



- [About](#)
- [Documentation](#)
  - [Reference](#)
  - [Book](#)
  - [Videos](#)
  - [External Links](#)
- [Blog](#)
- [Downloads](#)
  - [GUI Clients](#)
  - [Logos](#)
- [Community](#)

---

This book is translated into [Deutsch](#), [简体中文](#), [正體中文](#), [Français](#), [日本語](#), [Nederlands](#), [Русский](#), [한국어](#), [Português \(Brasil\)](#) and [Čeština](#).

Partial translations available in [Arabic](#), [Español](#), [Indonesian](#), [Italiano](#), [Suomi](#), [Македонски](#), [Polski](#) and [Türkçe](#).

Translations started for [Azərbaycan dili](#), [Беларуская](#), [Català](#), [Esperanto](#), [Español \(Nicaragua\)](#), [فارسی](#), [हिन्दी](#), [Magyar](#), [Norwegian Bokmål](#), [Română](#), [Српски](#), [ภาษาไทย](#), [Tiếng Việt](#), [Українська](#) and [Ўзбекча](#).

---

The source of this book is [hosted on GitHub](#).  
Patches, suggestions and comments are welcome.

## [Chapters ▼](#)

### 1. [1. 起步](#)

- 1.1 [关于版本控制](#)
- 1.2 [Git 简史](#)
- 1.3 [Git 基础](#)
- 1.4 [安装 Git](#)
- 1.5 [初次运行 Git 前的配置](#)
- 1.6 [获取帮助](#)
- 1.7 [小结](#)

### 2. [2. Git 基础](#)

- 2.1 [取得项目的 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 [技巧和窍门](#)
- 8. 2.8 [小结](#)

### 3. [3. Git 分支](#)

- 1. 3.1 [何谓分支](#)
- 2. 3.2 [分支的新建与合并](#)
- 3. 3.3 [分支的管理](#)
- 4. 3.4 [利用分支进行开发的工作流程](#)
- 5. 3.5 [远程分支](#)
- 6. 3.6 [分支的衍合](#)
- 7. 3.7 [小结](#)

### 1. [4. 服务器上的 Git](#)

- 1. 4.1 [协议](#)
- 2. 4.2 [在服务器上部署 Git](#)
- 3. 4.3 [生成 SSH 公钥](#)
- 4. 4.4 [架设服务器](#)
- 5. 4.5 [公共访问](#)
- 6. 4.6 [GitWeb](#)
- 7. 4.7 [Gitis](#)
- 8. 4.8 [Gitolite](#)
- 9. 4.9 [Git 守护进程](#)
- 10. 4.10 [Git 托管服务](#)
- 11. 4.11 [小结](#)

### 2. [5. 分布式 Git](#)

- 1. 5.1 [分布式工作流程](#)
- 2. 5.2 [为项目做贡献](#)
- 3. 5.3 [项目的管理](#)
- 4. 5.4 [小结](#)

### 3. [6. Git 工具](#)

1. 6.1 [修订版本 \(Revision\) 选择](#)
2. 6.2 [交互式暂存](#)
3. 6.3 [储藏 \(Stashing\)](#)
4. 6.4 [重写历史](#)
5. 6.5 [使用 Git 调试](#)
6. 6.6 [子模块](#)
7. 6.7 [子树合并](#)
8. 6.8 [总结](#)

## 1. [7. 自定义 Git](#)

1. 7.1 [配置 Git](#)
2. 7.2 [Git 属性](#)
3. 7.3 [Git 挂钩](#)
4. 7.4 [Git 强制策略实例](#)
5. 7.5 [总结](#)

## 2. [8. Git 与其他系统](#)

1. 8.1 [Git 与 Subversion](#)
2. 8.2 [迁移到 Git](#)
3. 8.3 [总结](#)

## 3. [9. Git 内部原理](#)

1. 9.1 [底层命令 \(Plumbing\) 和高层命令 \(Porcelain\)](#)
2. 9.2 [Git 对象](#)
3. 9.3 [Git References](#)
4. 9.4 [Packfiles](#)
5. 9.5 [The Refspec](#)
6. 9.6 [传输协议](#)
7. 9.7 [维护及数据恢复](#)
8. 9.8 [总结](#)

1st Edition

# 2.1 Git 基础 - 取得项目的 Git 仓库

## 取得项目的 Git 仓库

有两种取得 Git 项目仓库的方法。第一种是在现存的目录下，通过导入所有文件来创建新的 Git 仓库。第二种是从已有的 Git 仓库克隆出一个新的镜像仓库来。

## 在工作目录中初始化新仓库

要对现有的某个项目开始用 Git 管理，只需到此项目所在的目录，执行：

```
$ git init
```

初始化后，在当前目录下会出现一个名为 `.git` 的目录，所有 Git 需要的数据和资源都存放在这个目录中。不过目前，仅仅是按照既有的结构框架初始化好了里边所有的文件和目录，但我们还没有开始跟踪管理项目中的任何一个文件。（在第九章我们会详细说明刚才创建的 `.git` 目录中究竟有哪些文件，以及都起些什么作用。）

如果当前目录下有几个文件想要纳入版本控制，需要先用 `git add` 命令告诉 Git 开始对这些文件进行跟踪，然后提交：

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

稍后我们再逐一解释每条命令的意思。不过现在，你已经得到了一个实际维护着若干文件的 Git 仓库。

## 从现有仓库克隆

如果想对某个开源项目出一份力，可以先把该项目的 Git 仓库复制一份出来，这就需要用到 `git clone` 命令。如果你熟悉其他的 VCS 比如 Subversion，你可能已经注意到这里使用的是 `clone` 而不是 `checkout`。这是个非常重要的差别，Git 收取的是项目历史的所有数据（每一个文件的每一个版本），服务器上有的数据克隆之后本地也都有了。实际上，即便服务器的磁盘发生故障，用任何一个克隆出来的客户端都可以重建服务器上的仓库，回到当初克隆时的状态（虽然可能会丢失某些服务器端的挂钩设置，但所有版本的数据仍旧还在，有关细节请参考第四章）。

克隆仓库的命令格式为 `git clone [url]`。比如，要克隆 Ruby 语言的 Git 代码仓库 Grit，可以用下面的命令：

```
$ git clone git://github.com/schacon/grit.git
```

这会在当前目录下创建一个名为 `grit` 的目录，其中包含一个 `.git` 的目录，用于保存下载下来的所有版本记录，然后从中取出最新版本的文件拷贝。如果进入这个新建的 `grit` 目录，你会看到项目中的所有文件已经在里边了，准备好后续的开发和使用。如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

```
$ git clone git://github.com/schacon/grit.git mygrit
```

唯一的差别就是，现在新建的目录成了 `mygrit`，其他的都和上边的一样。

Git 支持许多数据传输协议。之前的例子使用的是 `git://` 协议，不过你也可以用 `http(s)://` 或者 `user@server:/path.git` 表示的 SSH 传输协议。我们会在第四章详细介绍所有这些协议在服务器端该如何配置使用，以及各种方式之间的利弊。

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

This [open sourced](#) site is [hosted on GitHub](#).

Patches, suggestions and comments are welcome.

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