



中国科学技术大学  
University of Science and Technology of China

# Git 简明教程



- 1 什么是配置管理
- 2 为什么我们需要git
- 3 git带来的好处
- 4 git与类似工具（SVN）的区别与联系
- 5 git的主要命令和原理（以 gitee 为例）



# 1 什么是配置管理

- **官方定义：**
  - 软件配置管理（Software Configuration Management，简称 SCM），是指在软件生命周期中，记录和控制各项变更的过程。
- **通俗理解：**
  - 它是整个项目的“高级后悔药”和“游戏存档系统”。
- **管理对象（配置项）：**
  - 不仅仅是源代码，还包括：需求文档、设计图、配置文件、测试脚本、甚至是运行环境配置。
- **核心目标：**
  - **版本控制：**随时能找回历史版本。
  - **变更追踪：**知道谁、在什么时候、因为什么原因、修改了哪行代码。
  - **团队协作：**保证多人并行开发时，产出是一致的。



## 2 为什么我们需要git

- **没有 Git 时的“日常痛点”：**
  - **靠重命名做版本控制：** 期末大作业.zip -> 大作业\_修改版.zip -> 大作业\_最终版\_v2.zip
  - **原始的多人协作：**
    - 通过微信/QQ互传代码，最后不知道谁的代码是最新的；
    - 一不小心就把队友写的代码覆盖了。
  - **Bug 溯源困难：** 程序突然跑不通了，不知道是哪次修改引入的错误。
- **Git 提供的核心能力：**
  - **时间机器：** 随时、安全地把整个项目回退到过去的任意一秒。
  - **平行宇宙（分支）：** 任何人都可以放心地去尝试开发新功能，就算代码写得再烂，也绝不会影响主干程序的正常运行。



## 3 git带来的好处

- **完全的分布式架构：**
  - 每台设备上都有一个**完整的版本库历史**。
  - 在网络不佳的环境下，依然可以正常查看历史、提交代码、切换分支。
- **极其轻量级的“分支 (Branch)”：**
  - 在 Git 中创建分支，就像新建一个标签一样快（毫秒级）。
  - **鼓励试错：**极大降低了开发新功能的成本和合并代码的痛苦。
- **性能极佳：**
  - 绝大多数操作（提交、查看历史、对比差异）都在本地磁盘和内存中完成，不需要等待网络延迟。
- **数据完整性极高：**
  - Git 使用 SHA-1 哈希算法对所有文件计算校验和。
  - **操作可溯源：**代码一旦提交，任何人都不可能在不改变哈希值的情况下悄悄篡改历史。



## 4 git与类似工具 (SVN) 的区别与联系

- **联系**：二者都是优秀的版本控制系统 (VCS) ， 都支持团队协作、代码追踪和版本回退。
- **区别 1 (核心区别) : 架构不同**
  - **SVN (集中式)** : 强依赖中央服务器。如果中央服务器宕机, 所有人都无法提交代码, 也无法查看历史。
  - **Git (分布式)** : 去中心化。每个开发者本地就是一个完整的服务器, 中央服务器 (如 GitHub/GitLab) 只是为了方便大家交换数据。
- **区别 2: 存储方式不同**
  - **SVN** 存储的是 “**差异/增量**” : 只记录每次文件变动了哪几行。
  - **Git** 存储的是 “**快照**” : 每次提交, 都相当于给整个项目拍了一张完整的照片 (微型文件系统) , 这使得 Git 的切换和回退速度极快。



# 5 git 的主要命令和原理 (以 gitee 为例)

5.1 远程仓库初始化

5.2 本地仓库初始化

5.3 本地工作流程

5.4 分支

5.5 深入学习



## 5.1 远程仓库初始化

希冀平台的 GitLab 上已经为大家创建了远程仓库，可以直接使用。本课程中，如果要使用 GitHub 等远程仓库托管服务，请确保仓库**私有**<sup>1</sup>。

### 查看远程仓库

```
$ git remote -v
```

由于 GitHub 等国外托管平台在校园网下可能存在访问不稳定的情况，日常学习和课设强烈推荐<sup>2</sup>使用国内的 Gitee（码云<sup>2</sup>）。

1. <https://stackoverflow.com/questions/10065526/github-how-to-make-a-fork-of-public-repository-private>
2. <https://gitee.com/>



# 5.1 远程仓库初始化—— gitee 账户的创建



开源

产品

客户案例

价格 特惠

模力方舟

服务与支持

搜开源

登录

注册

人与 AI 协作的 DevOps 平台

## 让每一行代码 都有改变世界的力量

请输入你的手机号码

企业免费使用

免费

个人账号注册



张晨  
开发工程师

新功能模块开发完成，  
已提交代码



AI 代码审查助手

按照企业规范，检测到 3 处代码  
异味，请修改代码



AI 项目管理助手

迭代已经进行两周，发现后端接口  
延误了 20%，及时上报给项目管理



AI 测试工程师

根据用户故事“订单超时取消”，生成 12 条测试路径，是否需要转化为自动化脚本？



购买咨询



联系方式

私有化



分析

程序

分支

预约演示



李静  
质量工程师

结合 AI 推荐的高频故障  
场景，补充 5 条边界测  
试用例，今晚执行全量  
回归。

众多知名企业及开源组织已在使用



## 5.1 远程仓库初始化—— gitee 账户的创建

**gitee**

### 企业级 DevOps 研发管理平台

阴明 掘金创始人

许多年轻的开发者使用 Gitee 共享和协作，Gitee 也是难得的为中国开发者的软件服务。期待未来 Gitee 继续发展，成为中国开发行业发展之基石！

Gitee <sup>hot</sup> 企业版 - 企业级 DevOps 研发管理平台 >

### 注册

已有帐号? [点此登录](#)

姓名

https://gitee.com/ 个人空间地址 AI ?

请输入手机号码

密码不少于8位

我已阅读并同意 [使用条款](#) 及 [非活跃帐号处理规范](#)

**立即注册**

 使用 OSChina 帐号登录

其他方式登录





# 5.1 远程仓库初始化—— gitee 工程的创建

我的工作台

仓库 7

- Charon\_az/KnowledgeOunce
- Charon\_az/2024ustc-jianmu-co...
- Charon\_az/ustc-compiler-princ...
- Charon\_az/Tidal
- 吴炳伦/unity\_game

查看全部

Pull Request 46

Issues

代码片段

我的星选集 1

我的企业/高校/组织

最近参与

全部 Issues Pull Request

## 动态

- 2026-01-05
- wkjokadj**
- 删除** 了分支 2 个月前
- Charon\_az/KnowledgeOunce  
wjh\_develop\_20260105\_2  
by Pull Request: #46 modify localhost
- 接受了 Pull Request 2 个月前
- #46 modify localhost**  
Charon\_az/KnowledgeOunce Charon\_az:wjh\_develop\_2026... → Charon\_az:develop  
审查: +2 测试: +2
- 接受了 Pull Request 2 个月前
- #46 modify localhost**  
Charon\_az/KnowledgeOunce Charon\_az:wjh\_develop\_2026... → Charon\_az:develop  
审查: +2 测试: +2
- 推送到了分支 2 个月前

所有动态

- + 新建仓库**
- 创建组织
- 开通企业版
- 从 GitHub / GitLab 导入仓库
- 发布代码片段

- 木香丘** 暂无简介 [关注](#)
- 商城** 开源商城。启山智软商城采用... [关注](#)
- dgiiot** 暂无简介 [关注](#)

- 推荐仓库** [换一批](#)
- flutter\_luckin\_coffee** ☆ 1808  
flutter luckin coffee application (仿瑞幸咖啡)
  - tinyriscv** ☆ 3432  
一个从零开始写的极简、非常易懂的RISC-V
  - LuatOS** **GVP** ☆ 1824  
Powerful embedded Lua Engine for IoT devices, ...
  - axmol** ☆ 54  
A Multi-platform Engine for Desktop, XBOX (U...
  - geektime-Rust** ☆ 225



# 5.1 远程仓库初始化—— gitee 工程的创建



## 新建仓库

在其他网站已经有仓库了吗? [点击导入](#)

仓库名称 \* ✓

测试仓库

AI

归属

Charon\_az

路径 \* ✓

test-repository

AI

仓库地址: [https://gitee.com/charon\\_az/test-repository](https://gitee.com/charon_az/test-repository)

仓库介绍

0/200

AI

用简短的语言来描述一下吧

开源 (所有人可见) ?

私有 (仅仓库成员可见)

初始化仓库 (设置语言、.gitignore、开源许可证)

设置模板 (添加 README、Issue、Pull Request 模板文件)

README 文件

Issue 模板文件 ?

Pull Request 模板文件 ?

选择分支模型 (仓库创建后将根据所选模型创建分支)

创建





## 5.2 本地仓库初始化——初始配置

### 名称和邮箱

```
$ git config --global user.name "Jon Doe"
```

注册/绑定 Gitee 的邮箱地址，务必与你本地配置的 Git 邮箱保持一致：

```
$ git config --global user.email "email@example.com"
```



# 5.2 本地仓库初始化——将 gitee 代码拷贝到本地

## Charon\_az/测试仓库

Watching 1 Star 0 Fork 0

代码 Issues 0 Pull Requests 0 Wiki 统计 流水线 服务 管理

master 分支 1 标签 0

克隆/下载

Charon_az	Initial commit	16028cf	刚刚	1 次提交
.gitee	Initial commit		刚刚	
README.en.md	Initial commit		刚刚	
README.md	Initial commit		刚刚	

### 简介

暂无描述

暂无标签

README

0 Stars

1 Watching

0 Forks

### 发行版

暂无发行版, [创建](#)

### 贡献者 (1)

全部



### 近期动态



不到1分钟前推送了新的 master 分支



不到1分钟前创建了仓库

### README

#### 测试仓库

- 介绍
- 软件架构
- 安装教程
- 使用说明
- 参与贡献
- 特技

## 测试仓库

### 介绍

{以下是 Gitee 平台说明, 您可以替换此简介} Gitee 是 OSCHINA 推出的基于 Git 的代码托管平台 (同时支持 SVN)。专为开发者提供稳定、高效、安全的云端软件开发协作平台 无论是个人、团队、或是企业, 都能够用 Gitee 实现代码托管、项目管理、协作开发。企业项目请看 <https://gitee.com/enterprises>

### 软件架构

软件架构说明

### 安装教程

1. xxxx



## 5.2 本地仓库初始化——将 gitee 代码拷贝到本地

克隆/下载



**HTTPS** SSH SVN SVN+SSH

下载ZIP

`https://gitee.com/charon_az/test-repository.git`



**提示**

下载代码请复制以下命令到终端执行

```
git clone https://gitee.com/charon_az/test-repository.git
```



为确保你提交的代码身份被 Gitee 正确识别，请执行以下命令完成配置

```
git config --global user.name 'Charon_az'  
git config --global user.email 'getchar404notfound@gmail.com'
```





## 5.2 本地仓库初始化——将 gitee 代码拷贝到本地

### 获取仓库

```
$ git clone https://example.com/example.git
```



## 5.2 本地仓库初始化

日常开发中的另一种开始方式

### 在当前目录初始化新仓库

```
$ cd my_project  
$ git init
```



## 5.3 分支——分支的分类

- 主分支 (main/master) : 用于发布稳定版本, 通常不允许直接提交。
- 开发分支 (develop) : 用于集成各个功能分支的代码。
- 功能分支 (feature branches) : 用于开发新功能, 通常从 develop 分支创建, 完成后合并回 develop 分支。
- 发布分支 (release branches) : 用于准备发布版本, 通常从 develop 分支创建, 完成后合并回 main 和 develop 分支。



## 5.3 分支——分支的查看

### 分支的查看

查看本地分支

```
$ git branch
```

查看远程分支

```
$ git branch -r
```

查看全部分支

```
$ git branch -a
```



## 5.3 分支——分支的创建和切换

### 分支的创建和切换

使用 如下命令可以创建一个新的分支并立即切换到该分支

```
$ git checkout -b new-feature
```



## 5.3 分支——分支的合并

### 分支的合并

切换到目标分支：例如要将 new-feature 分支合并到 main 分支，先切换到 main 分支

```
$ git checkout main
```

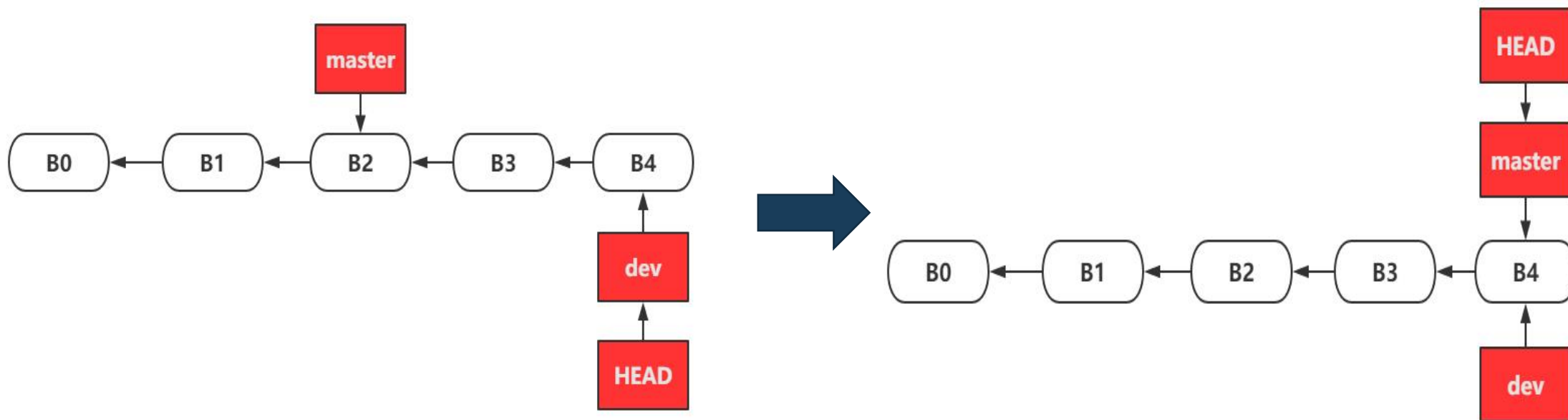
执行合并操作：将指定分支合并到当前分支

```
$ git merge new-feature
```



## 5.3 分支——git merge的三种情形

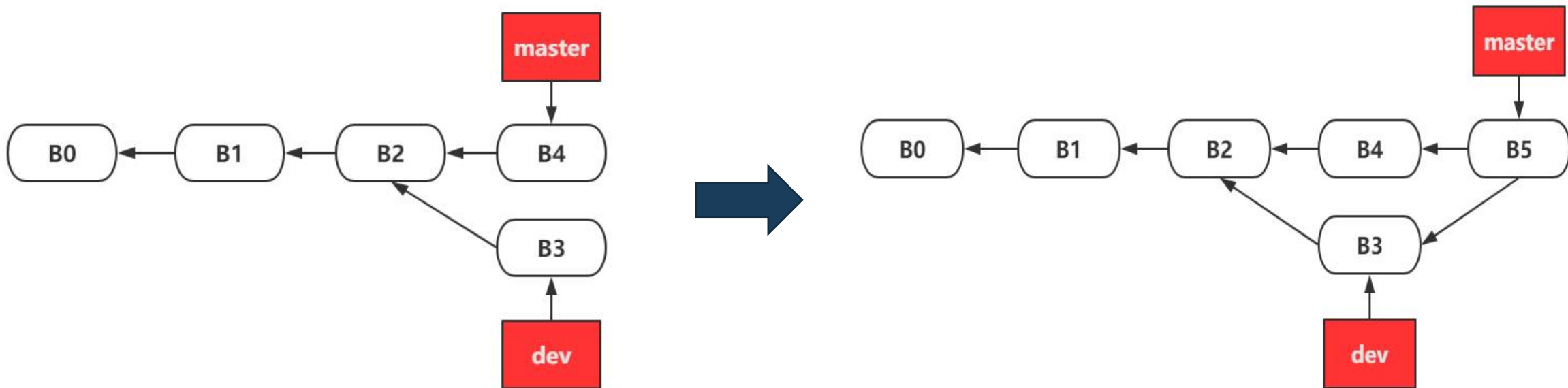
### 5.4.1 快进



在B2处新建了分支dev并提交了两个版本，将master和dev合并，只需要将master指针向前移动。这种情况下的合并操作没有需要解决的分歧。

## 5.3 分支——git merge的三种情形

### 5.4.2 非快进，但无冲突



当在新分支 dev 进行了一次提交B3，再回到分支 master 又进行一次提交 B4，两者修改的不同文件或修改的相同文件的不同部分，即不存在冲突。

则合并master和dev,Git 会使用两个分支的末端所指的快照（B3 和 B4）以及这两个分支的公共祖先（B2），做一个简单的**三方合并**。注意这里合并后 master 自动 commit 提交了一次，产生了**提交B5**。而B5中的结果是三方合并的结果。



## 5.3 分支——git merge的三种情形

### 5.4.2 非快进，但有冲突

当在新分支 dev 进行了一次提交B3，再回到分支 master 又进行一次提交 B4，两者修改的部分存在冲突。

此时 Git 做了合并，但是没有自动地创建一个新的合并提交。Git 会暂停下来，等待你去解决合并产生的冲突。

#### 查看分支状态

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
```

```
    both modified:   test-1.txt
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```



## 5.3 分支——git merge的三种情形

### 5.4.2 非快进，但有冲突

Git 会在有冲突的文件中加入标准的冲突解决标记，你可以打开这些包含冲突的文件然后手动解决冲突

手动解决冲突之后，手动提交

```
1 This is test-1.  
2 update test-1.  
3 add test-1.  
4 <<<<<< HEAD  
5 test master.  
6 =====  
7 test dev.  
8 >>>>>> dev
```



```
1 This is test-1.  
2 update test-1.  
3 add test-1.  
4 test master.  
5 test dev.
```



```
1 $ git add .  
2 $ git commit -m "connection"  
3 [master f7daa6b] connection
```



## 5.3 分支——分支的删除

### 分支的删除

删除本地分支：

```
$ git branch -d new-feature
```

删除远程分支

```
$ git push origin --delete new-feature
```



## 5.4 本地工作流程

假设我们修改了readme.txt

### 提交修改

```
$ git add readme.txt  
$ git commit -m "update readme"
```



## 5.4 本地工作流程

我们通常需要在提交前查看仓库的状态

### 查看状态

```
$ git status
On branch main
Changes to be committed:
.....
Untracked files:
.....
```



## 5.4 本地工作流程

我们有时需要进一步查看即将提交的修改内容

### 对比内容

```
$ git diff readme.txt
```



## 5.4 本地工作流程

我们有时需要查看提交的历史记录

### 提交历史

```
$ git log
```



## 5.4 本地工作流程

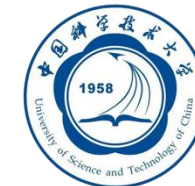
我们有时需要撤销未提交的修改

### 撤销未提交修改

```
$ git checkout -- readme.txt
```

2.23.0 以上版本的 git 可以使用

```
$ git restore readme.txt
```



## 5.4 本地工作流程

我们有时需要拉取最新的更改

### 与远程仓库同步

如果当前分支为想要同步（更新）的分支，可以直接执行：

```
$ git pull
```

将远程指定分支拉取到本地当前分支上：

```
$ git pull origin <远程分支名>
```

将远程指定分支拉取到本地指定分支上：

```
$ git pull origin <远程分支名>:<本地分支名>
```



## 5.4 本地工作流程

我们有时需要将本地的最新代码更新到 gitee 上

### 推送到远程仓库

如果当前分支为想要推送的分支，可以直接执行：

```
$ git push
```

如果本地分支与远程分支同名，可以直接执行：

```
$ git push origin <分支名>
```

当需要将本地分支推送到远程的其他分支时，可使用：

```
$ git push origin <本地分支>:<远程分支>
```

例如：`git push origin master` 用于将本地 **master** 分支的更改推送到远程仓库的 **master** 分支。如果远程分支不存在，Git 会自动创建它



## 5.5 深入学习

以上内容基本足够完成课程实验，但是我们省略了以下 git 特性：

- 暂存区
- 本地提交历史重写

如果想要了解这些特性，推荐阅读：

- Pro Git <sup>3</sup>
- Git User' s Manual <sup>4</sup>

3. <https://git-scm.com/book/en/v2>

4. <https://git-scm.com/docs/user-manual.html>

