

Git 版控系统

Git Version Control System

October 19, 2025

WHUAI Team

Ostrichbeta Chan 陈逸铭

1. Git 简介

2. Git 基本操作

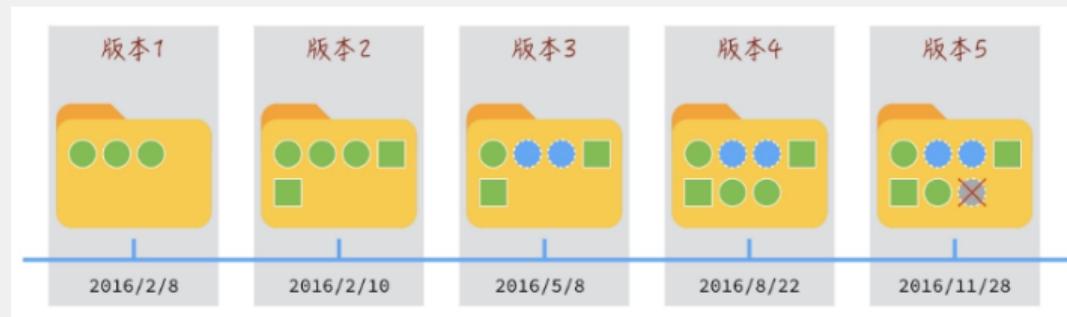
3. Git 远端联动

4. 练习

5. 附录

为什么我们要学习版控系统？

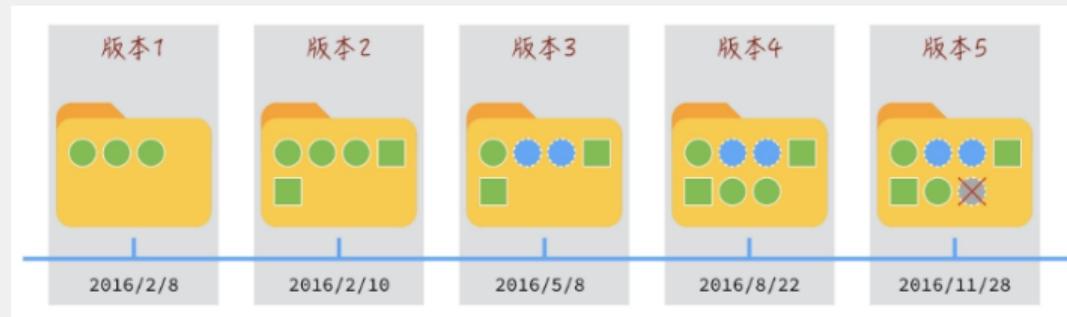
若没有版控系统，备份不同的版本，只能一个版本一个资料夹：



- Chan 基于版本 5 开发了版本 6
- Lee 由版本 4 修复了一些 Bug 希望同步到最新开发中
- Yang 希望能够快速的提炼出不同版本之间的区别给 Chang 老师汇报进度

为什么我们要学习版控系统？

若没有版控系统，备份不同的版本，只能一个版本一个资料夹：



- Chan 基于版本 5 开发了版本 6
- Lee 由版本 4 修复了一些 Bug 希望同步到最新开发中
- Yang 希望能够快速的提炼出不同版本之间的区别给 Chang 老师汇报进度

没有一个快速高效的方法，三个人都很头大

Git 的优点

开发管理 Linux 核心的 Linus Torvalds 在 2005 年也遇到了这个问题，加之当时版控系统不尽完善（有的还死贵），于是他花了 10 天时间完成了 Git 的初版并开源，经过多年发展，Git 系统已经十分完善。

Git 的优点

开发管理 Linux 核心的 Linus Torvalds 在 2005 年也遇到了这个问题，加之当时版控系统不尽完善（有的还死贵），于是他花了 10 天时间完成了 Git 的初版并开源，经过多年发展，Git 系统已经十分完善。

Git 有很多优点：

- **开放原始码** 可以免费用，且有社群维护
- **快速高效** Git 可以快速在不同版本间切换，特殊的设计使多版本下的目录的空间占用降低
- **分散式系统** Git 不依赖伺服器，可以不受网路影响的使用
- **易上手** 虽然 Git 有众多指令，但大概 20% 的指令可以完成 80% 的工作

Git 的安装

- Linux：使用各发行版的包管理器（apt, dnf, yum）直接安装或由原始码编译后安装
- macOS：安装 Xcode Command Line Tools 过程中自动安装
- Windows：由 Git 官网下载安装档安装

Git 官方网站 <https://git-scm.com/> 提供了 Windows, macOS 和 Linux 的安装档

预备知识

- git 预设的编辑器是 vim，可以了解 vim 的常用快速键，或使用 `git config --global core.editor nano` 改用更友善的 nano 编辑器。
- Bash 中可以在输入指令的过程中按 Tab 键自动补全
- Bash 中可使用上下方向键快速填入历史指令
- Bash 和 Zsh 中用 Ctrl-A 跳至行头，Ctrl-E 跳至行尾，Ctrl-C 终止当前程式，Ctrl-L 清除萤幕
- 在终端机中，可以使用 Shift-Ctrl-C 复制，Shift-Ctrl-V 貼上
- vim 的预设退出方法是按下 Esc 使 -- INSERT -- 字样消失，然后输入`:q` 退出

内容约定

- 以下的 git 指令大多数在 git v2.39 版本下于 macOS 15.1 下执行
- 指令中出现尖括弧的部分如 <text> 为指令中必须填入的项
- 指令中出现方括弧的部分如 [text] 为指令中的可选项
- 除非特别说明，否则**所有的括弧符号都不需要输入**

1. Git 简介

2. Git 基本操作

3. Git 远端联动

4. 练习

5. 附录

1. Git 简介

2. Git 基本操作

2.1 Git 本机工作流程

2.2 环境设定

2.3 Git 主要指令

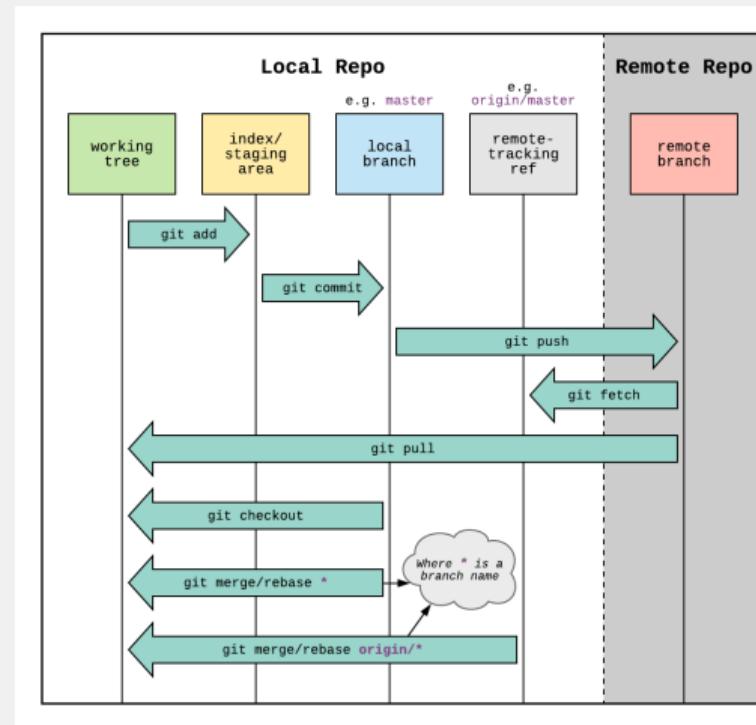
2.4 其他指令

3. Git 远端联动

4. 练习

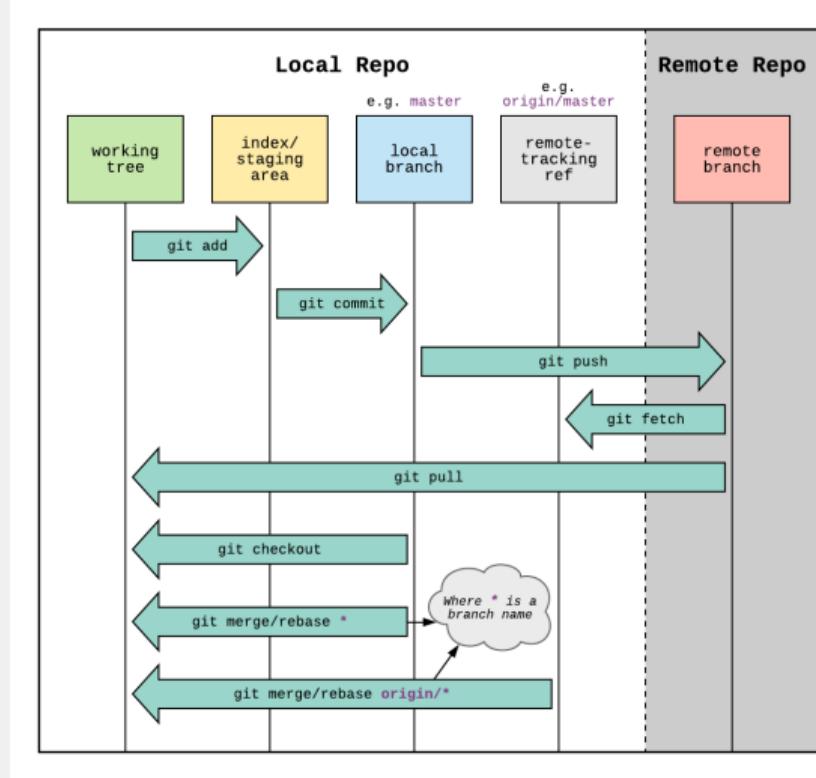
5. 附录

Git 的主要工作流程



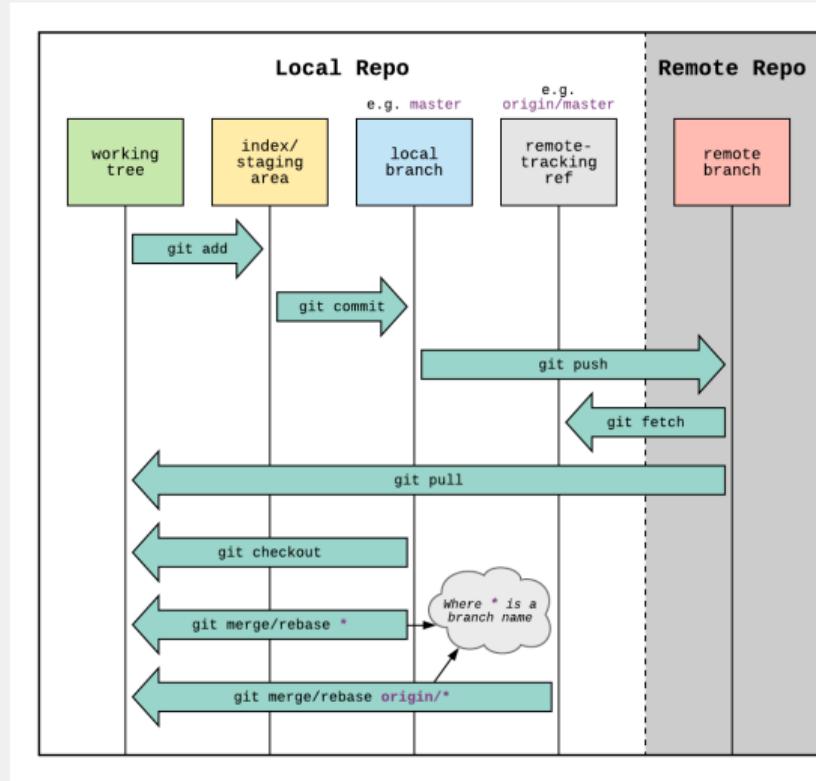
Git 流程简图

Git 的主要工作流程



Git 的主要工作流程有以下部分：

Git 的主要工作流程



Git 的主要工作流程有以下部分：

- 将文件加至 Git 版控
- 提交文件的变更
- 推送 commit 到远端仓库
- 取得远端代码变更
- 切换分支
- 合并不同分支
- etc.

Git 的主要工作流程

最简单的 Git 流程：

- 1 初始设定工作区
- 2 新增文件至版控系统
- 3 提交文件的修改

如果需要与远端仓库沟通：

- 4 将修改推至云端

1. Git 简介

2. Git 基本操作

2.1 Git 本机工作流程

2.2 环境设定

2.3 Git 主要指令

2.4 其他指令

3. Git 远端联动

4. 练习

5. 附录

设定个人资讯

使用 Git 的第一件事是设定我们的个人资讯，方便 Git 记录代码的提交者。

可以使用 `git config` 指令进行设定：

```
git config --global user.name <Name>
git config --global user.email <email>
```

执行后，可使用以下指令确认：

```
git config --global user.name
git config --global user.email
```

设定个人资讯

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:27:40]
[$ git config --global user.name "Ostrichbeta Chan"

# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:27:46]
[$ git config --global user.email "ostrichb@outlook.com"

# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:27:47]
[$ git config --global user.name
Ostrichbeta Chan

# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:27:49]
[$ git config --global user.email
ostrichb@outlook.com
```

执行范例

设定个人资讯

使用 `git config --list` 还可以检视当前 Git 的所有设定。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:06:50]
$ git --no-pager config --list
credential.helper=osxkeychain
init.defaultbranch=main
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
remote.origin.url=https://github.com/Ostrichbeta/whuai-git-test.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.main.remote=origin
branch.main.merge=refs/heads/main
```

1. Git 简介

2. Git 基本操作

2.1 Git 本机工作流程

2.2 环境设定

2.3 Git 主要指令

2.4 其他指令

3. Git 远端联动

4. 练习

5. 附录

本地工作常用指令

本章重点讲解以下指令：

- git init
- git add
- git commit
- git reset
- git branch
- git checkout
- git merge
- git status
- git log

新增仓库

git init

在工作空间中使用 `git init` 可以将当前目录交由 Git 接管。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test [14:03:59]
$ git init
Initialized empty Git repository in /Users/ostrichb/Documents/Projects/whuai-git-test/.git/
```

新增仓库

git init

在工作空间中使用 `git init` 可以将当前目录交由 Git 接管。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test [14:03:59]
$ git init
Initialized empty Git repository in /Users/ostrichb/Documents/Projects/whuai-git-test/.git/
```

- 一个 `.git` 目录会被新增，此为 Git 的核心
- 移除这个 `.git` 目录将会移除 Git 的控制

整个专案目录里，什么档案或目录删了都救得回来，但 `.git` 目录只要删了就没办法了。

状态查询

git status

取得当前工作空间的状态，是否与远端同步，档案的提交状态等。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main ✘ [14:04:02]
[$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LICENSE
    README.md
    a.txt
    convert.csv
    homo
    script.js
    token.txt

nothing added to commit but untracked files present (use "git add" to track)
```

状态查询

git status

取得当前工作空间的状态，是否与远端同步，档案的提交状态等。

```
[# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main o [14:06:40]
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

已与远端同步，且未有未提交的档案

往 Git 中新增档案

git add

将某些档案/目录加至 staging area (暂存区)，为提交作准备。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main x [14:24:04]
[$ git status
On branch main

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
    new file:   LICENSE

Untracked files:
(use "git add <file>..." to include in what will be committed)
    README.md
    a.txt
    convert.csv
    homo
    script.js
    token.txt
```

往 Git 中新增档案

git add

将某些档案/目录加至 staging area (暂存区)，为提交作准备。

- 使用 `git add .` 可以将当前目录所有档案加至暂存区
- 若档案有修改，需要再执行一次 `git add` 指令暂存变更
- 空目录是无法提交的
- 使用 `git diff` 可详细的查询档案具体的变更内容

往 Git 中新增档案

git add

将某些档案/目录加至 staging area (暂存区)，为提交作准备。

- 使用 `git add .` 可以将当前目录所有档案加至暂存区
- 若档案有修改，需要再执行一次 `git add` 指令暂存变更
- 空目录是无法提交的
- 使用 `git diff` 可详细的查询档案具体的变更内容

那要怎么移除档案？

- 用 `rm` 移除档案，然后 `git add` 追加变更至暂存区
- 或使用 `git rm` 指令完成上述两步

往 Git 中提交档案

git commit

将暂存区中的档案提交本地仓库存档。

```
git commit [-m <comment>]
```

<comment> 为这一次提交的内容简括，是代码沟通的重要资讯，以下是一些良好的范例：

```
git commit -m "feat(auth): add JWT-based authentication"
git commit -m "fix(login): resolve race condition in login flow"
```

...以及部分反面示范：

```
git commit -m "Update project"
git commit -m "Update readme and fix login issue"
```

往 Git 中提交档案

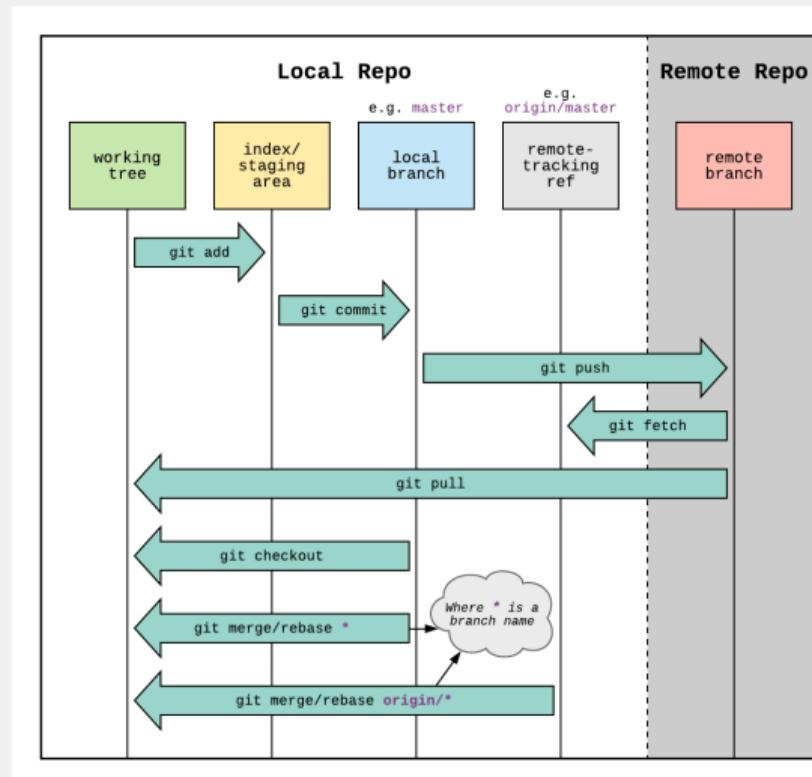
git commit

将暂存区中的档案提交本地仓库存档。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main ✘ [14:32:14]
|$ git commit -m "Init"
[main (root-commit) 3dfd2b1] Init
 6 files changed, 895 insertions(+)
  create mode 100644 LICENSE
  create mode 100644 README.md
  create mode 100644 a.txt
  create mode 100644 convert.csv
  create mode 100644 script.js
  create mode 100644 token.txt
```

- 未加入暂存区的档案将**不会被提交**
- 提交内容信息是必须的，若不带 `-m` 会弹出文字编辑器，需在其中加入文字
- 使用 `--amend` 不会新增 commit 而是追加到上一个 commit 中

往 Git 中提交档案



Always remember to git add !



忽略部分档案

.gitignore

一些仓库中会含有.gitignore 档，.gitignore 中记录的档案将不会被 Git 进行版本控制。

仓库中由编译生成的档案，或者是开发中所用的敏感存取密匙等通常都可加入.gitignore 中。

有以下几种格式：

```
/src/ # 忽略工作目录下最上层的src目录  
src/ # 忽略所有名为src的目录、子目录  
a.txt # 忽略所有名为 a.txt 的档案  
*.so # 忽略所有附档名为 *.so 的档案  
!homo.so # 强制保留名为 homo.so 的档案  
/ros2/**/lah/ # ros2开头 launch结尾的目录/ros2/lah/和/ros2/s/s/lah/都符合
```

设定好之后将 .gitignore 使用 git add 新增，其规则就会对未来档案生效。

GitHub 官方也整理了一些环境和语言下常用的.gitignore 档案集：

<https://github.com/github/gitignore>

忽略部分档案

.gitignore

.gitignore 设定的规则只对设定规则之后出现的档案有效，已经存在但符合.gitignore 规则的档案仍然会被 Git 控制。

如何将他们移出版控而不删去档案？

忽略部分档案

.gitignore

.gitignore 设定的规则只对设定规则之后出现的档案有效，已经存在但符合.gitignore 规则的档案仍然会被 Git 控制。

如何将他们移出版控而不删去档案？

```
git rm -r --cached . # 由版控中移除所有档案  
git add . # 新增所有档案，但是将忽略 .gitignore 规则中的档案  
git commit -am "Remove ignored files"
```

具体可以参考

<https://stackoverflow.com/questions/1274057/how-do-i-make-git-forget-about-a-file-that-was-tracked-but-is-now-in-gitignore>

检视提交记录

git log

检视过去的 commit 记录，由新至旧排列。

```
commit 09d8ceddf29add13289994560a33fe845323423 (HEAD -> main, origin/main, origin/HEAD)
Merge: 6acbb57 5cc69cb
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Thu May 16 05:48:20 2024 +0000

  Merge pull request #13 from WHUAI-Dog/image-server-dev

  Image server dev

commit 5cc69cb2c30d9d48d636051b83a4d57ed325cf60 (origin/image-server-dev)
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Thu May 16 05:34:54 2024 +0000

  added ball kicking

commit c882c21db16e4077f2bce39287d533d21ab3ee5d
Merge: ae50452 2b896d5
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Thu May 16 05:33:59 2024 +0000

  Merge branch 'image-server-dev' of https://github.com/WHUAI-Dog/tOS-Dog into image-server-dev

commit ae50452888f5830a7c651bf049e7a4df41904bbe
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Thu May 16 05:33:04 2024 +0000

  added override options

commit 2b896d553cf9a4e13ed8ca88462d25e2714642ee
:|
```

Log 预设会显示：

- 当前分支位置
- commit 编号
- commit 时间
- commit 用户

检视提交记录

git log

检视过去的 commit 记录，由新至旧排列。

- 使用 `--oneline` 可以更紧凑的显示记录
- 使用 `--graph` 可以加入分支线显示

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:48:13]
$ git --no-pager log --oneline
09d8ced (HEAD -> main, origin/main, origin/HEAD) Merge pull request #13 from WHUAI-Dog/image-server-dev
5cc69cb (origin/image-server-dev) added ball kicking
c882c21 Merge branch 'image-server-dev' of https://github.com/WHUAI-Dog/tOS-Dog
into image-server-dev
ae50452 added override options
2b896d5 Merge pull request #12 from WHUAI-Dog/main
362e2e6 Figure fix
6acbb57 Merge pull request #11 from WHUAI-Dog/image-server-dev
f9814ea added turnaround to football
0967be3 modified box detection
1c2a8aa Added box detect sound
b9fe80b Box detection
c795672 Box detection alpha
c6b3e9d Added gdelta to traverse three boxes, push to VM for further coding
98370ba Merge pull request #10 from WHUAI-Dog/image-server-dev
888e058 Add cones, signals and small_figures
cb12a6c (origin/odom-navi) Changed yaml points
bec2411 Test for yaml code
7779ae2 Merge pull request #9 from WHUAI-Dog/odom-navi
c5c21a7 Pos fix
```

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:48:36]
$ git --no-pager log --graph
* commit 09d8ced9f29add13289994560a33fe845323423 (HEAD -> main, origin/main,
origin/HEAD)
| \ Merge: 6acbb57 5cc69cb
| | Author: Ostrichbeta Chan <ostrichb@outlook.com>
| | Date: Thu May 16 05:48:20 2024 +0000
|
|   Merge pull request #13 from WHUAI-Dog/image-server-dev
|
|   Image server dev
|
* commit 5cc69cb2c30d9d48d636051b83a4d57ed325cf60 (origin/image-server-dev)
| Author: Ostrichbeta Chan <ostrichb@outlook.com>
| Date: Thu May 16 05:34:54 2024 +0000
|
|   added ball kicking
|
* commit c882c21db16e4077f2bce39287d533d21ab3ee5d
| \ Merge: ae50452 2b896d5
| | Author: Ostrichbeta Chan <ostrichb@outlook.com>
| | Date: Thu May 16 05:33:59 2024 +0000
|
|   Merge branch 'image-server-dev' of https://github.com/WHUAI-Dog/tOS-Dog
into image-server-dev
```

退回更新

git reset

可以将工作空间或目录还原到之前某一次提交时的状态。

```
git reset 0967be3 # Commit 号  
git reset 0967be3^  
git reset HEAD~2  
git reset main~4
```

退回更新

git reset

可以将工作空间或目录还原到之前某一次提交时的状态。

```
git reset 0967be3 # Commit 号  
git reset 0967be3^  
git reset HEAD~2  
git reset main~4
```

- 每一个 ^ 符号表示前一次，~ 表示前 n 次。如 HEAD~2 表示 HEAD 的前两次。
- HEAD 表示当前所在的分支状态，而 main 则指向 main 分支的最新状态。

退回更新

git reset

git reset 主要有三种工作模式。

```
git reset --mixed 0967be3 #若不带参数，预设就是mixed  
git reset --soft 0967be3  
git reset --hard 0967be3
```

退回更新

git reset

git reset 主要有三种工作模式。

```
git reset --mixed 0967be3 #若不带参数，预设就是mixed  
git reset --soft 0967be3  
git reset --hard 0967be3
```

- Mixed 是预设的工作模式，不改变工作目录，但将档案退回到未追踪的状态；
- Soft 不改变工作目录，将档案退回到暂存区；
- Hard 改变工作目录和暂存区，将档案复原到目标分支的状态（硬复原）。

退回更新 - git reset --mixed

- Mixed 是预设的工作模式，不改变工作目录，但将档案退回到未追踪的状态；

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:50:11]
[$ git reset 0967be3
Unstaged changes after reset:
M     src/tos/config/config.yaml
M     src/tos/launch/tos_core.launch
M     src/tos/src/image_server/image_server.cpp
M     src/tos/src/tos_core.cpp

# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main x [1:51:28]
[$ git status
On branch main
Your branch is behind 'origin/main' by 10 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   src/tos/config/config.yaml
    modified:   src/tos/launch/tos_core.launch
    modified:   src/tos/src/image_server/image_server.cpp
    modified:   src/tos/src/tos_core.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    src/tos/config/override.yaml

no changes added to commit (use "git add" and/or "git commit -a")
```

退回更新 - git reset --soft

- Soft 不改变工作目录，将档案退回到暂存区；

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:53:19] C
:130
[$ git reset --soft 0967be3
]
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main x [1:53:27]
[$ git status
On branch main
Your branch is behind 'origin/main' by 10 commits, and can be fast-forwarded
.
  (use "git pull" to update your local branch)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   src/tos/config/config.yaml
    new file:   src/tos/config/override.yaml
    modified:   src/tos/launch/tos_core.launch
    modified:   src/tos/src/image_server/image_server.cpp
    modified:   src/tos/src/tos_core.cpp
```

退回更新 - git reset --hard

- Hard 改变工作目录和暂存区，将档案复原到目标分支的状态（硬复原）。

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:53:59]
[$ git reset --hard 0967be3
HEAD is now at 0967be3 modified box detection

# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:54:10]
[$ git status
On branch main
Your branch is behind 'origin/main' by 10 commits, and can be fast-forwarded
.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

退回更新

Hard 复原之后，还可以回到没有执行 reset 之前的状态吗？

退回更新

Hard 复原之后，还可以回到没有执行 reset 之前的状态吗？

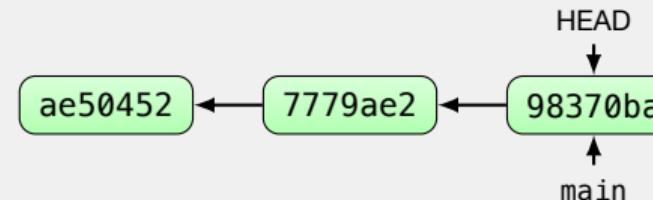
可以！

- 使用 git reflog 指令找出所需要的分支
- 使用 git reset --hard <commit> 复原到目标分支

```
# ostrichb @ Poppi in ~/Documents/Projects/tOS-Dog on git:main o [1:55:01]
$ git --no-pager reflog
09d8ced (HEAD -> main, origin/main, origin/HEAD) HEAD@{0}: reset: moving to 09d8ced
09d8ced (HEAD -> main, origin/main, origin/HEAD) HEAD@{1}: reset: moving to 09d8ced
0967be3 HEAD@{2}: reset: moving to 0967be3
09d8ced (HEAD -> main, origin/main, origin/HEAD) HEAD@{3}: reset: moving to 09d8ced
0967be3 HEAD@{4}: reset: moving to 0967be3
09d8ced (HEAD -> main, origin/main, origin/HEAD) HEAD@{5}: reset: moving to 09d8ced
0967be3 HEAD@{6}: reset: moving to HEAD
0967be3 HEAD@{7}: reset: moving to 0967be3
09d8ced (HEAD -> main, origin/main, origin/HEAD) HEAD@{8}: clone: from https://github
.com/WHUAI-Dog/tOS-Dog.git
```

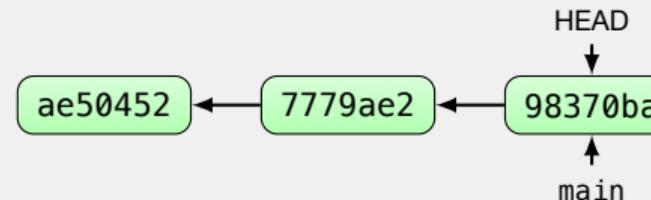
Commit 的更新流程

一开始的时候假设有三个 commit :

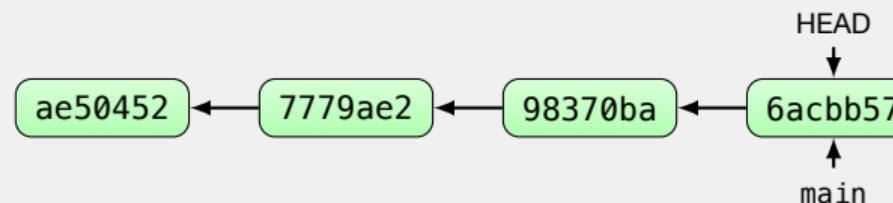


Commit 的更新流程

一开始的时候假设有三个 commit :

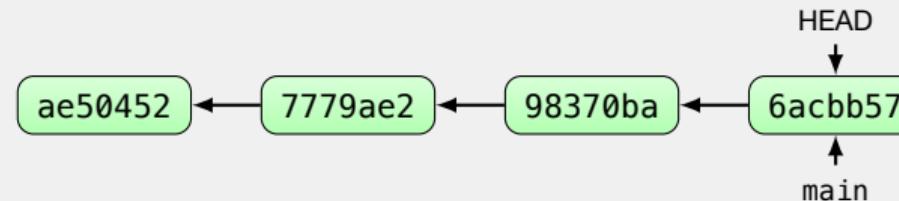


然后不小心错误的提交了一个不希望的 commit :



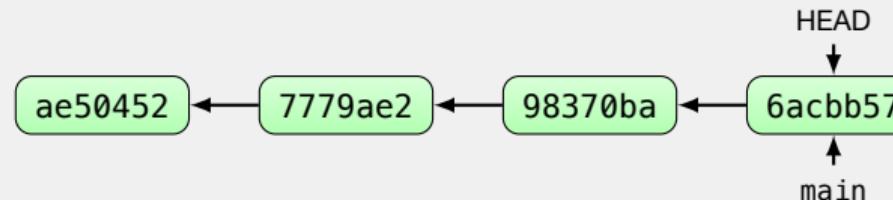
Commit 的更新流程

然后不小心错误的提交了一个不希望的 commit :

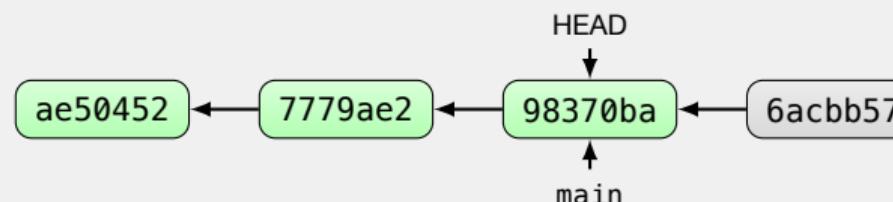


Commit 的更新流程

然后不小心错误的提交了一个不希望的 commit :

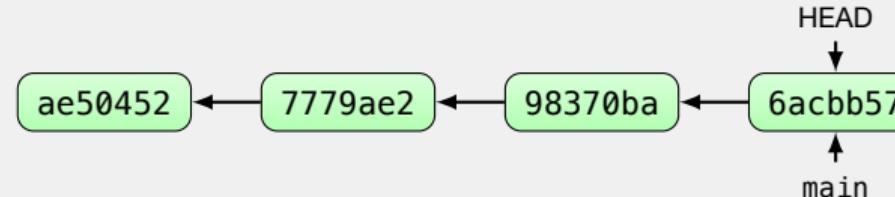


此时执行一次 `git reset --hard` :

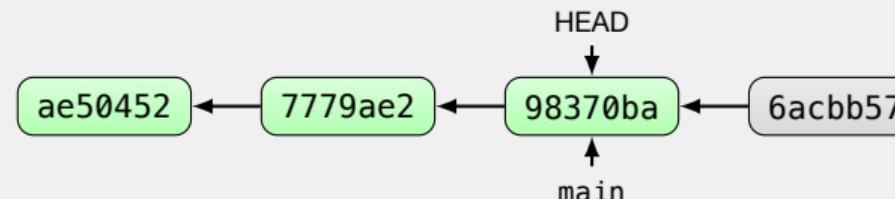


Commit 的更新流程

然后不小心错误的提交了一个不希望的 commit :

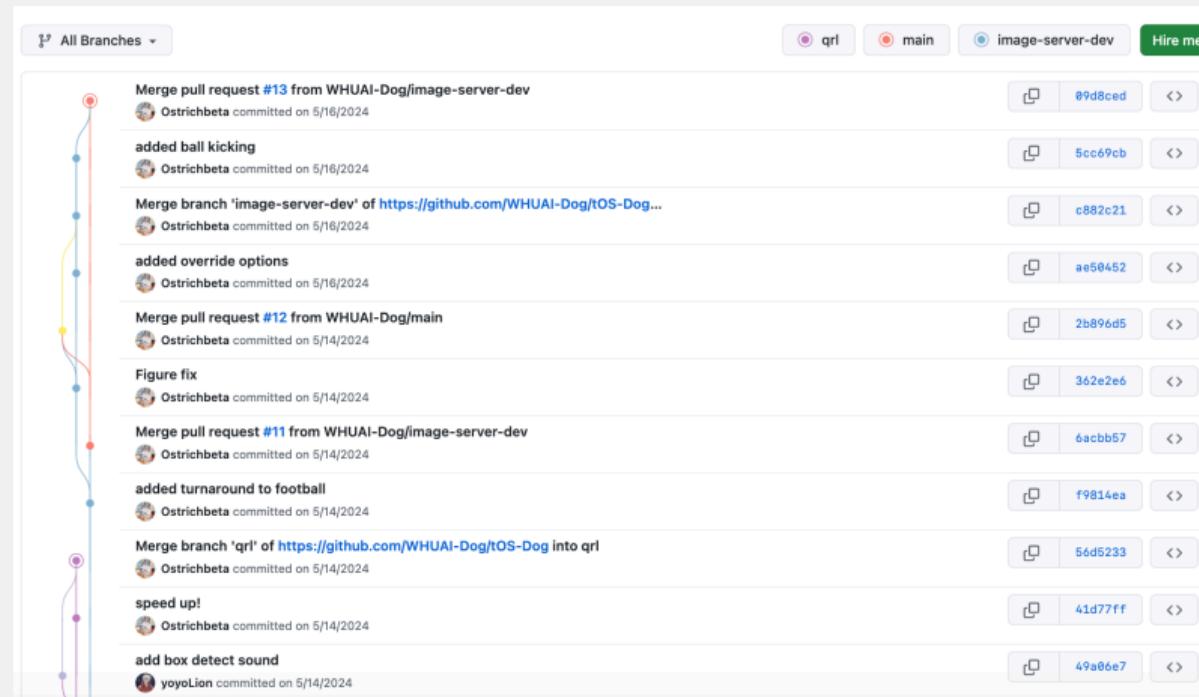


此时执行一次 `git reset --hard` :



`git reset --hard` 仅仅是跳至了已经储存在过去 / 未来的某个「状态」，所以可以快速复原

分支

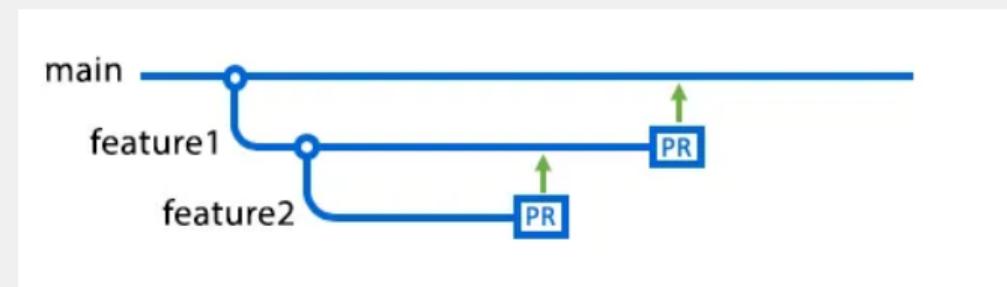


多分支开发范例

分支

在开发的过程中，一路往前 Commit 也没什么问题，但当开始越来越多同伴一起在同一个专案工作的时候，可能就不能这么随兴的想 Commit 就 Commit，这时候分支就很好用。例如想要增加新功能，或是**修正 Bug**，或是想**实验看看某些新的做法**，都可以另外做一个分支来进行，待做完确认没问题之后再合并回来，不会影响正在运行的产品线。

分支还可以方便的合并，就算不同分支之间可能存在冲突，Git 也会给出解决冲突的方法。



管理分支

git branch

- `git branch` : 检视当前本机存在的分支
- `git branch -a` : 检视本机和远端所有的分支 (仅远端可用`-r`)

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:12
:18]
$ git --no-pager branch
* main

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:12
:23]
$ git --no-pager branch -r
origin/HEAD -> origin/main
origin/image-server-dev
origin/line-follow
origin/main
origin/odom-navi
origin/param_yaml
origin/qrl
origin/revert-6-main
```

--no-pager 直接输出而不开启文字编辑器

管理分支

git branch

- git branch -m <old> <new> : 将 <old> 分支改名为 <new> 分支
- git branch <branch> : 新增分支 <branch>
- git branch -d <branch> : 移除分支 <branch>

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:22
:56]
[$ git branch newui
]

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:23
:33]
[$ git --no-pager branch
* main
  newui
```

分支切换

git checkout

- git checkout <branch> : 跳到 <branch> 分支
- git checkout -b <branch> : 跳到 <branch> 分支，若分支不存在则新增

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:35
:01]
[$ git checkout home
error: pathspec 'home' did not match any file(s) known to git
]

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:main o [7:35
:22] C:1
[$ git checkout -b home
Switched to a new branch 'home'

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/tOS-Dog on git:home o [7:35
:28]
$ ]
```

分支切换

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[4:34:50]
$ git checkout -b newui
Switched to a new branch 'newui'

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui o
[7:41:40]
$ ls
LICENSE      README.md      convert.csv script.js

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui o
[7:41:40]
$ touch praise.js

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui x
[7:41:48]
$ git add .

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui x
[7:41:54]
$ git commit -m "Add UI randering engine"
[newui 8e752de] Add UI randering engine
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 praise.js
```

分支切换

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui
o [7:42:12]
$ git --no-pager log
commit 8e752de5e53989ac105735d1f6bc884c68aa4701 (HEAD -> newui)
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Tue Oct 22 07:42:10 2024 +0000

    Add UI rendering engine

commit cbfc8d4dedb5e55561450a16f67fed82eed10ddf (main)
Author: Ostrichbeta Chan <ostrichb@outlook.com>
Date:   Tue Oct 22 01:35:17 2024 +0000

    Init
```

Commit 之后原分支没有改动，HEAD 指向当前的 newui 分支

分支合并

git merge

- git merge [-m <comment>] <branch> : 将 <branch> 分支往当前分支合并

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:newui
o [8:02:36]
[$ git checkout main
Switched to branch 'main'

# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[8:02:44]
[$ git merge newui
Updating cbfc8d4..8e752de
Fast-forward
praise.js | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 praise.js
```

由于分出 newui 后 main 没有新的 commit，合并时可以直接快转 (fast-forward)

分支合并

git merge

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[8:02:50]
$ git --no-pager log --oneline
8e752de (HEAD -> main, newui) Add UI randering engine
cbfc8d4 Init
```

合并后 HEAD, main 和 newui 都指向同一个 commit

分支合并

无冲突的合并

一开始 newui 由 main 分支分出去，然后在 newui 中新增两个档案：

```
git checkout newui
touch auth.js
touch ui.html
git add .
git commit -m "Created UI file"
```

main 中新增一个档案：

```
git checkout main
touch api.js
git add .
git commit -m "Included API entrypoint"
```

分支合并

现在分支树长得像这样：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[8:29:37]
$ git --no-pager log --oneline --all --graph
* 7e47655 (HEAD -> main) Included API entrypoint
| * 70c292c (newui) Created UI file
|/
* 8e752de Add UI randering engine
* cbfc8d4 Init
```

由于两边互不冲突也可以直接合并：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[8:29:40]
$ git merge newui
Merge made by the 'ort' strategy.
auth.js | 0
ui.html | 0
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 auth.js
create mode 100644 ui.html
```

分支合并

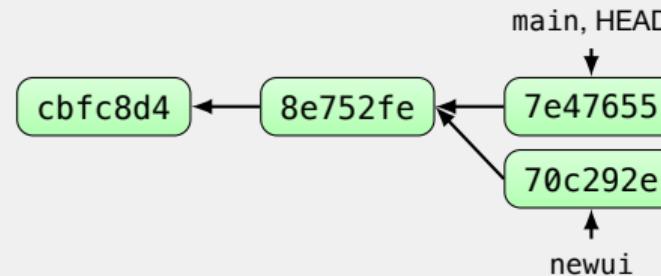
此时的 newui 是什么状态？

由于只是由 newui 并入 main 分支，main 分支包含了 newui 分支的变动（即 auth.js 和 ui.html）。

但是 newui 分支不会包含 main 在分出 newui 后的那一部分改变（即没有 api.js）。

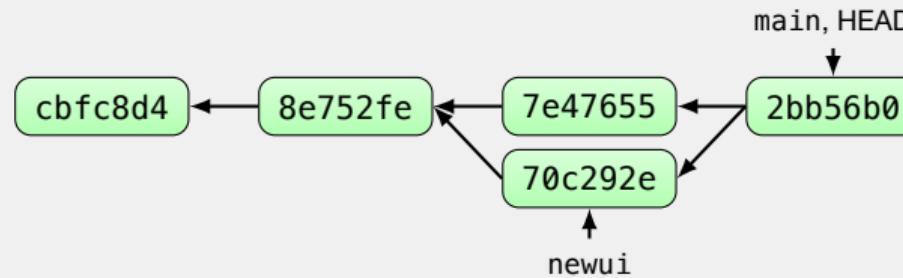
若要将 HEAD, main 和 newui 指向同一个 commit，可以切到 newui 后 merge main 分支。

分支合并



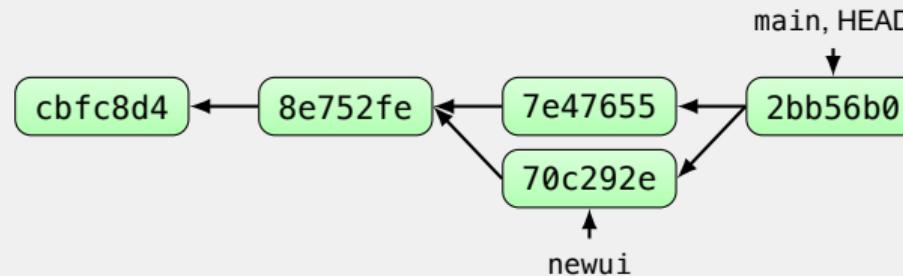
合并之前，两个分支指向各自分隔的 commit

分支合并



合并之后，`main` 指向合并后的 commit，但 `newui` 仍然指向原来合并之前的 commit

分支合并



合并之后，`main` 指向合并后的 commit，但 `newui` 仍然指向原来合并之前的 commit

分支本质上只是指向一个 **commit** 的指标

分支合并

有冲突合并

假设 main 分支下有一档案 EULA.txt :

```
This software is licensed.  
Created by:
```

此时分出分支 EULA，更改了其内容并 commit 如下：

```
This software is licensed.  
Created by:  
Jeder Chan
```

分支合并

有冲突合并

同时 main 分支中也更改了其内容：

```
This software is licensed.  
Created by:  
Ostrichbeta Chan
```

尝试进行合并，会提示自动合并失败，发生了内容冲突：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o  
[11:27:11]  
[$ git merge EULA  
Auto-merging EULA.txt  
CONFLICT (content): Merge conflict in EULA.txt  
Automatic merge failed; fix conflicts and then commit the result.
```

分支合并

有冲突合并

此时开启 EULA.txt，可以看到 Git 标记出了冲突处。

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main x
[11:27:24] C:1
[$ cat EULA.txt
This software is licensed.
<<<<< HEAD
Created by:
Ostrichbeta Chan
=====
Created by:
Jeder Chan
>>>>> EULA
```

分支合并

有冲突合并

此时开启 EULA.txt，可以看到 Git 标记出了冲突处。

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main x
[11:27:24] C:1
[$ cat EULA.txt
This software is licensed.
<<<<< HEAD
Created by:
Ostrichbeta Chan
=====
Created by:
Jeder Chan
>>>>> EULA
```

将其手动修改为不冲突的版本并去除所有标记：

```
[$ cat EULA.txt
This software is licensed.
Created by:
Ostrichbeta Chan
Jeder Chan
```

然后再 git add，git commit / git merge --continue 即可解决冲突。

分支合并

有冲突合并

Commit 后可以看到冲突已经解决，合并成功：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[11:34:30]
[$ git --no-pager log --graph --oneline --all
*   986e6c6 (HEAD -> main) Merge branch 'EULA'
|\ 
| * 73798bc (EULA) Changed EULA
* | 3f2682f Name added
|/
* 6915ecc Added EULA
```

分支合并

有冲突合并

若主分支对档案进行了改动，待合并分支将这个档案删除，此时合并不会发生修改/删除冲突：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-test on git:main o [0:25:46]
$ git merge del
CONFLICT (modify/delete): EULB.txt deleted in del and modified in HEAD. Version HEAD of EULB.txt left in tree.
Automatic merge failed; fix conflicts and then commit the result.
```

分支合并

有冲突合并

若主分支对档案进行了改动，待合并分支将这个档案删除，此时合并不会发生修改/删除冲突：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-test on git:main o [0:25:46]
$ git merge del
CONFLICT (modify/delete): EULB.txt deleted in del and modified in HEAD. Version HEAD of EULB.txt left in tree.
Automatic merge failed; fix conflicts and then commit the result.
```

有两种解决方式：

- 使用 `git add <file>` 保留档案，接受主分支的修改
- 使用 `git rm <file>` 删去档案，接受待合并分支的提交

解决冲突后使用 `git commit` 或 `git merge --continue` 继续合并过程

分支合并

有冲突合并

若主分支与待合并分支在合并时有一个内容不一样的**同名非文字档**，称为**新增/新增冲突**：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-test on git:main o [8
:16:54]
$ git merge image-submit
warning: Cannot merge binary files: sports.png (HEAD vs. image-submit)
Auto-merging sports.png
CONFLICT (add/add): Merge conflict in sports.png
Automatic merge failed; fix conflicts and then commit the result.
```

分支合并

有冲突合并

若主分支与待合并分支在合并时有一个内容不一样的同名非文字档，称为新增/新增冲突：

```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-test on git:main o [8
:16:54]
$ git merge image-submit
warning: Cannot merge binary files: sports.png (HEAD vs. image-submit)
Auto-merging sports.png
CONFLICT (add/add): Merge conflict in sports.png
Automatic merge failed; fix conflicts and then commit the result.
```

有两种解决方式：

- 使用 `git checkout --ours <file>` 使用主分支版本
- 使用 `git checkout --theirs <file>` 使用待合并分支版本

解决冲突后使用 `git commit` 或 `git merge --continue` 继续合并过程

分支合并

放弃合并

使用 `git merge --abort` 将会放弃当前的合并，将分支回到合并前状态。

1. Git 简介

2. Git 基本操作

2.1 Git 本机工作流程

2.2 环境设定

2.3 Git 主要指令

2.4 其他指令

3. Git 远端联动

4. 练习

5. 附录

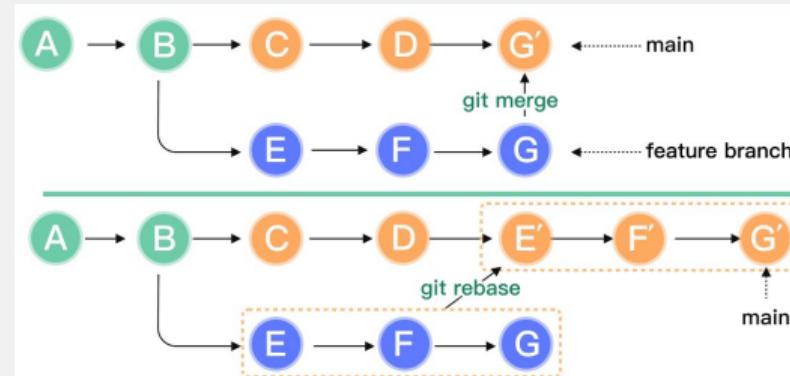
其他指令

git rebase

将另外一个分支作为基准合并。

```
git rebase <branch>
```

其与 git merge 的区别如下图所示。



其他指令

git stash

暂存已经进入暂存区但尚未 commit 的档案。

```
git stash # 暂存当前改动  
git stash list # 列出当前的 stash 列表  
git stash apply stash@{<id>} # 套用 ID 为 <id> 的 stash 到当前分支上  
git stash drop stash@{<id>} # 删去一个 stash  
git stash pop stash@{<id>} # 等同 apply + drop
```

使用 git stash 可以暂存所有「已被 git 追踪已经放入或未有放入暂存区 (staging area)」的变动，但不会暂存新增未被 git 管控或被忽略的档案。

其他指令

git cherry-pick

抽出一个 commit 并整合到当前的 HEAD

```
git cherry-pick <commit> # 拣一个分支整合  
git cherry-pick <commit1>...<commit2>  
# 拣 commit1 至 commit2 之间的整合 (不含 commit1)  
git cherry-pick <commit1>^..<commit2>  
# 拣 commit1 至 commit2 之间的整合 (含 commit1)
```

其他指令

git cherry-pick

抽出一个 commit 并整合到当前的 HEAD

```
git cherry-pick <commit> # 拣一个分支整合  
git cherry-pick <commit1>...<commit2>  
# 拣 commit1 至 commit2 之间的整合 (不含 commit1)  
git cherry-pick <commit1>^..<commit2>  
# 拣 commit1 至 commit2 之间的整合 (含 commit1)
```

整合之后，新的 commit 改变的内容会与旧 commit 一致，但会有一个新的 commit 号。

```
# ostrichb @ Poppi in ~/.config/nvim on git:main o [2:39:58]  
$ git --no-pager log --graph --all --oneline  
* 900b45a (HEAD -> main, origin/main, origin/HEAD) Change TeX-Fmt to match latest parameter format  
* 887632e Changed buffer manager and added fzf search utility  
* 55b528a Added latex to treesitter  
* b0f11fa Added copilot  
| * e6252f7 (origin/no-latex) Changed buffer manager and added fzf search utility  
| * 60ee641 Remove latex part for no-latex  
| * d4a08a7 Changed no-latex color from dark to light  
| * 7dcfc7d Added copilot
```

其他指令

git blame

检视一个档案中哪一行是由谁什么时候在哪个 commit 中新增。

```
git blame <path>
```

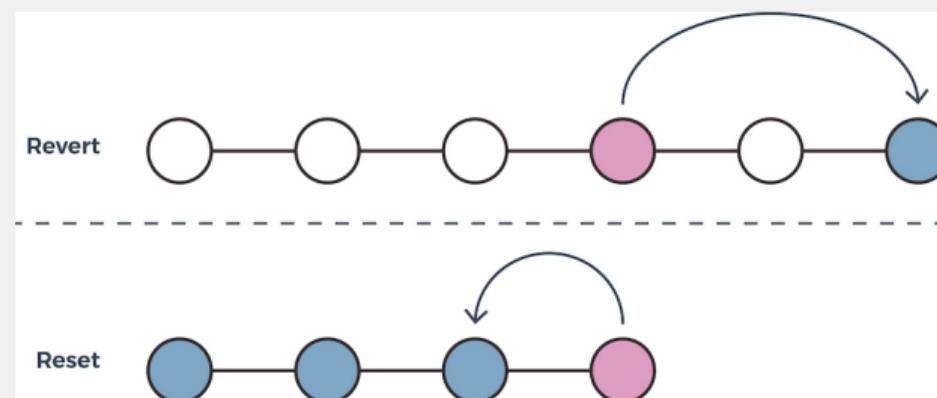
```
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 1) set number
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 2)
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 3) call plug#begin()
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 4) Plug 'nvim-tree/nvim-tree.lua'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 5) Plug 'nvim-lualine/lualine.nvim'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 6) Plug 'luochen1998/rainbow'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 7) Plug 'sbdchd/neofmt'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 8) Plug 'lervag/vimtex'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 9)
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 10) " Use release branch (recommended)
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 11) Plug 'neoclude/coc.nvim', {'branch': 'release'}
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 12) Plug 'honza/vim-snippets'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 13)
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 14) " Buffer Related
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 15) Plug 'nvim-lua/plenary.nvim'
887632e1 (Ostrichbeta Chan 2025-09-25 08:54:47 +0000 16) Plug 'wasabeef/bufferin.nvim'
887632e1 (Ostrichbeta Chan 2025-09-25 08:54:47 +0000 17) Plug 'willothy/nvim-cokeline'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 18)
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 19) Plug 'nvim-treesitter/nvim-treesitter', {'do': ':TSUpdate'}
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 20)
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 21) " Debug Stuff
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 22) Plug 'mfussenegger/nvim-dap'
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 23) Plug 'mfussenegger/nvim-dap-python'
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 24) Plug 'mxsdev/nvim-dap-vscode-js'
```

其他指令

git revert

使用一个 Commit 来取消不需要的 Commit。

```
git revert <commit>
```



git reset 和 git revert 的区别

其他指令

git checkout (补充用法)

git checkout 还可以由其他分支抽出档案。

```
git checkout <branch> <file> \dots # 由\text{branch}分支复原  
git checkout [--] <file> 放弃暂存区的更改，回到暂存前状态
```

为什么要加--？

git checkout 会优先匹配分支的名字。假若你有分支叫 c.cpp，但是同时又想复原 c.cpp 档案，使用 git checkout c.cpp 会切到 c.cpp 分支，唯有加上 -- 才可以跳过分支名字的匹配直接复原档案。

1. Git 简介

2. Git 基本操作

3. Git 远端联动

4. 练习

5. 附录

1. Git 简介

2. Git 基本操作

3. Git 远端联动

3.1 GitHub 简介

3.2 Git 远端连线

3.3 向他人的仓库贡献

4. 练习

5. 附录

GitHub 是什么

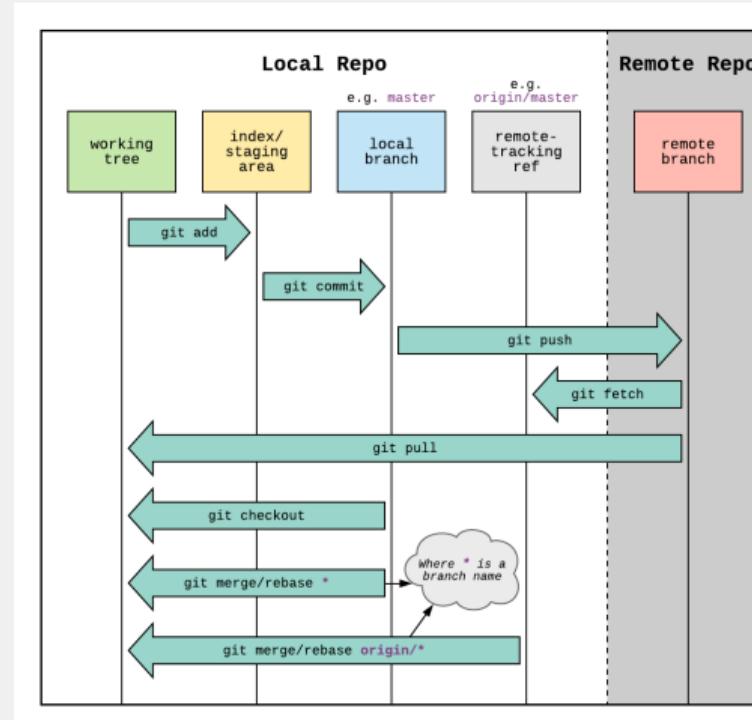
GitHub 是一个可以贮存、分享及与他人共同协力开发的 code 云端托管平台。使用 GitHub 存储的仓库可以：

- 展示或共享你的工作；
- 持续跟踪和管理对于原始码的更改；
- 让他人审阅你的原始码，给出改进的建议；
- 在公用专案中与他人合作，而毋须担心你与其他专案工作者的部分会发生冲突

—— GitHub 官方文档

GitHub 是什么

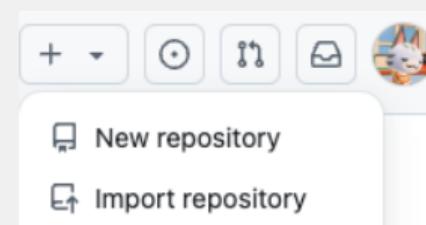
GitHub 上的仓库是一个远端仓库，与本地仓库联动可以更好的将仓库分享出去或与他人协作。



GitHub 仓库管理

新增或汇入仓库

按 GitHub 网页的右上角「+」号可以选择新增新仓库或由其他平台汇入已有仓库。



GitHub 新增仓库

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

● Repository template 根据范本新增仓库

GitHub 新增仓库

Owner * Repository name *

 Ostrichbeta /

Great repository names are short and memorable. Need inspiration? How about [stunning-succotash](#) ?

Description (optional)

 **Public**
Anyone on the internet can see this repository. You choose who can commit.

 **Private**
You choose who can see and commit to this repository.

● **Owner** 仓库拥有者（个人或组织）

● **Repo name** 仓库名字

● **Description** 仓库简介

● **Public** 公开仓库，任何人可检视

● **Private** 私有仓库，仅自己与部分人可见

GitHub 新增仓库

Initialize this repository with:

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

- **Add a README file** 自动新增 README 档案
- **Add .gitignore** 按照范本新增 .gitignore
- **Choose a license** 选择仓库授权规范 (Apache, BSD, GPL ...)

GitHub 新增仓库

新增仓库之后将进入仓库页面

The screenshot shows a section titled "Quick setup — if you've done this kind of thing before". It includes a code editor-like interface with three tabs: "Set up in Desktop" (selected), "HTTPS", and "SSH". The URL <https://github.com/Ostrichbeta/WHUAI-test.git> is displayed. Below the tabs, instructions advise starting by creating a new file or uploading an existing one, and recommend including a [README](#), [LICENSE](#), and [.gitignore](#). There are copy and link icons for the URL.

...or create a new repository on the command line

```
echo "# WHUAI-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Ostrichbeta/WHUAI-test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Ostrichbeta/WHUAI-test.git
git branch -M main
git push -u origin main
```

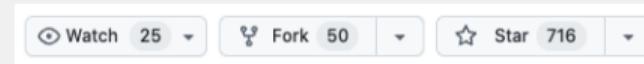
如要删除仓库，在仓库页面 Settings - General - Danger Zone - Delete this repository

GitHub 其他仓库功能

The screenshot shows a GitHub repository interface for 'WHUAI-Dog / WHUAI-test'. The top navigation bar includes links for Code, Commits, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. On the left, there are icons for issues, pull requests, and projects.

- **Issues** 向仓库回馈意见，反馈 Bug
- **Pull Requests** 合并自己及其他合作者提交的分支
- **Actions** 自动化编译及分发的流程
- **Projects** 链结专案版，列出开发方向等
- **Wiki** 可存放仓库使用文档

GitHub 其他仓库功能



- **Watch** 关注仓库，有新动向会发送电邮通知
- **Fork** 克隆仓库到自己的帐户下
- **Star** 给喜欢的项目加星标，星标更多的项目将获更多推荐

1. Git 简介

2. Git 基本操作

3. Git 远端联动

3.1 GitHub 简介

3.2 Git 远端连线

3.3 向他人的仓库贡献

4. 练习

5. 附录

Git 远端连线常用指令

涉及与远端仓库的联系时，主要会用到如下的指令：

- git remote
- git push
- git fetch
- git pull
- git clone

远端仓库管理

git remote

管理本地仓库所链结的远端仓库。

```
git remote add <remote-name> <remote-url>
```

- <remote-name>：远端仓库名字（可以自取），一般是 origin
- <remote-url>：远端仓库的 url

远端仓库管理

git remote

管理本地仓库所链结的远端仓库。

```
git remote add <remote-name> <remote-url>
```

- <remote-name>：远端仓库名字（可以自取），一般是 origin
- <remote-url>：远端仓库的 url

一个范例如下：

```
git remote add origin https://github.com/0strichbeta/WHUAI-test.git
```

远端仓库管理

git remote

git remote 还有其他一些指令：

```
git remote get-url <remote-name> # 取得远端仓库的 URL  
git remote rm <remote-name> # 移除远端仓库  
git remote rename <remote-name> <new-remote-name> # 重新命名远端仓库  
git remote set-url <remote-name> <remote-url> # 变更远端仓库的 URL  
git remote show <remote-name> # 显示远端仓库详细资讯
```

推送至远端仓库

git push

将本地仓库向远端仓库推送，常用：

```
git push -u <remote-name> <branch>
```

如：

```
git push -u origin main
```

- -u：设定远端仓库及分支，设定一次之后可以只用打 git push 就可推送
- 若目标仓库有帐户和密码认证，可能还需要输入对应的帐户和密码
- 若远端不存在分支 <branch>，会自动新增

推送至远端仓库

```
$ git push origin main
Username for 'https://github.com': Ostrichbeta
Password for 'https://Ostrichbeta@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/WHUAI-Dog/WHUAI-test.git/'
```

唉，为什么我 push 不上 GitHub ?

推送至远端仓库

```
$ git push origin main
Username for 'https://github.com': Ostrichbeta
Password for 'https://Ostrichbeta@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/WHUAI-Dog/WHUAI-test.git/'
```

唉，为什么我 push 不上 GitHub？

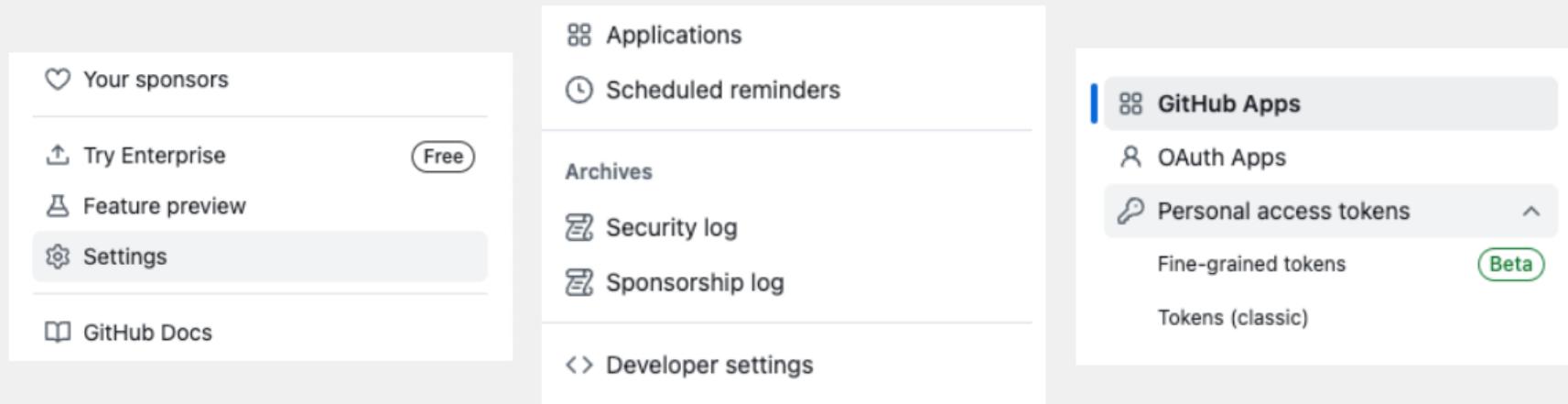
2021 年 8 月 13 日之后，GitHub 使用者无法再使用帐户和密码与远端仓库交互，只能：

- 1 生成并使用 Token，然后将 Token 用作 Git 推送时的密码
- 2 安装 Git Credential Manager 并登入，使用其存储帐户资讯
- 3 系统生成 SSH Key 后汇入 GitHub，使用 SSH 连线

推送至远端仓库

生成 Token

点按右上角用户头像 - Settings - 左侧栏最下方 Developer settings - Personal access tokens



推送至远端仓库

Token 种类

Personal access tokens (classic) 传统的 GitHub 使用 Token，使用上较简易，使用传统的 Personal access token 可以操作你所有有访问许可权的仓库。可以通过「Select scopes」可以选定 token 对于这些仓库所具有的作用域。

Fine-grained tokens 比较新的 Token 格式。与传统的 PAT 相比，对于权限的管理更加自由。

- 可以指定 Token 的适用范围（单用户还是组织，所有仓库还是仅限特定仓库）
- 组织管理者可以更方便下发新的 Token
- 与传统 PAT 相比，Fine-grained tokens 可以设定的作用域更广

推送至远端仓库

在 Personal access tokens 介面若是选择 Fine-grained tokens :

- 1 点「Generate new token」，填写 Token 名字、资源拥有者（个人或者组织）及到期日。
- 2 设定作用的仓库范围，选所有仓库「All repositories」或指定的仓库「Only select repositories」
- 3 下方的 Permissions 中设定权限，至少在「Repository permissions」中将「Contents」设定为「Read and write」，其他可以不设定
- 4 点按「Generate token」生成 Token，生成的 token 仅会显示一次，请进行记录



推送至远端仓库

在 Personal access tokens 介面若是选择 Tokens (classic) :

- 1 点「Generate new token」下的「Generate new token (classic)」
- 2 设定 Token 的备注，有效期及作用域（至少勾选「repo」项）
- 3 点按下面的「Generate token」生成 Token，生成的 token 仅会显示一次，请进行记录

Tokens you have generated that can be used to access the [GitHub API](#).



Make sure to copy your personal access token now. You won't be able to see it again!

✓ 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

Delete

推送至远端仓库

在取得对应的 Token 后，有两种方式使用：

- 在每一次进行 push 操作时，于密码处输入所生成的 Token
- 在 remote 的 URL 中加入 Token，每一次 push 操作时就不需要再输入密码。格式是 `https://<token>@<remote-url>`，如：

```
https://ghp_sMvY5xxxxx7od@github.com/WHUAI-Dog/WHUAI-test
```

推送至远端仓库

当 push 发生冲突时

假设现在云端比本地领先一个 commit (可能是其他开发者 push 的)：

```
$ git --no-pager log --oneline --graph
* b0b66ae (HEAD -> main, origin/main) Add a sentence to aa.txt
* b0a5fd1 Modified aa.txt
* 02504c5 Modified aa.txt
* 55754af Modified aa.txt
* aebda31 Add aa
```

```
$ git --no-pager log --oneline --graph
* b0a5fd1 (HEAD -> main, origin/main, origin/HEAD) Modified aa.txt
* 02504c5 Modified aa.txt
* 55754af Modified aa.txt
* aebda31 Add aa
* 986e6c6 Merge branch 'EULA'
```

左图：远端，右图：本地

如果此时本地提交了一个新的变更，并尝试推送：

```
$ git push
To https://github.com/WHUAI-Dog/WHUAI-test
! [rejected]          main -> main (fetch first)
error: failed to push some refs to 'https://github.com/WHUAI-Dog/WHUAI-test'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

推送至远端仓库

当 push 发生冲突时

假设现在云端比本地领先一个 commit (可能是其他开发者 push 的)：

```
$ git --no-pager log --oneline --graph
* b0b66ae (HEAD -> main, origin/main) Add a sentence to aa.txt
* b0a5fd1 Modified aa.txt
* 02504c5 Modified aa.txt
* 55754af Modified aa.txt
* aebda31 Add aa
```

```
$ git --no-pager log --oneline --graph
* b0a5fd1 (HEAD -> main, origin/main, origin/HEAD) Modified aa.txt
* 02504c5 Modified aa.txt
* 55754af Modified aa.txt
* aebda31 Add aa
* 986e6c6 Merge branch 'EULA'
```

左图：远端，右图：本地

如果此时本地提交了一个新的变更，并尝试推送：

```
$ git push
To https://github.com/WHUAI-Dog/WHUAI-test
! [rejected]          main -> main (fetch first)
error: failed to push some refs to 'https://github.com/WHUAI-Dog/WHUAI-test'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

远端有本地不存在的 commit，需要先拉取远端更新才可以进行 push

拉取远端更新

git pull

拉取远端的更新，并将其合并至本地仓库中。无参数时 merge 远端，加 --rebase 时进行 rebase 的动作

- 如果远端仓库比本地新（本地没有远端不存在的 commit 可以直接合并）
- 如果远端仓库和本地都有对方不存在的 commit，则需要手动 merge

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 445 bytes | 445.00 KiB/s, done.
From https://github.com/WHUAI-Dog/WHUAI-test
  b0a5fd1..b0bb6ae  main      -> origin/main
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false # merge (the default strategy)
hint:   git config pull.rebase true  # rebase
hint:   git config pull.ff only    # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

拉取远端更新

```
$ git merge origin/main
Auto-merging aa.txt
CONFLICT (content): Merge conflict in aa.txt
Automatic merge failed; fix conflicts and then commit the result.

# realvm @ realvm-Standard-PC-Q35-ICH9-2009 in ~/WHUAI-test-merge on
$ cat aa.txt
The quick brown fox jumps out of the lazy dog.
<<<<< HEAD
Blah blah
Blahaj
=====
Due to the conversions, some Kyujitai Kanjis were mapped to one sing
ton.
>>>>> origin/main
```

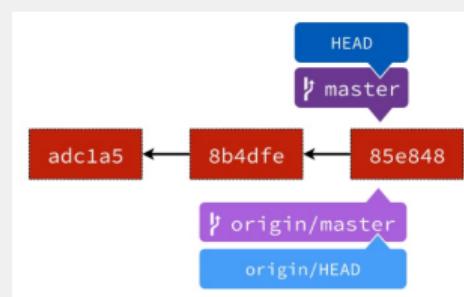
有冲突时需要手动合并

拉取远端更新

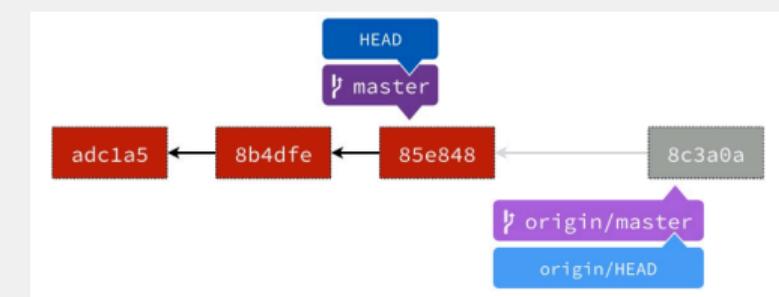
git fetch

仅拉取远端仓库的代码，但是不进行更新。

- 「git pull」实际上是「git fetch」与「git merge」或者「git rebase」的组合



git fetch 之前



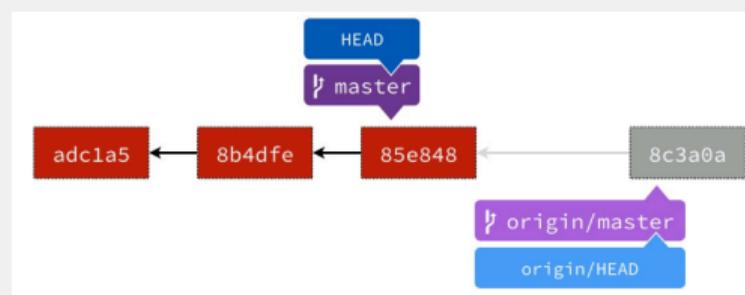
git fetch 之后

拉取远端更新

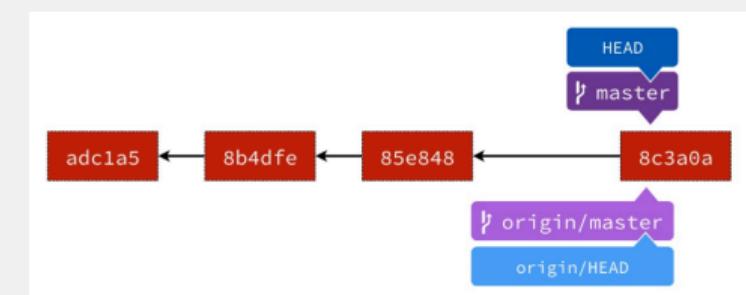
git fetch

仅拉取远端仓库的代码，但是不进行更新。

- 「git pull」实际上是「git fetch」与「git merge」或者「git rebase」的组合



git fetch 之后



git merge 之后

克隆仓库

git clone

将远端仓库克隆至本地。

```
git clone <url>
git clone <url> [path]
git clone [-b <branch>] [--single-branch] [--depth n] <url> [path]
```

- path如果不加，预设存储到仓库名字相同的目录
 - -b 将本地分支切换至 <branch>
 - --single-branch 仅克隆单一分支（有 -b 同步 <branch>，没有预设同步主分支）
 - --depth 仅同步最近 n 个 commit（预设将同步所有的 commit）
- single-branch 和--depth 指令在同步大仓库时尤为有用。

1. Git 简介

2. Git 基本操作

3. Git 远端联动

3.1 GitHub 简介

3.2 Git 远端连线

3.3 向他人的仓库贡献

4. 练习

5. 附录

提交至其他人的仓库

预设情况下，每一个帐户仅可以对自己的仓库直接进行 push 的操作，如果对于他人拥有的仓库进行交互，则：

- 成为其他人仓库具有直接存取（Direct access）许可权的合作者
- 克隆仓库到自己帐户，然后做出修改后提交回原始仓库

新增仓库合作者

仓库合作者可以在项目页面「Settings」 - 「Collaborators」 - 「Manage access」 中新增仓库具有直接存取权限的人。

The screenshot shows the GitHub repository settings interface. The left sidebar contains navigation links for Actions, Projects, Wiki, Security, Insights, and Settings. The 'Settings' tab is active. The main content area is titled 'Who has access'. It shows that the repository is a 'Public repository' (visible to anyone) and provides options for managing access. The 'Manage' button is highlighted. Below this, there are two sections: 'PUBLIC REPOSITORY' (repository is public and visible to anyone) and 'DIRECT ACCESS' (0 collaborators have access to this repository). A 'Manage' link is also present here. At the bottom, there is a section titled 'Manage access' with a message 'You haven't invited any collaborators yet' and a green 'Add people' button.

新增仓库合作者

组织所拥有的仓库可在「Settings」 - 「Collaborators and teams」 中新增仓库具有特定权限的人或团队。

The screenshot shows the 'Settings' page for a GitHub repository. The left sidebar lists various configuration sections: General, Collaborators (which is selected and highlighted in blue), Moderation options, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), and Security (Code security, Deploy keys, Secrets and variables). The main content area is titled 'Who has access' and indicates this is a 'Public repository'. It shows two sections: 'PUBLIC REPOSITORY' (This repository is public and visible to anyone) and 'DIRECT ACCESS' (0 collaborators have access to this repository. Only you can contribute to this repository). A 'Manage' button is available for both sections. Below this, another section titled 'Manage access' contains the message 'You haven't invited any collaborators yet' and a green 'Add people' button.

克隆、更改和提交

将其他人所有的仓库克隆后修改后提交，就算没有对原始仓库的直接访问许可权，也可以使原仓库获得批准后合并自己的修改。

- 1 在原仓库使用「Fork」功能复制一份原始仓库到自己帐户下
- 2 对自己帐户下的仓库进行修改
- 3 在自己仓库下使用「Pull requests」功能向原仓库发起合并申请
- 4 原仓库拥有者通过申请并处理可能的冲突之后，克隆的仓库会合并至原始仓库

自动合并

若克隆仓库与原始仓库之间没有冲突，在许可之后可以自动合并。

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

base repository: WHUAI-Dog/WHUAI-test | base: main | head repository: Ostrichbeta/WHUAI-test | compare: main

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) Create pull request

1 commit | 1 file changed | 1 contributor

Commits on Oct 27, 2024

Add eulb
Ostrichbeta committed 4 minutes ago

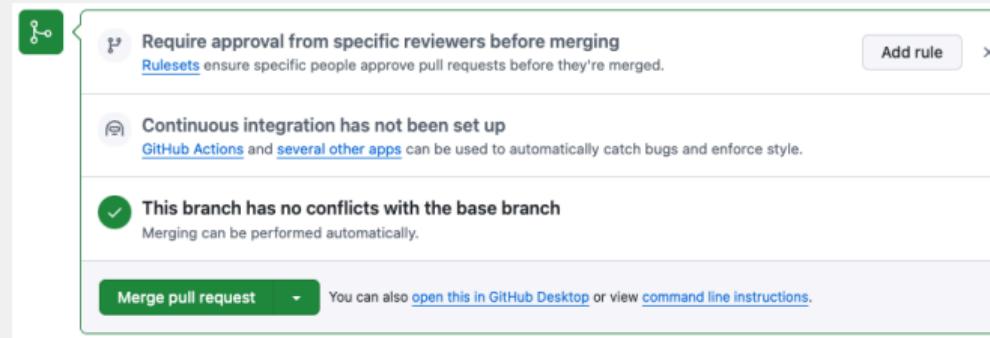
Show 1 changed file with 1 addition and 0 deletions. Split Unified

EULB.txt

...	...	@@ -0,0 +1 @@
1	+	114514

自动合并

若克隆仓库与原始仓库之间没有冲突，在许可之后可以自动合并。



手动合并

若克隆仓库与原始仓库之间有冲突：

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base repository: WHUAI-Dog/WHUAI-test base: main head repository: Ostrichbeta/WHUAI-test compare: main

✗ Can't automatically merge. Don't worry, you can still create the pull request.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) [Create pull request](#)

1 commit 1 file changed 1 contributor

Commits on Oct 27, 2024

Damn Ostrichbeta committed 1 minute ago

Showing 1 changed file with 1 addition and 1 deletion.

EULB.txt

...	...	@@ -1 +1 @@
1	-	1919810
1	+	1148141919810

手动合并

若克隆仓库与原始仓库之间有冲突：

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base repository: WHUAI-Dog/WHUAI-test base: main head repository: Ostrichbeta/WHUAI-test compare: main

✗ Can't automatically merge. Don't worry, you can still create the pull request.

Discuss and review the changes in this comparison with others. [Learn about pull requests](#) [Create pull request](#)

1 commit 1 file changed 1 contributor

Commits on Oct 27, 2024

Damn Ostrichbeta committed 1 minute ago

Showing 1 changed file with 1 addition and 1 deletion.

2 EULB.txt

...	...	@@ -1 +1 @@
1	- 1919810	+ 1148141919810

- 1 撤回合并申请，在克隆仓库中 pull 原始仓库解决冲突后再提交申请
- 2 由原始仓库的维护者解决冲突

手动合并

若冲突有原始仓库的维护者解决：

This branch has conflicts that must be resolved
Use the [web editor](#) or the [command line](#) to resolve conflicts.

Conflicting files

EULB.txt

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Checkout via command line
If the conflicts on this branch are too complex to resolve in the web editor, you can check it out via command line to resolve the conflicts.

HTTPS SSH Patch <https://github.com/Ostrichbeta/WHUAI-test.git>

Step 1: From your project repository, check out a new branch and test the changes.
`git checkout -b Ostrichbeta-main main
git pull https://github.com/Ostrichbeta/WHUAI-test.git main`

Step 2: Merge the changes and update on GitHub.
`git checkout main
git merge --no-ff Ostrichbeta-main
git push origin main`

- 在网页中按「Resolve conflicts」
手动编辑冲突部分解决
- 将冲突处移到本地进行处理
 - 1 原分支复制出一个合并用分支
 - 2 在复制的分支中解决冲突后合并
 - 3 在原分支中合并复制分支
 - 4 提交正常合并后的原分支

回退合并

原仓库拥有者如果在合并之后发现原仓库出现了重大问题，或只是单纯想撤回合并，可以在合并的页面按「Revert」将仓库回退到合并之前的状态。

Update EULB.txt #4

Merged Ostrichbeta merged 2 commits into [WHUAI-Dog:main](#) from [Ostrichbeta:main](#) 1 minute ago

Conversation 0 Commits 2 Checks 0 Files changed 1

Ostrichbeta commented 14 minutes ago
No description provided.

Ostrichbeta added 2 commits 14 minutes ago

- Update EULB.txt
- Merge branch 'main' into main

Ostrichbeta merged commit [e1d3218](#) into [WHUAI-Dog:main](#) 1 minute ago

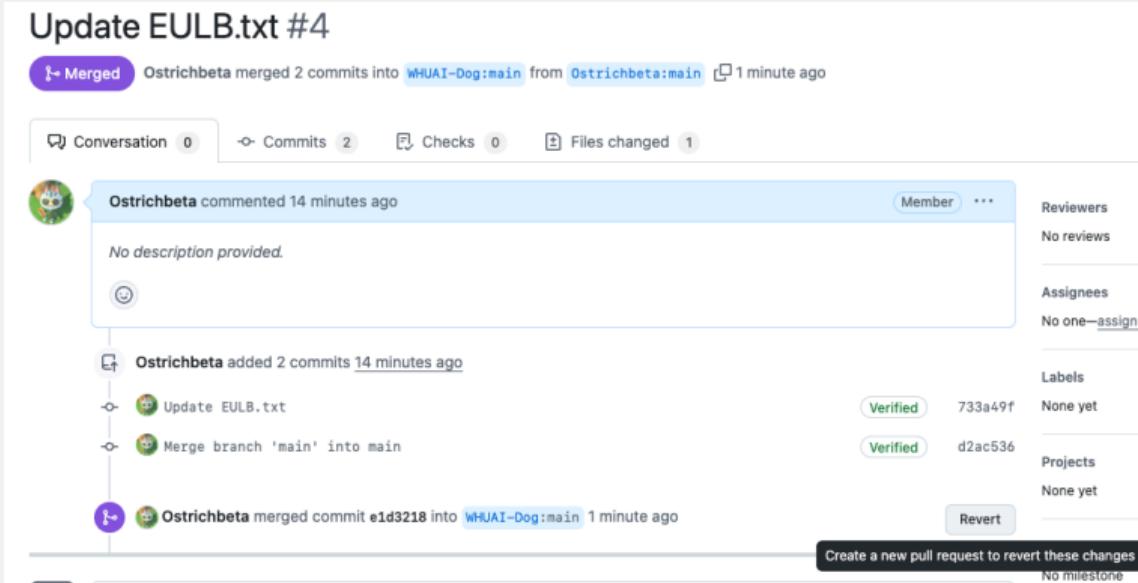
Revert Create a new pull request to revert these changes No milestone

Reviewers
No reviews

Assignees
No one—assign you

Labels
None yet

Projects
None yet



1. Git 简介

2. Git 基本操作

3. Git 远端联动

4. 练习

5. 附录

小练习

本仓库有一个 zip 包的练习仓库供练习 Git 指令。内含有 A, B, C 三个资料夹。

- 1 A 资料夹是一个未被 Git 管控的资料夹，初始化并完成一个 commit。
- 2 B 资料夹的最新一个 commit 中，token.txt 被加入了奇怪的内容。使用 git log 查询，并使用 git reset 或其他指令对其进行退回更新。
- 3 C 资料夹中，合并 hk 分支到 main 分支，解决可能出现的合并冲突。

若有不明之处，可以通过各种方式询问搜寻引擎或者 AI 解决问题。

1. Git 简介

2. Git 基本操作

3. Git 远端联动

4. 练习

5. 附录

Git 辅助资料

除了命令行的界面外，还可以使用一些 GUI 工具更方便 Git 的使用：

- SourceTree (Mac, Windows)
- GitHub Desktop (Mac, Windows)
- GitKraken Desktop (Mac, Linux, Windows)
- SmartGit (Mac, Linux, Windows)

可以前往 <https://git-scm.com/downloads/guis> 了解更多工具。

Git 速查表：<https://education.github.com/git-cheat-sheet-education.pdf>

参考资料

- 「为你自己学 Git」高见龙 - <https://gitbook.tw/>
- <https://stackoverflow.com/questions/1274057/how-do-i-make-git-forget-about-a-file-that-was-tracked-but-is-now-in-gitignore>
- https://www.reddit.com/r/git/comments/99ul9f/git_workflow_diagram_showcasing_the_role_of/
- <https://toanthien.com/blog/a-comprehensive-guide-to-git-git-merge-vs-git-rebase-and-essential-commands/>