

Git权威指南

作者：蒋鑫

Git权威指南

蒋鑫 著

ISBN: 978-7-111-34967-9

本书纸版由机械工业出版社于2011年出版，电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

目 录

[前言](#)

[本书的组织](#)

[适用读者](#)

[排版约定](#)

[在线资源](#)

[致谢](#)

[第1篇 初识Git](#)

[第1章 版本控制的前世和今生](#)

[1.1 黑暗的史前时代](#)

[1.2 CVS——开启版本控制大爆发](#)

[1.3 SVN——集中式版本控制集大成者](#)

[1.4 Git——Linus的第二个伟大作品](#)

[第2章 爱上Git的理由](#)

[2.1 每日工作备份](#)

[2.2 异地协同工作](#)

[2.3 现场版本控制](#)

[2.4 避免引入辅助目录](#)

[2.5 重写提交说明](#)

[2.6 想吃后悔药](#)

[2.7 更好用的提交列表](#)

[2.8 更好的差异比较](#)

[2.9 工作进度保存](#)

[2.10 代理SVN提交实现移动式办公](#)

[2.11 无处不在的分页器](#)

[2.12 快](#)

[第3章 Git的安装和使用](#)

[3.1 在Linux下安装和使用Git](#)

[3.1.1 包管理器方式安装](#)

[3.1.2 从源代码进行安装](#)

[3.1.3 从Git版本库进行安装](#)

[3.1.4 命令补齐](#)

[3.1.5 中文支持](#)

[3.2 在Mac OS X下安装和使用Git](#)

[3.2.1 以二进制发布包的方式安装](#)

[3.2.2 安装Xcode](#)

[3.2.3 使用Homebrew安装Git](#)

[3.2.4 从Git源码进行安装](#)

[3.2.5 命令补齐](#)

[3.2.6 其他辅助工具的安装](#)

[3.2.7 中文支持](#)

[3.3 在Windows下安装和使用Git（Cygwin篇）](#)

[3.3.1 安装Cygwin](#)

[3.3.2 安装Git](#)

[3.3.3 Cygwin的配置和使用](#)

[3.3.4 Cygwin下Git的中文支持](#)

[3.3.5 Cygwin下Git访问SSH服务](#)

[3.4 Windows下安装和使用Git（msysGit篇）](#)

[3.4.1 安装msysGit](#)

[3.4.2 msysGit的配置和使用](#)

[3.4.3 msysGit中shell环境的中文支持](#)

[3.4.4 msysGit中Git的中文支持](#)

[3.4.5 使用SSH协议](#)

[3.4.6 TortoiseGit的安装和使用](#)

[3.4.7 TortoiseGit的中文支持](#)

[第2篇 Git独奏](#)

[第4章 Git初始化](#)

[4.1 创建版本库及第一次提交](#)

[4.2 思考：为什么工作区根目录下有一个.git目录](#)

[4.3 思考：git config命令的各参数有何区别](#)

- [4.4 思考：是谁完成的提交](#)
- [4.5 思考：随意设置提交者姓名，是否太不安全](#)
- [4.6 思考：命令别名是干什么的](#)
- [4.7 备份本章的工作成果](#)
- [第5章 Git暂存区](#)
 - [5.1 修改不能直接提交吗](#)
 - [5.2 理解Git暂存区（stage）](#)
 - [5.3 Git Diff魔法](#)
 - [5.4 不要使用git commit-a](#)
 - [5.5 搁置问题，暂存状态](#)
- [第6章 Git对象](#)
 - [6.1 Git对象库探秘](#)
 - [6.2 思考：SHA1哈希值到底是什么，是如何生成的](#)
 - [6.3 思考：为什么不用顺序的数字来表示提交](#)
- [第7章 Git重置](#)
 - [7.1 分支游标master探秘](#)
 - [7.2 用reflog挽救错误的重置](#)
 - [7.3 深入了解git reset命令](#)
- [第8章 Git检出](#)
 - [8.1 HEAD的重置即检出](#)
 - [8.2 挽救分离头指针](#)
 - [8.3 深入了解git checkout命令](#)
- [第9章 恢复进度](#)
 - [9.1 继续暂存区未完成的实践](#)
 - [9.2 使用git stash](#)
 - [9.3 探秘git stash](#)
- [第10章 Git基本操作](#)
 - [10.1 先来合个影](#)
 - [10.2 删除文件](#)
 - [10.2.1 本地删除不是真的删除](#)
 - [10.2.2 执行git rm命令删除文件](#)
 - [10.2.3 命令git add-u快速标记删除](#)
 - [10.3 恢复删除的文件](#)
 - [10.4 移动文件](#)
 - [10.5 一个显示版本号Hello World](#)
 - [10.6 使用git add-i选择性添加](#)
 - [10.7 Hello World引发的新问题](#)
 - [10.8 文件忽略](#)
 - [10.9 文件归档](#)
- [第11章 历史穿梭](#)
 - [11.1 图形工具：gitk](#)
 - [11.2 图形工具：gitg](#)
 - [11.3 图形工具：qgit](#)
 - [11.4 命令行工具](#)
 - [11.4.1 版本表示法：git rev-parse](#)
 - [11.4.2 版本范围表示法：git rev-list](#)
 - [11.4.3 浏览日志：git log](#)
 - [11.4.4 差异比较：git diff](#)
 - [11.4.5 文件追溯：git blame](#)
 - [11.4.6 二分查找：git bisect](#)
 - [11.4.7 获取历史版本](#)
- [第12章 改变历史](#)
 - [12.1 悔棋](#)
 - [12.2 多步悔棋](#)
 - [12.3 回到未来](#)
 - [12.3.1 时间旅行一](#)
 - [12.3.2 时间旅行二](#)
 - [12.3.3 时间旅行三](#)
 - [12.4 丢弃历史](#)
 - [12.5 反转提交](#)

[第13章 Git克隆](#)

[13.1 鸡蛋不装在一个篮子里](#)

[13.2 对等工作区](#)

[13.3 克隆生成裸版本库](#)

[13.4 创建生成裸版本库](#)

[第14章 Git库管理](#)

[14.1 对象和引用哪里去了](#)

[14.2 暂存区操作引入的临时对象](#)

[14.3 重置操作引入的对象](#)

[14.4 Git管家：git-gc](#)

[14.5 Git管家的自动执行](#)

[第3篇 Git和声](#)

[第15章 Git协议与工作协同](#)

[15.1 Git支持的协议](#)

[15.2 多用户协同的本地模拟](#)

[15.3 强制非快进式推送](#)

[15.4 合并后推送](#)

[15.5 禁止非快进式推送](#)

[第16章 冲突解决](#)

[16.1 拉回操作中的合并](#)

[16.2 合并一：自动合并](#)

[16.2.1 修改不同的文件](#)

[16.2.2 修改相同文件的不同区域](#)

[16.2.3 同时更改文件名和文件内容](#)

[16.3 合并二：逻辑冲突](#)

[16.4 合并三：冲突解决](#)

[16.4.1 手工编辑完成冲突解决](#)

[16.4.2 图形工具完成冲突解决](#)

[16.5 合并四：树冲突](#)

[16.5.1 手工操作解决树冲突](#)

[16.5.2 交互式解决树冲突](#)

[16.6 合并策略](#)

[16.7 合并相关的设置](#)

[第17章 Git里程碑](#)

[17.1 显示里程碑](#)

[17.2 创建里程碑](#)

[17.2.1 轻量级里程碑](#)

[17.2.2 带说明的里程碑](#)

[17.2.3 带签名的里程碑](#)

[17.3 删除里程碑](#)

[17.4 不要随意更改里程碑](#)

[17.5 共享里程碑](#)

[17.6 删除远程版本库的里程碑](#)

[17.7 里程碑命名规范](#)

[第18章 Git分支](#)

[18.1 代码管理之殇](#)

[18.1.1 发布分支](#)

[18.1.2 特性分支](#)

[18.1.3 卖主分支](#)

[18.2 分支命令概述](#)

[18.3 "Hello World"开发计划](#)

[18.4 基于特性分支的开发](#)

[18.4.1 创建分支user1/getopt](#)

[18.4.2 创建分支user2/i18n](#)

[18.4.3 开发者user1完功能开发](#)

[18.4.4 将user1/getopt分支合并到主线](#)

[18.5 基于发布分支的开发](#)

[18.5.1 创建发布分支](#)

[18.5.2 开发者user1工作在发布分支](#)

[18.5.3 开发者user2工作在发布分支](#)

- [18.5.4 开发者user2合并推送](#)
- [18.5.5 发布分支的提交合并到主线](#)
- [18.6 分支变基](#)
 - [18.6.1 完成user2/i18n特性分支的开发](#)
 - [18.6.2 分支user2/i18n变基](#)
- [第19章 远程版本库](#)
 - [19.1 远程分支](#)
 - [19.2 分支追踪](#)
 - [19.3 远程版本库](#)
 - [19.4 PUSH和PULL操作与远程版本库](#)
 - [19.5 里程碑和远程版本库](#)
 - [19.6 分支和里程碑的安全性](#)
- [第20章 补丁文件交互](#)
 - [20.1 创建补丁](#)
 - [20.2 应用补丁](#)
 - [20.3 StGit和Quilt](#)
 - [20.3.1 StGit](#)
 - [20.3.2 Quilt](#)
- [第4篇 Git协同模型](#)
- [第21章 经典Git协同模型](#)
 - [21.1 集中式协同模型](#)
 - [21.1.1 传统集中式协同模型](#)
 - [21.1.2 Gerrit特殊的集中式协同模型](#)
 - [21.2 金字塔式协同模型](#)
 - [21.2.1 贡献者开放只读版本库](#)
 - [21.2.2 以补丁方式贡献代码](#)
- [第22章 Topgit协同模型](#)
 - [22.1 作者版本控制系统的三个里程碑](#)
 - [22.2 Topgit原理](#)
 - [22.3 Topgit的安装](#)
 - [22.4 Topgit的使用](#)
 - [22.5 用Topgit方式改造Topgit](#)
 - [22.6 Topgit使用中的注意事项](#)
- [第23章 子模组协同模型](#)
 - [23.1 创建子模组](#)
 - [23.2 克隆带子模组的版本库](#)
 - [23.3 在子模组中修改和子模组的更新](#)
 - [23.4 隐性子模组](#)
 - [23.5 子模组的管理问题](#)
- [第24章 子树合并](#)
 - [24.1 引入外部版本库](#)
 - [24.2 子目录方式合并外部版本库](#)
 - [24.3 利用子树合并跟踪上游改动](#)
 - [24.4 子树拆分](#)
 - [24.5 git-subtree插件](#)
- [第25章 Android式多版本库协同](#)
 - [25.1 关于repo](#)
 - [25.2 安装repo](#)
 - [25.3 repo和清单库的初始化](#)
 - [25.4 清单库和清单文件](#)
 - [25.5 同步项目](#)
 - [25.6 建立Android代码库本地镜像](#)
 - [25.7 repo的命令集](#)
 - [25.8 repo命令的工作流](#)
 - [25.9 好东西不能Android独享](#)
 - [25.9.1 repo+Gerrit模式](#)
 - [25.9.2 repo无审核模式](#)
 - [25.9.3 改进的repo无审核模式](#)
- [第26章 Git和SVN协同模型](#)
 - [26.1 使用git-svn的一般流程](#)

- 26.2 [git-svn的奥秘](#)
 - 26.2.1 [Git库配置文件的扩展及分支映射](#)
 - 26.2.2 [Git工作分支和Subversion如何对应](#)
 - 26.2.3 [其他辅助文件](#)
- 26.3 [多样的git-svn克隆模式](#)
- 26.4 [共享git-svn的克隆库](#)
- 26.5 [git-svn的局限](#)
- 第5篇 [搭建Git服务器](#)
- 第27章 [使用HTTP协议](#)
 - 27.1 [哑传输协议](#)
 - 27.2 [智能HTTP协议](#)
 - 27.3 [Gitweb服务器](#)
 - 27.3.1 [Gitweb的安装](#)
 - 27.3.2 [Gitweb的配置](#)
 - 27.3.3 [版本库的Gitweb相关设置](#)
 - 27.3.4 [即时Gitweb服务](#)
- 第28章 [使用Git协议](#)
 - 28.1 [Git协议语法格式](#)
 - 28.2 [Git服务软件](#)
 - 28.3 [以inetd方式配置运行](#)
 - 28.4 [以runit方式配置运行](#)
- 第29章 [使用SSH协议](#)
 - 29.1 [SSH协议语法格式](#)
 - 29.2 [服务架设方式比较](#)
 - 29.3 [关于SSH公钥认证](#)
 - 29.4 [关于SSH主机别名](#)
- 第30章 [Gitolite服务架设](#)
 - 30.1 [安装Gitolite](#)
 - 30.1.1 [服务器端创建专用账号](#)
 - 30.1.2 [Gitolite的安装/升级](#)
 - 30.1.3 [关于SSH主机别名](#)
 - 30.1.4 [其他的安装方法](#)
 - 30.2 [管理Gitolite](#)
 - 30.2.1 [管理员克隆gitolite-admin管理库](#)
 - 30.2.2 [增加新用户](#)
 - 30.2.3 [更改授权](#)
 - 30.3 [Gitolite授权详解](#)
 - 30.3.1 [授权文件的基本语法](#)
 - 30.3.2 [定义用户组和版本库组](#)
 - 30.3.3 [版本库ACL](#)
 - 30.3.4 [Gitolite授权机制](#)
 - 30.4 [版本库授权案例](#)
 - 30.4.1 [对整个版本库进行授权](#)
 - 30.4.2 [通配符版本库的授权](#)
 - 30.4.3 [用户自己的版本库空间](#)
 - 30.4.4 [对引用的授权：传统模式](#)
 - 30.4.5 [对引用的授权：扩展模式](#)
 - 30.4.6 [对引用的授权：禁用规则的使用](#)
 - 30.4.7 [用户分支](#)
 - 30.4.8 [对路径的写授权](#)
 - 30.5 [创建新版本库](#)
 - 30.5.1 [在配置文件中出现的版本库，即时生成](#)
 - 30.5.2 [通配符版本库，管理员通过推送创建](#)
 - 30.5.3 [直接在服务器端创建](#)
 - 30.6 [对Gitolite的改进](#)
 - 30.7 [Gitolite功能拓展](#)
 - 30.7.1 [版本库镜像](#)
 - 30.7.2 [Gitweb和Git daemon支持](#)
 - 30.7.3 [其他功能拓展和参考](#)
- 第31章 [Gitolite服务架设](#)

- [31.1 安装Gitolis](#)
 - [31.1.1 Gitosis的安装](#)
 - [31.1.2 服务器端创建专用账号](#)
 - [31.1.3 Gitosis服务初始化](#)
- [31.2 管理Gitolis](#)
 - [31.2.1 管理员克隆gitolit-admin管理库](#)
 - [31.2.2 增加新用户](#)
 - [31.2.3 更改授权](#)
- [31.3 Gitosis授权详解](#)
 - [31.3.1 Gitosis默认设置](#)
 - [31.3.2 管理版本库gitosis-admin](#)
 - [31.3.3 定义用户组和授权](#)
 - [31.3.4 Gitweb整合](#)
- [31.4 创建新版本库](#)
- [31.5 轻量级管理的Git服务](#)
- [第32章 Gerrit代码审核服务器](#)
 - [32.1 Gerrit的实现原理](#)
 - [32.2 架设Gerrit的服务器](#)
 - [32.3 Gerrit的配置文件](#)
 - [32.4 Gerrit的数据库访问](#)
 - [32.5 立即注册为Gerrit管理员](#)
 - [32.6 管理员访问SSH的管理接口](#)
 - [32.7 创建新项目](#)
 - [32.8 从已有的Git库创建项目](#)
 - [32.9 定义评审 workflow](#)
 - [32.10 Gerrit评审 workflow 实战](#)
 - [32.10.1 开发者在本地版本库中工作](#)
 - [32.10.2 开发者向审核服务器提交](#)
 - [32.10.3 审核评审任务](#)
 - [32.10.4 评审任务没有通过测试](#)
 - [32.10.5 重新提交新的补丁集](#)
 - [32.10.6 新修订集通过评审](#)
 - [32.10.7 从远程版本库更新](#)
 - [32.11 更多Gerrit参考](#)
- [第33章 Git版本库托管](#)
 - [33.1 Github](#)
 - [33.2 Gitorious](#)
- [第6篇 迁移到Git](#)
- [第34章 CVS版本库到Git的迁移](#)
 - [34.1 安装cvs2svn \(含cvs2git\)](#)
 - [34.1.1 Linux下cvs2svn的安装](#)
 - [34.1.2 Mac OS X下cvs2svn的安装](#)
 - [34.2 版本库转换的准备工作](#)
 - [34.2.1 版本库转换注意事项](#)
 - [34.2.2 文件名乱码问题](#)
 - [34.2.3 提交说明乱码问题](#)
 - [34.3 版本库转换](#)
 - [34.3.1 配置文件解说](#)
 - [34.3.2 运行cvs2git完成转换](#)
 - [34.4 迁移后的版本库检查](#)
- [第35章 更多版本控制系统的迁移](#)
 - [35.1 SVN版本库到Git的迁移](#)
 - [35.2 Hg版本库到Git的迁移](#)
 - [35.3 通用版本库迁移](#)
 - [35.4 Git版本库整理](#)
 - [35.4.1 环境变量过滤器](#)
 - [35.4.2 树过滤器](#)
 - [35.4.3 暂存区过滤器](#)
 - [35.4.4 父节点过滤器](#)
 - [35.4.5 提交说明过滤器](#)

- [35.4.6 提交过滤器](#)
- [35.4.7 里程碑名字过滤器](#)
- [35.4.8 子目录过滤器](#)
- [第7篇 Git的其他应用](#)
- [第36章 etckeeper](#)
- [36.1 安装etckeeper](#)
- [36.2 配置etckeeper](#)
- [36.3 使用etckeeper](#)
- [第37章 Gistore](#)
- [37.1 Gistore的安装](#)
- [37.1.1 软件依赖](#)
- [37.1.2 从源码安装Gistore](#)
- [37.1.3 用easy_install安装](#)
- [37.2 Gistore的使用](#)
- [37.2.1 创建并初始化备份库](#)
- [37.2.2 Gistore的配置文件](#)
- [37.2.3 Gistore的备份项管理](#)
- [37.2.4 执行备份任务](#)
- [37.2.5 查看备份日志](#)
- [37.2.6 查看及恢复备份数据](#)
- [37.2.7 备份回滚及设置](#)
- [37.2.8 注册备份任务别名](#)
- [37.2.9 自动备份: crontab](#)
- [37.3 Gistore双机备份](#)
- [第38章 补丁中的二进制文件](#)
- [38.1 Git版本库中二进制文件变更的支持](#)
- [38.2 对非Git版本库中二进制文件变更的支持](#)
- [38.3 其他工具对Git扩展补丁文件的支持](#)
- [第39章 云存储](#)
- [39.1 现有云存储的问题](#)
- [39.2 Git式云存储畅想](#)
- [第8篇 Git杂谈](#)
- [第40章 跨平台操作Git](#)
- [40.1 字符集问题](#)
- [40.2 文件名大小写问题](#)
- [40.3 换行符问题](#)
- [第41章 Git的其他特性](#)
- [41.1 属性](#)
- [41.1.1 属性定义](#)
- [41.1.2 属性文件及优先级](#)
- [41.1.3 常用属性介绍](#)
- [41.2 钩子和模板](#)
- [41.2.1 Git钩子](#)
- [41.2.2 Git模板](#)
- [41.3 稀疏检出和浅克隆](#)
- [41.3.1 稀疏检出](#)
- [41.3.2 浅克隆](#)
- [41.4 嫁接和替换](#)
- [41.4.1 提交嫁接](#)
- [41.4.2 提交替换](#)
- [41.5 Git评注](#)
- [41.5.1 评注的奥秘](#)
- [41.5.2 评注相关命令](#)
- [41.5.3 评注相关配置](#)
- [第9篇 附录](#)
- [附录A Git命令索引](#)
- [A.1 常用的Git命令](#)
- [A.2 对象库操作相关命令](#)
- [A.3 引用操作相关命令](#)
- [A.4 版本库管理相关命令](#)

- [A.5 数据传输相关命令](#)
- [A.6 邮件相关命令](#)
- [A.7 协议相关命令](#)
- [A.8 版本库转换和交互相关命令](#)
- [A.9 合并相关的辅助命令](#)
- [A.10 杂项](#)
- [附录B Git与CVS面对面](#)
- [B.1 面对面访谈录](#)
- [B.2 Git和CVS命令对照](#)
- [附录C Git与SVN面对面](#)
- [C.1 面对面访谈录](#)
- [C.2 Git和SVN命令对照](#)
- [附录D Git与Hg面对面](#)
- [D.1 面对面访谈录](#)
- [D.2 Git和Hg命令对照](#)

前言

版本控制是管理数据变更的艺术，无论数据变更是来自同一个人，还是来自不同的人（一个团队）。版本控制系统不但要忠实地记录数据的每一次变更，还要能够帮助还原任何一次历史变更，以及实现团队的协同工作等。Git就是版本控制系统中的佼佼者。

我对版本控制系统的兴趣源自于我的个人知识管理实践，其核心就是撰写可维护的文档，并保存于版本控制系统中。可维护文档的格式可以是DocBook、FreeMind、reStructuredText等。我甚至还对FreeMind加以改造以便让其文档格式更适合于版本控制系统，这就是我的第一个开源实践：托管于SourceForge上的FreeMind-MMX项目^[1]。文档书写格式的问题解决之后，就是文档的存储问题了。通过版本控制系统，很自然地就可以实现对文档历史版本的保存，但是如何避免因为版本控制系统瘫痪而导致数据丢失呢？Git用其崭新的分布式的版本控制设计提供了最好的解决方案。使用Git，我的知识库不再只有唯一的版本库与之对应，而是可以通过克隆操作分发到不同的磁盘或主机上，克隆的版本库之间通过推送（PUSH）和拉回（PULL）等操作进行同步，数据安全得到了极大的提升。在版本控制系统的忠实呵护下，我的知识库中关于Git的FreeMind脑图在日积月累中变得越来越翔实，越来越清晰，最终成为本书的雏形。

版本控制能决定项目的成败，甚至是公司的生死，此言不虚。我在推广开源项目管理工具和为企业提供咨询服务的过程中看到，有很多团队因为版本控制系统管理的混乱导致项目延期、修正的Bug重现、客户的问题不能在代码中定位……无论他们使用的是什么版本控制系统（开源的或是商业的）都是如此。这是因为传统的集中式版本控制系统不能有效地管理分支和进行分支间合并。集中管理的版本库只有唯一的分支命名空间，需要专人管理，从而造成分支创建的不自由；分支间的合并要么因为缺乏追踪导致重复合并、引发严重冲突，要么因为版本控制系统本身蹩脚的设计导致分支合并时效率低下和陷阱重重。Git凭借其灵活的设计让项目摆脱分支管理的梦魇。

我的公司也经历过代码管理的生死考验。因为公司的开发模式主要是基于开源软件的二次开发，所以最早在使用SVN（Subversion）做版本控制时，很自然地使用了SVN卖主分支模型来管理代码。随着增加和修改的代码越来越多，我们开发的软件与开源软件上游的偏离也越来越远，当上游有新版本发布时，最早可能只用几个小时就可以将改动迁移过去，但是如果对上游的改动多达几十甚至上百处时，迁移的过程就会异常痛苦，基本上和重新做一遍差不多。那时似乎只有一种选择：不再与上游合并，不再追踪上游的改动，而这与公司的价值观“发动全球智慧为客户创造价值”相违背。迷茫之中，分布式版本控制系统飘然而至，原来版本控制还可以这么做。

我最先尝试的分布式版本控制系统是Hg（Mercurial），当发现Hg和MQ（Hg的一个插件）这一对宝贝儿的时候，我如获至宝。逐渐地，公司的版本库都迁移到了Hg上。但随着新的开发人员的加入，问题又出现了，一个人使用Hg和MQ很好，但多个人使用时则会出现难以协同的问题。于是我们大胆地采用了Git，并在实践中结合Topgit等工具进行代码的管理。再一次，也许是最后一次，我们的代码库迁移到了Git。

最早认识分布式版本控制，源自于我们看到了众多开源项目的版本控制系统大迁移，这场迁移还在进行中。

MoinMoin是我们关注的一个开源的维基软件，2006年，它的代码库从SVN迁移到了Hg^[2]。

Mailman同样是我们关注的一个开源邮件列表软件。2007年，它的代码库从SVN迁移到了Bazaar^[3]。

Linux采用Git作为版本控制系统（一点都不奇怪，因为Git就是Linus Torvalds开发的）。

Android是目前最为流行的开源项目之一，因为潜在市场巨大，已经吸引了越来越多的开发者进入这个市场，而Android就是用Git维护的。

欢迎访问：电子书学习和下载网站 (<https://www.shgis.cn>)

文档名称：《Git权威指南》蒋鑫.epub

请登录 <https://shgis.cn/post/1836.html> 下载完整文档。

手机端请扫码查看：

