大白话解释 Git 和 GitHub

2017/05/18 · IT技术 · 3 评论 · Git, Github

 本文由 伯乐在线 - 听风 翻译，艾凌风 校稿。未经许可，禁止转载！
英文出处：Red Radger。欢迎加入翻译组。

本文旨在使用通俗易懂的文字，讲解版本控制背后的理论，以便你能对程序员们如何工作有个全局概念。本文不涉及代码，不用下载啥东西，循序渐进，不关注繁复细节，只有文字和一些不怎么漂亮的手绘涂鸦。

学习任何东西都能在网络上找到如此之多的指导教程，这一直令我惊讶不已。Git 和 Github 也不例外，网络上有大量优秀资源，这些资源要么只对其中一个，要么兼顾二者引导你开始学习。以下是我特别喜欢的一些资源：

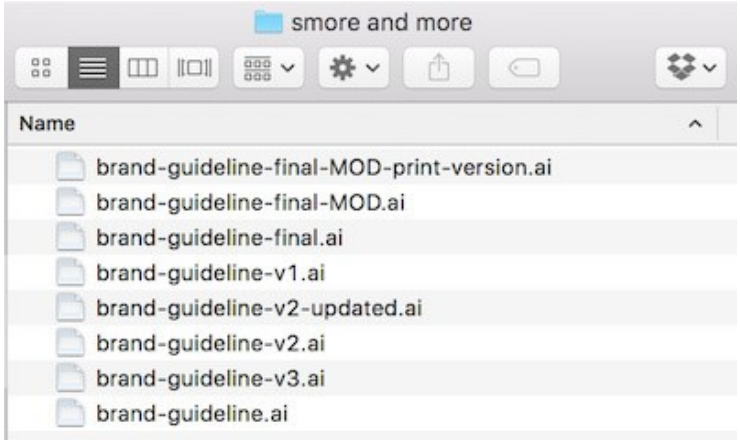
- Treehouse – 写给设计师的 Git 入门介绍
- Roger Dudler – Git 简易教程
- Pluralsight – Github：初学者指南

然而，我发现这些教程总是跳过许多理论知识，直接解释如何通过命令行或 Github 桌面应用程序使用 Git。坦白说，如果你只是想知道你的开发团队谈论的到底是什么，这些指导教程也绰绰有余了。如上所述，我的目标是对版本控制的整体概念进行简明扼要地讲解，同时希望能让你了解到版本控制是如此酷。

让我们从最基础的开始：版本控制

Image credit: weebletheringskite, WordPress

版本控制（Version control）:学习它，爱上它，享受它。顾名思义，**版本控制系统是任何能让你了解到一个文件的历史，以及它的发展过程的系统。**之前作为平面设计师时，我常常会遇到这种文件：



看起来眼熟？尽管上述系统不是一个好用的系统 ，但它确实是一个版本控制系统。更复杂点的例子就像，Google 文档的 “修订历史” 或 Photoshop 的 “历史” 工具。

开始 Git

Git 是一种专为处理文本文件而设计的版本控制系统。因为，归根到底，这就是代码的本质：一堆堆以某

建易于导航的系统历史。

（附： “Git” 也是工程师取的名字，我们对市场部同仁感到抱歉）

那么， Git 做了什么，是简单地保存文件所做不到的呢？从根本上讲，文件保存就是一个简化的版本控制系统，但坦率地说，它并不是一个好用的系统，因为它只能前进。当然，你也许会争论“撤消”按钮可以让你的文件回滚到以前的状态。但我们都清楚，“撤消”按钮有其局限性，最明显示的是，在关闭文件时，文件的过去也随之丢失。

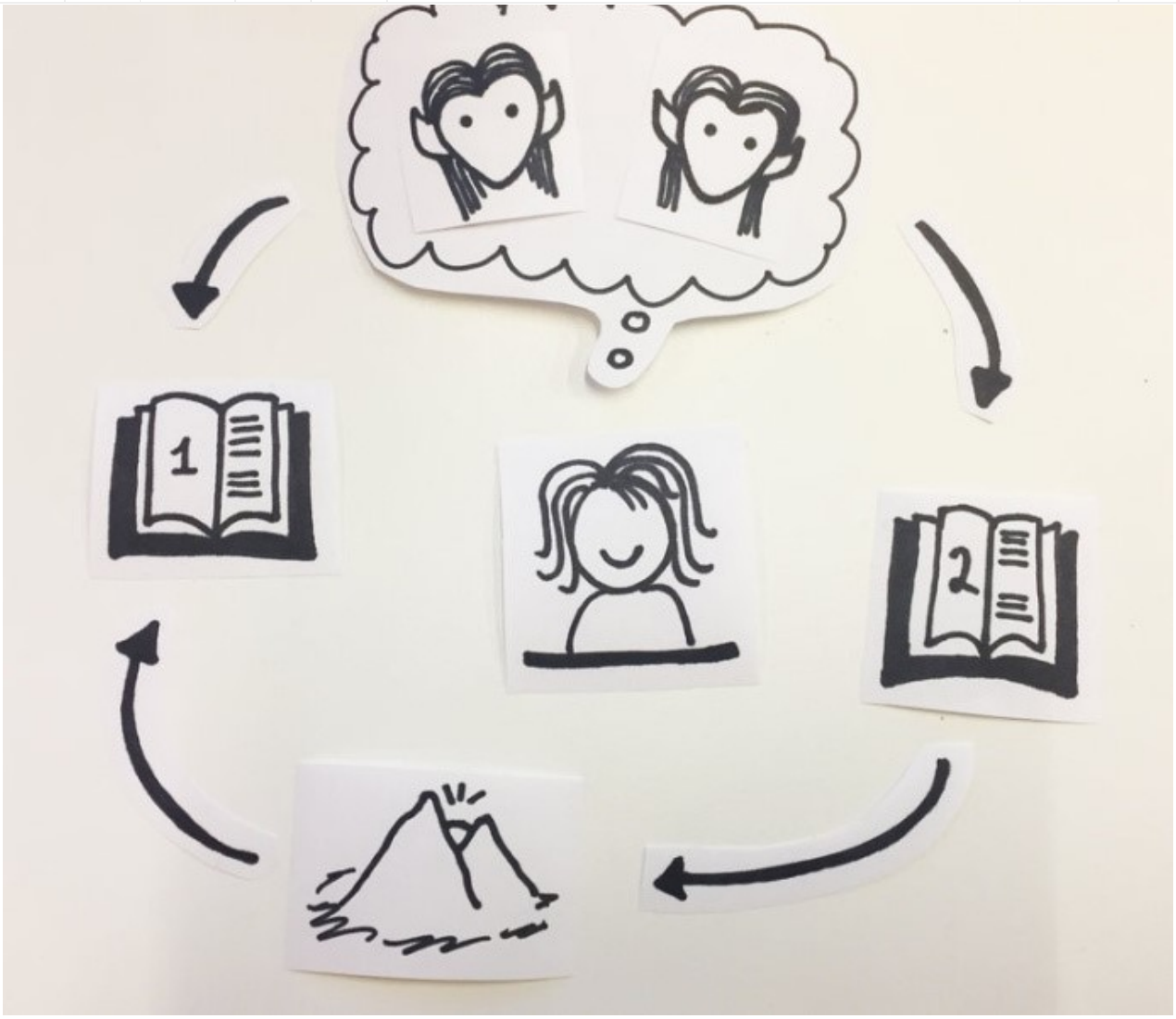
另外，文件保存是非常个人化的。它不能够显示整个系统的历史，只能够显示该文件的。针对这一点，你可能会想，“嗯，我不是一个工程师，我不需要为系统烦恼”。我愿意花些时间来解释一下，很多事情你认为不是“系统”，而实际上它们就是。

以 Sally为例，她是一个正在写下一个大冒险奇幻小说系列的作家。Sally 已经写完该系列小说中的第一本，并把它传给了她的编辑。此外，由于她才华出众，在等待编辑的反馈的同时，她还写了第二本书的前三章。每本书都建立了独立的 World 文件。

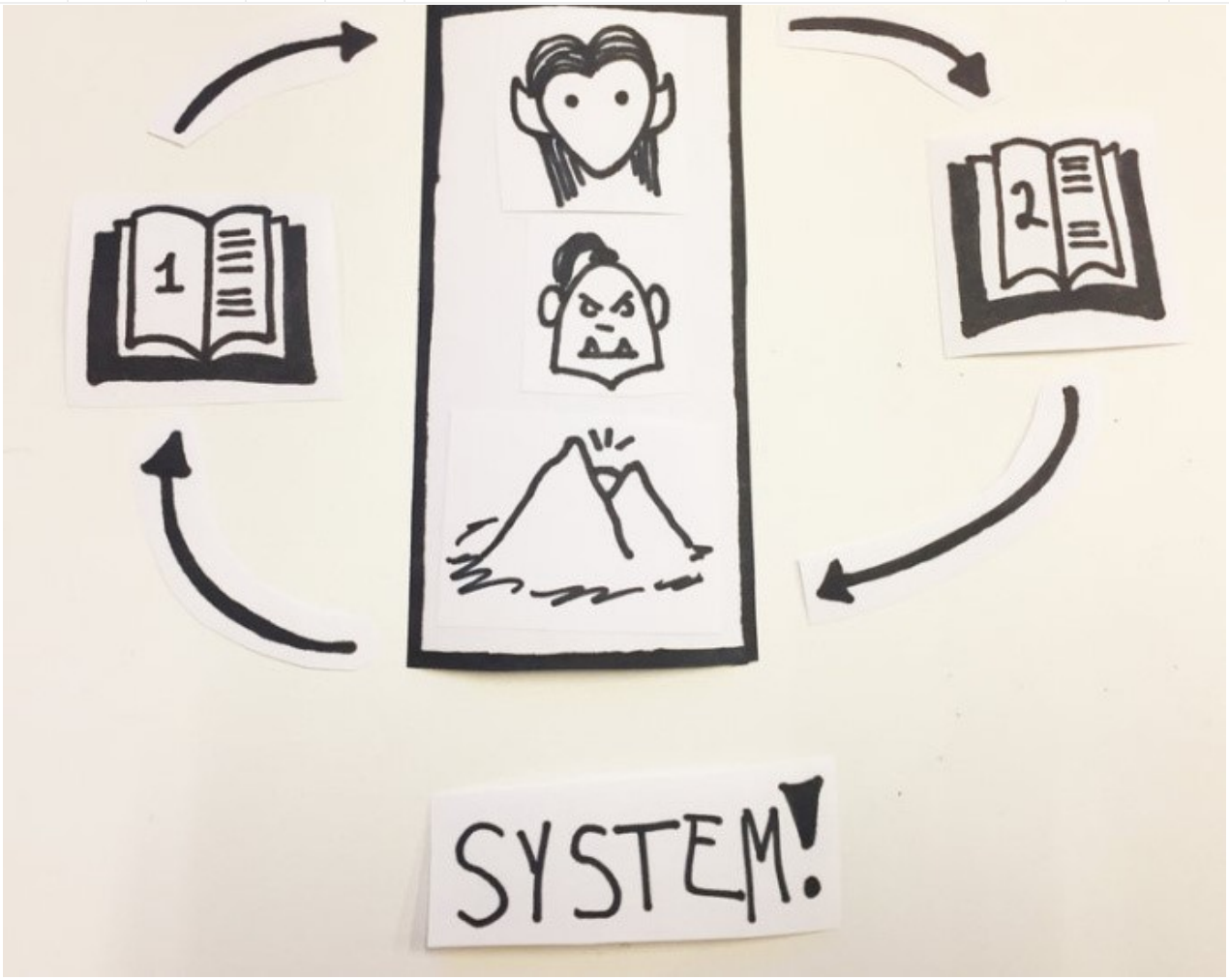
在某个快乐的日子，Sally 等来了她的编辑关于第一本书的反馈。他担心年轻的读者不想读一系列专写兽人故事的书，希望她在这个故事中引入一些精灵。关于这点，Sally 叹了口气，但很快意识到，她的精灵新角色将带来始料未及的冲突和曲折的情节。然后，她做了以下事情：

- 在第一本书中加入新角色，并修改故事情节。
- 完成第一本书之后，对第二本书的故事情节，进行必要的修改。
- 所有的这些修改，导致她需要引入某个地理位置到第一本书中，而不是第二本书。
- 重新编辑第一本书，让它包含新的地理位置。

终于，她推开了她的键盘，确信已经把精灵融入到了她的奇幻世界之中。



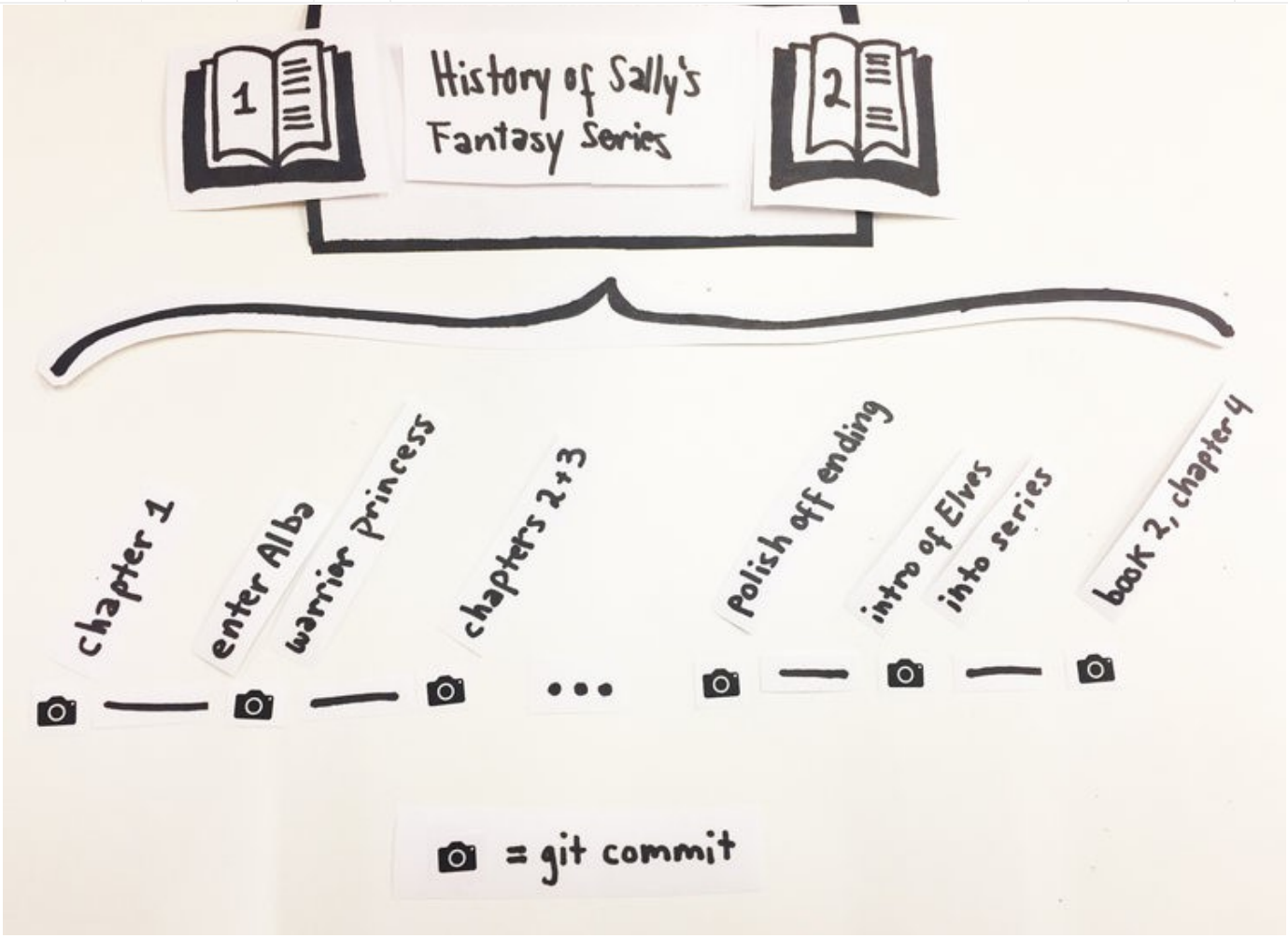
你瞧，Sally 实际上在处理一个系统。她的两本书互相影响。角色、地理位置和故事情节在两本书中流动交织。然而，遗憾的是，一个月后，她的文件系统里什么都没有了。Word 的 “文档历史”工具，或她曾经粘贴在显示屏边缘用于记录修改过程的便签纸，将把所有的变化过程都揉合在一起。



这正是Git 大放光芒之处。如果 Sally 一直结合 Git 使用Word，她就能对所有这些相关的变化做一个关于“将精灵引入到系列”的简洁小结。她可以看到所有穿插在页、章节、文件，以及每本书中的修改记录，让她真正地了解引入精灵对她的奇幻系列产生的影响。这个“简洁的小结”就是我们在 Git 领域中所讲的**提交（commit）**。

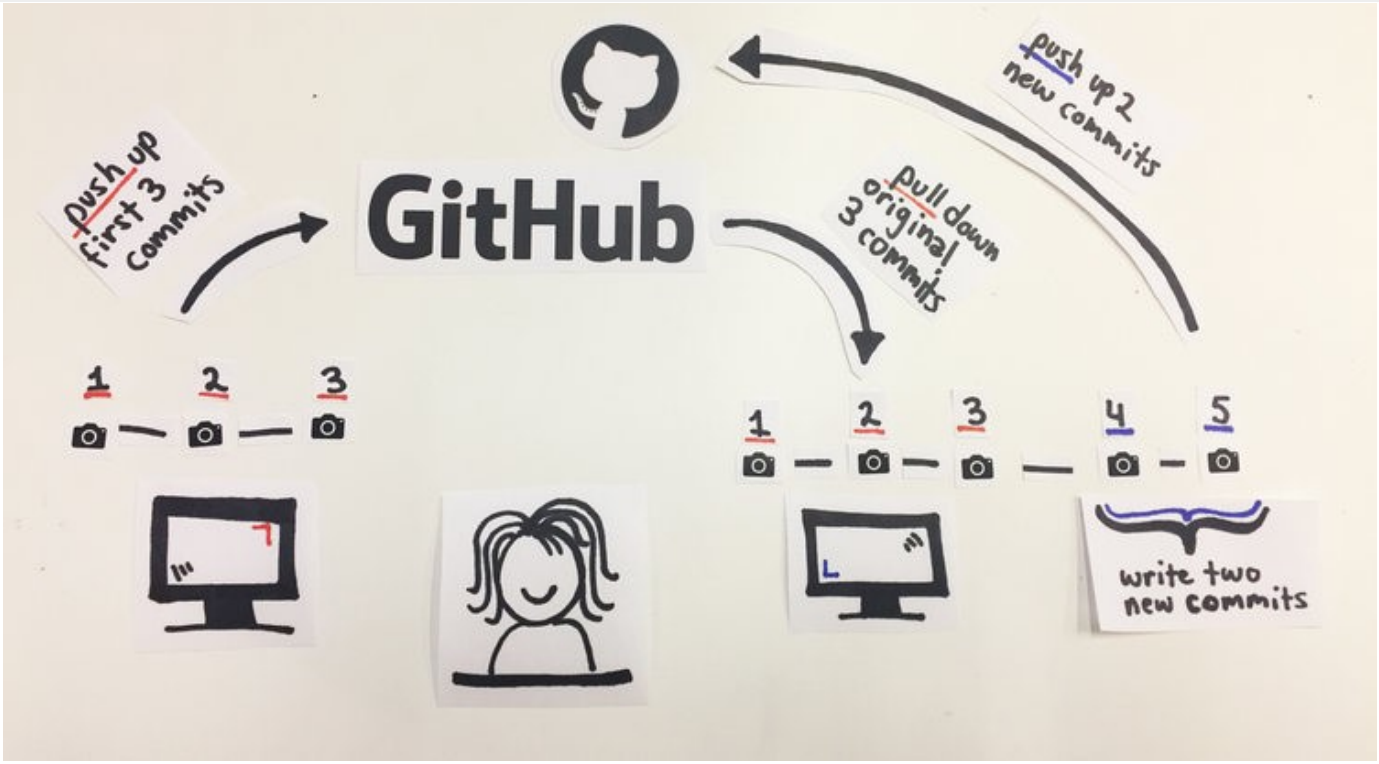
回顾一下。 Git 是一个软件，它允许你通过提交对一个系统（或一组）文件的历史进行注释。这些提交便是在给定时间点对系统做出的差异“快照”。

那么，如果我是Sally，我的**提交历史**看起来是这样子的：



Github

到目前为止，一切都还不错。但是，**如果 Sally 同时用到两台电脑工作，将会发生什么呢？**问得好。这时，就该用到 Github了。注意，不要和 Git 混淆了。Github 获取 Git 中的提交历史，并将其存储在互联网上，因此你可以从任一一台电脑访问它。你在本机（例如：你当前正在使用的电脑）**推送（pushing）**提交到 Github，然后，从另一台新的或不同的电脑上**拉取（pulling）**这些提交。



让我们假设上图为 Sally 的工作流程。她在家里的台式电脑（左边，橘黄色的）上开启她一天的工作。接下来，她完成了几个章节的写作，又做了一些编辑工作，等等。整个过程中，她对工作总共进行了三次策略性的“快照”（Git 提交）。

下午，Sally 常常喜欢带着她的笔记本电脑（上图中的右侧，蓝色的）去咖啡馆写作。今天也不例外。因此，在关闭家里的台式电脑之前，她需要确认当前的Git 提交历史已**推送（push）**到了在线Github。一旦被上传到 Github，这些提交记录就被存储在**远程仓库（remote repository）**中。

我们先来分析一下几个计算机术语：**远程（remote）**仅仅意味着联网（与“本地”的意思相反，和之前我们理解到的意思一样的，代表当前正在使用的电脑）。而**仓库（repository，经常简写为“repo”）**，就是一个具备 Git 超级权限的文件夹。

因此，**Github 就是让你把工作（通过Git提交进行注解）存储在了一个指定的在线文件夹（repo）**。明白了吧？简单。

午餐之后，在当地的一家咖啡馆中，Sally 拿出了她的笔记本电脑。很明显，她想接着家里的工作进度继续。因些，她从 Github 仓库上获取到最新进度的工作。“从 Github 上获取她的工作”，这一过程就叫**拉取（pulling）**。再看一下上面这幅图片，你将看到 Sally 拉取了之前她在家时进行的三个提交。

现在，在她的笔记本电脑上，Sally 有整个系统（包含她的幻想系列的所有文本文件）的最新的完整副本，并能够基于上次的进度，继续工作。她写了更多的章节，对工作进行了两次以上的策略“快照”（提交）。最后，Sally 把这些提交**推送**（**push**）到 Github 上，结束了这一天的工作。这样第二天上午的时候，在家里的台式电脑上就可以取得这些最新进度的工作。

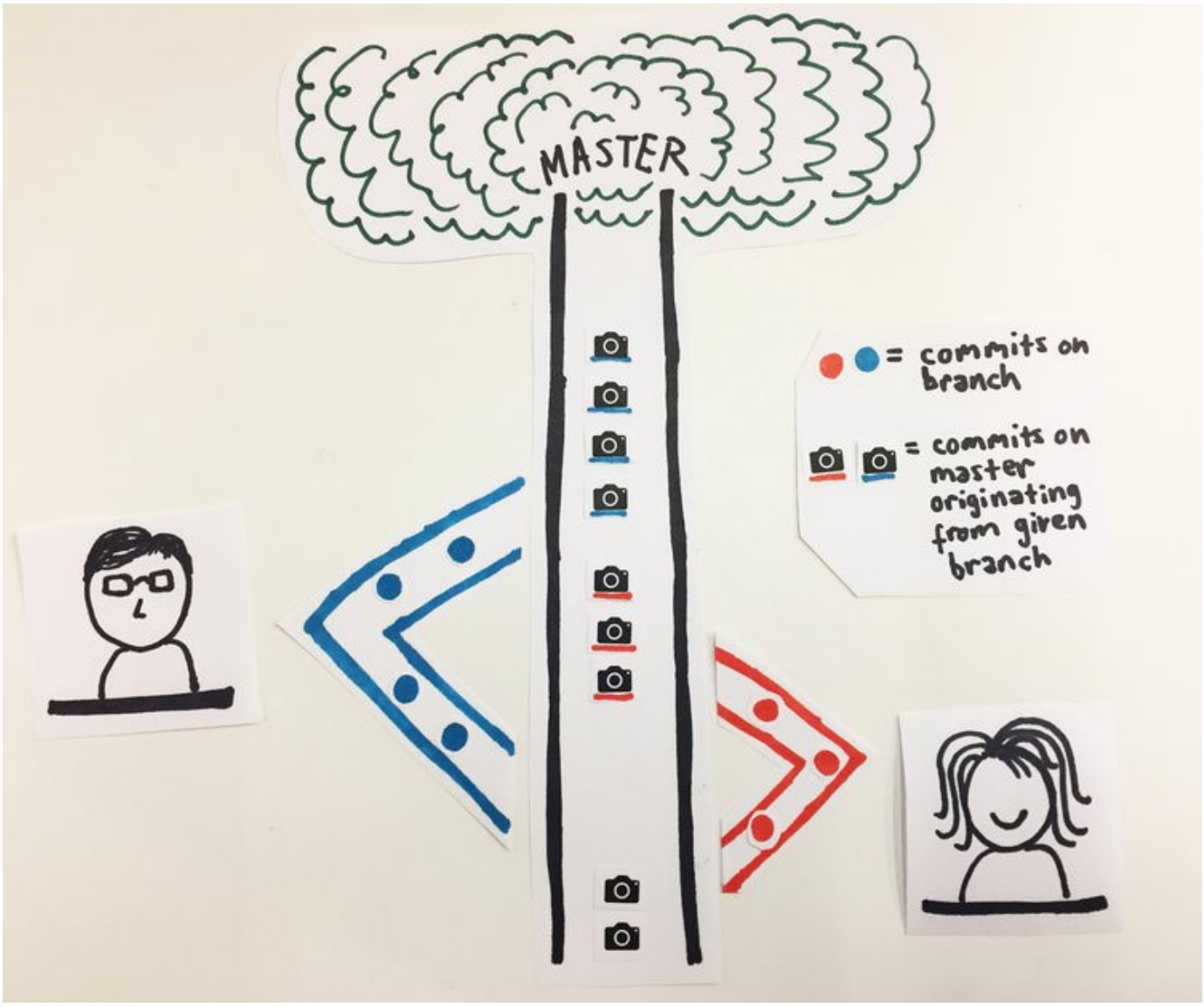
协同工作

好吧，这一切都能说得通。但是， Sally再如何酷，整个项目也只有她一人而已。**工程团队要如何确保他们的工作不会重叠？**

简而言之，创建分支。将你的 Git 提交历史想像成一棵树。树的主干就是我们谈到的主分支。为了让团队成员避免彼此牵扯，他们在独立于他人的隔离区（在一个功能分支）进行工作，然而最终，每个人的工作

现在，回到 Sally 的例子。她加入了奇幻作家协会，在这里每个人都与他人合作完成这本书——《奇幻系列生物辞典》。这本辞典更像一本教材，由多个作者共同完成：Sally、Tom 和 Adam。

让我们来看看《奇幻系列生物辞典》项目的在线 Github 仓库，现在的情况是：



如上图所示，树的类比完全适用于奇幻作家协会在这个项目上的合作情况，仓库历史沿主分支向上移动。常规工作流始于每个作者为完成一个工作任务（例如编写章节内容，或排版章节）而在主分支上创建分支。只有当更改得到其他合作作家的批准时，分支才会被合并到主分支上（请谨记，主分支上的内容，才是最终要发布的内容）。

当一个分支的内容**合并（merged）**到主分支时，意味着该分支的内容会覆盖主分支上的。因此，现有内容的任何更改都将会替代之前的。当然，任何新添加的内容也会添加到主分支。实际上，当分支合并到主分支时，该分支的提交历史被添加到主分支提交历史的顶部。

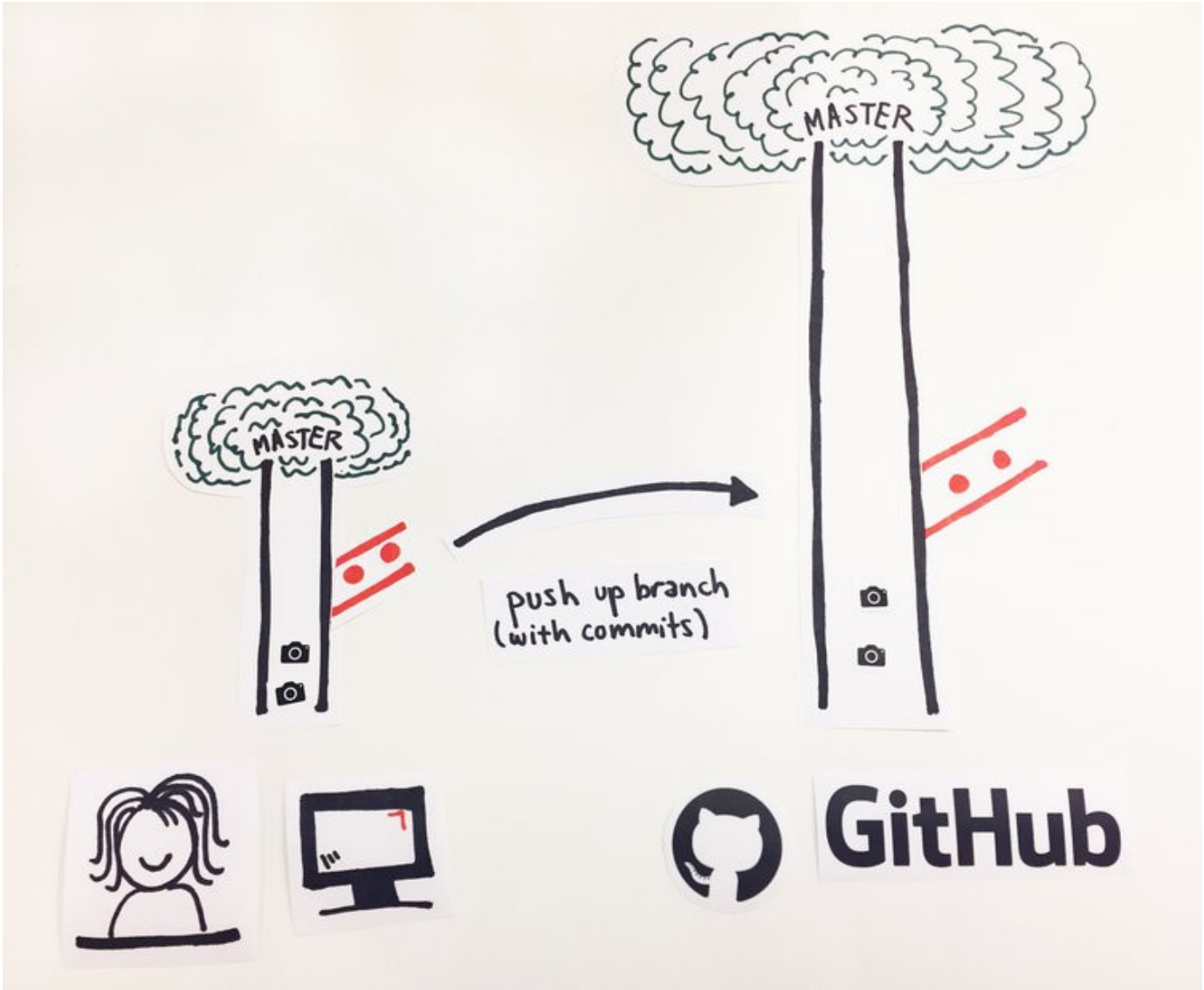
然而，你可能正在思考：**人们在本机的工作和之后才推送到 Github 的工作变更是如何连接到一起的呢？**

关于这个问题，重点在于：你在 Github 的远程仓库是你本机工作项目的一个镜像。这意味着，你在自己的电脑里存储了该项目（例如：一个已设置可进行 Git 提交的文件夹）的本地 Git 仓库。在这个本地的 Git 仓库（再次，这是一个特定术语，指你的电脑里某个启用了 Git 功能的文件夹）中，你拥有与该项目相关的所有文件，在本文的例子中，即《奇幻系列生物辞典》。

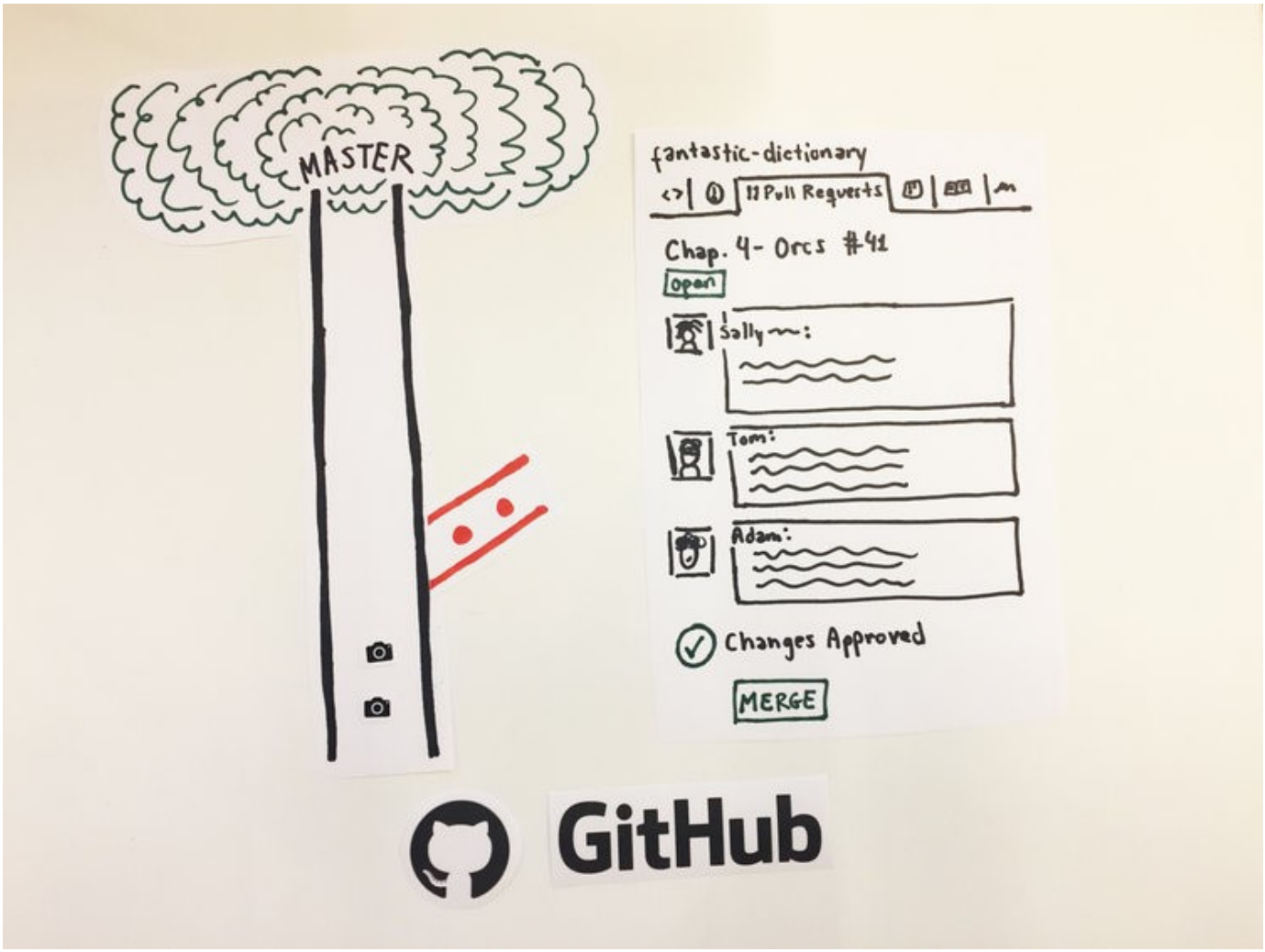
关，进行工作并更新这些文件。最后，这些操作被同步到网络上。然而，我们知道，Git/Github 工作流还包含了一些额外的步骤。首先，你必须有意识地对某一时刻的工作执行“快照”（即执行一次提交）。然后，你必须特意地推送这些提交（push）到 Github。只有这样，你的工作才被同步到网络上的位置（Github 版本库）。

既然如此，为什么不自动化该工作流呢？为什么不让它像 Dropbox一样，当你更新本地文件时，同时自动更新 Github 上的文件？有很多理由让我们不这么做。最主要的理由是——bugs 。同出版界一样，软件工程中也不是所有写过的东西都要保留。有时，你希望实验一下你的想法，如果实验失败，你希望有一种简单的方式能让工作快速回滚到之前的正确状态上。这也是为什么我们提倡这个经验法则，即在你试图用不同的方法编辑或实验之前，先对当前你希望保留的修改进行提交。频繁地提交小块工作有益无害，事实上，许多工程师为自己能做到这一点而感到自豪。

现在，回到《奇幻系列生物辞典》。由于 Sally 对兽族有较深的了解，她被挑选为写兽族章节。但她不想在没有经过其它合作人员允许的情况下去修改这本书，于是，她创建了一个本地分支，并在该分支上进行写作和提交。然后，**她将本地分支推送到 Github**。像往常一样，Github 的远程仓库是本地库的一个镜像，最新进展显示 Sally 已创建了一个包含部分提交的分支（如下图所示）。



随着她对本章节的持续写作，Sally 进行了更多的提交，并将它们推送到 Github 的在线镜像分支。终于，她准备请 Tom 和 Adam 一起对她的工作进行评审。因此，她在 Github 上发布了一个 **Pull Request（发布请求）**，这是一个 Github 功能，允许她解释该分支相对于主分支做了哪些修改。Github 还提供了一个简易平台，合作人员可以在该平台上针对分支的修改内容进行讨论，并要求 Sally 在分支合并到主分支之



在对部分内容请求修改后，如上图所示，Tom 和 Adam 对 Sally 的分支内容很满意，并决定将她的工作成果合并到 Github 的主分支上。此时，他们所要做的就是将 Sally 之前独立提交的内容，添加到主分支的提交历史顶部：

此时，Sally可以切换（或“**check out**”）到本地计算机上的主分支，并将先前在功能分支（兽人章节分支）中的独立提交拉取下来。现在她又要在新的主分支上重新开始了：以该主分支为基础为她的下一步工作创建一个新的本地分支，帮助汤姆编辑有关妖精的章节。因此，这一过程又将重复:

1. 创建本地分支
2. 在本地分支上编辑修改，然后提交
3. 推送提交（Push）到 Github
4. 创建发布请求（Pull Request），说明该分支包含了哪些更改
5. 合并（Merge）分支内容到主分支
6. 将主分支上的最新提交拉取（pull）到本地
7. 重复上述步骤

正如你所看到的，这是一个非常流畅的工作流，完美地结合了独立工作与团队协作。你本机的 Git 提供了一个绝妙的方法，即通过由你自己控制和策划的丰富的历史提交，来创建你工作的各种版本。Github 是一个非常棒的在线版本控制工具，不仅存储和提供了清晰的可视化历史记录，而且还能进行协同工作和质量控制。

总而言之，我希望我已经说服你去**尝试使用 Git 和 Github 进行任何项目**。没有理由只有工程师能从这个很棒的工具中受益。毕竟，我们也想看到更多有关兽人的故事。

致谢：

非常感谢 Common Craft 对本文的涂鸦和解释风格的启发。还要感谢这个视频《 saving me from the horror of having to explain Twitter to my mum 》。

本文涉及的术语

统。

- **Git：**一个版本控制程序，通过对变更进行注释，以创建一个易于遍历的系统历史。
- **Commit（提交）：**在指定时间点对系统差异进行的注释 “快照”。
- **Local（本地）：**指任意时刻工作时正在使用的电脑。
- **Remote（远程）：** 指某个联网的位置。
- **Repository (仓库，简称 repo)：**配置了Git超级权限的特定文件夹，包含了你的项目或系统相关的所有文件。
- **Github：**获取本地提交历史记录，并进行远程存储，以便你可以从任何计算机访问这些记录。
- **Pushing（推送）：**取得本地Git提交（以及相关的所有工作），然后将其上传到在线Github。
- **Pulling（拉取）：**从在线的Github上获取最新的提交记录，然后合并到本地电脑上。
- **Master (branch)：主分支，**提交历史 “树”的 “树干”，包含所有已审核的内容/代码。
- **Feature branch（功能分支/特性分支）：**一个基于主分支的独立的位置，在再次并入到主分支之前，你可以在这里安全地写工作中的新任务。
- **Pull Request（发布请求）：**一个 Github 工具，允许用户轻松地查看某功能分支的更改 （the difference或 “diff” ），同时允许用户在该分支合并到主分支之前对其进行讨论和调整。
- **Merging（合并）：**该操作**指**获取功能分支的提交，加入到主分支提交历史的顶部。
- **Checking out（切换）：**该操作指从一个分支切换到另一个分支。

关于作者：听风

简介还没来得及写：）

个人主页 · 我的文章 · 23

相关文章

- 技术面试中常被问到是否参与开源，那如何清晰展示 GitHub 项目呢？ · 2
- GitHub 简易入门指南
- 可能是迄今为止最好的 GitHub 代码浏览插件 · 2
- Github 对程序员职业生涯的影响 · 4
- 研发团队 GIT 开发流程新人学习指南 · 1

可能感兴趣的话题

- 三十而立，可是很迷茫，大家畅所欲言。 · 6
- 申请加入翻译小组 · 1
- 如果有一天不做前端或者互联网行业了，能做什么 · 8
- 200 OK (from memory cache) 为什么会使浏览速度变很慢 · 4
- 分享 国产超轻量级组件化数据渲染框架Eng JS 数据双向绑定的简单应用 · 1
- 有哪些进入公司，你才发现的坑 · 12

直接登录

最新评论

砍柴少年（ 1 ） 05/25

good for you

赞 回复

silkshadow（ 1 ） 06/17

通俗易懂

赞 回复

晴平乐 07/17

棒 谢谢~~~

赞 回复

- 本周热门文章
- 本月热门文章
- 热门标签

0 每周 70+ 小时，吴恩达的招人要求...

1 比尔·盖茨评审产品时说 F**k ...

2 React 许可证的五宗罪

3 微博爬虫“免登录”技巧详解及 Jav...

4 di：比 df 更有用的磁盘信息工具

5 Linux 大爆炸：一个内核，无数发行版

6 没有报酬，有多少开源项目维护者能坚...

7 自上而下，逐步揭开 PHP 解析大...

9 为什么招不到最好的程序员？SO 创...

业界热点资讯 **更多 »**

FB 妥协了，下周把 React 协议改成 MIT
2 天前 · 24 · 1

比尔·盖茨再次为 CTRL+ALT+DEL 表示歉意
3 天前 · 13

忘记 PHP！Facebook 的 HHVM 引擎将转用 Hac...
4 天前 · 18 · 1

Ubuntu 17.10 已确认使用 4.13 内核和 GCC 7...
3 天前 · 4

Swift 4.0 正式发布，更快更兼容更好用
5 天前 · 8 · 2

精选工具资源 **更多资源 »**

Whitewidow：SQL 漏洞自动扫描工具
数据库 · 2

Caffe：一个深度学习框架
机器学习

静态代码分析工具清单：公司篇
静态代码分析

HotswapAgent：支持无限次重定义运行时类与资源
开发流程增强工具

静态代码分析工具清单：开源篇（各语言）
静态代码分析

关于伯乐在线博客

在这个信息爆炸的时代，人们已然被大量、快速并且简短的信息所包围。然而，我们相信：过多“快餐”式的阅读只会令人“虚胖”，缺乏实质的内涵。伯乐在线内容团队正试图以我们微薄的力量，把优秀的原创文章和译文分享给读者，为“快餐”添加一些“营养”元素。

快速链接
网站使用指南 »
问题反馈与求助 »
加入我们 »
网站积分规则 »
网站声望规则 »

关注我们

新浪微博：@伯乐在线官方微博
RSS：订阅地址
推荐微信号

合作联系
Email：bd@Jobbole.com
QQ： 2302462408（加好友请注明来意）

更多频道

小组 – 好的话题、有启发的回复、值得信赖的圈子
头条 – 分享和发现有价值的内容与观点
相亲 – 为IT单身男女服务的征婚传播平台
资源 – 优秀的工具资源导航
翻译 – 翻译传播优秀的外文文章
文章 – 国内外的精选文章
设计 – UI,网页，交互和用户体验
iOS – 专注iOS技术分享
安卓 – 专注Android技术分享
前端 – JavaScript, HTML5, CSS
Java – 专注Java技术分享
Python – 专注Python技术分享

 文章 小组 相亲 加入我们 反馈 沪ICP备14046347号-1