



--distributed--is--the--new--centralized

▢

- [About](#)
 - [Branching and Merging](#)
 - [Small and Fast](#)
 - [Distributed](#)
 - [Data Assurance](#)
 - [Staging Area](#)
 - [Free and Open Source](#)
 - [Trademark](#)
 - [Documentation](#)
 - [Reference](#)
 - [Book](#)
 - [Videos](#)
 - [External Links](#)
 - [Downloads](#)
 - [GUI Clients](#)
 - [Logos](#)
 - [Community](#)
-

This book is available in [English](#).

Full translation available in

[azərbaycan dili](#),

[български език](#),

[Deutsch](#),

[Español](#),

[Français](#),
[Ελληνικά](#),
[日本語](#),
[한국어](#),
[Nederlands](#),
[Русский](#),
[Slovenščina](#),
[Tagalog](#),
[Українська](#)
[简体中文](#),

Partial translations available in

[Čeština](#),
[Македонски](#),
[Polski](#),
[Српски](#),
[Ўзбекча](#),
[繁體中文](#),

Translations started for

[Беларуская](#),
[فارسی](#),
[Indonesian](#),
[Italiano](#),
[Bahasa Melayu](#),
[Português \(Brasil\)](#),
[Português \(Portugal\)](#),

[Svenska](#),

[Türkçe](#).

The source of this book is [hosted on GitHub](#).

Patches, suggestions and comments are welcome.

[Chapters](#) ▼

1. [1. 起步](#)

1. 1.1 [关于版本控制](#)

2. 1.2 [Git 简史](#)

3. 1.3 [Git 是什么?](#)

4. 1.4 [命令行](#)

5. 1.5 [安装 Git](#)

6. 1.6 [初次运行 Git 前的配置](#)

7. 1.7 [获取帮助](#)

8. 1.8 [总结](#)

2. [2. Git 基础](#)

1. 2.1 [获取 Git 仓库](#)

2. 2.2 [记录每次更新到仓库](#)

3. 2.3 [查看提交历史](#)

4. 2.4 [撤销操作](#)

5. 2.5 [远程仓库的使用](#)

6. 2.6 [打标签](#)

7. 2.7 [Git 别名](#)

8. 2.8 [总结](#)

3. [3. Git 分支](#)

1. 3.1 [分支简介](#)
2. 3.2 [分支的新建与合并](#)
3. 3.3 [分支管理](#)
4. 3.4 [分支开发工作流](#)
5. 3.5 [远程分支](#)
6. 3.6 [变基](#)
7. 3.7 [总结](#)

4. [4. 服务器上的 Git](#)

1. 4.1 [协议](#)
2. 4.2 [在服务器上搭建 Git](#)
3. 4.3 [生成 SSH 公钥](#)
4. 4.4 [配置服务器](#)
5. 4.5 [Git 守护进程](#)
6. 4.6 [Smart HTTP](#)
7. 4.7 [GitWeb](#)
8. 4.8 [GitLab](#)
9. 4.9 [第三方托管的选择](#)
10. 4.10 [总结](#)

5. [5. 分布式 Git](#)

1. 5.1 [分布式工作流程](#)
2. 5.2 [向一个项目贡献](#)
3. 5.3 [维护项目](#)
4. 5.4 [总结](#)

1. [6. GitHub](#)

1. 6.1 [账户的创建和配置](#)

- 2. 6.2 [对项目做出贡献](#)
- 3. 6.3 [维护项目](#)
- 4. 6.4 [管理组织](#)
- 5. 6.5 [脚本 GitHub](#)
- 6. 6.6 [总结](#)

2. [7. Git 工具](#)

- 1. 7.1 [选择修订版本](#)
- 2. 7.2 [交互式暂存](#)
- 3. 7.3 [贮藏与清理](#)
- 4. 7.4 [签署工作](#)
- 5. 7.5 [搜索](#)
- 6. 7.6 [重写历史](#)
- 7. 7.7 [重置揭密](#)
- 8. 7.8 [高级合并](#)
- 9. 7.9 [Rerere](#)
- 10. 7.10 [使用 Git 调试](#)
- 11. 7.11 [子模块](#)
- 12. 7.12 [打包](#)
- 13. 7.13 [替换](#)
- 14. 7.14 [凭证存储](#)
- 15. 7.15 [总结](#)

3. [8. 自定义 Git](#)

- 1. 8.1 [配置 Git](#)
- 2. 8.2 [Git 属性](#)
- 3. 8.3 [Git 钩子](#)
- 4. 8.4 [使用强制策略的一个例子](#)
- 5. 8.5 [总结](#)

4. **9. Git 与其他系统**

1. 9.1 [作为客户端的 Git](#)
2. 9.2 [迁移到 Git](#)
3. 9.3 [总结](#)

5. **10. Git 内部原理**

1. 10.1 [底层命令与上层命令](#)
2. 10.2 [Git 对象](#)
3. 10.3 [Git 引用](#)
4. 10.4 [包文件](#)
5. 10.5 [引用规范](#)
6. 10.6 [传输协议](#)
7. 10.7 [维护与数据恢复](#)
8. 10.8 [环境变量](#)
9. 10.9 [总结](#)

1. **A1. 附录 A: 在其它环境中使用 Git**

1. A1.1 [图形界面](#)
2. A1.2 [Visual Studio 中的 Git](#)
3. A1.3 [Visual Studio Code 中的 Git](#)
4. A1.4 [Eclipse 中的 Git](#)
5. A1.5 [IntelliJ / PyCharm / WebStorm / PhpStorm / RubyMine 中的 Git](#)
6. A1.6 [Sublime Text 中的 Git](#)
7. A1.7 [Bash 中的 Git](#)
8. A1.8 [Zsh 中的 Git](#)
9. A1.9 [Git 在 PowerShell 中使用 Git](#)
10. A1.10 [总结](#)

2. [A2. 附录 B: 在你的应用中嵌入 Git](#)

1. [A2.1 命令行 Git 方式](#)
2. [A2.2 Libgit2](#)
3. [A2.3 JGit](#)
4. [A2.4 go-git](#)
5. [A2.5 Dulwich](#)

3. [A3. 附录 C: Git 命令](#)

1. [A3.1 设置与配置](#)
2. [A3.2 获取与创建项目](#)
3. [A3.3 快照基础](#)
4. [A3.4 分支与合并](#)
5. [A3.5 项目分享与更新](#)
6. [A3.6 检查与比较](#)
7. [A3.7 调试](#)
8. [A3.8 补丁](#)
9. [A3.9 邮件](#)
10. [A3.10 外部系统](#)
11. [A3.11 管理](#)
12. [A3.12 底层命令](#)

2nd Edition

10.1 Git 内部原理 – 底层命令与上层命令

无论是从之前的章节直接跳到本章，还是读完了其余章节一直到这——你都将在本章见识到 Git 的内部工作原理和实现方式。我们认为学习这部分内容对于理解 Git 的用途和强大至关重要。不过也有人认为这些内容对于初学者而言可能难以理解且过于复杂。因此我们把这部分内容放在最后一章，在学习过程中可以先阅读这部分，也可以晚点阅读这部分，这取决于你自己。

无论如何，既然已经读到了这里，就让我们开始吧。首先要弄明白一点，从根本上来讲 Git 是一个内容寻址（content-addressable）文件系统，并在此之上提供了一个版本控制系统的用户界面。马上你就会学到这意味着什么。

早期的 Git（主要是 1.5 之前的版本）的用户界面要比现在复杂的多，因为它更侧重于作为一个文件系统，而不是一个打磨过的版本控制系统。不时会有一些陈词滥调抱怨早期那个晦涩复杂的 Git 用户界面；不过最近几年来，它已经被改进到不输于任何其他版本控制系统地清晰易用了。

内容寻址文件系统层是一套相当酷的东西，所以在本章我们会先讲解这部分内容。随后我们会学习传输机制和版本库管理任务——你迟早会和它们打交道。

底层命令与上层命令

本书主要涵盖了 checkout、branch、remote 等约 30 个 Git 的子命令。然而，由于 Git 最初是一套面向版本控制系统的工具集，而不是一个完整的、用户友好的版本控制系统，所以它还包含了一部分用于完成底层工作的子命令。这些命令被设计成能以 UNIX 命令行的风格连接在一起，抑或藉由脚本调用，来完成工作。这部分命令一般被称作“底层（plumbing）”命令，而那些更友好的命令则被称作“上层（porcelain）”命令。

你或许已经注意到了，本书前九章专注于探讨上层命令。然而在本章中，我们将主要面对底层命令。因为，底层命令得以让你窥探 Git 内部的工作机制，也有助于说明 Git 是如何完成工作的，以及它为何如此运作。多数底层命令并不面向最终用户：它们更适合作为新工具的组件和自定义脚本的组成部分。

当在一个新目录或已有目录执行 `git init` 时，Git 会创建一个 `.git` 目录。这个目录包含了几乎所有 Git 存储和操作的东西。如若想备份或复制一个版本库，只需把这个目录拷贝至另一处即可。本章探讨的所有内容，均位于这个目录内。新初始化的 `.git` 目录的典型结构如下：

```
$ ls -Fl
config
description
HEAD
hooks/
info/
objects/
refs/
```

随着 Git 版本的不同，该目录下可能还会包含其他内容。不过对于一个全新的 `git init` 版本库，这将是看到的默认结构。`description` 文件仅供 GitWeb 程序使用，我们无需关心。`config` 文件包含项目特有的配置选项。`info` 目录包含一个全局性排除（global exclude）文件，用以放置那些不希望被记录在 `.gitignore` 文件中的忽略模式（ignored patterns）。`hooks` 目录包含客户端或服务端的钩子脚本（hook scripts），在 [Git 钩子](#) 中这部分话题已被详细探讨过。


剩下的四个条目很重要：`HEAD` 文件、（尚待创建的）`index` 文件，和 `objects` 目录、`refs` 目录。它们都是 Git 的核心组成部分。`objects` 目录存储所有数据内容；`refs` 目录存储指向数据（分支、远程仓库和标签等）的提交对象的指针；`HEAD` 文件指向目前被检出的分支；`index` 文件保存暂存区信息。我们将详细地逐一检视这四部分，来理解 Git 是如何运转的。

[prev](#) | [next](#)

[About this site](#)

Patches, suggestions, and comments are welcome.

Git is a member of [Software Freedom Conservancy](#).

A red arrow originates from the text box and points to the word 'HEAD' in the paragraph above.

实际存储的内容是一个路径，表示refs中一个提交对象