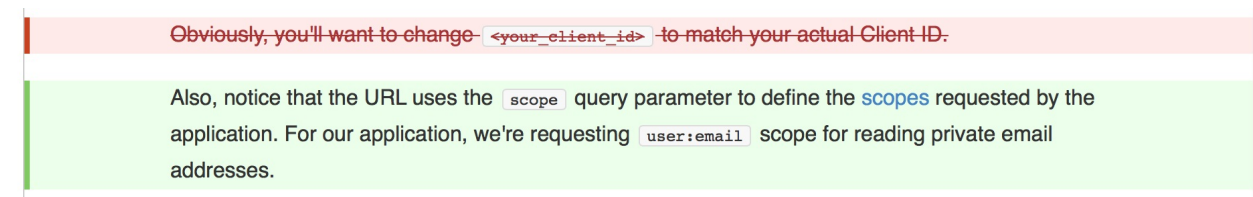
## Diffs

### 可渲染文档的Diffs

提交和Pull Requests里包含有Github支持的可渲染文档（比如Markdown）会提供source 和 rendered 两个视图功能。



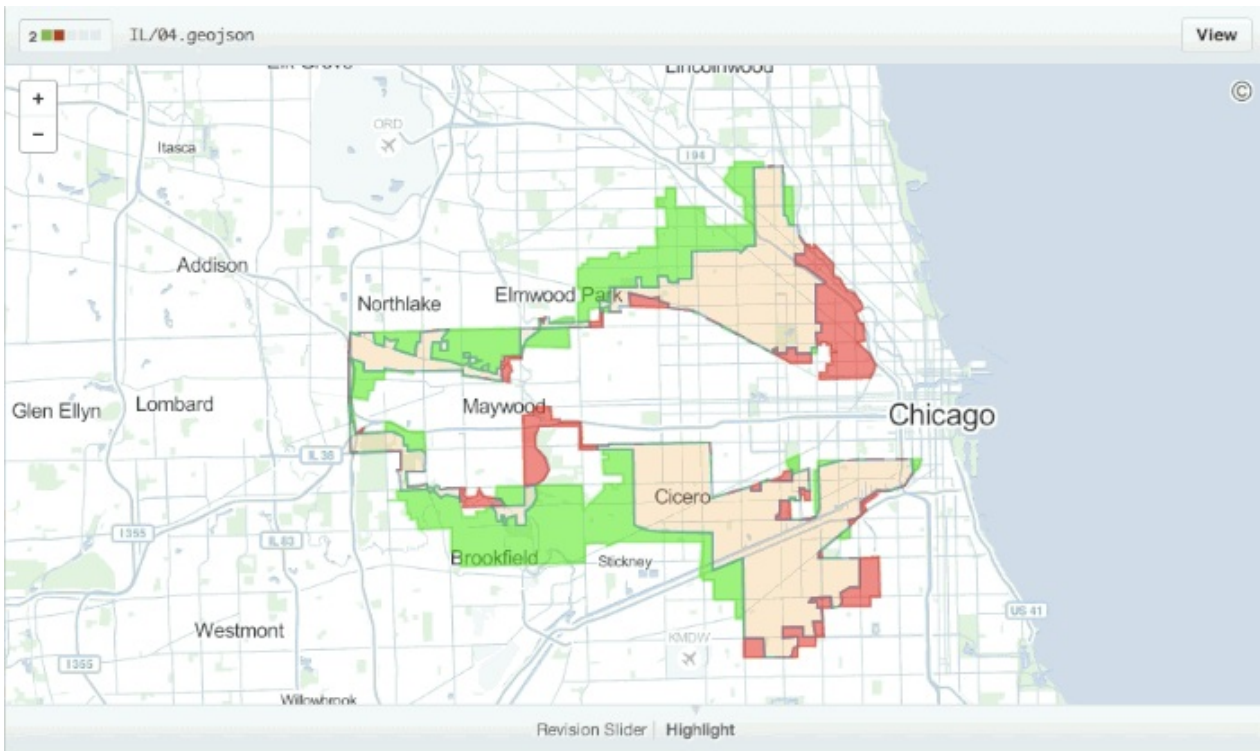
点击 "rendered" 按钮，看看改动在渲染后的显示效果。当你添加、删除或修改文本时，渲染纯文本视图非常方便。



进一步了解渲染纯文本视图Diffs.

### 可变化地图

当你在GitHub上查看一个包含地理数据的提交或pull request时，Github 将以可视化的方式对比版本之间的差异。



[进一步了解可比较地图.](#)

## 在diff中折叠与扩展代码

你可以通过点击diff边栏里的 *unfold* 按钮来多显示几行上下文。你可以一直点击 *unfold* 按钮直到显示了文件的全部内容。这个功能在所有GitHub产生的diff界面都可以使用。

```

94  +- (RACSignal *)enqueueRequest:(NSURLRequest *)request fetchAllPages:(BOOL)fetchAllPages;
95  +
82  96  // Enqueues a request to fetch information about the current user by accessing
83  97  // a path relative to the user object.
84  98  //
⌵  @@ -241,11 +255,13 @@ - (id)initWithServer:(OCTServer *)server {
241 255     NSString *userAgent = self.class.userAgent;
242 256     if (userAgent != nil) [self setDefaultHeader:@"User-Agent" value:userAgent];
243 257
244     - self.parameterEncoding = AFJSONParameterEncoding;
245     - [self setDefaultHeader:@"Accept" value:@"application/vnd.github.beta+json"];
246     -
247 258     [AFHTTPRequestOperation addAcceptableStatusCodes:[NSIndexSet indexSetWithIndex:OCTClientNotModifiedStatusCode]];
248     - [AFJSONRequestOperation addAcceptableContentTypes:[NSSet setWithObject:@"application/vnd.github.beta+json"]];
259     +
260     + NSString *contentType = [NSString stringWithFormat:@"application/vnd.github.%@+json", OCTClientAPIVersion];
261     + [self setDefaultHeader:@"Accept" value:contentType];
262     + [AFJSONRequestOperation addAcceptableContentTypes:[NSSet setWithObject:contentType]];
263     +
264     + self.parameterEncoding = AFJSONParameterEncoding;
249 265     [self registerHTTPOperationClass:AFJSONRequestOperation.class];
250 266
251 267     return self;

```

[进一步了解扩展Diff上下文.](#)

## 查看Pull Request的diff和patch

在Pull Request的URL后面加上 `.diff` 或 `.patch` 的扩展名就可以得到它的diff或patch文件，例如：

```
https://github.com/tiimgreen/github-cheat-sheet/pull/15
https://github.com/tiimgreen/github-cheat-sheet/pull/15.diff
https://github.com/tiimgreen/github-cheat-sheet/pull/15.patch
```

.diff 扩展会使用普通文本格式显示如下内容：

```
diff --git a/README.md b/README.md
index 88fcf69..8614873 100644
--- a/README.md
+++ b/README.md
@@ -28,6 +28,7 @@ All the hidden and not hidden features of Git and GitHub. This cheat sheet was i
- [Merged Branches](#merged-branches)
- [Quick Licensing](#quick-licensing)
- [TODO Lists](#todo-lists)
+- [Relative Links](#relative-links)
- [.gitconfig Recommendations](#gitconfig-recommendations)
  - [Aliases](#aliases)
  - [Auto-correct](#auto-correct)
@@ -381,6 +382,19 @@ When they are clicked, they will be updated in the pure Markdown:
- [ ] Sleep

(...)
```

### 渲染图像发生的变动

GitHub可以显示包括PNG、JPG、GIF、PSD在内的多种图片格式并提供了几种方式来比较这些格式的图片文件版本间的不同。





[查看更多关于渲染图像变动的内容](#)

## Hub

Hub是一个对Git进行了封装的命令行工具，可以帮助你更方便的使用Github。

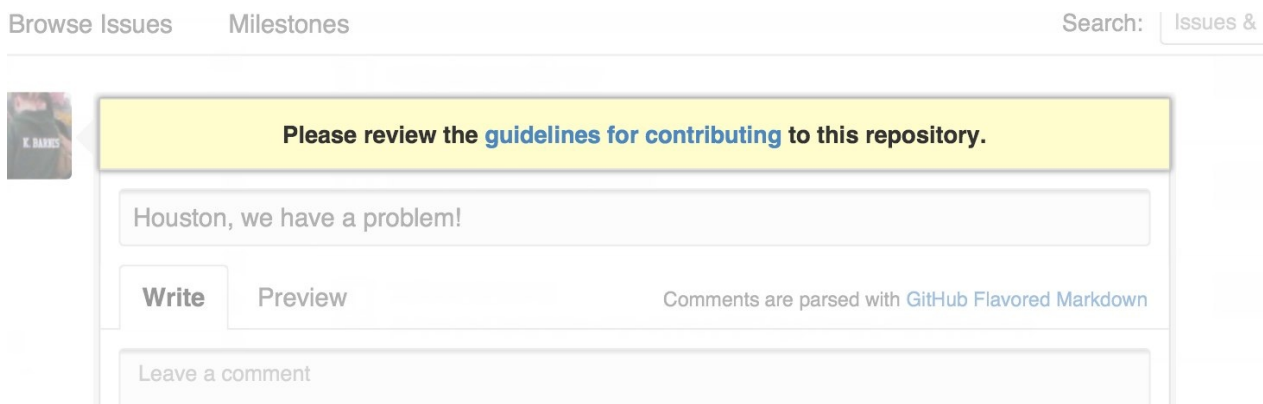
这使得你可以像下面这样进行克隆：

```
$ hub clone tiimgreen/toc
```

[查看更多Hub提供的超酷命令.](#)

## 贡献者指南

在你的仓库的根目录添加一个名为 `CONTRIBUTING` 的文件后，贡献者在新建Issue或Pull Request时会看到这个文件的链接。



[进一步了解贡献者指南.](#)

## GitHub资源

| Title            | Link                                                                      |
|------------------|---------------------------------------------------------------------------|
| GitHub Explore   | <a href="https://github.com/explore">https://github.com/explore</a>       |
| GitHub Blog      | <a href="https://github.com/blog">https://github.com/blog</a>             |
| GitHub Help      | <a href="https://help.github.com/">https://help.github.com/</a>           |
| GitHub Training  | <a href="http://training.github.com/">http://training.github.com/</a>     |
| GitHub Developer | <a href="https://developer.github.com/">https://developer.github.com/</a> |

## GitHub讨论

| Title                                           | Link                                                                                                  |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| How GitHub Uses GitHub to Build GitHub          | <a href="https://www.youtube.com/watch?v=qyz3jkOBbQY">https://www.youtube.com/watch?v=qyz3jkOBbQY</a> |
| Introduction to Git with Scott Chacon of GitHub | <a href="https://www.youtube.com/watch?v=ZDR433b0HJY">https://www.youtube.com/watch?v=ZDR433b0HJY</a> |
| How GitHub No Longer Works                      | <a href="https://www.youtube.com/watch?v=gXD1ITW7iZI">https://www.youtube.com/watch?v=gXD1ITW7iZI</a> |
| Git and GitHub Secrets                          | <a href="https://www.youtube.com/watch?v=Foz9yvMkvIA">https://www.youtube.com/watch?v=Foz9yvMkvIA</a> |
| More Git and GitHub Secrets                     | <a href="https://www.youtube.com/watch?v=p50xsL-iVgU">https://www.youtube.com/watch?v=p50xsL-iVgU</a> |

# Git

## 前一个分支

快速检出上一个分支：

```
$ git checkout -
# Switched to branch 'master'

$ git checkout -
# Switched to branch 'next'

$ git checkout -
# Switched to branch 'master'
```

[进一步了解 Git 分支.](#)

## Stripspace命令

Git Stripspace命令可以:

- 去掉行尾空白符
- 多个空行压缩成一行
- 必要时在文件末尾增加一个空行

使用此命令时必须传入一个文件，像这样：

```
$ git stripspace < README.md
```

[进一步了解 Git `stripspace` 命令.](#)

## 检出Pull Requests

Pull Request是一种GitHub上可以通过以下多种方式在本地被检索的特别分支：

检索某个分支并临时储存在本地的 `FETCH_HEAD` 中以便快速查看更改(diff)以及合并(merge)：

```
$ git fetch origin refs/pull/[PR-Number]/head
```

通过refspec获取所有的Pull Request为本地分支：

```
$ git fetch origin '+refs/pull/*/head:refs/remotes/origin/pr/*'
```

或在仓库的 `.git/config` 中加入下列设置来自动获取远程仓库中的Pull Request

```
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = git@github.com:tiimgreen/github-cheat-sheet.git
```

```
[remote "origin"]
```

```
fetch = +refs/heads/*:refs/remotes/origin/*  
url = git@github.com:tiimgreen/github-cheat-sheet.git  
fetch = +refs/pull/*/head:refs/remotes/origin/pr/*
```

对基于派生库的Pull Request, 可以通过先 `checkout` 代表此Pull Request的远端分支再由此分支建立一个本地分支:

```
$ git checkout pr/42 pr-42
```

操作多个仓库的时候, 可以在Git中设置获取Pull Request的全局选项。

```
git config --global --add remote.origin.fetch "+refs/pull/*/head:refs/remotes/origin/pr/*"
```

此时可以在任意仓库中使用以下命令:

```
git fetch origin
```

```
git checkout pr/42
```

[进一步了解如何检出pull request到本地.](#)

## 提交空改动 :trollface:

可以使用 `--allow-empty` 选项强制创建一个没有任何改动的提交:

```
$ git commit -m "Big-ass commit" --allow-empty
```

这样做在如下几种情况下是有意义的:

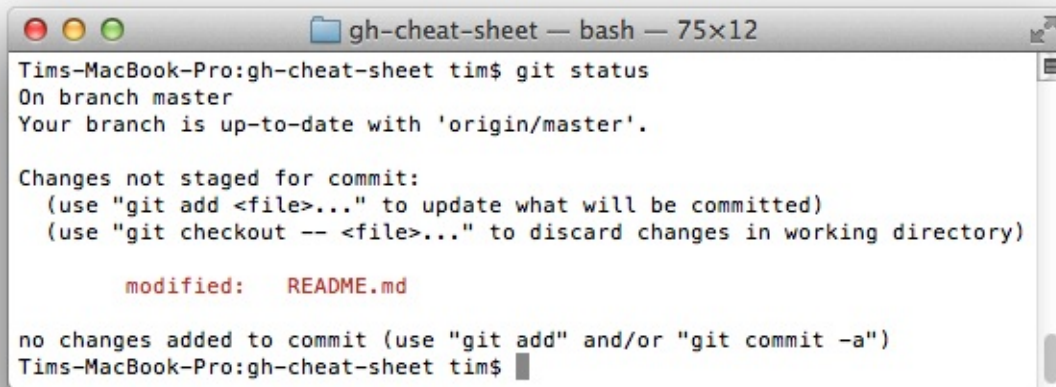
- 标记新的工作或一个新功能的开始。
- 记录对项目的跟代码无关的改动。
- 跟使用你仓库的其他人交流。
- 作为仓库的第一次提交, 因为第一次提交后不能被rebase: `git commit -m "init repo" --allow-empty .`

## 更直观的Git Status

在命令行输入如下命令:

```
$ git status
```

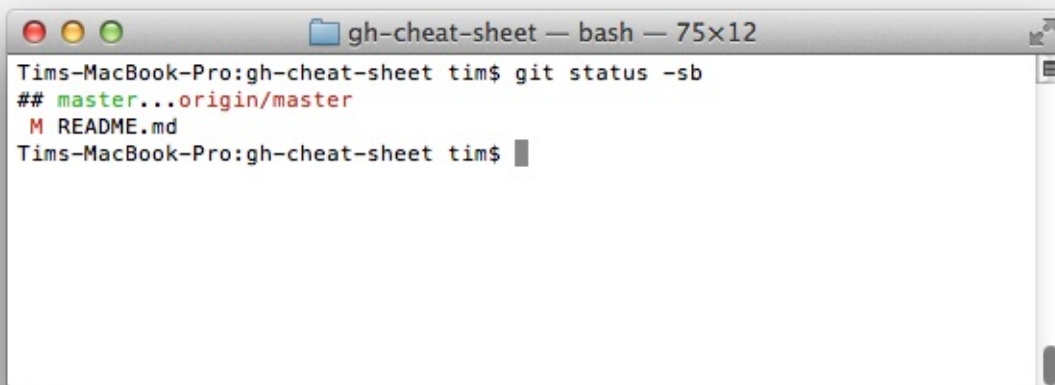
可以看到:



加上 -sb 选项:

```
$ git status -sb
```

这会得到:



进一步了解 `Git status` 命令.

## 更直观的Git Log

输入如下命令:

```
$ git log --all --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<an>%Creset'
```

可以看到:

```

Tims-MacBook-Pro:ghcs tim$ git log --all --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<an>%Cre
set' --abbrev-commit --date=relative
* d3b0586 - (HEAD, origin/master, origin/HEAD, master) Use different link to Git docs (26 hours ago) <Rafal Chmiel>
* 6ca33bd - Stripspace (26 hours ago) <Tim Green>
* 8e5b0bd - Remove verbose Hub example (27 hours ago) <Tim Green>
* bdb3f77 - Format link better (27 hours ago) <Tim Green>
* 2cbbc3 - Literally link emoji lists to add clarity (27 hours ago) <Tim Green>
* 003319c - Add date format helper for comparing branches (27 hours ago) <Tim Green>
* 7e669d3 - Add Ignoring Whitespace example (27 hours ago) <Tim Green>
* 7bbbe8a - Merge pull request #27 from hariadi/gh-pages-support (30 hours ago) <Rafal Chmiel>
|
| * fea81db - URL: Metadata and Plugin Support for GitHub Pages (30 hours ago) <Hariadi Hintia>
|/
* cb5058e - Add 'CI Status on Pull Requests' section (fixes #22) (32 hours ago) <Rafal Chmiel>
* 404f919 - Change title of one section (34 hours ago) <Rafal Chmiel>
* 3ce96f8 - Improve recently merged section (34 hours ago) <Rafal Chmiel>
* 58d4f6d - Merge pull request #26 from rb2/pull-as-diff (34 hours ago) <Rafal Chmiel>
|
| * 2e4446f - Add ToC entry for Getting content of Pull Request in Diff or Patch format (3 days ago) <Ruslan Brest>
| * a347b23 - Getting content of Pull Request in Diff or Patch format (3 days ago) <Ruslan Brest>
| * 9274e62 - Add another branch compare example (34 hours ago) <Rafal Chmiel>
* 957019c - Add example forked repo compare (34 hours ago) <Rafal Chmiel>
* 3d24fe7 - Add commit history example (34 hours ago) <Rafal Chmiel>
* ac15804 - Add compare branch example (34 hours ago) <Rafal Chmiel>
* ef29a13 - Add keyboard screenshot example (34 hours ago) <Rafal Chmiel>
* 97a5328 - Add Git.io screenshot example (34 hours ago) <Rafal Chmiel>

```

这要归功于Palesz在stackoverflow的回答。

这个命令可以被用作别名，详细做法见[这里](#)。

进一步了解 `Git log` 命令。

## Git 查询

Git 查询运行你在之前的所有提交信息里进行搜索，找到其中和搜索条件相匹配的最近的一条。

```
$ git show :/query
```

这里 `query`（区别大小写）是你想要搜索的词语，这条命令会找到包含这个词语的最后那个提交并显示变动详情。

```
$ git show :/typo
```

```

Tims-MacBook-Pro:ghcs tim$ git show :/typo
commit e70c204e3d7cb154f5866b575d01c1da14b68a6c
Author: Rafal Chmiel <hi@rafalchmiel.com>
Date:   Mon Apr 14 12:37:57 2014 +0100

    Fix typo

diff --git a/README.md b/README.md
index 2bc494e..40022d2 100644
--- a/README.md
+++ b/README.md
@@ -118,7 +118,7 @@ To see all of the shortcuts for the current page press `?`.

## Closing Issues with Commits

-If a particular commit fixes an issue, any of the keywords `fix/fixes/fixed` or `close/closes/closed`, followed by the issue number, will c
+If a particular commit fixes an issue, any of the keywords `fix/fixes/fixed` or `close/closes/closed`, followed by the issue number, will c

```bash
$ git commit -m "Fix cock up, fixes #12"
(END)

```

- 按 `q` 键退出命令。\*

## 合并分支

输入命令:

```
$ git branch --merged
```

这会显示所有已经合并到你当前分支的分支列表。

相反地:

```
$ git branch --no-merged
```

会显示所有还没有合并到你当前分支的分支列表。

[进一步了解 Git `branch` 命令.](#)

## 使用网页查看本地仓库

使用Git的 `instaweb` 可以立即在 `gitweb` 中浏览你的工作仓库。这个命令是个简单的脚本，配置了 `gitweb` 和用来浏览本地仓库的Web服务器。（译者注：默认需要`lighttpd`支持）

```
$ git instaweb
```

执行后打开:

projects / .git / summary +++ git

summary | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#) commit ↕ search:  re

---

description Unnamed repository; edit this file 'description' to name the repository.  
owner Rafal Chmiel  
last change Mon, 14 Apr 2014 20:04:18 +0100 (20:04 +0100)

---

**shortlog**

2 min ago Rafal Chmiel Add use-cases for empty commits, thanks to @davej ... [master](#) | [origin/HEAD](#) | [origin/master](#) | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
10 min ago Tim Green Revert title back to original | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
12 min ago Rafal Chmiel Add 'Git' to title | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
15 min ago Rafal Chmiel Update contributing guide to fit new organization | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
17 min ago Rafal Chmiel :star2: Organize sections (fixes #21) :star: | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
68 min ago Rafal Chmiel :star: Don't push, I'll be reorganizing the whole file | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
2 hours ago Tim Green Update intro paragraph | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
2 hours ago Tim Green Clarify Rendered Prose Diffs | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
3 hours ago Rafal Chmiel Add wiki images section under images/GIFs | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
3 hours ago Rafal Chmiel Add 'Embedding Images in GitHub Wiki' section thanks... | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
3 hours ago Rafal Chmiel Add info about 'pibb' thanks to @jballanc :heart: | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
3 hours ago Rafal Chmiel Make list of emojis prettier | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
3 hours ago Rafal Chmiel Just a little :trollface: | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
4 hours ago Rafal Chmiel :trollface: :trollface: :trollface: | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
4 hours ago Rafal Chmiel Add 'Contributing Guidelines' section :feelsgood: | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
4 hours ago Rafal Chmiel Search bar -> command bar | [commit](#) | [commitdiff](#) | [tree](#) | [snapshot](#)  
...

---

**heads**

2 min ago [master](#) | [shortlog](#) | [log](#) | [tree](#)

---

**remotes**

进一步了解 [Git instaweb](#) 命令。

## Git配置

所有Git配置都保存在你的 `.gitconfig` 文件中。

## Git命令自定义别名

别名用来帮助你定义自己的git命令。比如你可以定义 `git a` 来运行 `git add --all`。

要添加一个别名，一种方法是打开 `~/.gitconfig` 文件并添加如下内容：

```
[alias]
  co = checkout
  cm = commit
  p = push
  # Show verbose output about tags, branches or remotes
  tags = tag -l
  branches = branch -a
  remotes = remote -v
```

...或者在命令行里键入：

```
$ git config --global alias.new_alias git_function
```

例如：

```
$ git config --global alias.cm commit
```

指向多个命令的别名可以用引号来定义：

```
$ git config --global alias.ac 'add -A . && commit'
```

下面列出了一些有用的别名：

别名 Alias	命令 Command	如何设置 What to Type
git cm	git commit	git config --global alias.cm commit
git co	git checkout	git config --global alias.co checkout
git ac	git add . -A git commit	git config --global alias.ac '!git add -A && git commit'
git st	git status -sb	git config --global alias.st 'status -sb'
git tags	git tag -l	git config --global alias.tags 'tag -l'
git branches	git branch -a	git config --global alias.branches 'branch -a'
git remotes	git remote -v	git config --global alias.remotes 'remote -v'
git lg	git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --	git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --"

## 自动更正

如果键入 `git comit` 你会看到如下输出：

```
$ git comit -m "Message"
# git: 'comit' is not a git command. See 'git --help'.

# Did you mean this?
#   commit
```

为了在键入 `comit` 调用 `commit` 命令，只需启用自动纠错功能：

```
$ git config --global help.autocorrect 1
```

现在你就会看到：

```
$ git comit -m "Message"
# WARNING: You called a Git command named 'comit', which does not exist.
# Continuing under the assumption that you meant 'commit'
# in 0.1 seconds automatically...
```

## 带颜色输出

要在你的Git命令输出里加上颜色的话，可以用如下命令：

```
$ git config --global color.ui 1
```

进一步了解 [Git config](#) 命令。

## Git资源

Title	Link
-------	------



Official Git Site	<a href="http://git-scm.com/">http://git-scm.com/</a>
Official Git Video Tutorials	<a href="http://git-scm.com/videos">http://git-scm.com/videos</a>
Code School Try Git	<a href="http://try.github.com/">http://try.github.com/</a>
Introductory Reference & Tutorial for Git	<a href="http://gitref.org/">http://gitref.org/</a>
Official Git Tutorial	<a href="http://git-scm.com/docs/gittutorial">http://git-scm.com/docs/gittutorial</a>
Everyday Git	<a href="http://git-scm.com/docs/everyday">http://git-scm.com/docs/everyday</a>
Git Immersion	<a href="http://gitimmersion.com/">http://gitimmersion.com/</a>
Ry's Git Tutorial	<a href="http://rypress.com/tutorials/git/index.html">http://rypress.com/tutorials/git/index.html</a>
Git for Designer	<a href="http://hoth.entp.com/output/git_for_designers.html">http://hoth.entp.com/output/git_for_designers.html</a>
Git for Computer Scientists	<a href="http://eagain.net/articles/git-for-computer-scientists/">http://eagain.net/articles/git-for-computer-scientists/</a>
Git Magic	<a href="http://www-cs-students.stanford.edu/~blynn/gitmagic/">http://www-cs-students.stanford.edu/~blynn/gitmagic/</a>

## Git参考书籍

Title	Link
Pragmatic Version Control Using Git	<a href="http://www.pragprog.com/titles/tsgit/pragmatic-version-control-using-git">http://www.pragprog.com/titles/tsgit/pragmatic-version-control-using-git</a>
Pro Git	<a href="http://git-scm.com/book">http://git-scm.com/book</a>
Git Internals Peepcode	<a href="http://peepcode.com/products/git-internals-pdf">http://peepcode.com/products/git-internals-pdf</a>
Git in the Trenches	<a href="http://cbx33.github.com/gitt/">http://cbx33.github.com/gitt/</a>
Version Control with Git	<a href="http://www.amazon.com/Version-Control-Git-collaborative-development/dp/1449316387">http://www.amazon.com/Version-Control-Git-collaborative-development/dp/1449316387</a>
Pragmatic Guide to Git	<a href="http://www.pragprog.com/titles/pg_git/pragmatic-guide-to-git">http://www.pragprog.com/titles/pg_git/pragmatic-guide-to-git</a>
Git: Version Control for Everyone	<a href="http://www.packtpub.com/git-version-control-for-everyone/book">http://www.packtpub.com/git-version-control-for-everyone/book</a>

# 附录

---

## 计算机科学与技术

---

Computer Science, 大学的专业, 但也没有很认真的学。

### 参考资料

---

- [网易云课堂](#)
- [Github干货：计算机科学公开课](#)

## 网站

---

- [码农周刊](#)
- [码农周刊整理](#)

## 书籍

---

- [【年终特刊】2013年我们读过的好书](#)
- [【年终特刊】2014 年最受欢迎的技术干货](#)

## 个人提升

---

- [知道创宇研发技能表v2.2: 国内著名黑客余弦总结。](#)

## 数据可视化

---

- [data-design: 提高我们对于数据和设计的理解的集体行动的开始。](#)
- [一位前端开发者的计算机视觉学习之路——专访《Chrome扩展及应用开发》作者李喆](#)