



--everything-is-local

- [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 [撤消操作](#)
- 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

2.1 Git 基础 - 获取 Git 仓库

如果你只想通过阅读一章来学习 Git，那么本章将是你的不二选择。本章涵盖了你在使用 Git 完成各种工作时将会用到的各种基本命令。在学习完本章之后，你应该能够配置并初始化一个仓库（repository）、开始或停止跟踪（track）文件、暂存（stage）或提交（commit）更改。本章也将向你演示了如何配置 Git 来忽略指定的文件和文件模式、如何迅速而简单地撤销错误操作、如何浏览你的项目的历史版本以及不同提交（commits）之间的差异、如何向你的远程仓库推送（push）以及如何从你的远程仓库拉取（pull）文件。

获取 Git 仓库

通常有两种获取 Git 项目仓库的方式：

1. 将尚未进行版本控制的本地目录转换为 Git 仓库；
2. 从其它服务器 **克隆** 一个已存在的 Git 仓库。

两种方式都会在你的本地机器上得到一个工作就绪的 Git 仓库。

在已存在目录中初始化仓库

如果你有一个尚未进行版本控制的项目目录，想要用 Git 来控制它，那么首先需要进入该项目目录中。如果你还没这样做过，那么不同系统上的做法有些不同：

在 Linux 上：

```
$ cd /home/user/my_project
```

在 macOS 上:

```
$ cd /Users/user/my_project
```

在 Windows 上:

```
$ cd /c/user/my_project
```

之后执行:

```
$ git init
```

该命令将创建一个名为 `.git` 的子目录, 这个子目录含有你初始化的 Git 仓库中所有的必须文件, 这些文件是 Git 仓库的骨干。但是, 在这个时候, 我们仅仅是做了一个初始化的操作, 你的项目里的文件还没有被跟踪。(参见 [Git 内部原理](#) 来了解更多关于到底 `.git` 文件夹中包含了哪些文件的信息。)

如果在一个已存在文件的文件夹 (而非空文件夹) 中进行版本控制, 你应该开始追踪这些文件并进行初始提交。可以通过 `git add` 命令来指定所需的文件来进行追踪, 然后执行 `git commit` :

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'initial project version'
```

稍后我们再逐一解释这些指令的行为。现在, 你已经得到了一个存在被追踪文件与初始提交的 Git 仓库。

克隆现有的仓库

如果你想获得一份已经存在了的 Git 仓库的拷贝, 比如说, 你想为某个开源项目贡献自己的一份力, 这时就要用到 `git clone` 命令。如果你对其它的 VCS 系统 (比如说 Subversion) 很熟悉, 请留心一下你所使用的命令是 "clone" 而不是 "checkout"。这是 Git 区别于其它版本控制系统的一个重要特性, Git 克隆的是该 Git 仓库服务器上的几乎所有数据, 而不是仅仅复制完成你的工作所需要文件。当你执行 `git clone` 命令的时候, 默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。事实上, 如果你的服务器的磁盘坏掉了, 你通常可以使用任何一个克隆下来的用户端来重建服务器上的仓库 (虽然可能会丢失某些服务器端的钩子 (hook) 设置, 但是所有版本的数据仍在, 详见 [在服务器上搭建 Git](#)) 。

克隆仓库的命令是 `git clone <url>`。比如, 要克隆 Git 的链接库 `libgit2`, 可以用下面的命令:

```
$ git clone https://github.com/libgit2/libgit2
```

这会在当前目录下创建一个名为 "libgit2" 的目录, 并在这个目录下初始化一个 `.git` 文件夹, 从远程仓库拉取下所有数据放入 `.git` 文件夹, 然后从中读取最新版本的文件的拷贝。如果你进入到这个新建的 `libgit2` 文件夹, 你会发现所有的项目文件已经在里面了, 准备就绪等待后续的开发和使用。

如果你想在克隆远程仓库的时候, 自定义本地仓库的名字, 你可以通过额外的参数指定新的目录名:

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

这会执行与上一条命令相同的操作, 但目标目录名变为了 `mylibgit`。

Git 支持多种数据传输协议。上面的例子使用的是 `https://` 协议，不过你也可以使用 `git://` 协议或者使用 SSH 传输协议，比如 `user@server:path/to/repo.git`。 [在服务器上搭建 Git](#) 将会介绍所有这些协议在服务器端如何配置使用，以及各种方式之间的利弊。

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

[About this site](#)

Patches, suggestions, and comments are welcome.

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