Git 零基础入门使用手册

一、Git 是什么？

Git 是一个免费的代码版本管理工具，简单说就是：

- 帮你记录文件的每一次修改（像给文件拍快照）
- 可以随时恢复到之前的版本（后悔药功能）
- 多人协作时不会互相覆盖代码（团队合作必备）
- 核心概念解释：

1. 仓库（Repository）——你的“项目文件夹”

想象你写软件代码，所有代码放在一个文件夹里。

这个文件夹就是 仓库。

它分两种：

本地仓库：在你电脑上的文件夹。

远程仓库：放在网上的（比如 GitHub），方便和别人共享。

✓ 注：用 git init 指令可以把普通文件夹变成 Git 仓库。

---

2. 工作区（Working Directory）——你正在写的草稿

这就是你打开文件、修改内容的地方。

比如你改了某个功能，或者修复了某个问题，但还没决定要不要保留这个改动——这些都叫“工作区的修改”。

---

3. 暂存区（Staging Area）——准备提交的“购物车”

你想提交哪些改动？不是所有修改都要一次提交。

暂存区就像购物车：把想保存的改动“放进购物车”。

然后才统一结账（提交）。

4. 提交（Commit）

当你对购物车里的内容满意了，就点“提交”。

给你这次改动起个名字（提交信息），比如：“添加 xxx 功能”。

---

5. 分支（Branch）——在平行宇宙写代码

主干叫 main（或 master）。一般这个 main 是总是可以正常运行的

如果你想要加入新功能，但是担心一加入这个功能整个软件就没法正常运行了，那就创建一个分支

分支可以理解成是主干的一个副本，一个复制粘贴的文件夹

在分支里随便改，提交新的功能，不会影响主干

---

6. HEAD ——你现在站在哪条时间线上？

HEAD 就是你当前所在的“位置”。

默认在 main 分支的最新提交上。

切换分支或回退历史时，HEAD 会跟着动。

你可以把它想象成“光标”，告诉你现在在哪。

---

7. 合并（Merge）——把支线剧情加回主线

你发现你的分支可以正常运行，那么就向管理员申请把你的分支合并到主干

---

8. 远程仓库（Remote）——云端备份 & 团队协作

你写的代码上传到 GitHub/Gitcode，这就是远程仓库。

别人也能下载、修改、再传回来。

---

9. 冲突（Conflict）——两个人同时改了同一段

比如你和朋友同时修改了第 5 章开头。

Git 不知道该听谁的，就会标记“冲突”。

你需要打开文件，手动选择保留哪一段，然后重新提交。

✅ 冲突不可怕，是协作的正常现象！

二、安装 Git

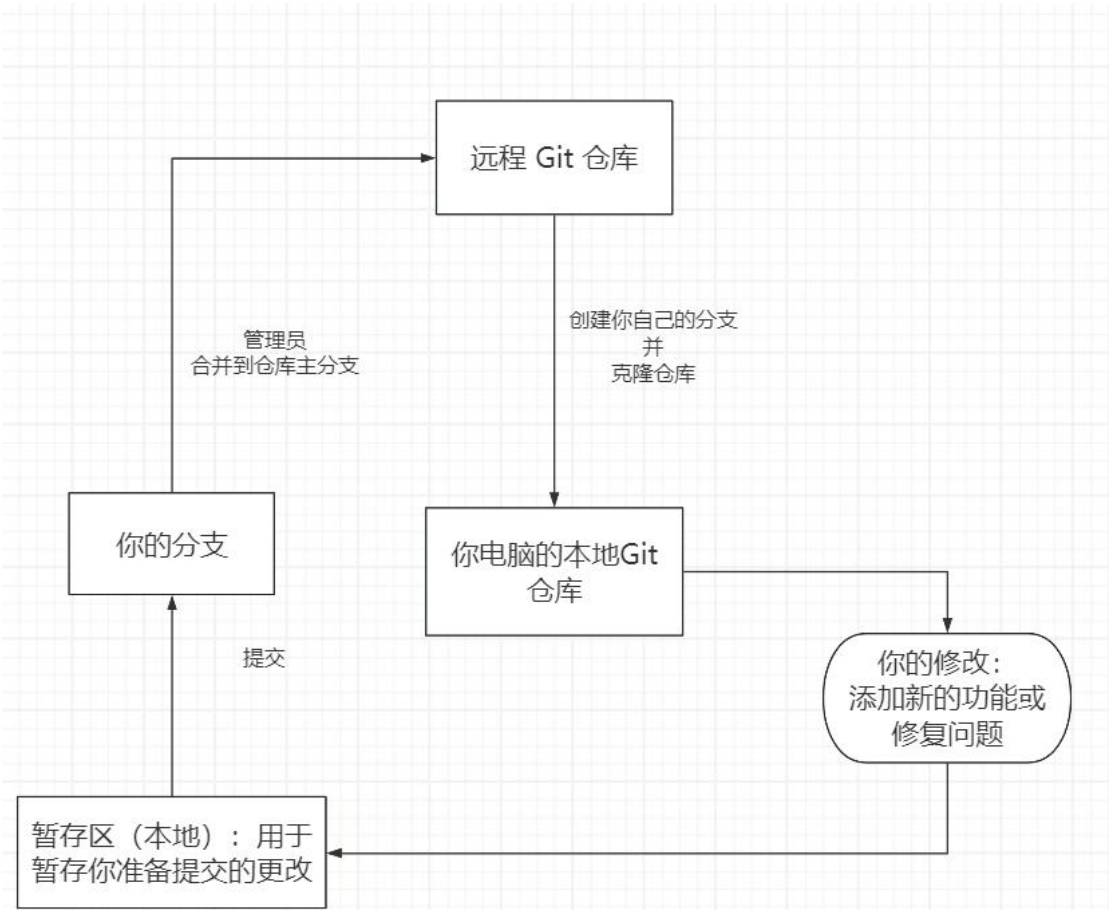
1. **Windows 用户：**
   - 访问 git-scm.com/download/win
   - 下载后双击安装，所有选项保持默认，一路点击"Next"即可
2. **Mac 用户：**
   - 打开终端，输入 git --version
   - 如果没有安装，会自动提示安装命令，按提示操作
3. **验证安装：**
   - 安装完成后，打开终端（Windows 用 Git Bash）
   - 输入 git --version，显示版本号即安装成功

三、基础配置（只需做一次）

打开终端，输入以下命令（替换成你的信息）：
git config --global user.name "你的名字" # 例如：git config --global user.name "张三"
git config --global user.email "你的邮箱" # 例如：git config --global user.email "zhangsan@example.com"

四、核心概念图解

工作区（你正在编辑的文件）→ 暂存区（临时存放修改）→ 本地仓库（保存历史版本）→ 远程仓库（如 GitHub，多人共享）



五、常用操作步骤

1. 创建/初始化仓库

场景：新建项目时使用
方法 1：新建文件夹并初始化
mkdir 项目名称 # 创建文件夹（例如 mkdir myproject）
cd 项目名称 # 进入文件夹
git init # 初始化 Git 仓库（会生成 .git 隐藏文件夹）
方法 2：克隆远程仓库（下载别人的项目）
git clone 仓库地址 # 例如：git clone https://github.com/xxx/xxx.git

2. 日常开发流程（最重要！）

1. 查看文件状态（每次修改后建议先看这个）
git status # 红色文字：未暂存的修改；绿色文字：已暂存的修改
2. 将修改添加到暂存区
git add 文件名 # 添加单个文件，例如：git add index.html
git add . # 添加所有修改（最常用，注意有个点）

3. 将暂存区内容提交到本地仓库
git commit -m "这里写修改说明" # 例如：git commit -m "修复登录按钮 bug"
4. （多人协作时）拉取远程最新代码（避免冲突）
git pull # 拉取并合并远程仓库的更新
5. 将本地提交推送到远程仓库
git push # 推送到远程（第一次可能需要输入账号密码或者 Access Token）

3. 查看历史记录

git log # 查看详细历史，按 q 键退出
git log --oneline # 简洁显示历史（只显示 commit id 和说明）

4. 版本回退（后悔药）

方法 1：回退到上一个版本
git reset --hard HEAD~1
方法 2：回退到指定版本（需要 commit id，从 git log 中获取）
git reset --hard 提交 ID # 例如：git reset --hard a1b2c3d

5. 分支操作（多人协作必备）

查看所有分支
git branch # 当前分支前会有 * 号
创建并切换到新分支（开发新功能时用）
git checkout -b 分支名 # 例如：git checkout -b feature/login
切换分支
git checkout 分支名 # 例如：git checkout main
合并分支（功能开发完成后）
git checkout 主分支 # 先切换到主分支，例如：git checkout main
git merge 分支名 # 合并开发分支，例如：git merge feature/login
删除分支（合并后可删除）
git branch -d 分支名 # 例如：git branch -d feature/login

六、解决冲突（多人协作常见）

当多人修改同一文件同一位置时会产生冲突，解决步骤：

1. 执行 git pull 时提示冲突
2. 打开冲突文件，寻找类似以下标记的内容：<<<<<<< HEAD

   你的修改内容

=======

别人的修改内容

>>>>>>> 分支名

3. 手动编辑保留正确内容，删除冲突标记（<<<<<<<, =======, >>>>>>>）
4. 重新执行 git add .和 git commit -m "解决冲突"

七、实用技巧

- **忽略文件**：在项目根目录创建.gitignore 文件，写入不需要跟踪的文件，例如：node_modules/ # 依赖文件夹

  .env # 环境配置文件

  *.log # 日志文件

- **撤销修改**：git checkout -- 文件名 # 撤销工作区的修改（未 add 前有效）

  git reset HEAD 文件名 # 从暂存区撤销到工作区

- **设置快捷命令**：git config --global alias.st status # 以后可用 git st 代替 git status

  git config --global alias.co checkout # git co 代替 git checkout

八、常见问题解决

1. **忘记 commit 就切换分支**：先 git commit 提交，再切换
2. **提交信息写错**：git commit --amend 可修改最后一次提交信息
3. **push 失败**：先 git pull 拉取最新代码，解决冲突后再 push

九、学习资源

- 官方文档：git-scm.com/book/zh/v2
- 图形化工具：推荐 GitKraken（新手友好）或 SourceTree

提示：刚开始可能记不住命令，建议把常用命令抄在纸上，操作几次就会了！遇到问题先看 git status 的提示，大部分问题都能解决。

附录：核心使用指令示例

```
# 1. 首次获取项目（只需做一次）
git clone https://github.com/xxx/project.git
cd project

# 2. 确保本地 main 分支是最新的（重要！）
git checkout main
git pull origin main        # 拉取远程最新代码

# 3. 基于最新的 main 创建自己的功能/修复分支
git checkout -b fix/login-error

# 4. 修改代码……（比如修了一个登录 bug）

# 5. 查看改了哪些文件
git status

# 6. 暂存改动
git add src/auth.js         # 或 git add .（谨慎使用）

# 7. 提交改动（写清楚做了什么）
git commit -m "修复登录接口返回 500 的问题"

# 8. 推送到远程（创建同名远程分支）
git push origin fix/login-error
```

一些 AI 托管平台的指令有所不同：

```
# 1. 进入项目目录（如果还没初始化 Git）
cd my-project

# 2. 初始化本地 Git 仓库（如果还没有 .git）
git init

# 3. 添加远程仓库地址（只需做一次）
#    把下面的 URL 换成你在 GitHub / GitLab / Hugging Face Spaces 上的仓库地址
git remote add origin https://github.com/your-username/my-project.git

# 4. （可选但推荐）先拉取远程已有内容（比如 README、配置文件等）
#    注意：如果是全新空仓库，这一步可能报错，可以跳过
git pull origin main        # 或 master，取决于默认分支名

#    如果远程是空的，你可以先创建初始提交：
echo "# My Project" > README.md
git add .
git commit -m "初始提交"

# 5. 确保你在 main 分支，并基于最新代码开始工作
git checkout -b main        # 如果刚 init，默认不在任何分支，需显式创建
# 或者如果已存在 main 分支：
git checkout main

# 6. 创建你的功能分支
git checkout -b feat/user-login

# 7. 修改代码……

# 8. 暂存并提交
git add .
git commit -m "添加用户登录功能"

# 9. 推送到远程（首次推送需指定上游分支）
git push -u origin feat/user-login
```