

Git 入门学习指南 —— Japluto

这份指南根据我与Gemini之间的对话整理而成，从基础概念出发，重点解决在使用过程过程中遇到的真实问题。希望对大家有所帮助！

• Git 入门学习指南 —— Japluto

- 1. Git 核心概念入门
 - 1.1. Git 是什么？(版本控制)
 - 1.2. Git vs GitHub (工具 vs 网站)
 - 1.3. 核心理念：三个区域
- 2. 本地仓库操作 (从 0 到 1)
 - 2.1. 本地工作流三部曲 (`init` , `add` , `commit`)
 - 2.2. 【Q&A】 为什么我 `add` 了一堆文件夹，`git log` 却只显示一条记录？
 - 2.3. 如何查看 `commit` 到底提交了什么？
- 3. 项目管理与文件整理
 - 3.1. 【Q&A】 仓库根目录太乱 (很丑)，如何创建文件夹？
 - 3.2. 【方法一：本地】在本地整理，然后 `push` (最佳实践)
 - 3.3. 【方法二：网页】按 `.` 键使用网页版 VS Code
 - 3.4. 【专业建议】使用 `.gitignore` 忽略 "产物" (如 `.ckpt` , `.log`)
- 4. 远程仓库与团队协作 (GitHub)
 - 4.1. 关联远程仓库 (`remote add`)
 - 4.2. 推送与拉取 (`push` , `pull`)
 - 4.3. 【Q&A】 如何将 GitHub 仓库从 Public 改为 Private？
 - 4.4. 【Q&A】 我创建仓库时勾选了 README，本地的 `git init` 仓库怎么办？
- 5. 必备技能：分支 (Branch)
 - 5.1. 为什么需要分支？
 - 5.2. 核心操作 (`switch -c` , `merge`)
- 6. 常见“陷阱”与问题排查 (Troubleshooting)
 - 6.1. 【问题】 `git add` 报 LF will be replaced by CRLF 警告
 - 6.2. 【问题】 `cd` 文件夹失败，No such file or directory (中文路径/斜杠问题)
 - 6.3. 【问题】 `git add` 报 warning: adding embedded git repository 警告 (仓库“套娃”)
 - 6.4. 【问题】 如何把“套娃”仓库分离成两个独立仓库？
 - 6.5. 【问题】 `git push` 失败，Failed to connect to github.com port 443
- 7. VS Code 与 Git 协同开发

- 7.1. VS Code 如何识别 Git 仓库
 - 7.2. 核心界面：源代码管理面板
 - 7.3. 可视化日常流程 (`add` , `commit` , `push`)
 - 7.4. 可视化分支管理
 - 7.5. 杀手锏：查看差异 (Diff)
 - 8. GitLens 插件使用详解
 - 8.1. GitLens 侧栏 (仓库仪表盘)
 - 8.2. GitLens Inspect 侧栏 (历史调查器)
 - 8.3. 核心精髓：“隐形”能力 (代码追溯)
 - 9. Git 常用操作指令速查表
-

1. Git 核心概念入门

1.1. Git 是什么？(版本控制)

Git 是一个“版本控制系统”(VCS)，一个“超级存档神器”。它帮你记录项目文件的 每一次 修改（称为“快照”），你可以随时“穿越”回任何一个历史版本，而不用手动复制 `v1` , `v2` , 最终版 这样的文件。

1.2. Git vs GitHub (工具 vs 网站)

- **Git**： 是一个**工具/软件**，安装在你本地电脑上，帮你管理本地文件。
- **GitHub**： 是一个**网站/服务**，一个“云端仓库托管平台”。

比喻： Git 是 Word 软件，GitHub 是腾讯文档。你用 Git (Word) 在本地写好，然后 `push` (上传) 到 GitHub (腾讯文档) 来备份、分享和协作。

1.3. 核心理念：三个区域

这是 Git 最核心理念，你的所有操作都围绕这三个区域进行：

1. **工作区 (Working Directory)**： 你在电脑上能看到的、正在编辑的项目文件夹。
2. **暂存区 (Staging Area)**： 一个临时的“待提交”区域。像一个公文包，你把修改好的文件 `add` 进来，准备一次性提交。
3. **本地仓库 (Local Repository)**： Git 在你本地的“历史档案馆”(`.git` 文件夹)，存放你所有 `commit` 过的历史版本。

2. 本地仓库操作 (从 0 到 1)

2.1. 本地工作流三部曲 (`init` , `add` , `commit`)

1. 初始化 (只做一次): `git init`

- 在你的项目文件夹 (如 `F:/code`) 里运行, 告诉 Git: “从现在开始, 请你管理这个文件夹”。

2. 添加 (`add`): `git add .`

- 把**工作区**中 *所有* 的修改 (新增、改动) 都放进**暂存区** (公文包)。

3. 提交 (`commit`): `git commit -m "你的提交信息"`

- 把**暂存区** (公文包) 里的所有内容, 打上一个标签 (`-m` 后面的信息), 正式存入**本地仓库** (档案馆)。

2.2. 【Q&A】 为什么我 `add` 了一堆文件夹, `git log` 却只显示一条记录?

回答: 这是 100% 正确的。

`git log` 显示的是你的**“提交记录” (Commit)**，而不是你提交的“文件列表”。

你把“一大堆文件夹” `add` 进暂存区, 然后用**一次** `commit` 命令, 把它们**打包**成了一个“快照”存入仓库。因此, `git log` 里只显示**一条**记录, 即你这一次“打包”的动作。

2.3. 如何查看 `commit` 到底提交了什么?

- `git log --stat`: 查看 `log`, 并**统计**每次 `commit` 改动了哪些文件。
- `git show`: 显示**最新一次** `commit` 的**所有**详细改动 (包括代码)。
- `git show --name-only`: 只看最新一次 `commit` 改动了哪些**文件名**。

3. 项目管理与文件整理

3.1. 【Q&A】 仓库根目录太乱 (很丑), 如何创建文件夹?

回答: Git 只跟踪文件, 不跟踪空文件夹。

你不能“创建”一个空文件夹再 `add`。你必须在本地**创建文件夹**, 并把**文件移动**进去, 然后 Git 才能识别到这个“移动”操作。

3.2. 【方法一: 本地】在本地整理, 然后 `push` (最佳实践)

1. 在你的**本地电脑** (如 `F:/code`) 里, 用 Windows 资源管理器 (鼠标拖拽) 创建新文件夹 (如 `code/`, `data/`, `reports/`)。
2. 把所有文件拖拽到对应的文件夹里。
3. 回到 **Git Bash** 终端。
4. 运行 `git status`, 你会看到一堆“已删除”(旧路径) 和“未跟踪”(新路径) 的文件。
5. 运行 `git add .`, Git 会自动理解你是在“移动”文件。
6. 运行 `git commit -m "Organize project structure"`
7. 运行 `git push` 推送到 GitHub, 你的 GitHub 页面就会焕然一新。

3.3. 【方法二: 网页】按 `.` 键使用网页版 VS Code

1. 在你的 GitHub 仓库主页, 按一下键盘上的 `.` (英文句号) 键。
2. GitHub 会在浏览器里打开一个 VS Code 界面。
3. 在左侧的文件列表里, 你可以像在本地一样, 用鼠标**拖拽文件**到你**新建的文件夹**里。
4. 整理完毕后, 点击左侧的 **Git 图标**。
5. 在“消息”框里输入提交信息 (如 "Organize files")。
6. 点击蓝色的 **"提交并推送" (Commit & Push)** 按钮。

3.4. 【专业建议】使用 `.gitignore` 忽略 "产物" (如 `.ckpt`, `.log`)

像 `.ckpt` (模型权重) 或 `.log` (日志) 这种由代码**生成**的、体积**巨大**的文件, **不应该**被上传到 Git 仓库。

1. 在项目**根目录** (`F:/code`) 下, 创建一个名为 `.gitignore` 的文件。
2. 用记事本打开, 写入你要忽略的文件类型, 例如:

```
# 忽略所有 .log 文件
*.log

# 忽略所有 .ckpt 文件
*.ckpt
```

3. 保存文件, 然后 `git add .gitignore` 并 `commit` 它。

4. 从此以后，Git 会自动忽略所有这些文件，`git status` 会保持得很干净。

4. 远程仓库与团队协作 (GitHub)

4.1. 关联远程仓库 (`remote add`)

当你本地用 `git init` 创建了仓库，你需要把它和 GitHub 上的空仓库关联起来。

```
# 'origin' 是你给远程仓库起的一个别名，'https://...' 是它的地址
git remote add origin https://github.com/YourName/my-project.git
```

4.2. 推送与拉取 (`push` , `pull`)

- `git push` : 把**本地仓库**（档案馆）里的新提交，推送到**远程仓库**（GitHub）。
- `git pull` : 从**远程仓库**（GitHub）拉取最新的提交，并与你的**本地仓库**合并。

4.3. 【Q&A】 如何将 GitHub 仓库从 Public 改为 Private?

回答：在 GitHub 仓库页面 -> `Settings` -> 滚到底部 `Danger Zone` ->

`Change repository visibility`。

【严重警告】

将 Public 仓库转为 Private，会导致你**永久丢失**所有的 **Stars** (点赞) 和 **Watchers** (关注者)。
即使你未来再改回 Public，这些 Stars 也**不会**回来。

4.4. 【Q&A】 我创建仓库时勾选了 README，本地的 `git init` 仓库怎么办？

回答：你遇到了两种不同工作流的冲突。

1. **本地优先流**：你本地 `git init`，这时你需要一个**空**的远程仓库，才能 `push`。
2. **远程优先流**：你远程创建了带 README 的仓库，这时它已经有了自己的“历史”。

你遇到的问题：本地有一套历史，远程也有一套历史，它们“互不相干”(unrelated histories)，Git 拒绝合并。

你的解决方案：

你后来使用了“方法二（网页版 VS Code）”，你所有的文件整理**都是在远程完成的**。

正确的做法是：

你本地的 `F:/code` 文件夹现在是**过时的**。你应该**删掉**它，然后使用 `git clone` 把 GitHub 上那个**最新、最正确**的仓库重新下载到本地。

```
# 1. 删掉本地 F:/code 文件夹
# 2. cd 到 F:/
# 3. 重新克隆
git clone https://github.com/Japluto/HITSZ-machine-learning-course-work.git
```

以后你就在这个新克隆下来的文件夹里工作。

5. 必备技能：分支 (Branch)

5.1. 为什么需要分支？

分支就是“平行宇宙”。当你想开发一个新功能，或者修复一个 Bug 时，你应该创建一个新分支，在这个“平行宇宙”里尽情修改。

- 改好了，就把分支**合并 (Merge)** 回主分支。
- 改砸了，就直接**丢弃**这个分支。

你的主分支 `main` 永远是干净、可用的。

5.2. 核心操作 (`switch -c` , `merge`)

```
# 1. 创建一个叫 'new-feature' 的新分支，并立刻切换过去
git switch -c new-feature

# 2. (在新分支上修改、add、commit...)

# 3. 开发完成，切回主分支
git switch main

# 4. 把 'new-feature' 分支的修改合并到 'main' 分支
git merge new-feature

# 5. 删除已经没用的分支
git branch -d new-feature
```

6. 常见“陷阱”与问题排查 (Troubleshooting)

这是学习中最有价值的部分。

6.1. 【问题】 `git add` 报 `LF will be replaced by CRLF` 警告

- **原因：** 换行符不统一。Windows 用 `CRLF` (`\r\n`)，Linux/Mac 用 `LF` (`\n`)。
- **这是致命错误吗？ 不是。** 这只是一个警告，Git 在告诉你它会自动帮你转换，你的 `add` 已经成功了。
- **一劳永逸的方案：** 在项目根目录创建 `.gitattributes` 文件，内容写 `* text=auto`，然后 `add` 和 `commit` 这个文件。

6.2. 【问题】 `cd` 文件夹失败， `No such file or directory` (中文路径/斜杠问题)

- **原因：** 你在 `MINGW64` (Bash) 环境中，使用了 Windows 的**反斜杠** `\` 作为路径分隔符。
- **解决方案：**
 - i. Bash 环境中必须使用**正斜杠** `/`。
 - 错误： `cd F:\code\markdown`

- o 正确: `cd F:/code/markdown`
- ii. 命令是区分大小写的 (`cd` 不是 `CD`)。
- iii. 最佳实践: 使用 `Tab` 键自动补全路径, 它会自动处理中文、空格和斜杠。

6.3. 【问题】 `git add` 报 `warning: adding embedded git repository` 警告 (仓库“套娃”)

- 原因: 你的“外层”仓库 (如 `F:/code`) 在 `add` 时, 发现了一个**“内层”文件夹 (如 `.../2026-season...`), 这个内层文件夹它自己也是一个 Git 仓库** (它里面也有个 `.git` 文件夹)。
- Git 的困惑: 外层 Git 不知道是否应该把内层当作一个独立的“子模块”来引用。

6.4. 【问题】 如何把“套娃”仓库分离成两个独立仓库?

- 你的目标: 你不希望“套娃”, 你希望它们是两个独立的项目 (机器学习作业 和 硬件组资料)。
- 解决方案: 这是最专业的做法!
 - i. 去 GitHub 为“硬件组资料”创建第二个新仓库。
 - ii. `cd` 到上一级目录 (如 `F:/`)。
 - iii. `git clone` 这个新仓库到本地 (如 `F:/NGXY-Hardware...`)。
 - iv. 用文件资源管理器 (鼠标) 把“硬件组资料”从 `F:/code/MARKDOWN/...` 剪切 (Cut) 并 粘贴 (Paste) 到 `F:/NGXY-Hardware...` 文件夹里。
 - v. 在新仓库 (`NGXY-Hardware...`) 里: `git add . -> git commit -> git push`。
 - vi. 在旧仓库 (`code`) 里: `git add .` (Git 会检测到文件被删除了) -> `git commit -m "Move hardware files to new repo" -> git push`。

6.5. 【问题】 `git push` 失败, `Failed to connect to github.com port 443`

- 原因: 这是一个纯粹的网络问题, 和 Git 无关。你的电脑无法连接到 GitHub 的服务器。
- 最可能的原因: 你的代理 (VPN) 设置问题。
- 排查步骤:
 - i. 测试浏览器: 浏览器能打开 `github.com` 吗?
 - ii. 浏览器打不开: 检查你的 VPN 是否正常, 或尝试用手机热点。
 - iii. 浏览器能打开: 说明 Git Bash 没有走代理。你需要手动为 Git 配置代理 (`127.0.0.1` 是本地, `7890` 是你的代理端口号):

```
git config --global http.proxy http://127.0.0.1:7890
git config --global https.proxy http://127.0.0.1:7890
```


iv. 取消代理：

```
git config --global --unset http.proxy
git config --global --unset https.proxy
```

7. VS Code 与 Git 协同开发

VS Code 内置了强大的 Git 图形化界面 (GUI)，90% 的日常操作（`add`，`commit`，`push`，`pull`，`branch`）都可以用鼠标完成。

7.1. VS Code 如何识别 Git 仓库



当你用 VS Code “打开文件夹”时，只要这个文件夹是一个 Git 仓库（内含 `.git` 文件夹），VS Code 就会自动识别，并在左侧“活动栏”显示“源代码管理”图标（分叉状）。

7.2. 核心界面：源代码管理面板

点击“源代码管理”图标，你会看到 Git 的主界面：

- **消息框：** 对应 `git commit -m "..."`，在此输入提交信息。
- **✓ 提交 (Commit) 按钮：** 对应 `git commit`，点击后提交**暂存区**的文件。
- **更改 (Changes)：** 对应**工作区**。显示已修改、但**未暂存**的文件。
 - 点击文件名旁的 **+** (加号)，即可将其**暂存**（`git add`）。
- **暂存的更改 (Staged Changes)：** 对应**暂存区**。显示已暂存、准备提交的文件。
 - 点击文件名旁的 **-** (减号)，即可将其**取消暂存**（`git reset HEAD ...`）。

7.3. 可视化日常流程（`add`，`commit`，`push`）

1. **修改文件：** 正常在编辑器里写代码。
2. **暂存 (add)：** 去“源代码管理”面板，在“更改”列表里，点击你想 `add` 的文件旁的 **+** 号。
3. **提交 (commit)：** 文件进入“暂存的更改”后，在**消息框**输入信息，点击 **✓ (提交)** 按钮。
4. **推送 (push)：** 提交后，看 VS Code **左下角**的状态栏，会有一个**“云朵+向上箭头”**  图标。点击它，即可 `git push`。
5. **拉取/同步 (pull)：** 当远程有更新时，左下角会显示**“循环箭头”**  图标。点击它，即可 `git pull`（并 `push` 你本地的提交）。

7.4. 可视化分支管理

- **查看/切换分支：** 点击 VS Code **左下角**的状态栏上的**分支名**（如 `main`），顶部会弹出列表，点击你想切换的分支即可。
- **创建新分支：** 点击状态栏上的分支名，在弹出列表里选择 **+ 创建新分支...**，然后输入新分支名称。

7.5. 杀手锏：查看差异 (Diff)

在“源代码管理”面板的“更改”列表里，**单击**任何一个文件名，VS Code 会自动打开一个**分屏对比**视图，左红（旧）右绿（新），清晰显示你的所有改动。

8. GitLens 插件使用详解

GitLens 是 VS Code 上的“神器”级 Git 扩展。它把你仓库的历史和上下文“注入”到你的编辑器中。

8.1. GitLens 侧栏 (仓库仪表盘)

- **定位：** 整个仓库的“宏观”指挥中心，用于**浏览和管理**。
- **功能：**
 - **Commits (提交)：** 图形化的 `git log`，浏览提交历史。
 - **Branches (分支)：** 图形化的 `git branch`，可双击切换、右键合并。
 - **Git Graph (Git 图)：** **(核心)** 用可视化分支图，清晰查看分支的分叉与合并关系。

8.2. GitLens Inspect 侧栏 (历史调查器)

- **定位：** 用于“调查”和“搜索”特定问题（例如“谁动了我的代码？”）。
- **功能：**
 - **Search & Compare (搜索与比较)：** **(核心)**
 - **搜索：** 按提交信息、作者、文件名搜索 `commit`。
 - **比较：** 选择两个分支或 `commit`，清晰对比它们之间的所有文件差异。

8.3. 核心精髓：“隐形”能力 (代码追溯)

你 90% 的时间会用到的功能，它们在编辑器内部：

1. 当前行追溯 (Current Line Blame):

- 功能： 你的光标点到任何一行代码，行尾都会出现一行灰色的字： [作者], [时间] (via [提交信息])。
- 作用： 立刻知道这行代码是谁、何时、为何写下的。

2. 悬停信息 (Hover Info):

- 功能： 鼠标悬停在那行灰色的“追溯信息”上。
- 作用： 弹出一个详细卡片，可一键查看该 commit 的所有修改 (Diff)。

9. Git 常用操作指令速查表

功能分类	命令	作用 (大白话)
首次配置	git config --global user.name "..."	设置你的签名（提交时显示的名字）
	git config --global user.email "..."	设置你的邮箱（GitHub 靠它识别你）
开始项目	git init	在当前文件夹创建“本地仓库”
	git clone [url]	把云端的项目完整“克隆”一份到本地
日常工作	git status	(必用) 检查工作区/暂存区的状态
	git add .	把所有修改（增/删/改）放进“暂存区”
	git add [file]	只把某个文件放进“暂存区”
	git commit -m "..."	把暂存区的所有内容“提交”到本地仓库
	git log	查看所有“提交历史”
	git log --stat	查看历史，并显示改动了哪些文件
分支管理	git branch	列出所有分支
	git switch -c [name]	创建一个叫 [name] 的新分支并切换过去
	git switch [name]	切换到已有的 [name] 分支
	git merge [name]	把 [name] 分支的修改合并到你当前所在的分支

功能分类	命令	作用 (大白话)
	<code>git branch -d [name]</code>	删除一个已经合并过的分支
远程协作	<code>git remote -v</code>	查看当前关联的“远程仓库”地址
	<code>git remote add origin [url]</code>	关联一个远程仓库，并叫它 'origin'
	<code>git push</code>	把本地的新提交 推送 到云端 (GitHub)
	<code>git pull</code>	把云端的最新提交 拉取 到本地并合并
撤销/ 恢复	<code>git checkout -- [file]</code>	(危险) 丢弃在“工作区”对某个文件的修改
	<code>git reset HEAD [file]</code>	把文件从“暂存区”拿回到“工作区”
	<code>git rm --cached [file]</code>	告诉 Git“不再跟踪这个文件” (比如从套娃仓库中移除)