

**TheUnknownThing**

# **Git Operations**





# 为什么要用 Git?





# 我需要**一个强大的时光机**

- 你会重写你的代码
- 有很多人会贡献代码
- 手贱了怎么办?
- 谁写了哪行代码?
- 万一重构失败了怎么办😭
- 我和别人改了同一个文件怎么办🤔
- 我补药重写一遍😭
- 让我康康谁写的代码出问题了😈

**“ 如果有一台时光机， 该多好？ ”**



# 为什么要用 Git?





**什么是 Git?**

**为什么要用 Git?**



**git**

**git** 1 of 2 **noun**

'git 

[Synonyms of \*git\* >](#)

**British**

: a foolish or worthless person

<https://www.merriam-webster.com/dictionary/git>

**“I’m an egotistical bastard, and I name all my projects after myself. First ‘linux’, now ‘git’.”**

**Linus Torvalds**

# Git 是版本控制系统

版本控制系统能：

- 帮助我们管理代码的修改历史
- 让协作变得更方便。
- 自动维护一系列快照（文件 & 文件夹）

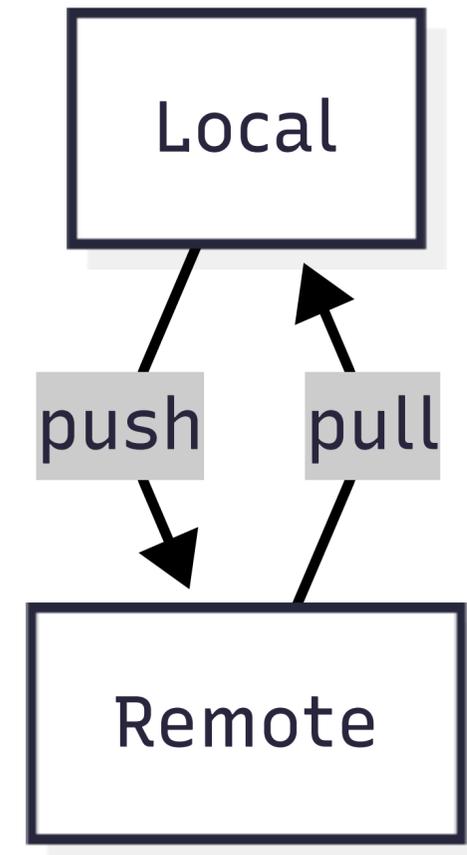
每一个仓库就是一个“时光机”

这些时光机“看起来”就是一个文件夹

```
> ls -la .
total 80
drwxr-xr-x@  theunknownthing  .
drwxr-x---+  theunknownthing  ..
drwxr-x---@  theunknownthing  .cache
-rw-r--r--@  theunknownthing  .clangd
drwxr-xr-x@  theunknownthing  .git
drwxr-xr-x@  theunknownthing  .github
```

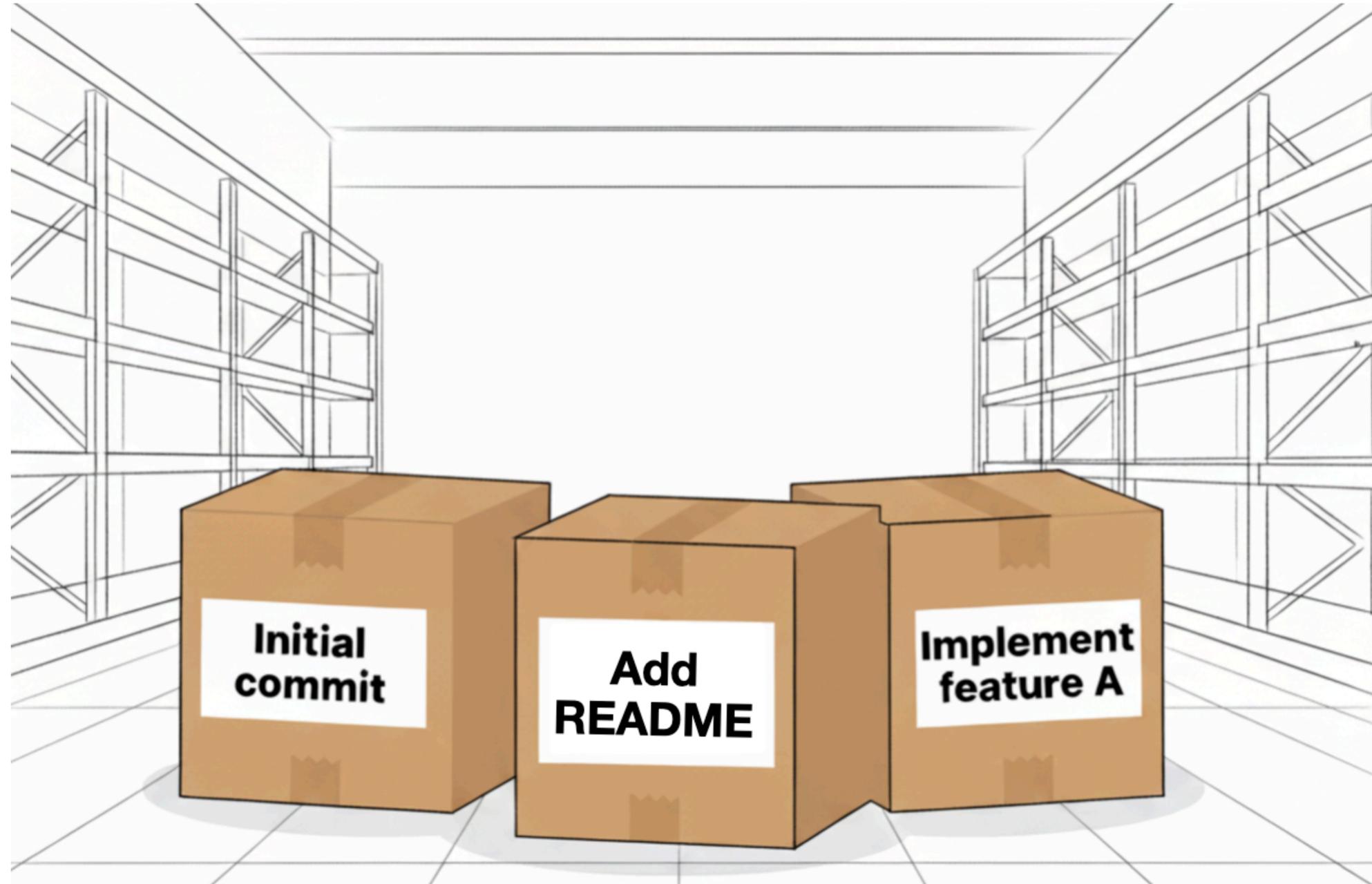
# 你可能会问：

- 是什么魔法让 Git 与众不同？  
Git 仓库 v.s. 普通文件夹
- Git 的时光机是存放在哪里的？  
.git 文件夹
- 我要怎么和别人合作？  
本地仓库 + 云端仓库  + 



# 认识一些 Git 中的名词

仓库



# 认识一些 Git 中的名词

## Commits

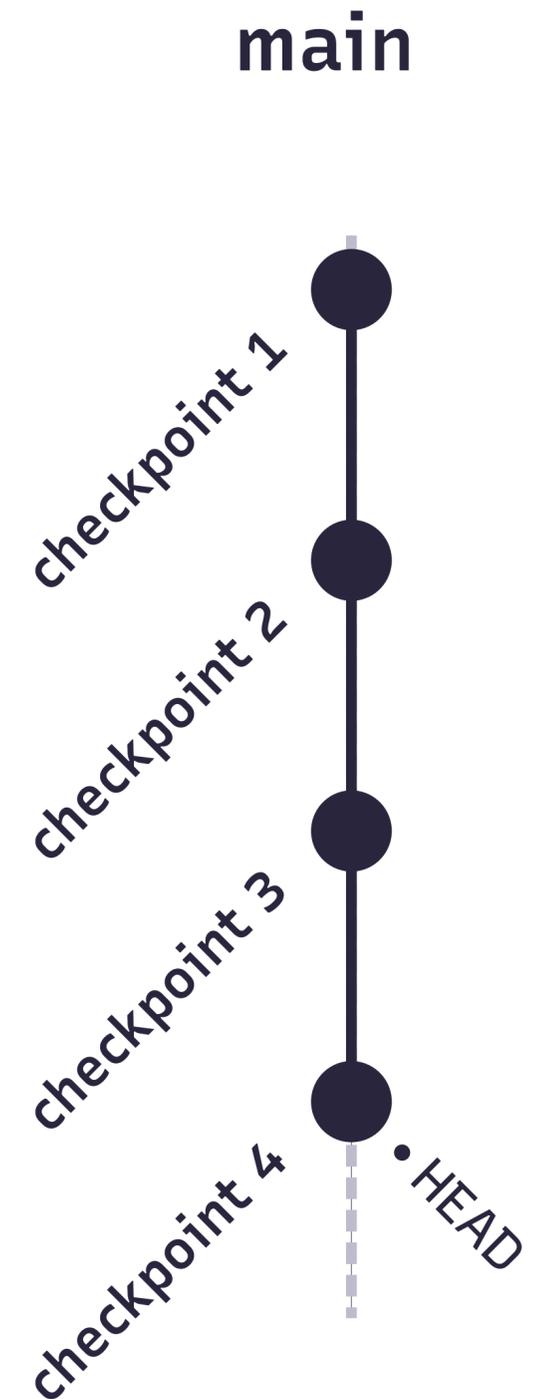
黑点代表 Commits，相当于“保存仓库某时刻的快照”

## HEAD

告诉 Git，你现在处于哪个版本上工作？

## Parent

之前的一系列快照，“父节点”



# 认识一些 Git 中的名词

## Branch & Main Branch

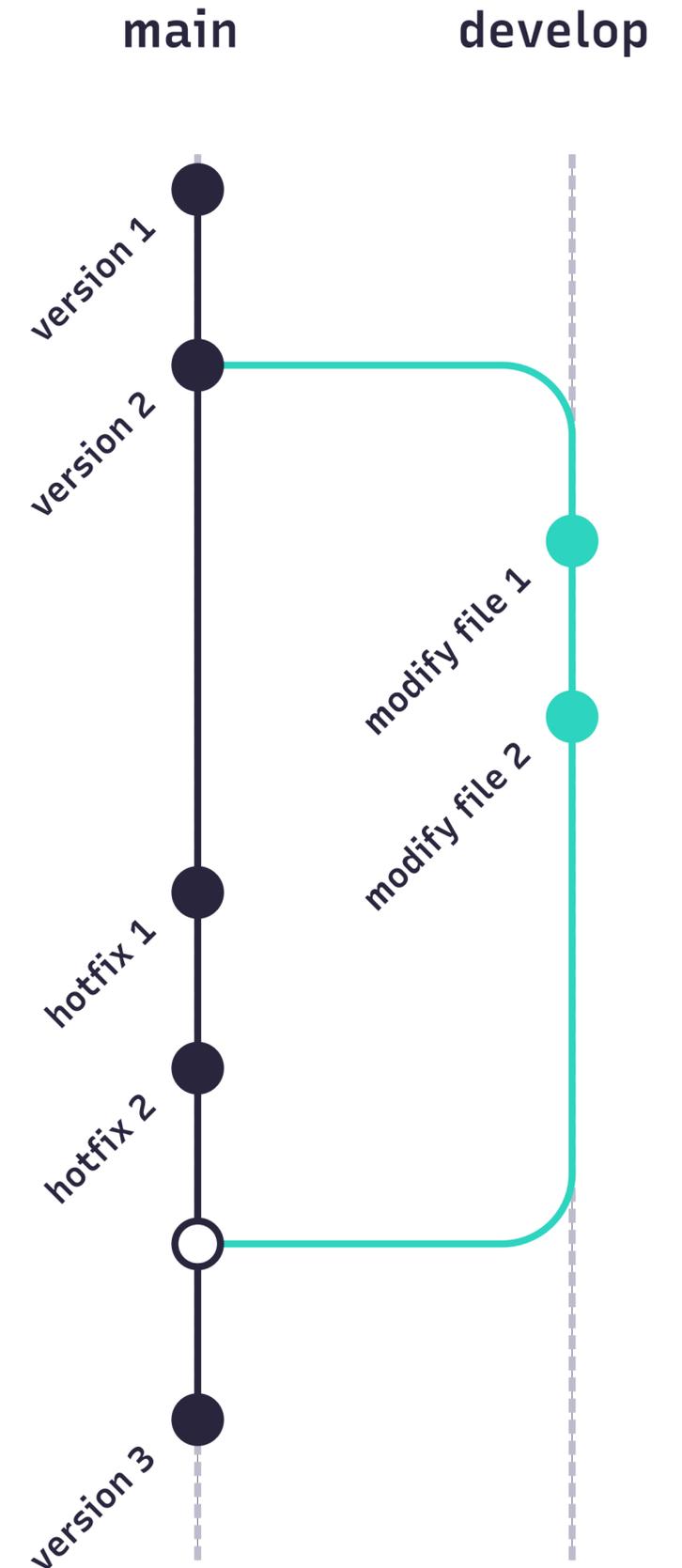
创建了一个分支，就相当于创建了一条独立的“时间线”

## Merge

在分支上的工作结束后，合并到主线

## Merge Commit

空心圆点；合并两条历史；有两个 PARENT



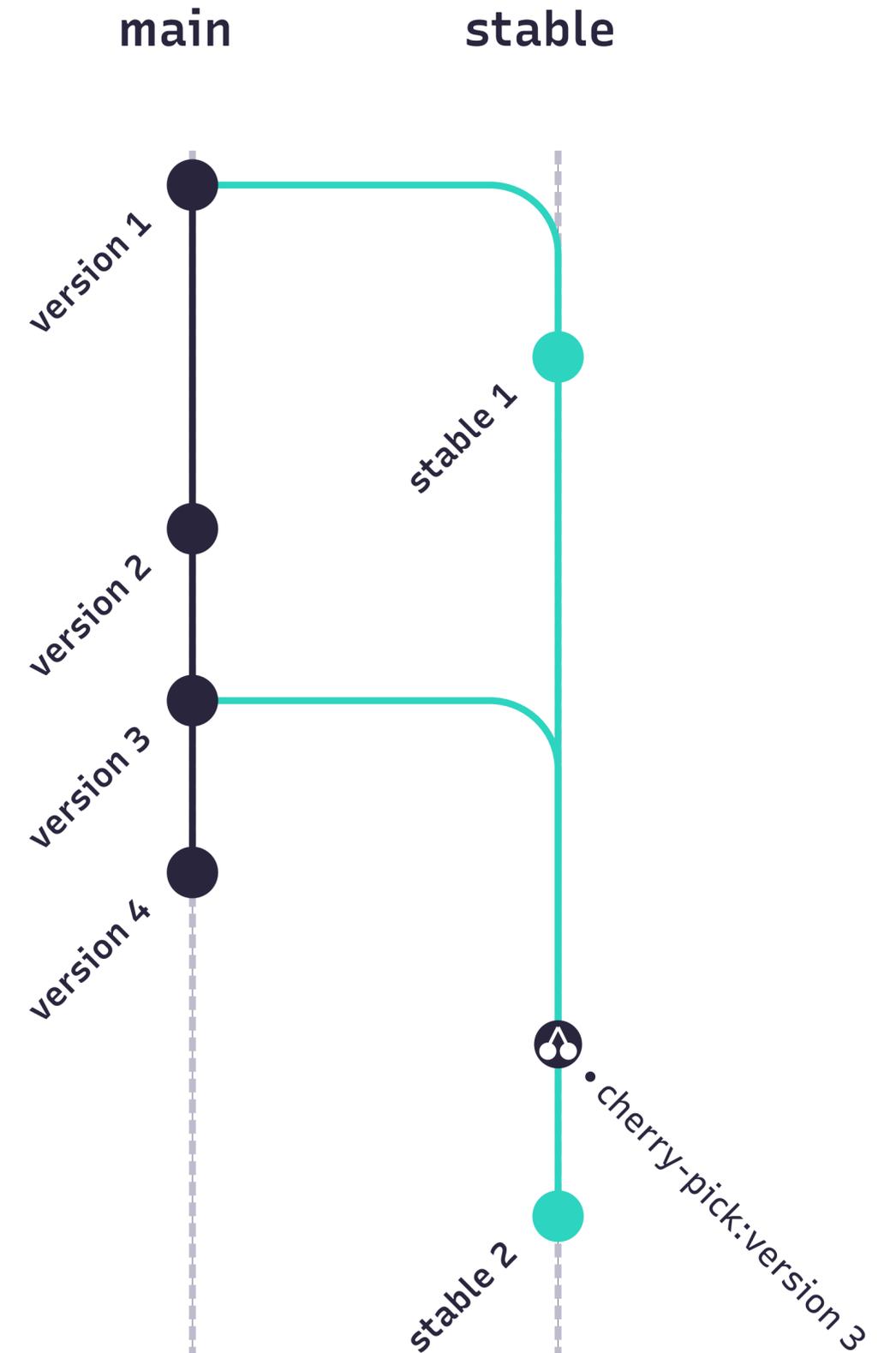
# 认识一些 Git 中的名词

## Cherry-Pick

把某一个 Commit 的修改应用于所在的分支

## Parallel Development

做版本与分支管理不是无用功，会在关键时刻救你命





**什么是 Git?**

**为什么要用 Git?**



**git**



# 怎么用 Git?

什么是 Git?

为什么要用 Git?





# 怎么用 Git?

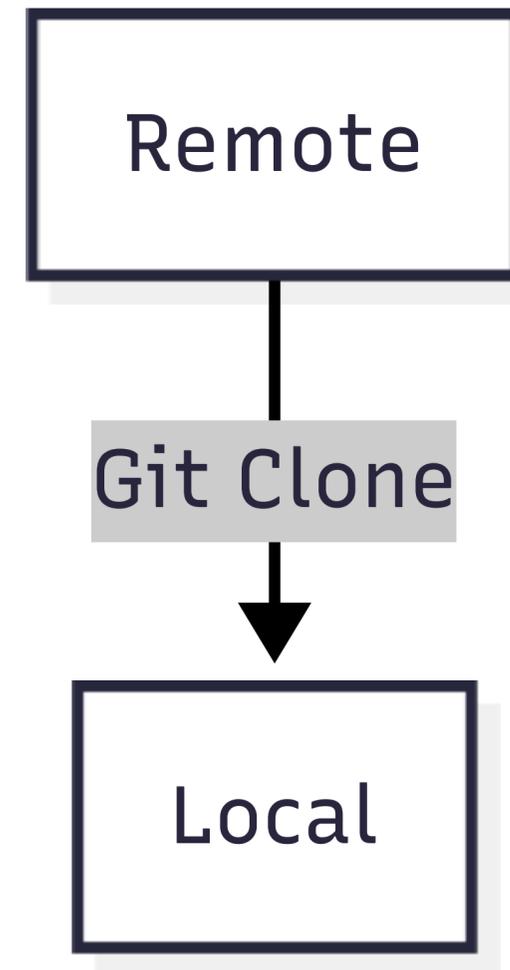
- 这就是 Git。它通过一个优美的分布式图论树状模型来跟踪项目的协作。🧐
- **Cool!** 我们该怎么用它? 💕
- 我也不知道。大概就是把几个 shell 命令背下来，然后输入它们。如果遇到报错，就把你的工作备份到别处，删掉整个项目，再重新下载一份新的。🤪

# git clone

Clone an existing repo

克隆一个存在的仓库

下载包含所有历史时光机



# 把几个 shell 命令背下来

你可能会用到的命令

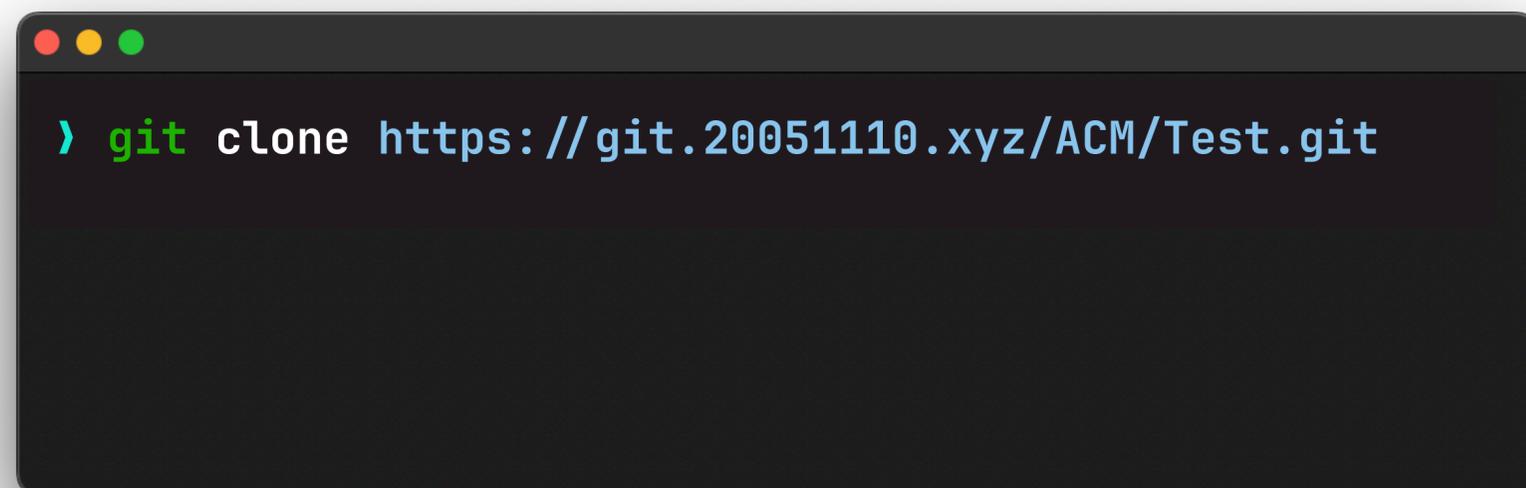
```
git clone url
```

克隆仓库

```
git clone url directory
```

克隆仓库到指定目录

动手做一做：

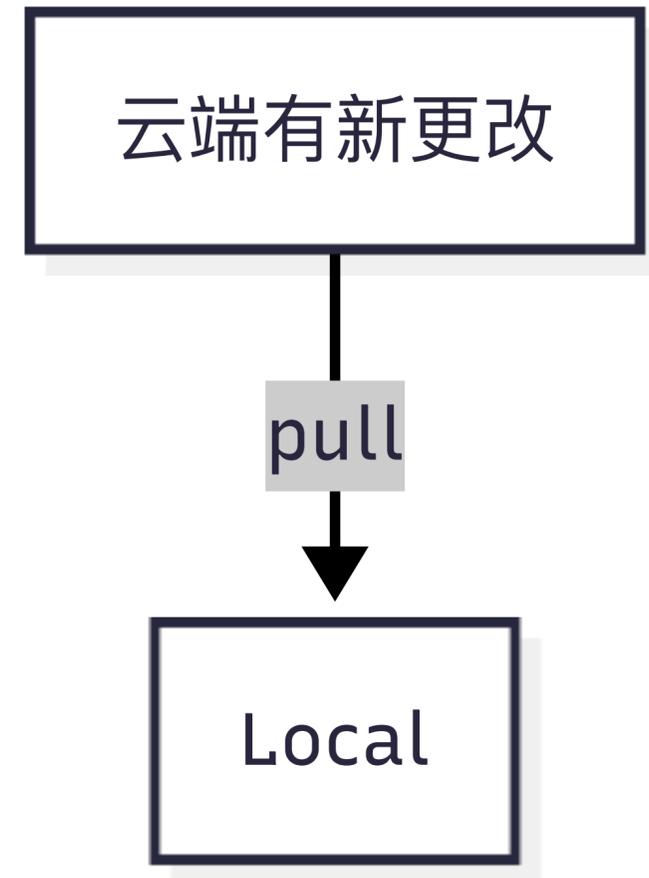
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal shows a single command: `> git clone https://git.20051110.xyz/ACM/Test.git`.

```
> git clone https://git.20051110.xyz/ACM/Test.git
```

# git pull

Fetch remote, merge it with local

拉取更改并与本地合并  
有人提交了新的代码!



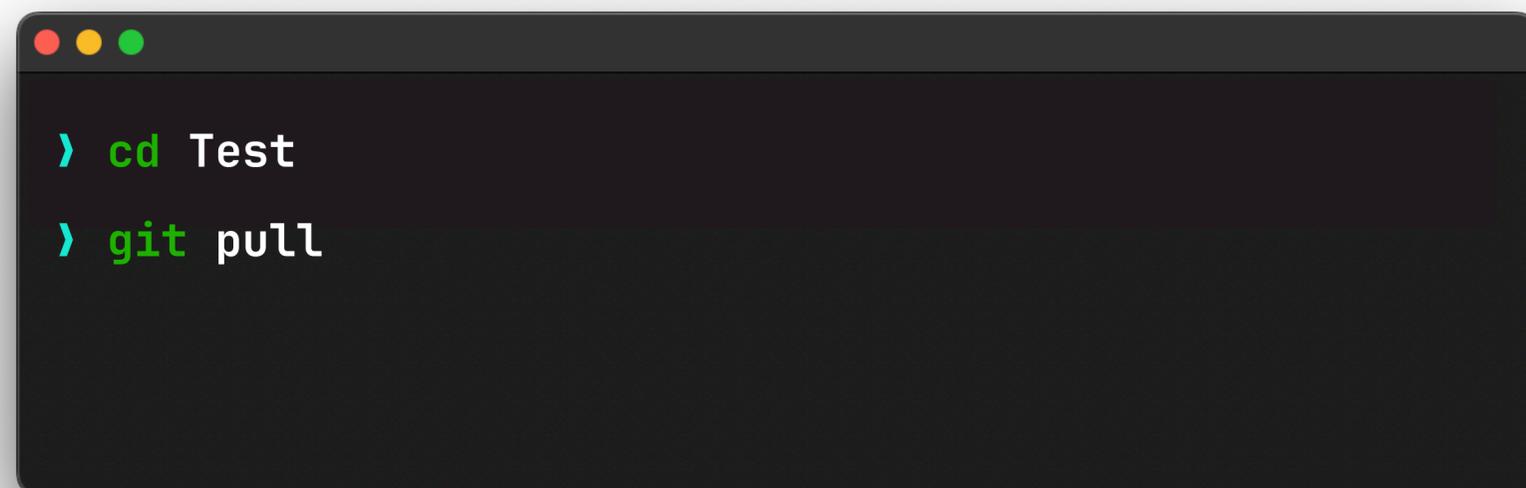
# 把几个 shell 命令背下来

你可能会用到的命令

`git pull`

拉取更改

动手做一做：

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of shell commands: the first line is a prompt followed by `cd Test`, and the second line is a prompt followed by `git pull`.

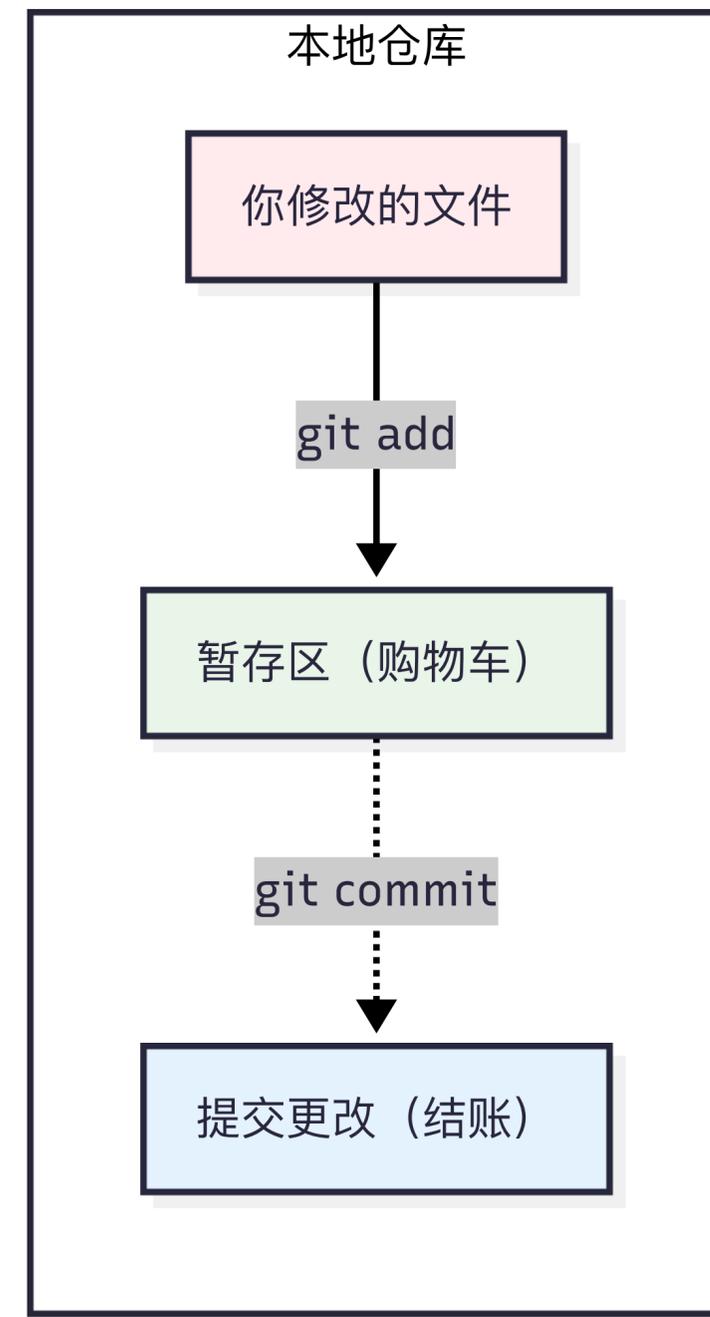
```
> cd Test
> git pull
```

# git add

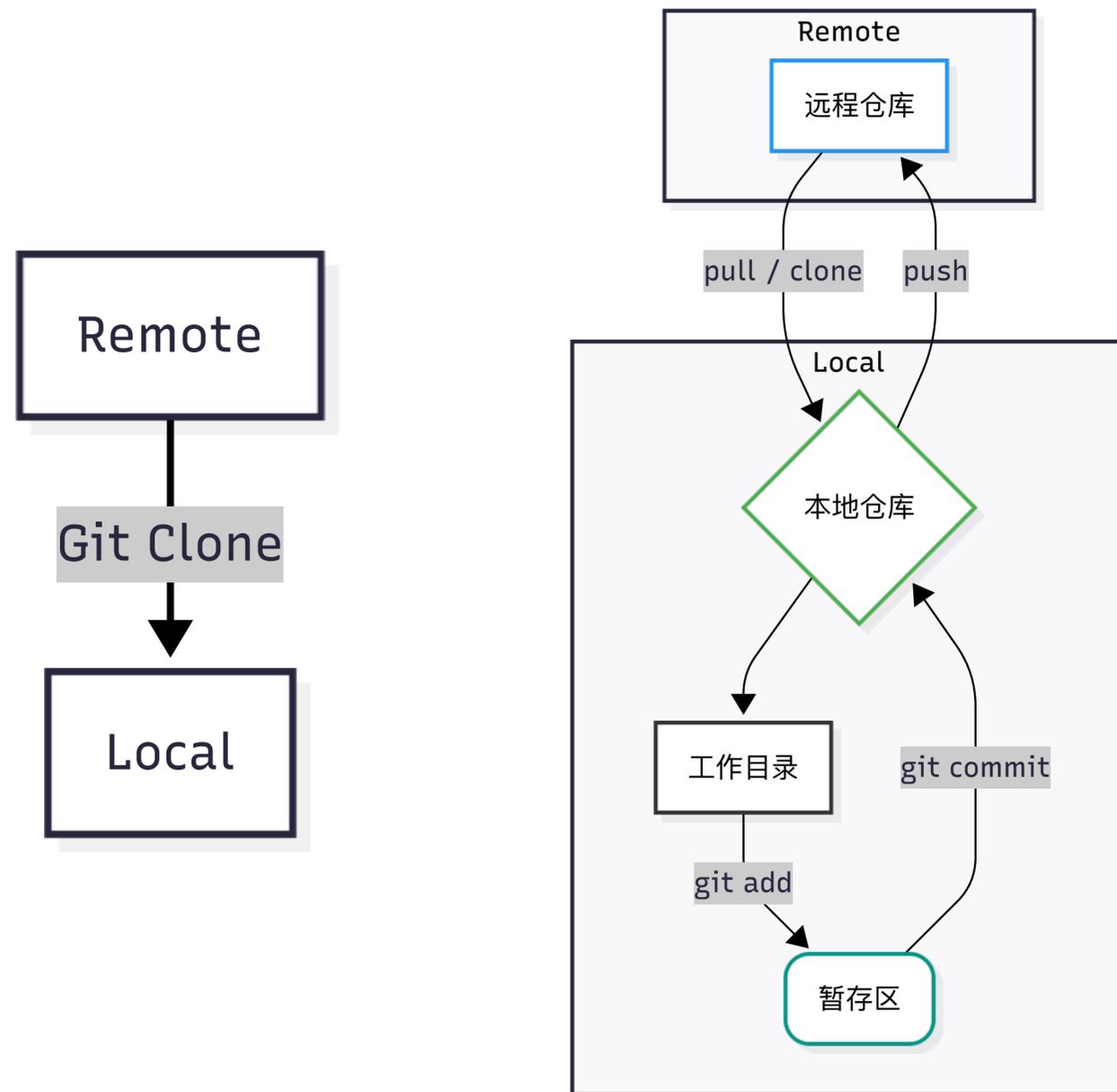
Adds changed files to the index

添加更改的文件至 Staging Area

结账之前放到购物车里 🛒



# 暂存区是什么 🤔



工作目录 📁:

- 正如名字所说，是一个你电脑上的目录
- 存储着你代码的文件夹

暂存区 🛒:

- 存储 准备在下一次提交中修改的内容
- 你可以选择你要提交什么更改

本地仓库 📦:

- 就是那个 `.git` 文件夹
- 它存储着你仓库的一切历史记录

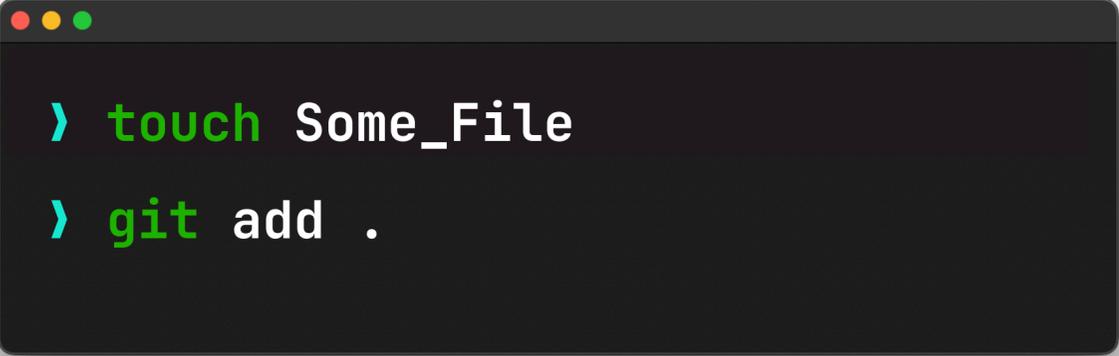
# 把几个 shell 命令背下来

## 你可能会用到的命令

`git add file`      添加文件至暂存区

`git add .`      添加目录下所有文件至暂存区

## 动手做一做：

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains two lines of shell commands: the first line is a green prompt character followed by the command 'touch Some\_File', and the second line is a green prompt character followed by the command 'git add .'.

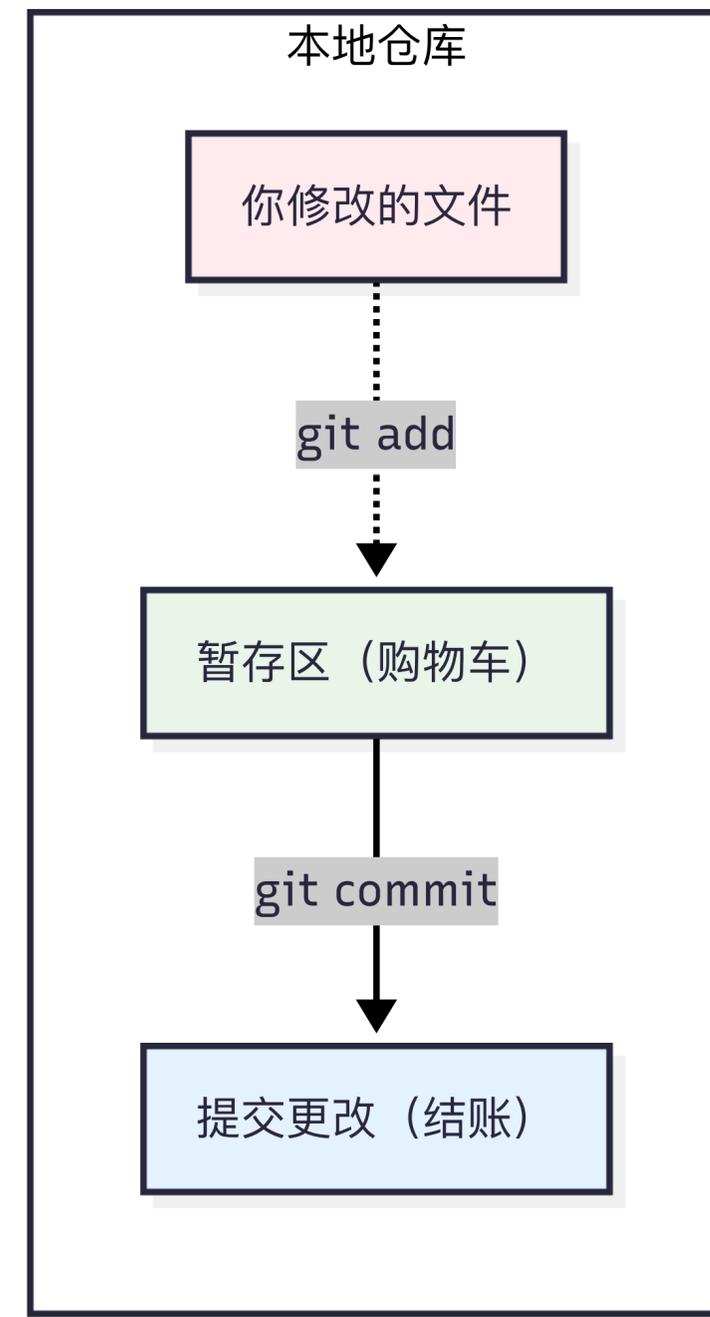
```
> touch Some_File  
> git add .
```

# git commit

Commit changes to local repo

提交更改至仓库中

你将更改的代码合并到仓库中（本地）



# 把几个 shell 命令背下来

## 你可能会用到的命令

```
git commit -m "message"
```

提交暂存区中的文件并附上描述

## 动手做一做：

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal shows a prompt character followed by the command `git commit -m "这里写你做了什么"`.

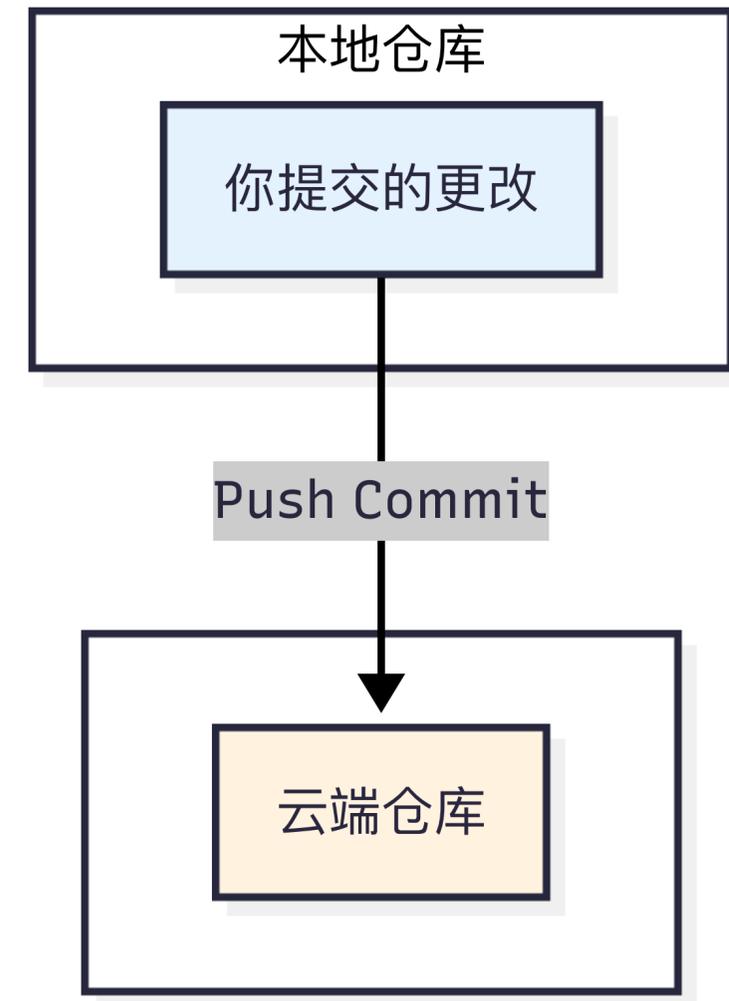
```
> git commit -m "这里写你做了什么"
```

# git push

Push Commits to a **remote** repo

推送更改至**远端**仓库

把本地的更改上传（大家都能看见）



# 把几个 shell 命令背下来

你可能会用到的命令

`git push`

推送更改

这次不动手实操了

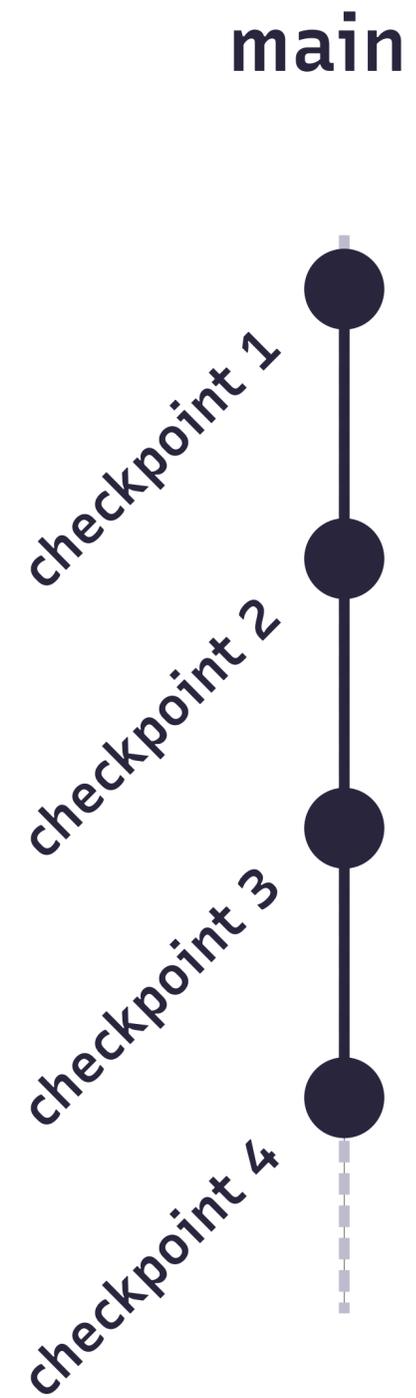
你们也没我仓库的写入权限

# git log

View the commit history of repo

检查仓库的历史提交

谁在什么时候做了什么 🙄



# 把几个 shell 命令背下来

## 你可能会用到的命令

`git log`

查看提交历史（时间顺序）

`git log --stat`

查看具体更改了哪些文件

## 动手做一做：



```
> git log --stat
```

Tips: 按 `Q` 退出

# Git Conventions

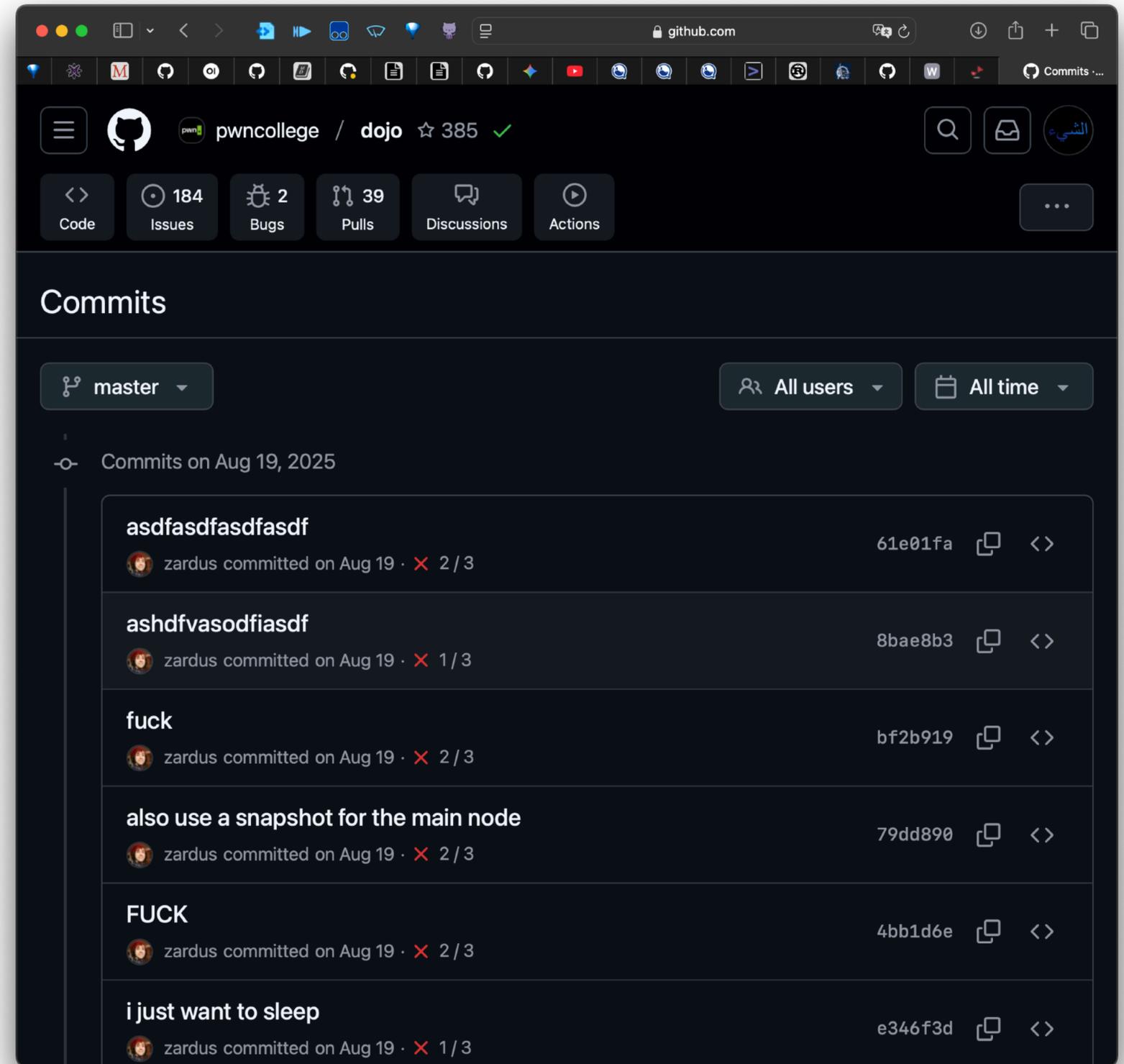
```
Git — theunknownthing@TheUnknownThingdeMacBook-Air...
~/Git 15:58:03
> git commit -m "这里写你做了什么"
```

Commit Message 不许发疯!

坏的: *F\*\*k; ASDF; LOL;*

好的: feat: add Polish language

[www.conventionalcommits.org](http://www.conventionalcommits.org)



# Conventional Commits

```
<type>[optional scope]: <description>
```

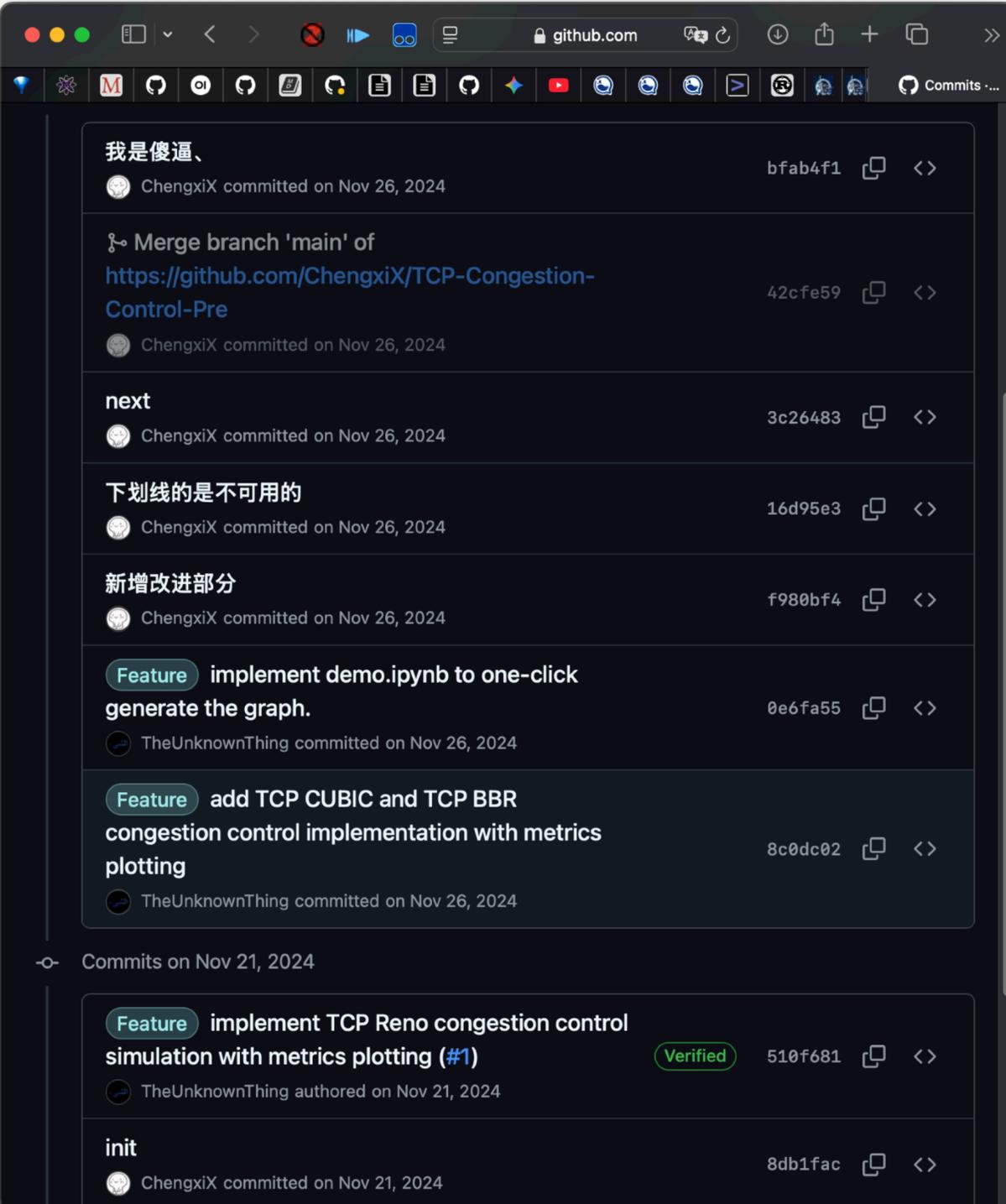
```
[optional body]
```

```
[optional footer(s)]
```

## 👍 Examples

```
docs: correct spelling of CHANGELOG
```

```
feat(lang): add Polish language
```



The screenshot shows a GitHub commit history page for a repository. The commits are listed in reverse chronological order. The most recent commit is by ChengxiX, with the message "我是傻逼、" and hash bfab4f1. Below it is a merge commit by ChengxiX with the message "Merge branch 'main' of https://github.com/ChengxiX/TCP-Congestion-Control-Pre" and hash 42cfe59. This is followed by a commit by ChengxiX with the message "next" and hash 3c26483. Then another commit by ChengxiX with the message "下划线的是不可用的" and hash 16d95e3. Next is a commit by ChengxiX with the message "新增改进部分" and hash f980bf4. This is followed by a commit by TheUnknownThing with the message "Feature implement demo.ipynb to one-click generate the graph." and hash 0e6fa55. Then another commit by TheUnknownThing with the message "Feature add TCP CUBIC and TCP BBR congestion control implementation with metrics plotting" and hash 8c0dc02. Below this is a section for "Commits on Nov 21, 2024". The first commit in this section is by TheUnknownThing with the message "Feature implement TCP Reno congestion control simulation with metrics plotting (#1)" and hash 510f681, which is marked as "Verified". The final commit shown is by ChengxiX with the message "init" and hash 8db1fac.

Commit Message	Author	Hash
我是傻逼、	ChengxiX	bfab4f1
Merge branch 'main' of https://github.com/ChengxiX/TCP-Congestion-Control-Pre	ChengxiX	42cfe59
next	ChengxiX	3c26483
下划线的是不可用的	ChengxiX	16d95e3
新增改进部分	ChengxiX	f980bf4
Feature implement demo.ipynb to one-click generate the graph.	TheUnknownThing	0e6fa55
Feature add TCP CUBIC and TCP BBR congestion control implementation with metrics plotting	TheUnknownThing	8c0dc02
Commits on Nov 21, 2024		
Feature implement TCP Reno congestion control simulation with metrics plotting (#1)	TheUnknownThing	510f681
init	ChengxiX	8db1fac

# Git Ignore

包含密钥的文件

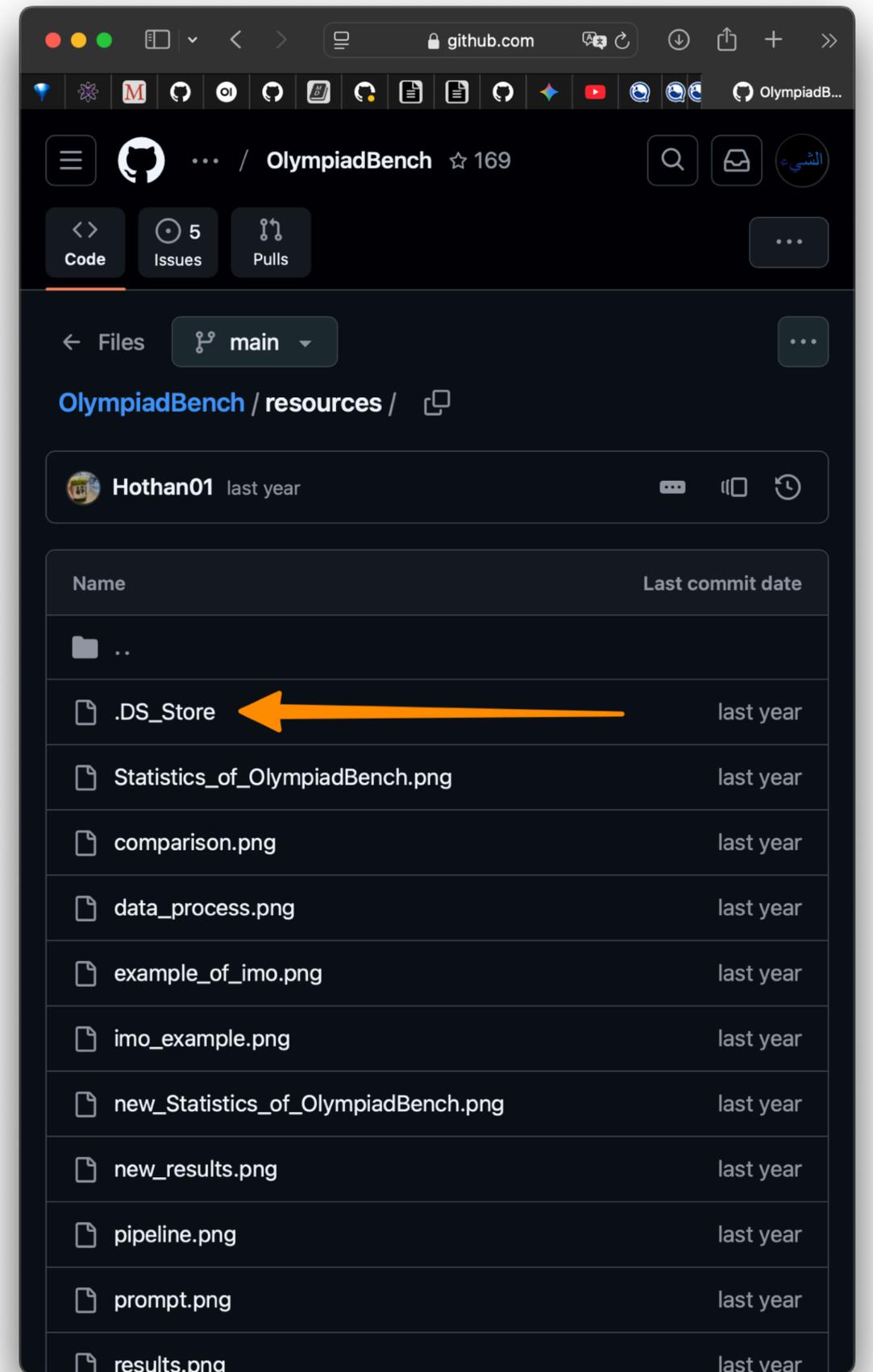
macOS 会拉屎 🍌

代码的编译会产生很多文件

.....

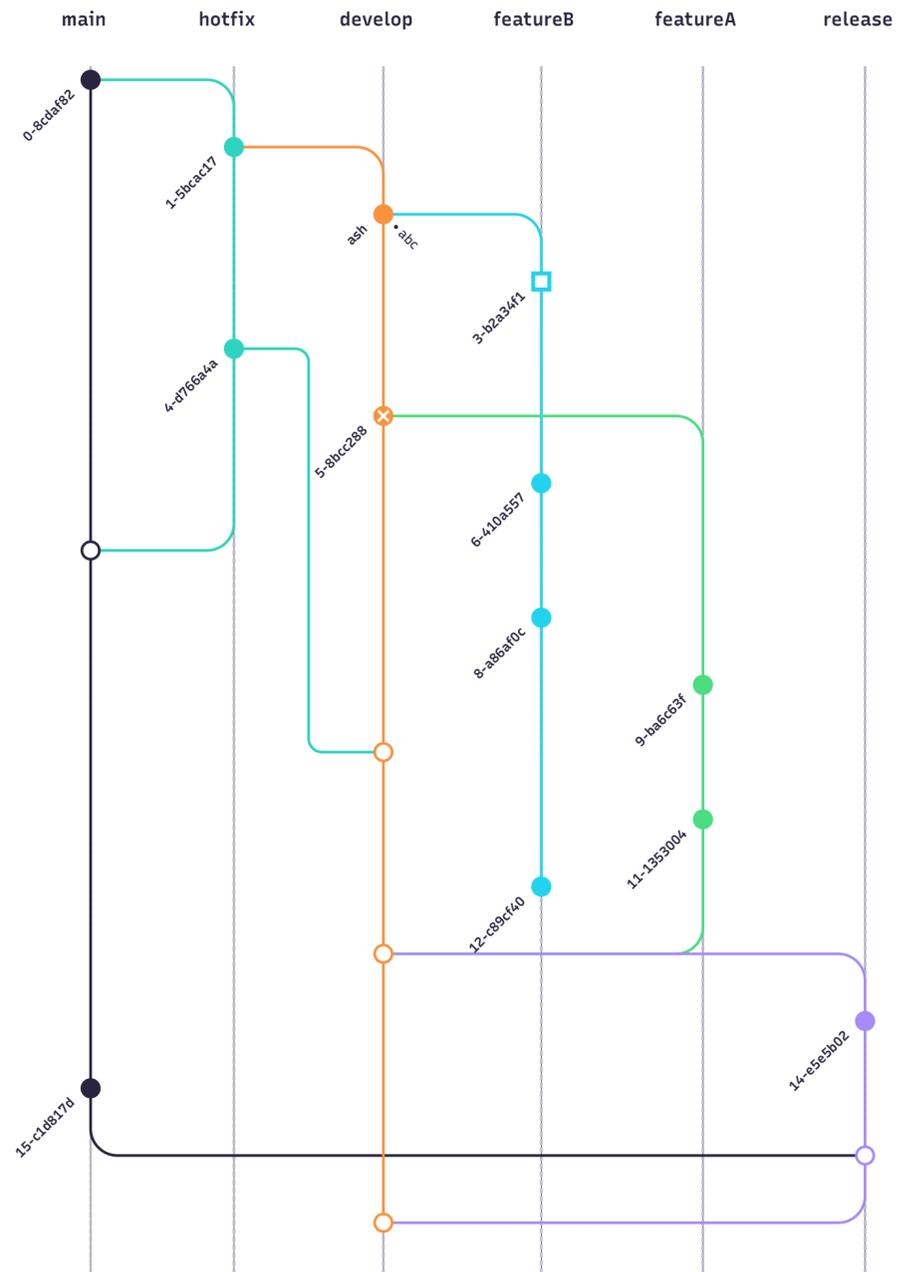
不想把他们传到仓库中，难道每次要手动删除？

🔗 `.gitignore`



```
1 # Build directories ..... 编译目录
2 build/
3 bin/
4 lib/
5
6 # CMake generated files ..... CMake 生成的文件
7 CMakeCache.txt
8 CMakeFiles/
9 cmake_install.cmake
10 Makefile
11
12 # Object files ..... 库文件
13 *.o
14 *.obj
15
16 # Executables ..... 可执行文件
17 *.exe
18 *.out
19 rcompiler
20 unit_tests
21 integration_tests
```

# Branches



The screenshot shows the GitHub web interface for the 'WebAssembly / spec' repository. The 'Branches' page is active, displaying a list of branches with their update times, check statuses, and commit counts. The interface includes a search bar and navigation tabs for Overview, Active, Stale, and All branches.

Branch	Updated	Check status	Behind	Ahead
main	3 days ago	9 / 12		Default
mem4g	6 hours ago	1 / 1	0	1
gh-pages	3 days ago	3 / 3	5947	763
update-testharness	4 days ago		10	1
wasm-3.0	4 days ago	2 / 2	2717	1
fix-readme	4 days ago		13	1
retry-echidna	last week	5 / 8	4923	6
xref-jsapi	last month	8 / 11	2727	1
i32x4_dot_testcase	last month	1 / 1	4924	1
wasm-3.0+relaxed.fix	4 months ago	9 / 12	2773	2
fix-1889	5 months ago	1 / 1	2788	1

# git checkout

Navigating between different versions

| 切换时光机的检查点

```
git checkout <branch> # 切换已有分支
```

```
git checkout -b <branch> # 新建并切换分支
```

```
git checkout <commit_hash> # 查看该提交时仓库的状态
```

# git branch

Managing your branches

| 管理分支的主要命令 |

```
git branch -a # 显示所有分支
```

```
git branch --show-current # 显示当前分支
```

```
git branch <name> # 在当前提交基础上新建分支
```

```
git branch <name> <commit_hash> # 在制定提交基础上新建分支
```

**git** 1 of 2 **noun**

'git ◀▶

[Synonyms of \*git\* >](#)

**British**

: a foolish or worthless person

**Noun**

That ***git*** of a brother of yours has ruined everything!

oh, don't be such a silly ***git***, of course your mates want you around

<https://www.merriam-webster.com/dictionary/git>

# Oh Shit, Git!?!

<https://ohshitgit.com/>

# Oh Shit, Git!?!

Git is hard: screwing up is easy, and figuring out how to fix your mistakes is fucking impossible. Git documentation has this chicken and egg problem where you can't search for how to get yourself out of a mess, *unless you already know the name of the thing you need to know about* in order to fix your problem.

So here are some bad situations I've gotten myself into, and how I eventually got myself out of them *in plain english*.

## Oh shit, I did something terribly wrong, please tell me git has a magic time machine!?!

```
git reflog
# you will see a list of every thing you've
# done in git, across all branches!
# each one has an index HEAD@{index}
# find the one before you broke everything
git reset HEAD@{index}
# magic time machine
```

You can use this to get back stuff you accidentally deleted, or just to remove some stuff you tried that broke the repo, or to recover after a bad merge, or just to go back to a time when things actually worked. I use reflog A LOT. Mega hat tip to the many many many many many people who suggested adding it!

## Oh shit, I committed and immediately realized I need to make one small change!

```
# make your change
git add . # or add individual files
git commit --amend --no-edit
# now your last commit contains that change!
# WARNING: never amend public commits
```

# Oh Shit, Git!?!

用好 Git 很难: 很容易就犯错了, 然后想自己弥补犯下的错, 简直太难了。查阅 Git 文档简直就像是个“鸡生蛋 蛋生鸡”的问题, *你得知道你要的是啥*, 但如果我知道的话, 我还他妈查个毛文档啊!

所以接下来我会分享一些我遇到过的抓狂的经历, 然后用 *白话* 来说说我是如何解决的。

## 哎呦我去, 我刚才好像犯了个大错, 能不能给我台时光机啊!?!

```
git reflog
# 你将看到你在 git 上提交的所有改动记录被列
# 了出来, 而且囊括了所有的分支, 和已被删除的
# commit 哦!
# 每一条记录都有一个类似 HEAD@{index} 的索
# 引编号
# 找到在犯错前的那个提交记录的索引号, 然后执
# 行:
git reset HEAD@{index}
# 哈哈, 这就是你要的时光机!
```

你可以用这个方法找回那些你不小心删除的东西、恢复一些你对 repo 改动、恢复一次错误的 merge 操作、或者仅仅想退回到你的项目还能正常工作的那一时刻。我经常使用 reflog, 在此我要向那些提案添加这个功能的人们表示感谢, 太谢谢他们了!

## 哎呦我去, 我刚提交 commit 就发现还有一个小改动需要添加!

```
# 继续改动你的文件
git add . # 或者你可以添加指定的文件
git commit --amend --no-edit
# 你这次的改动会被添加进最近一次的 commit 中
# 警告: 千万别对公共的 commit 做这种操作
```



# 怎么用 Git?

什么是 Git?

为什么要用 Git?

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



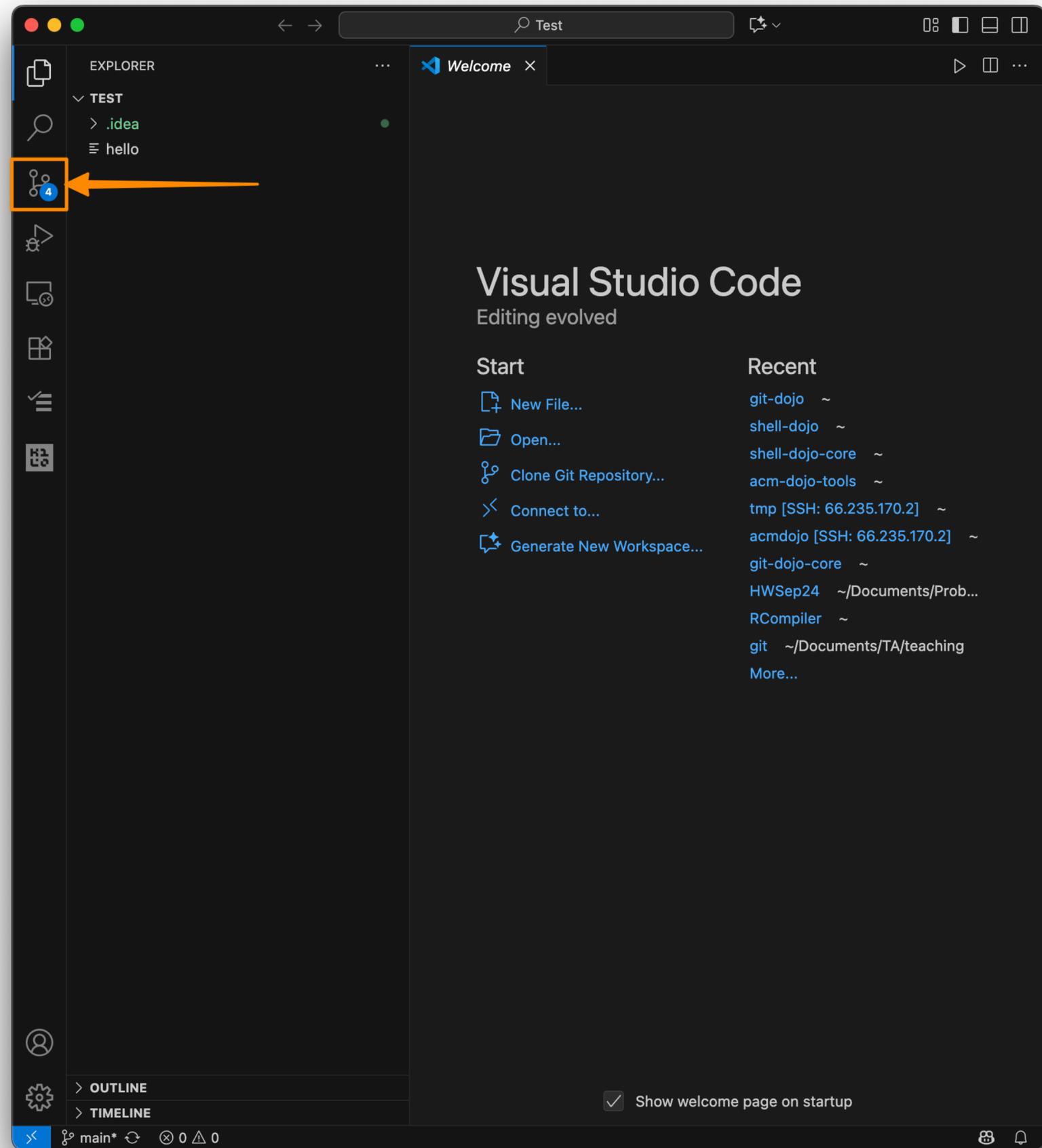


# 图形化操作 Git

怎么用 Git?

什么是 Git?

为什么要用 Git?



Test main

仓库差异

602f47d7 hello 849ef4b6

```
1 Hello there! If you see this, it means that
2 Git is a time machine, and I made some chan
3
4 Git is a foolish person.
5
```

Git 日志

feat: origin & main TheUnknownThing 1小时之前

feat: do some modific: TheUnknownThing 今天 18:30

feat: hello world TheUnknownThing 今天 18:29

Test 1个文件

hello

feat: do some modifications to hello

Test

SOURCE CONTROL

CHANGES

Message (%Enter to commit on "...)

Commit

Changes 4

- .gitignore .idea U
- modules.xml .idea U
- Test.iml .idea U
- vcs.xml .idea U

hello (602f47d) ↔ hello (849ef4b)

```
1 Hello there! If you see this, it means that you have successfu
2 Git is a time machine, and I made some changes to hello.
3+
4+ Git is a foolish person.
3 5
```

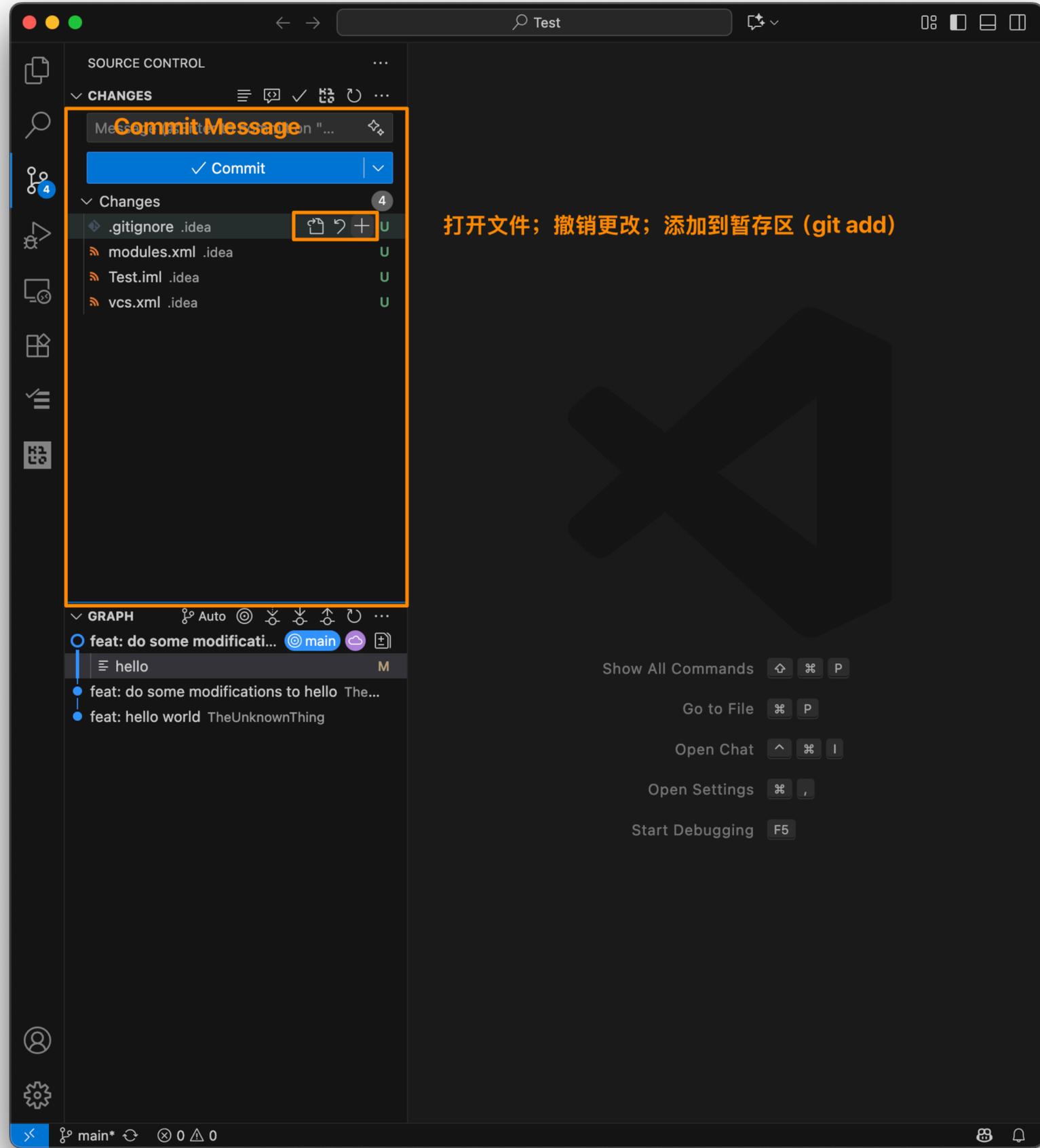
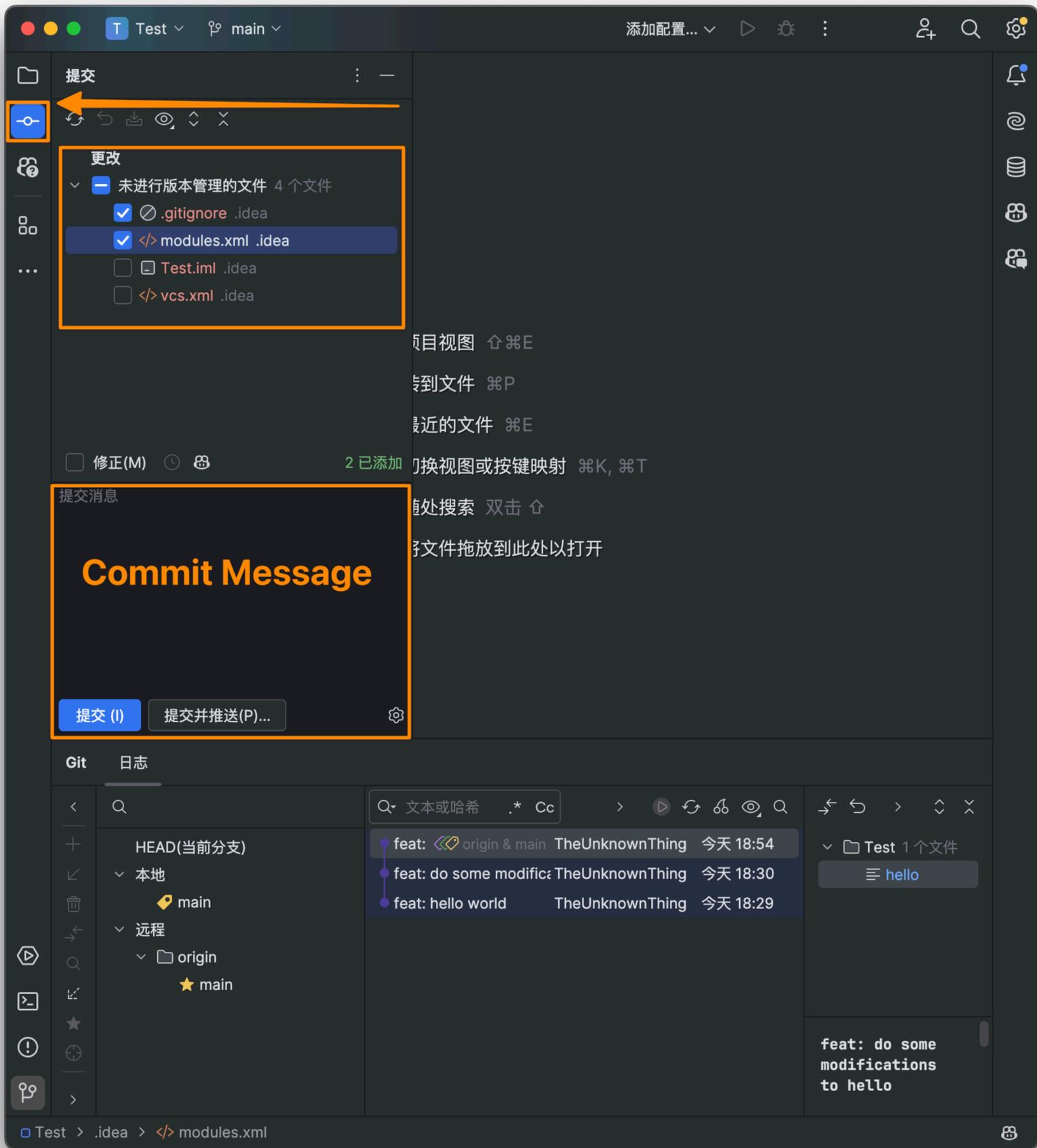
GRAPH

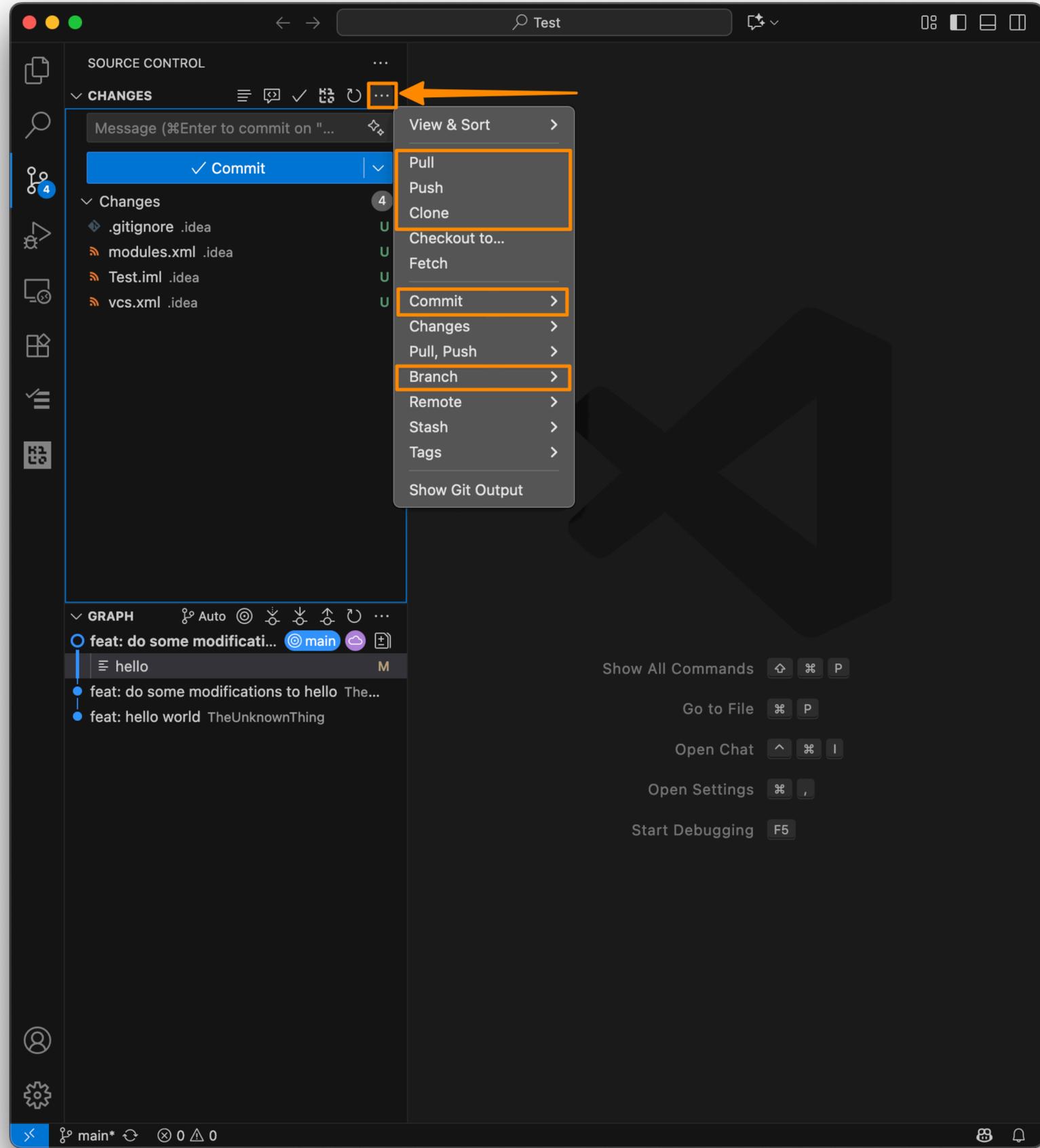
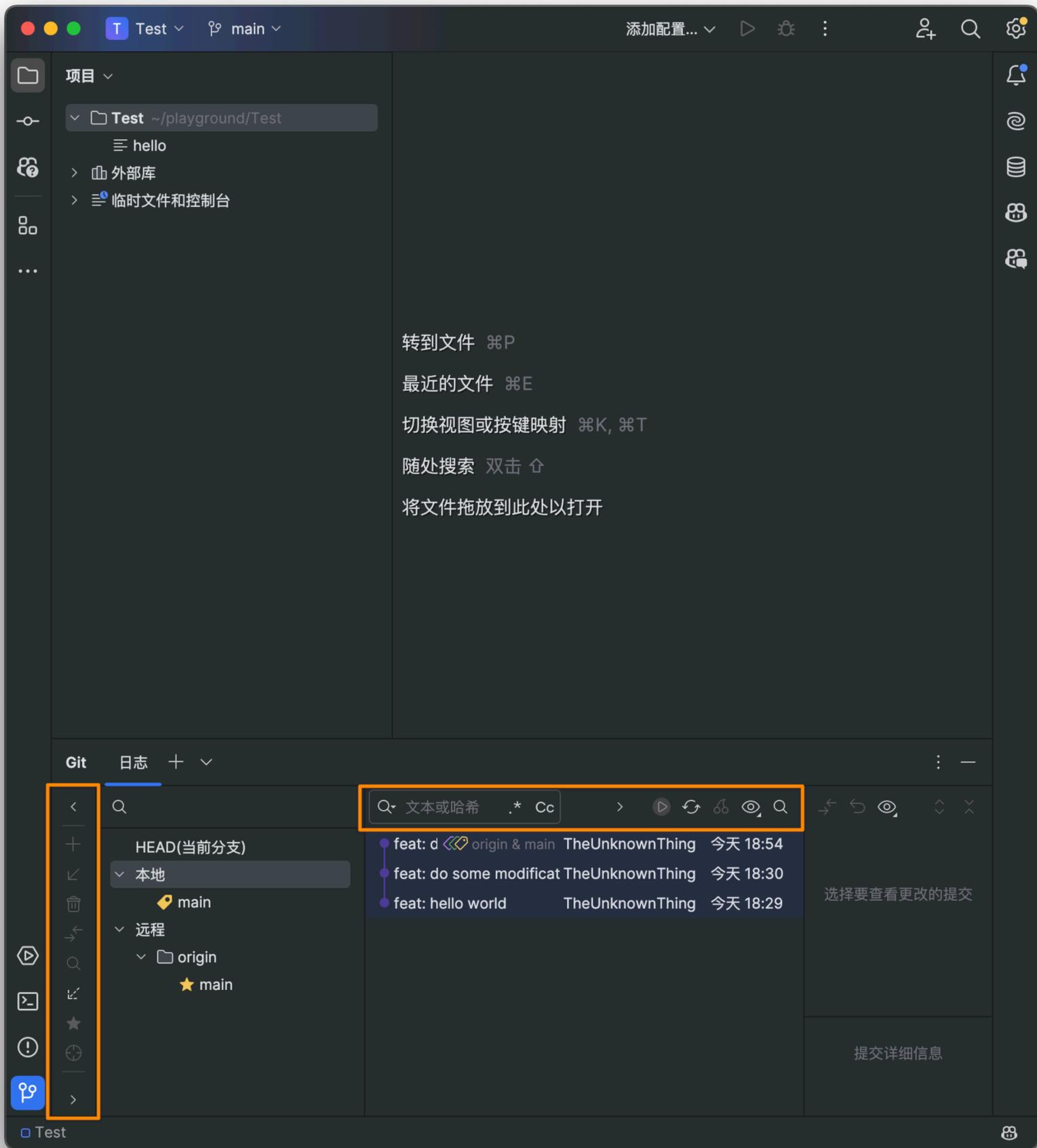
feat: do some modifications ... main

hello M

- feat: do some modifications to hello The...
- feat: hello world TheUnknownThing

main\* 0 0 TheUnknownThing (1 hour ago) Ln 3, Col 1 Spaces: 4 UTF-8 Plain Text Prettier





**“不要让使用 Git 成为你的负担。养成版本管理的好习惯非常重要，况且这不难。”**

我·自己说的

A vertical line on the left side of the slide features a yellow bracket that highlights the top section, which includes the main title and subtitle.

# 图形化操作 Git

## 怎么用 Git?

### 什么是 Git?

### 为什么要用 Git?



# GitHub

图形化操作 Git

怎么用 Git?

什么是 Git?

为什么要用 Git?



github.com

ACMClassCourse-2024 / Minesweeper-2024 ⭐ 3

Code Issues Pull requests 1

Minesweeper-2024 Public Watch 1 Fork 7 Starred 3

main Go to file Code

xiaoh105 Fix: Fix bugs in advanced.cpp 2aaf3f9 · last year

figures	initial commit	last year
src	Fix: Fix bugs in advanced.c...	last year
testcases	initial commit	last year
.clang-format	initial commit	last year
.gitignore	initial commit	last year
CMakeLists.txt	Upd: Add CMakeLists.txt	last year
LICENSE	initial commit	last year
README.md	initial commit	last year

Course project of Programming (CS1953, 2024 Fall), ACM Honor Class

- Readme
- MIT license
- Activity
- Custom properties
- 3 stars
- 1 watching
- 7 forks
- 1 year old

C++ 97.9% CMake 2.1%

Report repository

README MIT license

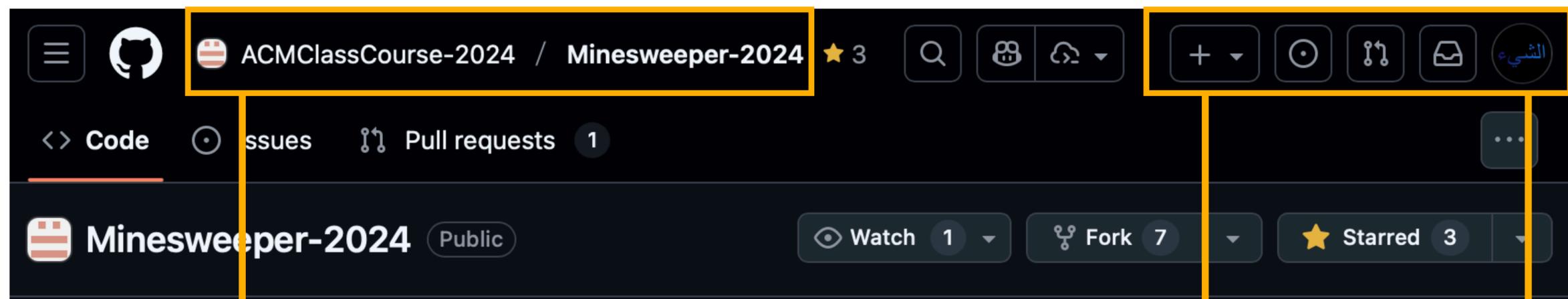
# Minesweeper-2024

ACM 班 2024 级程序设计第一次大作业

大作业 扫雷，启动!

## 目录

# GitHub 上你会使用...

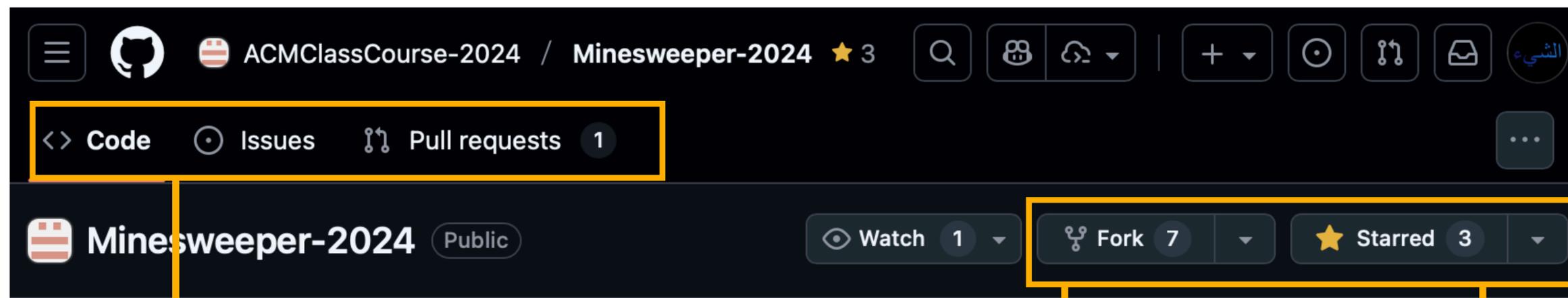


用户（组织）名 / 仓库名

创建仓库 / Issue

个人设置

# GitHub 上你会使用...



代码 / 议题 / 拉取请求

Fork 仓库

小星星

在你的账户中创建仓库副本

# GitHub 上你会使用...

The screenshot shows a GitHub repository interface with several key elements highlighted by yellow boxes:

- Branch selector:** A dropdown menu showing 'main' with a refresh icon.
- Repository description:** 'Course project of Programming (CS1953, 2024 Fall), ACM Honor Class'.
- Commit header:** 'xiaoh105 Fix: Fix bugs in advanced.cpp' with commit hash '2aaf3f9' and 'last year'.
- File list:** A table of files and folders with their commit history.
- Repository metadata:** 'Readme', 'MIT license', 'Activity', 'Custom properties', '3 stars', '1 watching', '7 forks', '1 year old', and a language usage bar for C++ (97.9%) and CMake (2.1%).

File/Folder	Commit	Time
figures	initial commit	last year
src	Fix: Fix bugs in advanced.c...	last year
testcases	initial commit	last year
.clang-format	initial commit	last year
.gitignore	initial commit	last year
CMakeLists.txt	Upd: Add CMakeLists.txt	last year
LICENSE	initial commit	last year

当前分支

仓库描述

提交记录

# GitHub 上你会使用...



README

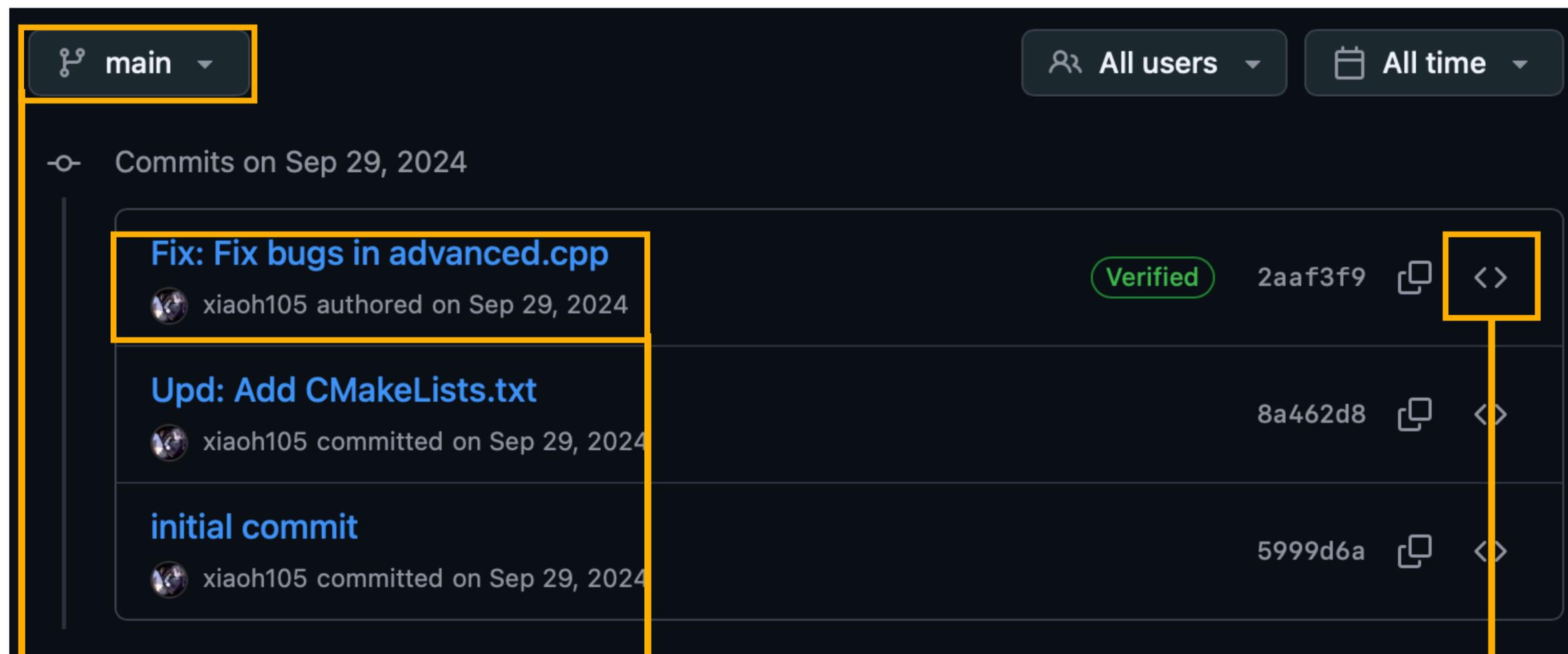
许可证

显示 README.md

如果你还不会 Markdown 的话:

<https://www.markdownguide.org/cheat-sheet/>

# GitHub 上你会使用...



当前分支

看提交的更改

看那时的整个代码仓库

# GitHub 上你会使用...

## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Required fields are marked with an asterisk (\*).

Owner \*

Choose an owner ▾

Repository name \*

Minesweeper-2024

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Course project of Programming (CS1953, 2024 Fall), ACM Honor Class

Copy the `main` branch only

Contribute back to ACMClassCourse-2024/Minesweeper-2024 by adding your own branch. [Learn more.](#)

Disable Projects and Wikis

After creating the repository disable the projects and wiki. [Suggestion by Refined GitHub.](#)

Create fork

你 Fork 后仓库的信息

创建你的Fork

# 发现项目 · 提出议题 · 贡献代码

The screenshot shows the GitHub repository page for 'tinode' (chat). The repository has 12.8k stars, 2k forks, and 301 watchers. The main content area displays the repository description: 'Instant messaging platform. Backend in Go. Clients: Swift iOS, Java Android, JS webapp, scriptable command line; chatbots'. Below the description are links for 'Contributing', 'Security policy', and 'Public repository'. A list of files and folders is visible, including: 'or-else remove debug printing', '.github/ISSUE\_TEMPLATE', 'chatbot', 'docker', 'docs', 'keygen', 'loadtest', 'monitoring', 'pbx', and 'py\_grpc'.

The screenshot shows a GitHub issue page for 'Subject of the issue' opened by 'rogerflowey (WuTong)' on Aug 4. The issue title is 'On the web client all json files sent cannot be received.' The 'Your environment' section is divided into 'Server-side' and 'Client-side'. The 'Server-side' section lists: 'web.tinode.co, api.tinode.co', 'sandbox.tinode.co', and 'Your own setup: Ubuntu 24.04, docker compose deployed, using mysql, default config.' The 'Client-side' section lists: 'TinodeWeb/tinodejs: javascript client'. The 'Steps to reproduce' section states: 'Simply send any json file. It can be sent but nothing is displayed at the receiving end. The same issue occurs when you rename any file to \*.json'. The 'Proposed Reason' section states: 'Perhaps this behavior is linked to json files being regarded as part of the control flow, see logs, where for json files: "mime":"application/json" is observed.' The 'Server-side log' section shows a log entry: 'I2025/08/04 13:42:04 in: '{"pub":{"id":"90373","topic":"usr3ZtncFE4iFk","noecho":'.

The screenshot shows a GitHub pull request page for 'fix:fix the render of "No examples provided." #6' by 'Seven-Streams (Lin Zhang Li)' on Apr 20. The pull request is merged and closed. The description of the pull request states: 'The extension will fail to render "No examples provided." correctly. Use `md.render()` may solve the problem.' The pull request is merged into 'TheUnknownThing:main' on Apr 20. A notification at the bottom states: 'Pull request successfully merged and closed. You're all set — the branch has been merged.' A note at the bottom indicates: 'This pull request first appeared in 0.5.1'.

# 装点门面

**TheUnknownThing**

Overview | Repositories 50 | Projects | Packages | Stars 66 | Gists

**Profile:** Hanning Wang, TheUnknownThing, Shanghai Jiao Tong University, Shanghai, China. Website: https://theunknown.site

**Pinned / Top repositories:**

- volcengine/verl** (Public): Python, 13.7k stars, 2.4k forks. Description: verl: Volcano Engine Reinforcement Learning for LLMs
- vscode-acmoj** (Public): TypeScript, 19 stars, 3 forks. Description: A simple integration of ACMOJ into VSCode, which allows you to browse problemsets (contests/homework), view problems, submit your code, and check submission results without leaving your editor.
- ACMClassOJ/TesutoHime** (Public): HTML, 29 stars, 4 forks. Description: テスト姫 (评测姬)
- QUIC-Transproxy** (Public): Go, 1 star. Description: A quic implementation of transparent proxy
- RCompiler** (Public): C++, 2 stars, 1 fork. Description: A compiler designed for the subset of Rust (https://github.com/peterzheng98/RCompiler-Spec).
- RISC-V-Simulator** (Public): C++, 2 stars, 1 fork. Description: A toy RISC-V simulator.

**Achievements:** 5 icons including GitHub, C++, and others.

**802 contributions in the last year:** Contribution grid showing activity from Nov to Sep.

**WXRIW / README .md**

**Hi there 🙌**

- I'm currently working on [Lyricify](#) (along with [Lyricify Lyrics Helper](#)) and [Ink Canvas](#)
- I'm currently learning C#
- How to reach me: [wxriw@foxmail.com](mailto:wxriw@foxmail.com)
- Secret of Life: People change people
- Business: Contact me via email

**XY Wang's GitHub Stats:** Total Stars Earned: 6.6k, Total Commits (last year): 188, Total PRs: 20, Total Issues: 21, Contributed to (last year): 2. Rating: **B+**

**Profile:** XY Wang, WXRIW, Nanjing, Anhui, China. 473 followers, 8 following.

**Pinned / Top repositories:**

- Lyricify-App** (Public): 6k stars, 109 forks. Description: Lyricify (لتريسافاز), a fantastic app to provide scroll lyrics for Spotify and other apps. 一款为 Spotify 等各种应用提供滚动歌词的软件。
- Ink-Canvas** (Public): C#, 303 stars, 66 forks. Description: A fantastic Ink Canvas in WPF/C#, with fantastic support for Seewo Boards.
- Lyricify-Lyrics-Helper** (Public): C#, 128 stars, 13 forks. Description: 集成 Lyricify 所需的歌词相关功能

**Achievements:** 5 icons including GitHub, C#, and others.

**589 contributions in the last year:** Contribution grid showing activity from Sep 2024 to Sep 2021.



# GitHub

图形化操作 Git

怎么用 Git?

什么是 Git?

为什么要用 Git?



- Dojo 
- GitHub
- 图形化操作 Git
- 怎么用 Git?
- 什么是 Git?
- 为什么要用 Git?



ACM | DOJO

# Reference

<http://ohshitgit.com/>

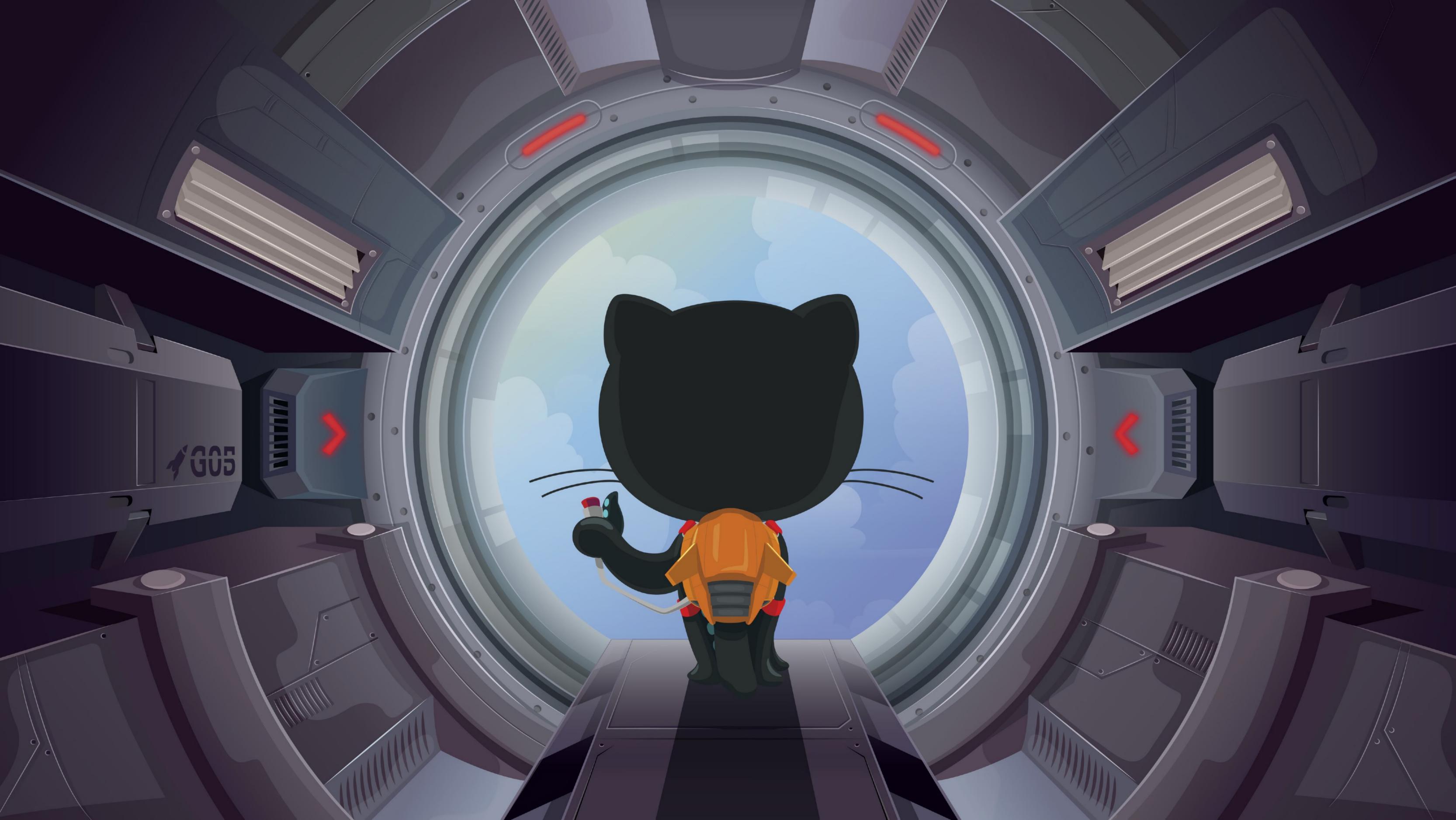
<https://git-scm.com/docs>

<http://merriam-webster.com/dictionary/git>

<https://www.wolai.com/zymbox/gYBwBGsBpfqnrnsXFiUBqFo>

<https://lau.yeeyu.org/blog-zh-cn/git-basic-usage-zh-cn/>

<https://missing-semester-cn.github.io/2020/version-control>



G05