

## 第一周实验报告 1

24 计算机科学与技术吴虹霖 学号: 24020007135

2025 年 8 月 30 日

# 一、Git 练习实例

## 实例 1.1 Git 命令和数据模型相关内容

**Q:** 如果您之前从来没有用过 Git, 推荐您阅读 Pro Git 的前几章, 或者完成像 Learn Git Branching 这样的教程。重点关注 Git 命令和数据模型相关内容;

**A:** 1. 数据模型: Git 通过“工作区 → 暂存区 → 本地版本库”三层结构管理代码, 分支本质是指向特定提交记录的引用, 所有历史版本通过提交对象链串联, 形成可追溯的变更记录。

2. 核心命令: 围绕数据流转和分支操作, `git add` 将工作区文件移至暂存区, `git commit` 将暂存区内容提交到版本库形成历史记录; `git branch` 创建分支, `git checkout` 切换分支, `git merge` 合并分支, 支持多线开发与历史整合。

## 实例 1.2 Git 命令和数据模型相关内容

**Q:** 克隆本课程网站的仓库, 将版本历史可视化并进行探索

**A:** step1: 克隆仓库

```
C:\Users\吴虹霖\Desktop\2025小学期\系统开发工具基础\clone> git clone https://github.com/missing-semester-cn/missing-semester-cn.github.io.git
Cloning into 'missing-semester-cn.github.io'...
remote: Enumerating objects: 3284, done.
remote: Counting objects: 100% (54/54), done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 3284 (delta 12), reused 23 (delta 7), pack-reused 3230 (from 1)
Receiving objects: 100% (3284/3284), 15.71 MiB | 1.87 MiB/s, done.
Resolving deltas: 100% (2052/2052), done.
```

图 1: 克隆仓库截图

step2: 可视化版本历史

```
C:\Users\吴虹霖\Desktop\2025小学期\系统开发工具基础\clone> cd missing-semester-cn.github.io
C:\Users\吴虹霖\Desktop\2025小学期\系统开发工具基础\clone\missing-semester-cn.github.io> git log --all --graph --oneline
* 3e31dd9 (HEAD -> master, origin/master, origin/HEAD) Merge pull request #197 from yueneiqi/fix/shell-tools-terminology
* c0d8274 Synchronize Chinese translations with English repository updates
* 50365f3 Fix multiple translation issues across course materials
* 9a5fa75 Fix terminology in Shell tools exercises
* d893328 Merge pull request #196 from Jd010012/master
* f43d642 删除了about.md中多余的换行
* 1107558 Merge pull request #195 from VictorZhangAI/master
* 6fd4b5 Update qa.md
* 9cde6ae Update qa.md
* ea6be58 Merge pull request #194 from yhc999/yhc
* bdaf003 fix: 删除_2020/version-control.md结尾分号前空格
* 3ab2ec1 Merge pull request #190 from Fortuna233/master
* 373a740 修改about.md
* 17ecce7 Merge pull request #189 from hulyoo/master
* 594ddca 修改_2020/editors.md, 把视角更改为缓冲区的视图.
```

图 2: 查看历史版本截图

### 实例 1.3 查看修改

**Q:** 是谁最后修改了 README.md 文件? (提示: 使用 `git log` 命令并添加合适的参数)

**A:** 如图所示。

```
C:\Users\吴虹霖\Desktop\2025小学期\系统开发工具基础\clone\missing-semester-cn.github.io> git log -1 README.md
commit fc93d7c8660cee7ac2dfeb23fd85f9ec741ff3a8
Author: Zhengner233 <2042712521@qq.com>
Date:   Fri Nov 15 00:01:20 2024 +0800
```

图 3: 查看修改人

### 实例 1.4 Git 远程仓库关联 (GitHub/Gitee)

- 操作目的: 实现本地仓库与远程仓库 (如 GitHub) 的连接, 推送本地代码到远程。
- 操作步骤: 1. 在 GitHub 创建新仓库 (如 `git-practice-2025`, 不勾选 “Initialize this repository with a README”); 2. 本地终端执行 `git remote add origin https://github.com/你的用户名/git-practice-2025.git` (关联远程仓库); 3. 执行 `git push -u origin main` (推送主分支到远程, 首次推送需输入 GitHub 账号密码/Token); 4. 浏览器打开 GitHub 仓库页面, 验证是否看到本地提交的 `readme.md`。
- 结果说明: 终端显示 “Writing objects: 100”

### 实例 1.5 Git 拉取远程仓库更新

- 操作目的: 学会从远程仓库拉取他人/自己在其他设备的更新, 保持本地与远程同步。
- 操作步骤: 1. 在 GitHub 网页端直接修改 `readme.md` (如添加 “remote update: 2025-08-31”), 并提交; 2. 本地终端执行 `git pull origin main` (拉取远程主分支的更新); 3. 打开本地 `readme.md`, 查看是否包含网页端添加的内容。
- 结果说明: 终端显示 “Updating a1b2c3d..e4f5g6h” (拉取成功), 本地文件与远程完全同步。

### 实例 1.6 Git 撤销暂存区文件

- 操作目的: 掌握 “将文件从暂存区撤回工作区” 的操作 (误 `add` 后补救)。
- 操作步骤: 1. 本地创建新文件 `test.txt`, 写入 “test file”; 2. 执行 `git add test.txt` (将文件加入暂存区); 3. 执行 `git status` (查看状态, 显示 “Changes to be committed: test.txt”); 4. 执行 `git rm --cached test.txt` (撤销暂存区的 `test.txt`); 5. 再次执行 `git status`, 验证是否显示 “Untracked files: test.txt” (回到未暂存状态)。
- 结果说明: `test.txt` 从 “待提交” 状态变回 “未追踪” 状态, 工作区文件未被删除。

### 实例 1.7 Git 查看历史提交记录 (筛选与格式化)

- 操作目的: 高效筛选历史记录 (如指定作者、时间), 简化输出格式。
- 操作步骤: 1. 执行 `git log --oneline` (简化显示: 短哈希值 + 提交信息, 一行一条); 2. 执行 `git log --author=" 你的用户名 "` (只显示你提交的记录); 3. 执行 `git log --since="2025-08-30" --until="2025-08-31"` (显示指定日期范围内的记录); 4. 执行 `git log -p readme.md` (查看 `readme.md` 文件的所有修改历史, 含具体内容差异)。
- 结果说明: 不同命令对应不同筛选结果, 能快速定位目标提交记录。

## 实例 1.8 Git 解决简单合并冲突

- 操作目的：处理“同一文件同一行被不同分支修改”导致的合并冲突。
- 操作步骤：1. 切到 dev 分支：git checkout dev，修改 readme.md 第 1 行为“dev: conflict test”，提交；2. 切回 main 分支：git checkout main，修改 readme.md 第 1 行为“main: conflict test”，提交；3. 执行 git merge dev，终端显示“Automatic merge failed; fix conflicts and then commit the result.”（冲突提示）；4. 打开 readme.md，看到冲突标记：

```
<<<<<<< HEAD (当前分支: main)
main: conflict test
=====
dev: conflict test
>>>>>>> dev (待合并分支: dev)
```

5. 删除冲突标记，修改内容为“fixed conflict: main + dev”，执行 git add readme.md && git commit -m "fix merge conflict"（提交解决冲突）；
- 结果说明：合并成功，readme.md 保留修改后的内容，git log 能看到“fix merge conflict”的提交记录。

## 实例 1.9 Git 删除本地与远程分支

- 操作目的：清理无用分支（本地 + 远程），保持仓库整洁。
- 操作步骤：1. 切回 main 分支：git checkout main（删除分支前需退出该分支）；2. 执行 git branch -d dev（删除本地 dev 分支，若分支有未合并修改，用 -D 强制删除）；3. 执行 git push origin --delete dev（删除远程 dev 分支）；4. 执行 git branch（验证本地 dev 已删除），访问 GitHub 仓库查看“Branches”，验证远程 dev 已删除。
- 结果说明：本地和远程的 dev 分支均被删除，仅保留 main 分支。

## 实例 1.10 Git 忽略文件（.gitignore 配置）

- 操作目的：让 Git 自动忽略不需要追踪的文件（如日志、缓存、IDE 配置文件）。
- 操作步骤：1. 本地创建 log.txt（日志文件）和 vscode/ 文件夹（IDE 配置文件夹）；2. 在仓库根目录创建 .gitignore 文件，写入以下内容：

```
# 忽略日志文件
log.txt
# 忽略VSCode配置文件夹
vscode/
# 忽略所有.csv文件
*.csv
```

3. 执行 git add .gitignore && git commit -m "add .gitignore: ignore log/vscode/csv"（提交配置）；4. 执行 git status，验证 log.txt 和 vscode/ 是否显示“Untracked files”（Git 已忽略它们，不提示追踪）。
- 结果说明：log.txt、vscode/ 等文件不会被 Git 追踪，避免提交无用文件到仓库。