GitHub 代码提交指南

一、前置准备（仅需做一次）

1. 注册 GitHub 账号

打开 GitHub 官网，注册账号（记住用户名和绑定邮箱）。

2. 安装 Git 并配置

- 下载 Git：Git 官网（下载后默认安装即可）；
- 配置 Git（和 GitHub 账号绑定）：

```
\# 设置用户名（必须和 GitHub 用户名一致）

git config --global user.name "你的GitHub用户名"

\# 设置邮箱（必须和 GitHub 绑定的邮箱一致）

git config --global user.email "你的GitHub绑定邮箱"
```

1. 打开电脑终端（Windows 用「命令提示符」或「PowerShell」，Mac 用「终端」）；
2. 复制以下命令，替换成你的 GitHub 用户名和邮箱，逐行执行：

3. 接受仓库邀请

- 仓库管理员会给你发邮件邀请，点击邮件里的「Accept invitation」；
- 接受后，在 GitHub 首页就能看到共享仓库了。

二、核心操作流程（每天都要用）

第一步：拉取最新代码（重中之重！修改前必做）

每次开始写代码前，一定要先拉取远程仓库的最新代码，避免和队友的代码冲突！

1. 打开终端，进入你想存放代码的文件夹（比如桌面的「项目文件夹」）：

```
\# 示例：进入桌面的项目文件夹（Windows 和 Mac 通用，替换成你的文件夹路径）

cd Desktop/项目文件夹
```

1. 第一次拉取：克隆仓库到本地（仅需做一次）

```
\# 复制仓库的 HTTPS 链接，替换下面的地址

git clone https://github.com/仓库管理员用户名/仓库名.git
```

- 如何复制仓库链接？打开 GitHub 仓库主页 → 点击绿色「→ 复制 HTTPS 链接；

1. 后续拉取：更新本地代码（每天写代码前都要做）

```
\# 先进入仓库文件夹（如果已经在文件夹里，可跳过这步）

cd 仓库名

\# 拉取最新代码（直接复制执行，不用改）

git pull origin main
```

- 执行成功后，本地代码就和 GitHub 上的最新代码同步了。

第二步：本地修改代码（写代码环节）

在本地仓库文件夹里，直接修改、新增、删除文件（比如写 `main.py`、改 `index.html` 等），和平时在电脑上操作文件一样。

第三步：提交修改到本地 Git（告诉 Git 你改了什么）

代码写好后，把修改「告诉」Git，准备推送到 GitHub：

1. 查看修改状态（可选，帮你确认改了哪些文件）：

```
git status
```

- 终端会显示红色的修改文件（未暂存），绿色的是已暂存文件；

1. 暂存所有修改（把修改「打包」，准备提交）：

```
\#  "." 表示暂存所有修改的文件（推荐新手用这个，不用一个个选文件）

git add .

\# 如果只想暂存某个文件，比如只暂存 main.py，可替换成：git add main.py
```

1. 提交修改（写清楚你改了什么，方便队友查看）：

```
\# 把引号里的文字换成你的修改说明，必须写！
```

```
git commit -m "新增登录功能"
```

- 说明要简单明了，比如：「修复计算错误」「新增用户注册页面」「修改按钮样式」。

第四步：推送代码到 GitHub（共享给队友）

提交到本地后，把修改推送到 GitHub，队友就能看到你的代码了：

```
\# 直接复制执行，不用改（推送到 main 分支）

git push origin main
```

- 第一次推送可能会弹出登录窗口，输入你的 GitHub 用户名和密码（如果提示密码无效，用「个人访问令牌」登录，下文有教程）；
- 执行成功后，打开 GitHub 仓库主页，就能看到你的修改了！

三、常见问题解决（萌新必看）

问题 1：推送时提示「密码无效」（GitHub 不再支持密码登录）

解决方案：用「个人访问令牌」代替密码登录

1. 生成令牌：

- 打开 GitHub → 点击右上角头像 → 「Settings」→ 左侧「Developer settings」→ 「Personal access tokens」→ 「Generate a token」；
- 填写备注（比如「团队协作令牌」）→ 勾选「repo」相关权限（全选 repo 下的选项）→ 拉到最下面点击「Generate token」；
- 生成后，**立即复制令牌**（只显示一次，刷新页面就没了，保存到记事本里）；

1. 推送时，用户名输入 GitHub 用户名，密码粘贴刚才的令牌，就能登录成功。

问题 2：拉取 / 推送时提示「冲突」（和队友改了同一个文件的同一行）

1. 终端会提示「Automatic merge failed」，并告诉你哪个文件冲突了（比如 `main.py`）；
2. 打开冲突文件，会看到类似这样的标记：

```
<<<<<<你本地写的代码

print("我的代码")

\=======

\# 队友写的代码

print("队友的代码")
```

```
\>>>>>>> 一串字符
```

1. 手动修改文件：删除冲突标记（`<<<<<<`、`=======`、`>>>>>>>`），保留正确的代码（可以和队友商量后修改）；
2. 修改后，重新执行「提交→推送」步骤：

```
git add .

git commit -m "解决代码冲突"

git push origin main
```

问题 3：忘记拉取就推送，提示「failed to push some refs」

解决方案：先拉取，再推送

```
\# 先拉取最新代码（会自动尝试合并）

git pull origin main

\# 如果出现冲突，按上面的「冲突解决」步骤处理

\# 冲突解决后，再推送

git push origin main
```

四、新手避坑指南（一定要看！）

1. 「拉取代码」是第一步！每次写代码前、推送代码前，都要执行 `git pull origin main`；
2. 不要直接在 GitHub 网页上修改代码（除非紧急情况），所有修改都在本地做，然后通过「拉取 - 提交 - 推送」流程；
3. 每次提交只改一个小功能 / 一个小修复，比如「新增登录按钮」「修复注册接口 bug」，不要一次改一大堆；
4. 提交说明要写清楚，比如「新增用户登录接口」，不要写「修改了文件」「更新代码」这种没用的话；
5. 遇到报错不要慌，把终端的报错信息复制给仓库管理员，或者百度报错关键词，大部分问题都有现成解决方案。

五、常用命令总结（直接复制用）

命令作用	命令代码
克隆仓库（第一次拉取）	git clone 仓库 HTTPS 链接

命令作用	命令代码
拉取最新代码	git pull origin main
查看修改状态	git status
暂存所有修改	git add .
提交修改（写说明）	git commit -m "修改说明"
推送代码到 GitHub	git push origin main