

Git 版控系統

Git Version Control System

October 19, 2025

WHUAI Team

Ostrichbeta Chan 陳逸銘

1. Git 簡介

2. Git 基本操作

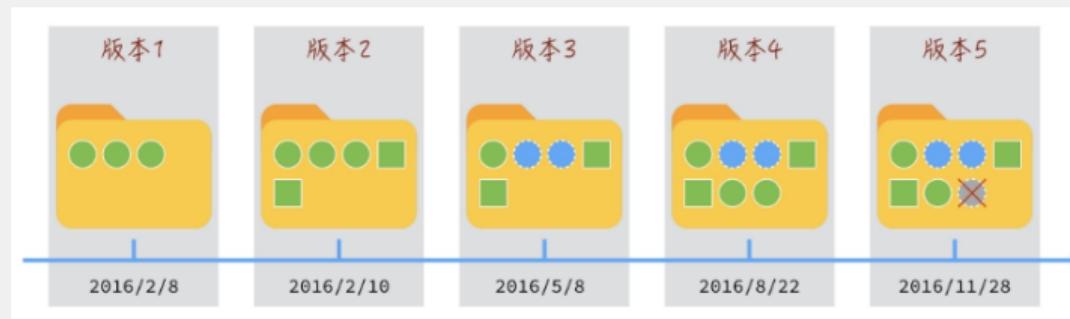
3. Git 遠端聯動

4. 練習

5. 附錄

為什麼我們要學習版控系統？

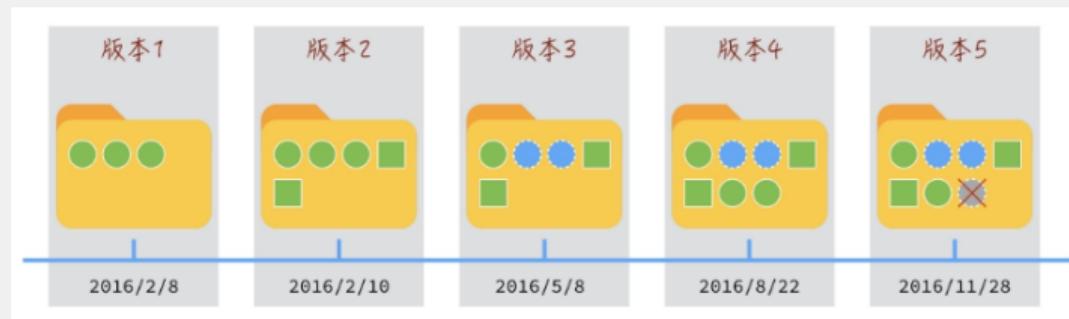
若沒有版控系統，備份不同的版本，只能一個版本一個資料夾：



- Chan 基於版本 5 開發了版本 6
- Lee 由版本 4 修復了一些 Bug 希望同步到最新開發中
- Yang 希望能夠快速的提煉出不同版本之間的區別給 Chang 老師彙報進度

為什麼我們要學習版控系統？

若沒有版控系統，備份不同的版本，只能一個版本一個資料夾：



- Chan 基於版本 5 開發了版本 6
- Lee 由版本 4 修復了一些 Bug 希望同步到最新開發中
- Yang 希望能夠快速的提煉出不同版本之間的區別給 Chang 老師彙報進度

沒有一個快速高效的方法，三個人都很頭大

Git 的優點

開發管理 Linux 核心的 Linus Torvalds 在 2005 年也遇到了這個問題，加之當時版控系統不盡完善（有的還死貴），於是他在 10 天時間完成了 Git 的初版並開源，經過多年發展，Git 系統已經十分完善。

Git 的優點

開發管理 Linux 核心的 Linus Torvalds 在 2005 年也遇到了這個問題，加之當時版控系統不盡完善（有的還死貴），於是他在 10 天時間完成了 Git 的初版並開源，經過多年發展，Git 系統已經十分完善。

Git 有很多優點：

- **開放原始碼** 可以免費用，且有社群維護
- **快速高效** Git 可以快速在不同版本間切換，特殊的設計使多版本下的目錄的空間佔用降低
- **分散式系統** Git 不依賴伺服器，可以不受網路影響的使用
- **易上手** 雖然 Git 有眾多指令，但大概 20% 的指令可以完成 80% 的工作

Git 的安裝

- Linux：使用各發行版的包管理器（apt, dnf, yum）直接安裝或由原始碼編譯後安裝
- macOS：安裝 Xcode Command Line Tools 過程中自動安裝
- Windows：由 Git 官網下載安裝檔安裝

Git 官方網站 <https://git-scm.com/> 提供了 Windows, macOS 和 Linux 的安裝檔

預備知識

- git 預設的編輯器是 vim，可以瞭解 vim 的常用快速鍵，或使用 `git config --global core.editor nano` 改用更友善的 nano 編輯器。
- Bash 中可以在輸入指令的過程中按 Tab 鍵自動補全
- Bash 中可使用上下方向鍵快速填入歷史指令
- Bash 和 Zsh 中用 Ctrl-A 跳至行頭，Ctrl-E 跳至行尾，Ctrl-C 終止當前程式，Ctrl-L 清除螢幕
- 在終端機中，可以使用 Shift-Ctrl-C 複製，Shift-Ctrl-V 貼上
- vim 的預設退出方法是按下 Esc 使 -- INSERT -- 字樣消失，然後輸入`:q` 退出

內容約定

- 以下的 git 指令大多數在 git v2.39 版本下於 macOS 15.1 下執行
- 指令中出現尖括弧的部分如 <text> 為指令中必須填入的項
- 指令中出現方括弧的部分如 [text] 為指令中的可選項
- 除非特別說明，否則**所有的括弧符號都不需要輸入**

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

4. 練習

5. 附錄

1. Git 簡介

2. Git 基本操作

2.1 Git 本機工作流程

2.2 環境設定

2.3 Git 主要指令

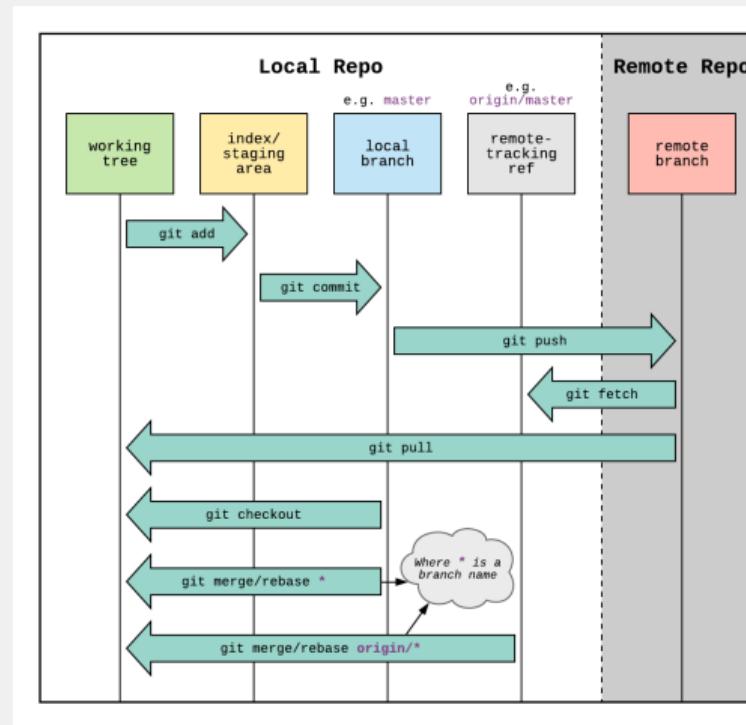
2.4 其他指令

3. Git 遠端聯動

4. 練習

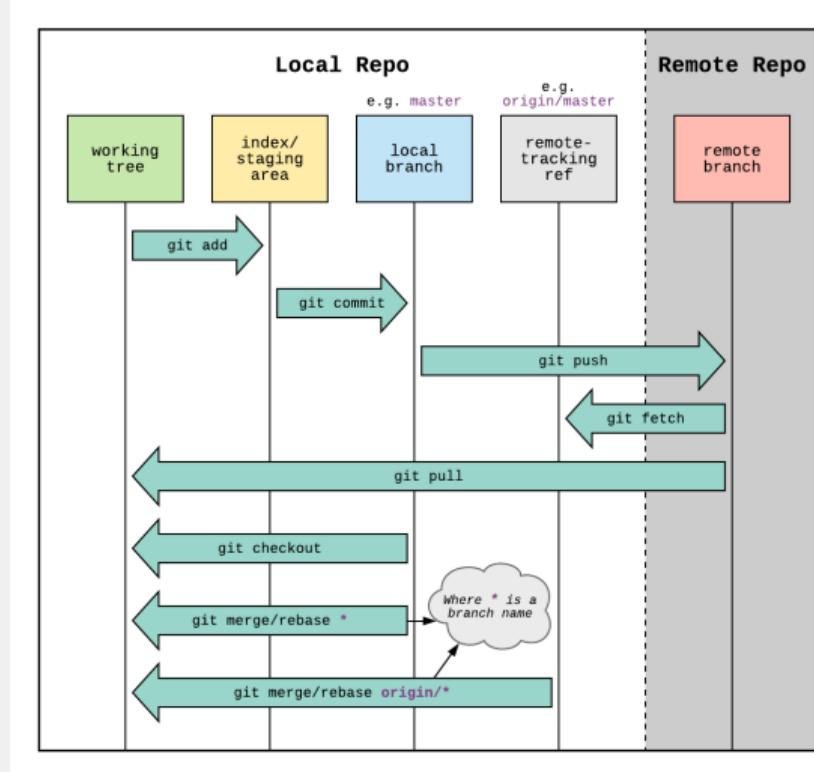
5. 附錄

Git 的主要工作流程



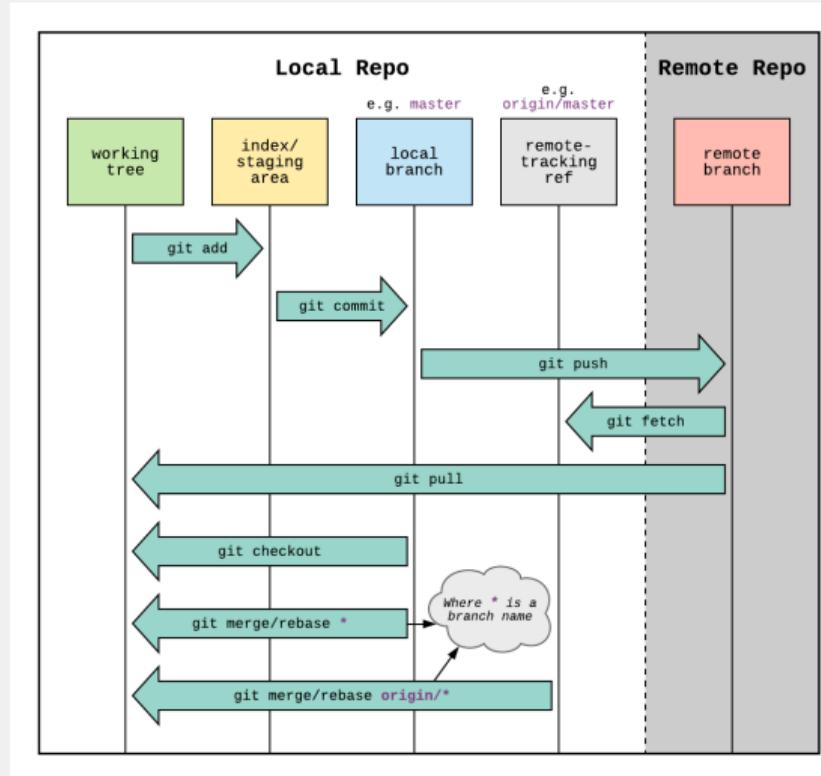
Git 流程簡圖

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：

- 將文件加至 Git 版控
- 提交文件的變更
- 推送 commit 到遠端倉庫
- 取得遠端代碼變更
- 切換分支
- 合併不同分支
- etc.

Git 的主要工作流程

最簡單的 Git 流程：

- 1 初始設定工作區
- 2 新增文件至版控系統
- 3 提交文件的修改

如果需要與遠端倉庫溝通：

- 4 將修改推至雲端

1. Git 簡介

2. Git 基本操作

2.1 Git 本機工作流程

2.2 環境設定

2.3 Git 主要指令

2.4 其他指令

3. Git 遠端聯動

4. 練習

5. 附錄

設定個人資訊

使用 Git 的第一件事是設定我們的個人資訊，方便 Git 記錄代碼的提交者。

可以使用 `git config` 指令進行設定：

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

執行後，可使用以下指令確認：

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

設定個人資訊

```
# 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
```

執行範例

設定個人資訊

使用 `git config --list` 還可以檢視當前 Git 的所有設定。

```
# 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 簡介

2. Git 基本操作

2.1 Git 本機工作流程

2.2 環境設定

2.3 Git 主要指令

2.4 其他指令

3. Git 遠端聯動

4. 練習

5. 附錄

本地工作常用指令

本章重點講解以下指令：

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

新增倉庫

git init

在工作空間中使用 git init 可以將當前目錄交由 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/
```

新增倉庫

git init

在工作空間中使用 `git init` 可以將當前目錄交由 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/
```

- 一個 `.git` 目錄會被新增，此為 Git 的核心
- 移除這個 `.git` 目錄將會移除 Git 的控制

整個專案目錄裡，什麼檔案或目錄刪了都救得回來，但 `.git` 目錄只要刪了就沒辦法了。

狀態查詢

git status

取得當前工作空間的狀態，是否與遠端同步，檔案的提交狀態等。

```
# 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)
```

狀態查詢

git status

取得當前工作空間的狀態，是否與遠端同步，檔案的提交狀態等。

```
[# 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
```

已與遠端同步，且未有未提交的檔案

往 Git 中新增檔案

git add

將某些檔案/目錄加至 staging area (暫存區)，為提交作準備。

```
# ostrichb @ Poppi in ~/Documents/Projects/whuai-git-test on git:main ✘ [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 中新增檔案

git add

將某些檔案/目錄加至 staging area (暫存區)，為提交作準備。

- 使用 `git add .` 可以將當前目錄所有檔案加至暫存區
- 若檔案有修改，需要再執行一次 `git add` 指令暫存變更
- 空目錄是無法提交的
- 使用 `git diff` 可詳細的查詢檔案具體的變更內容

往 Git 中新增檔案

git add

將某些檔案/目錄加至 staging area (暫存區)，為提交作準備。

- 使用 `git add .` 可以將當前目錄所有檔案加至暫存區
- 若檔案有修改，需要再執行一次 `git add` 指令暫存變更
- 空目錄是無法提交的
- 使用 `git diff` 可詳細的查詢檔案具體的變更內容

那要怎麼移除檔案？

- 用 `rm` 移除檔案，然後 `git add` 追加變更至暫存區
- 或使用 `git rm` 指令完成上述兩步

往 Git 中提交檔案

git commit

將暫存區中的檔案提交本地倉庫存檔。

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

<comment> 為這一次提交的內容簡括，是代碼溝通的重要資訊，以下是一些良好的範例：

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

...以及部分反面示範：

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

往 Git 中提交檔案

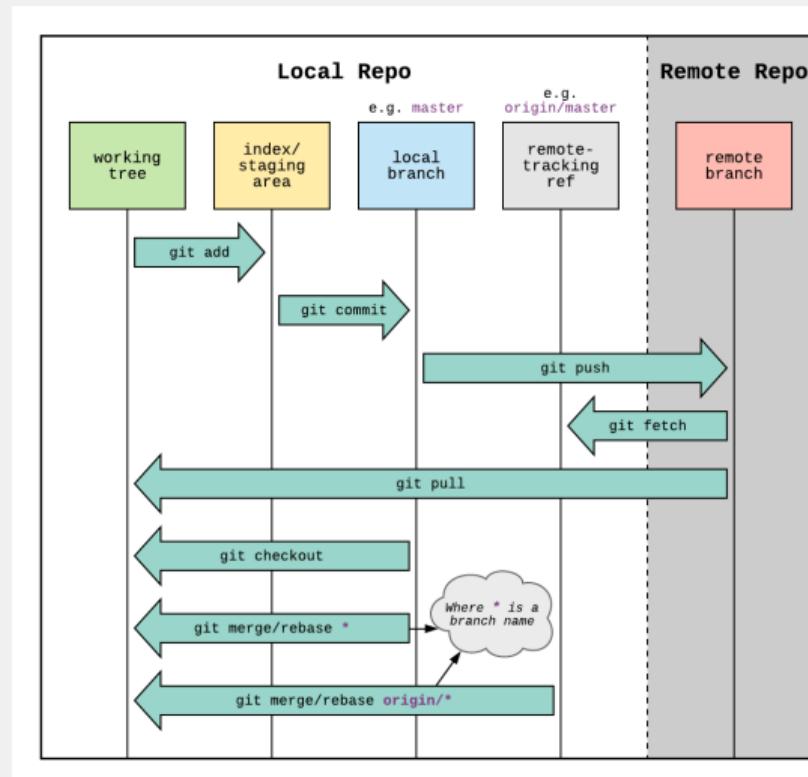
git commit

將暫存區中的檔案提交本地倉庫存檔。

```
# 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
```

- 未加入暫存區的檔案將不會被提交
- 提交內容信息是必須的，若不帶 -m 會彈出文字編輯器，需在其中加入文字
- 使用 --amend 不會新增 commit 而是追加到上一個 commit 中

往 Git 中提交檔案



Always remember to git add !



忽略部分檔案

.gitignore

一些倉庫中會含有.gitignore 檔，.gitignore 中記錄的檔案將不會被 Git 進行版本控制。

倉庫中由編譯生成的檔案，或者是開發中所用的敏感存取密匙等通常都可加入.gitignore 中。

有以下幾種格式：

```
/src/ # 忽略工作目錄下最上層的src目錄  
src/ # 忽略所有名為src的目錄、子目錄  
a.txt # 忽略所有名為 a.txt 的檔案  
*.so # 忽略所有附檔名為 *.so 的檔案  
!homo.so # 強制保留名為 homo.so 的檔案  
/ros2/**/lah/ # ros2開頭 launch結尾的目錄/ros2/lah/和/ros2/s/s/lah/都符合
```

設定好之後將 .gitignore 使用 git add 新增，其規則就會對未來檔案生效。

GitHub 官方也整理了一些環境和語言下常用的.gitignore 檔案集：

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

忽略部分檔案

.gitignore

.gitignore 設定的規則只對設定規則之後出現的檔案有效，已經存在但符合.gitignore 規則的檔案仍然會被 Git 控制。

如何將他們移出版控而不刪去檔案？

忽略部分檔案

.gitignore

.gitignore 設定的規則只對設定規則之後出現的檔案有效，已經存在但符合.gitignore 規則的檔案仍然會被 Git 控制。

如何將他們移出版控而不刪去檔案？

```
git rm -r --cached . # 由版控中移除所有檔案  
git add . # 新增所有檔案，但是將忽略 .gitignore 規則中的檔案  
git commit -am "Remove ignored files"
```

具體可以參考

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

檢視提交記錄

git log

檢視過去的 commit 記錄，由新至舊排列。

```
commit 09d8cedd9f29add13289994560a33fe845323423 (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 預設會顯示：

- 當前分支位置
- commit 編號
- commit 時間
- commit 用戶

檢視提交記錄

git log

檢視過去的 commit 記錄，由新至舊排列。

- 使用 `--oneline` 可以更緊湊的顯示記錄
- 使用 `--graph` 可以加入分支線顯示

```
# 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 WHU
AI-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
808e058 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

可以將工作空間或目錄復原到之前某一次提交時的狀態。

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

退回更新

git reset

可以將工作空間或目錄復原到之前某一次提交時的狀態。

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

- 每一個 ^ 符號表示前一次，~ 表示前 n 次。如 HEAD~2 表示 HEAD 的前兩次。
- HEAD 表示當前所在的分支狀態，而 main 則指向 main 分支的最新狀態。

退回更新

git reset

git reset 主要有三種工作模式。

```
git reset --mixed 0967be3 #若不帶參數，預設就是mixed  
git reset --soft 0967be3  
git reset --hard 0967be3
```

退回更新

git reset

git reset 主要有三種工作模式。

```
git reset --mixed 0967be3 # 若不帶參數，預設就是 mixed  
git reset --soft 0967be3  
git reset --hard 0967be3
```

- Mixed 是預設的工作模式，不改變工作目錄，但將檔案退回到未追蹤的狀態；
- Soft 不改變工作目錄，將檔案退回到暫存區；
- Hard 改變工作目錄和暫存區，將檔案復原到目標分支的狀態（硬復原）。

退回更新 - git reset --mixed

- Mixed 是預設的工作模式，不改變工作目錄，但將檔案退回到未追蹤的狀態；

```
# 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 不改變工作目錄，將檔案退回到暫存區；

```
# 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 改變工作目錄和暫存區，將檔案復原到目標分支的狀態（硬復原）。

```
# 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 復原之後，還可以回到沒有執行 reset 之前的狀態嗎？

退回更新

Hard 復原之後，還可以回到沒有執行 reset 之前的狀態嗎？

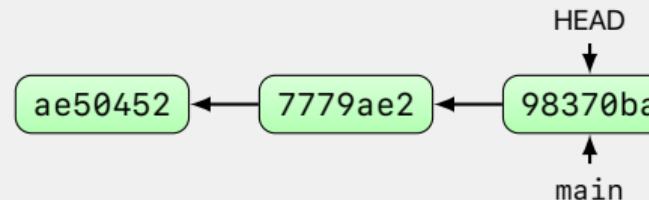
可以！

- 使用 git reflog 指令找出所需要的分支
- 使用 git reset --hard <commit> 復原到目標分支

```
# 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
```

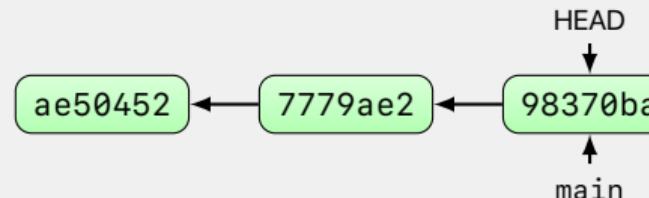
Commit 的更新流程

一開始的時候假設有三個 commit：

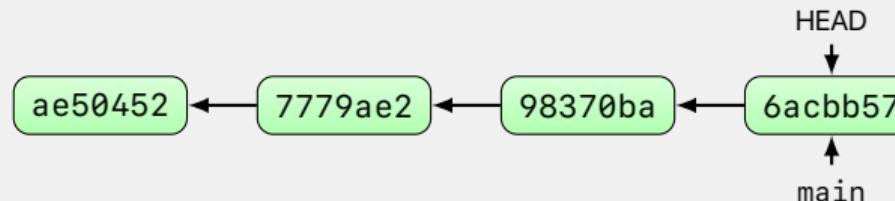


Commit 的更新流程

一開始的時候假設有三個 commit：

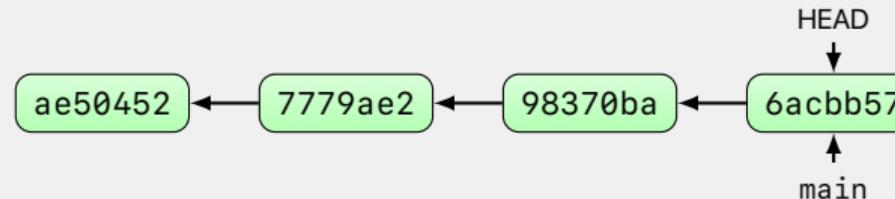


然後不小心錯誤的提交了一個不希望的 commit：



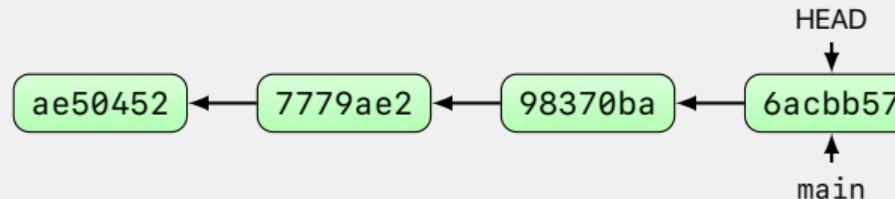
Commit 的更新流程

然後不小心錯誤的提交了一個不希望的 commit :

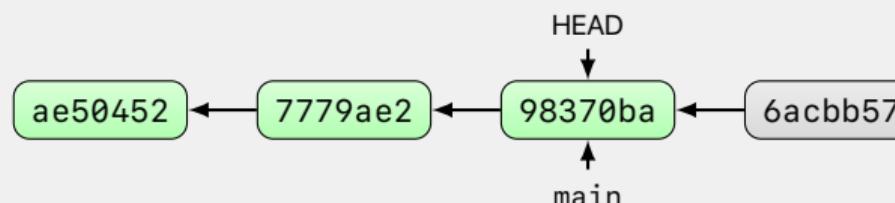


Commit 的更新流程

然後不小心錯誤的提交了一個不希望的 commit :

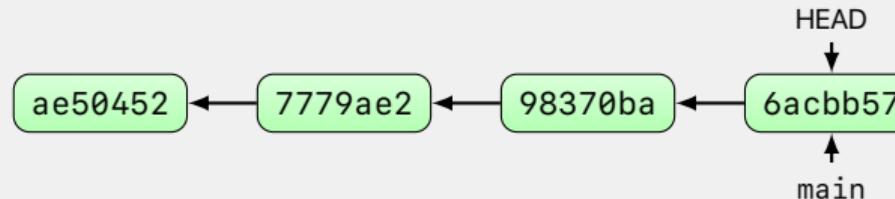


此時執行一次 `git reset --hard` :

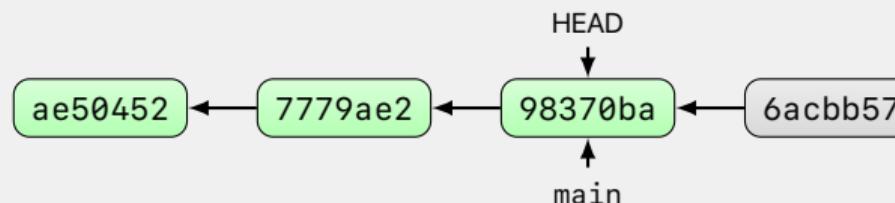


Commit 的更新流程

然後不小心錯誤的提交了一個不希望的 commit：

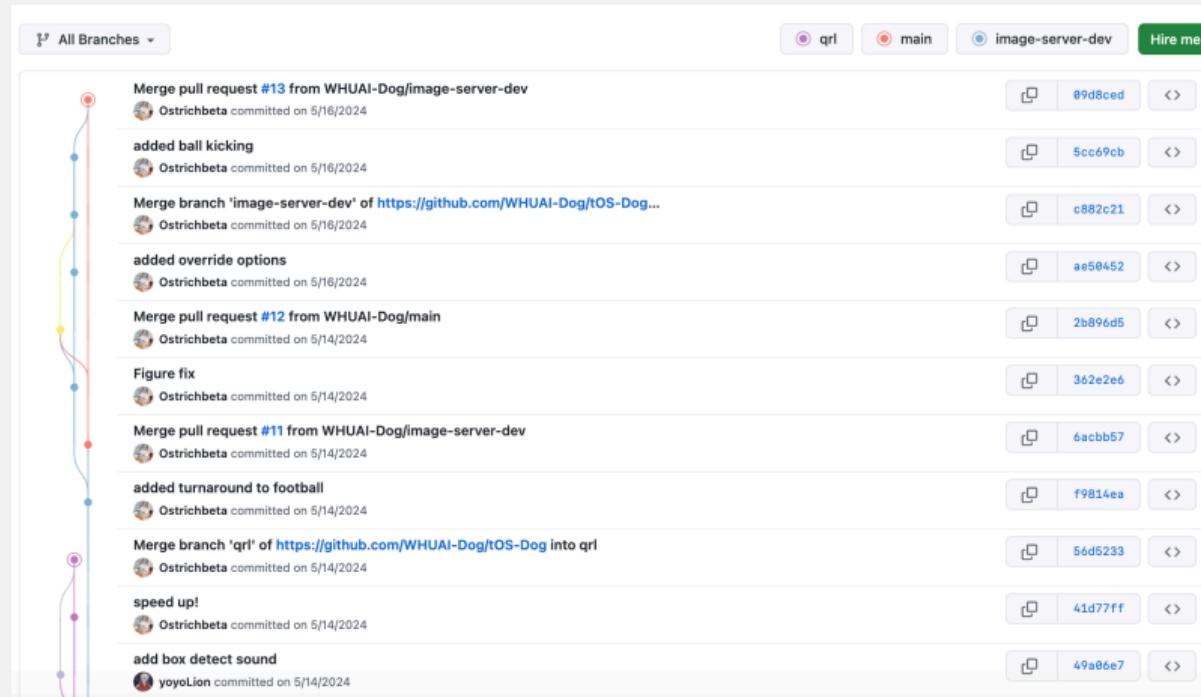


此時執行一次 `git reset --hard`：



`git reset --hard` 僅僅是跳至了已經儲存在過去 / 未來的某個「狀態」，所以可以快速復原

分支



多分支開發範例

分支

在開發的過程中，一路往前 Commit 也沒什麼問題，但當開始越來越多同伴一起在同一個專案工作的時候，可能就不能這麼隨興的想 Commit 就 Commit，這時候分支就很好用。例如想要增加新功能，或是**修正 Bug**，或是**實驗看看某些新的做法**，都可以另外做一個分支來進行，待做完確認沒問題之後再合併回來，不會影響正在運行的產品線。

分支還可以方便的合併，就算不同分支之間可能存在衝突，Git 也會給出解決衝突的方法。



管理分支

git branch

- git branch : 檢視當前本機存在的分支
- git branch -a : 檢視本機和遠端所有的分支（僅遠端可用-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 直接輸出而不開啟文字編輯器

管理分支

git branch

- git branch -m <old> <new> : 將 <old> 分支改名為 <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
```

分支切換

git checkout

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

```
# 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]
$ ]
```

分支切換

```
# 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
```

分支切換

```
# 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 randering engine

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

    Init
```

Commit 之後原分支沒有改動，HEAD 指向當前的 newui 分支

分支合併

git merge

- git merge [-m <comment>] <branch> : 將 <branch> 分支往當前分支合併

```
# 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
```

由於分出 newui 後 main 沒有新的 commit，合併時可以直接快轉 (fast-forward)

分支合併

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
```

合併後 HEAD, main 和 newui 都指向同一個 commit

分支合併

無衝突的合併

一開始 newui 由 main 分支分出去，然後在 newui 中新增兩個檔案：

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

main 中新增一個檔案：

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

分支合併

現在分支樹長得像這樣：

```
# 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
```

由於兩邊互不衝突也可以直接合併：

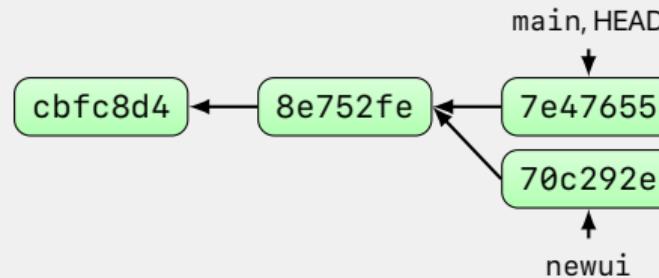
```
# ostrichb @ Ostrichbetas-MacBook-Pro in ~/Documents/WHUAI-Git-test on git:main o
[8:29:40]
$ git merge newui
Merge made by the 'ort' strategy.
auth.js | 0
ui.html | 0
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 auth.js
create mode 100644 ui.html
```

分支合併

此時的 newui 是什麼狀態？

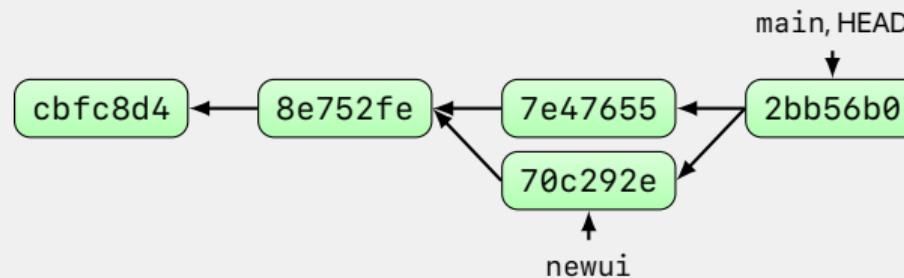
由於只是由 newui 併入 main 分支，main 分支包含了 newui 分支的更動（即 auth.js 和 ui.html）。但是 newui 分支不會包含 main 在分出 newui 後的那一部分改變（即沒有 api.js）。若要將 HEAD, main 和 newui 指向同一個 commit，可以切到 newui 後 merge main 分支。

分支合併



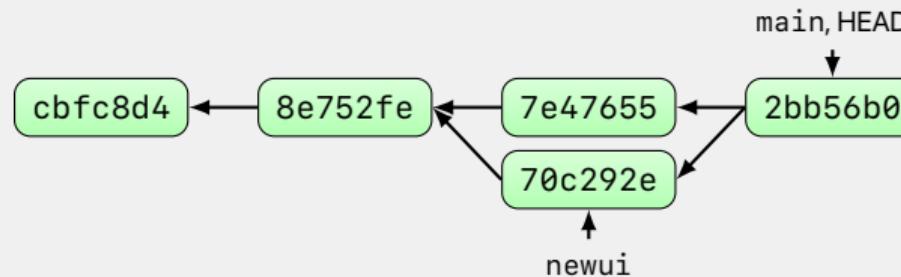
合併之前，兩個分支指向各自分隔的 commit

分支合併



合併之後，main 指向合併後的 commit，但 newui 仍然指向原來合併之前的 commit

分支合併



合併之後，main 指向合併後的 commit，但 newui 仍然指向原來合併之前的 commit

分支本質上只是指向一個 commit 的指標

分支合併

有衝突合併

假設 main 分支下有一檔案 EULA.txt：

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

此時分出分支 EULA，更改了其內容並 commit 如下：

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

分支合併

有衝突合併

同時 main 分支中也更改了其內容：

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

嘗試進行合併，會提示自動合併失敗，發生了**內容衝突**：

```
# 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.
```

分支合併

有衝突合併

此時開啟 EULA.txt，可以看到 Git 標記出了衝突處。

```
# 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
Oreated by:
Ostrichbeta Chan
=====
Created by:
Jeder Chan
>>>>> EULA
```

分支合併

有衝突合併

此時開啟 EULA.txt，可以看到 Git 標記出了衝突處。

```
# 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
```

將其手動修改為不衝突的版本並去除所有標記：

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

然後再 git add , git commit / git merge --continue 即可解決衝突。

分支合併

有衝突合併

Commit 後可以看到衝突已經解決，合併成功：

```
# 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
```

分支合併

有衝突合併

若主分支對檔案進行了改動，待合併分支將這個檔案刪除，此時合併會發生修改/刪除衝突：

```
# 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.
```

分支合併

有衝突合併

若主分支對檔案進行了改動，待合併分支將這個檔案刪除，此時合併會發生修改/刪除衝突：

```
# 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.
```

有兩種解決方式：

- 使用 `git add <file>` 保留檔案，接受主分支的修改
- 使用 `git rm <file>` 刪去檔案，接受待合併分支的提交

解決衝突後使用 `git commit` 或 `git merge --continue` 繼續合併過程

分支合併

有衝突合併

若主分支與待合併分支在合併時有一個內容不一樣的**同名非文字檔**，稱為**新增/新增衝突**：

```
# 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.
```

分支合併

有衝突合併

若主分支與待合併分支在合併時有一個內容不一樣的**同名非文字檔**，稱為**新增/新增衝突**：

```
# 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.
```

有兩種解決方式：

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

解決衝突後使用 `git commit` 或 `git merge --continue` 繼續合併過程

分支合併

放棄合併

使用 `git merge --abort` 將會放棄當前的合併，將分支回到合併前狀態。

1. Git 簡介

2. Git 基本操作

2.1 Git 本機工作流程

2.2 環境設定

2.3 Git 主要指令

2.4 其他指令

3. Git 遠端聯動

4. 練習

5. 附錄

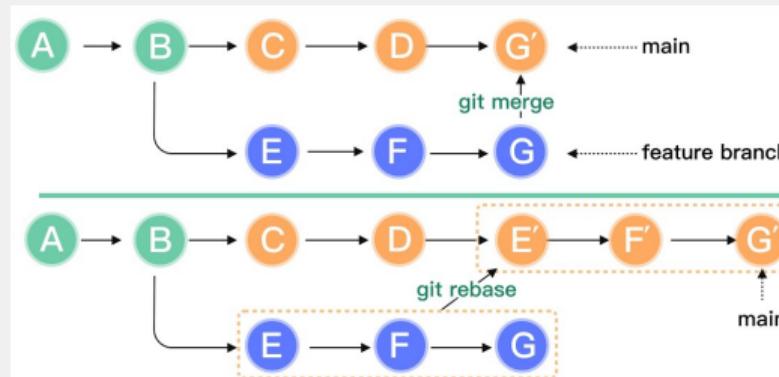
其他指令

git rebase

將另外一個分支作為基準合併。

```
git rebase <branch>
```

其與 git merge 的區別如下圖所示。



其他指令

git stash

暫存已經進入暫存區但尚未 commit 的檔案。

```
git stash # 暫存當前改動  
git stash list # 列出當前的 stash 清單  
git stash apply stash@{<id>} # 套用 ID 為 <id> 的 stash 到當前分支上  
git stash drop stash@{<id>} # 刪去一個 stash  
git stash pop stash@{<id>} # 等同 apply + drop
```

使用 git stash 可以暫存所有「已被 git 追蹤已經放入或未有放入暫存區 (staging area)」的變動，但不會暫存新增未被 git 管控或被忽略的檔案。

其他指令

git cherry-pick

抽出一個 commit 並整合到當前的 HEAD

```
git cherry-pick <commit> # 擷一個 分支 整合  
git cherry-pick <commit1>...<commit2>  
# 擷 commit1 至 commit2 之間的 整合 (不含 commit1)  
git cherry-pick <commit1>^..<commit2>  
# 擷 commit1 至 commit2 之間的 整合 (含 commit1)
```

其他指令

git cherry-pick

抽出一個 commit 並整合到當前的 HEAD

```
git cherry-pick <commit> # 擷一個 分支 整合  
git cherry-pick <commit1>...<commit2>  
# 擷 commit1 至 commit2 之間的 整合 (不含 commit1)  
git cherry-pick <commit1>^..<commit2>  
# 擷 commit1 至 commit2 之間的 整合 (含 commit1)
```

整合之後，新的 commit 改變的內容會與舊 commit 一致，但會有一個新的 commit 號。

```
# 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

檢視一個檔案中哪一行是由誰什麼時候在哪個 commit 中新增。

```
git blame <path>
```

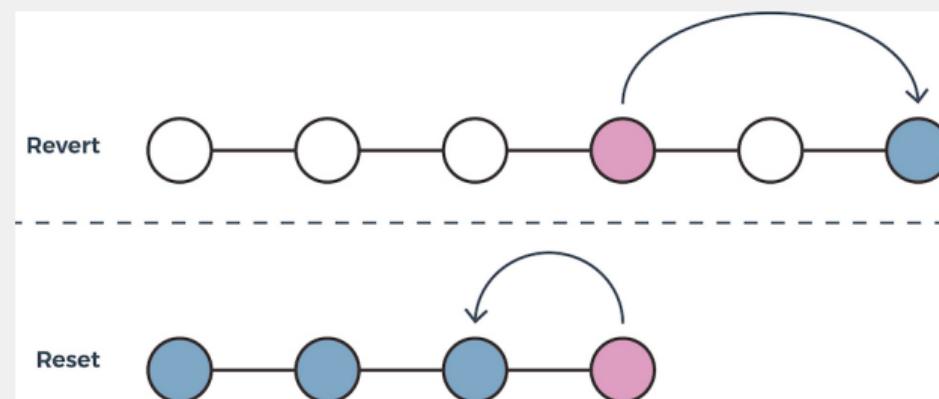
```
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 1) set number
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 2)
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 3) call plug#begin()
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 4) Plug 'nvim-tree/nvim-tree.lua'
^9b5ab4e (Ostrichbeta 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 'luochen1990/rainbow'
^9b5ab4e (Ostrichbeta Chan 2024-11-22 13:12:09 +0000 7) Plug 'sbdchd/neoformat'
^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 'neoclide/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 2025-09-28 08:54:47 +0000 16) Plug 'wasabeef/bufferin.nvim'
887632e1 (Ostrichbeta Chan 2025-09-28 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 'mfussenenegger/nvim-dap'
f0faaff29 (Ostrichbeta Chan 2024-11-25 10:21:54 +0000 23) Plug 'mfussenenegger/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

使用一個 Commit 來取消不需要的 Commit。

```
git revert <commit>
```



git revert 和 git reset 的區別

其他指令

git checkout (補充用法)

git checkout 還可以由其他分支抽出檔案。

```
git checkout <branch> <file> \dots # 由\text{branch}分支復原  
git checkout [--] <file> 放棄暫存區的更改，回到暫存前狀態
```

為什麼要加--？

git checkout 會優先匹配分支的名字。假若你有分支叫 c.cpp，但是同時又想復原 c.cpp 檔案，使用 git checkout c.cpp 會切到 c.cpp 分支，唯有加上 -- 才可以跳過分支名字的匹配直接復原檔案。

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

4. 練習

5. 附錄

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

3.1 GitHub 簡介

3.2 Git 遠端連線

3.3 向他人的倉庫貢獻

4. 練習

5. 附錄

GitHub 是什麼

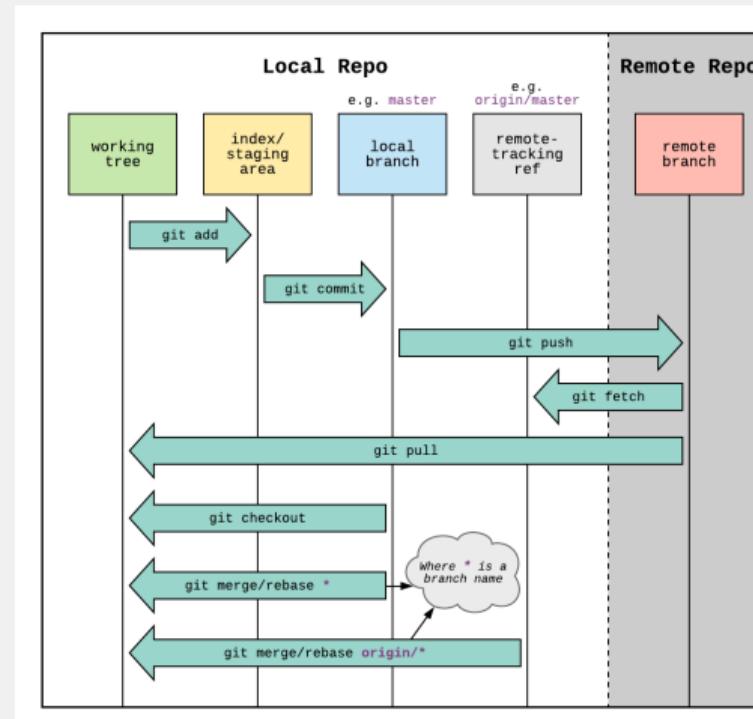
GitHub 是一個可以貯存、分享及與他人共同協力開發的 code 雲端託管平臺。使用 GitHub 存儲的倉庫可以：

- 展示或共享你的工作；
- 持續跟蹤和管理對於原始碼的更改；
- 讓他人審閱你的原始碼，給出改進的建議；
- 在公用專案中與他人合作，而毋須擔心你與其他專案工作者的部分會發生衝突

— GitHub 官方文檔

GitHub 是什麼

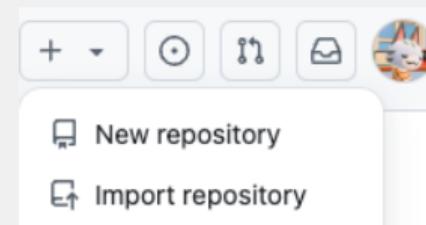
GitHub 上的倉庫是一個遠端倉庫，與本地倉庫聯動可以更好的將倉庫分享出去或與他人協作。



GitHub 倉庫管理

新增或匯入倉庫

按 GitHub 網頁的右上角「+」號可以選擇新增新倉庫或由其他平臺匯入已有倉庫。



GitHub 新增倉庫

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 根據範本新增倉庫

GitHub 新增倉庫

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 倉庫擁有者（個人或組織）

● Repo name 倉庫名字

● Description 倉庫簡介

● Public 公開倉庫，任何人可檢視

● Private 私有倉庫，僅自己與部分人可見

GitHub 新增倉庫

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 自動新增 README 檔案
- Add .gitignore 按照範本新增 .gitignore
- Choose a license 選擇倉庫授權規範 (Apache, BSD, GPL ...)

GitHub 新增倉庫

新增倉庫之後將進入倉庫頁面

Quick setup — if you've done this kind of thing before

or <https://github.com/Ostrichbeta/WHUAI-test.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...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
```

如要刪除倉庫，在倉庫頁面 Settings - General - Danger Zone - Delete this repository

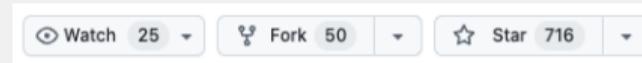
GitHub 其他倉庫功能



A screenshot of a GitHub repository page for 'WHUAI-Dog / WHUAI-test'. The top navigation bar includes links for Code, Commits, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The Issues link is highlighted.

- **Issues** 向倉庫回饋意見，反饋 Bug
- **Pull Requests** 合併自己及其他合作者提交的分支
- **Actions** 自動化編譯及分發的流程
- **Projects** 鏈結專案版，列出開發方向等
- **Wiki** 可存放倉庫使用文檔

GitHub 其他倉庫功能



- **Watch** 關注倉庫，有新動向會發送電郵通知
- **Fork** 克隆倉庫到自己的帳戶下
- **Star** 紿喜歡的項目加星標，星標更多的項目將獲更多推薦

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

3.1 GitHub 簡介

3.2 Git 遠端連線

3.3 向他人的倉庫貢獻

4. 練習

5. 附錄

Git 遠端連線常用指令

涉及與遠端倉庫的聯繫時，主要會用到如下的指令：

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

遠端倉庫管理

git remote

管理本地倉庫所鏈結的遠端倉庫。

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

- <remote-name>：遠端倉庫名字（可以自取），一般是 origin
- <remote-url>：遠端倉庫的 url

遠端倉庫管理

git remote

管理本地倉庫所鏈結的遠端倉庫。

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

- <remote-name>：遠端倉庫名字（可以自取），一般是 origin
- <remote-url>：遠端倉庫的 url

一個範例如下：

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

遠端倉庫管理

git remote

git remote 還有其他一些指令：

```
git remote get-url <remote-name> # 取得遠端倉庫的 URL  
git remote rm <remote-name> # 移除遠端倉庫  
git remote rename <remote-name> <new-remote-name> # 重新命名遠端倉庫  
git remote set-url <remote-name> <remote-url> # 變更遠端倉庫的 URL  
git remote show <remote-name> # 顯示遠端倉庫詳細資訊
```

推送至遠端倉庫

git push

將本地倉庫向遠端倉庫推送，常用：

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

如：

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

- -u：設定遠端倉庫及分支，設定一次之後可以只用打 git push 就可推送
- 若目標倉庫有帳戶和密碼認證，可能還需要輸入對應的帳戶和密碼
- 若遠端不存在分支 <branch>，會自動新增

推送至遠端倉庫

```
$ 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，為什麼我 push 不上 GitHub ?

推送至遠端倉庫

```
$ 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，為什麼我 push 不上 GitHub ?

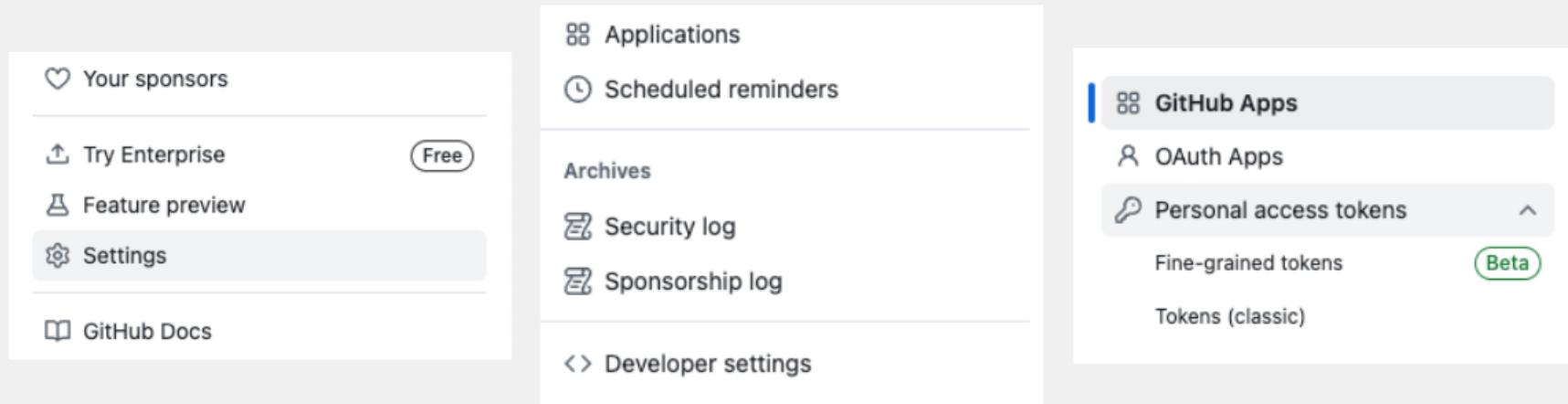
2021 年 8 月 13 日之後，GitHub 使用者無法再使用帳戶和密碼與遠端倉庫交互，只能：

- 1 生成並使用 Token，然後將 Token 用作 Git 推送時的密碼
- 2 安裝 Git Credential Manager 並登入，使用其存儲帳戶資訊
- 3 系統生成 SSH Key 後匯入 GitHub，使用 SSH 連線

推送至遠端倉庫

生成 Token

點按右上角用戶頭像 - Settings - 左側欄最下方 Developer settings - Personal access tokens



推送至遠端倉庫

Token 種類

Personal access tokens (classic) 傳統的 GitHub 使用 Token，使用上較簡易，使用傳統的 Personal access token 可以操作你所有有訪問許可權的倉庫。可以通過「Select scopes」可以選定 token 對於這些倉庫所具有的作用域。

Fine-grained tokens 比較新的 Token 格式。與傳統的 PAT 相比，對於權限的管理更加自由。

- 可以指定 Token 的適用範圍（單用戶還是組織，所有倉庫還是僅限特定倉庫）
- 組織管理者可以更方便下發新的 Token
- 與傳統 PAT 相比，Fine-grained tokens 可以設定的作用域更廣

推送至遠端倉庫

在 Personal access tokens 介面若是選擇 Fine-grained tokens：

- 1 點「Generate new token」，填寫 Token 名字、資源擁有者（個人或者組織）及到期日。
- 2 設定作用的倉庫範圍，選所有倉庫「All repositories」或指定的倉庫「Only select repositories」
- 3 下方的 Permissions 中設定權限，至少在「Repository permissions」中將「Contents」設定為「Read and write」，其他可以不設定
- 4 點按「Generate token」生成 Token，生成的 token 僅會顯示一次，請進行記錄



推送至遠端倉庫

在 Personal access tokens 介面若是選擇 Tokens (classic)：

- 1 點「Generate new token」下的「Generate new token (classic)」
- 2 設定 Token 的備註，有效期及作用域（至少勾選「repo」項）
- 3 點按下面的「Generate token」生成 Token，生成的 token 僅會顯示一次，請進行記錄

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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 Copy	Delete
--	---------------------

推送至遠端倉庫

在取得對應的 Token 後，有兩種方式使用：

- 在每一次進行 push 操作時，於密碼處輸入所生成的 Token
- 在 remote 的 URL 中加入 Token，每一次 push 操作時就不需要再輸入密碼。格式是 `https://<token>@<remote-url>`，如：

```
https://ghp_sMvYSxxxxx7od@github.com/WHUAI-Dog/WHUAI-test
```

推送至遠端倉庫

當 push 發生衝突時

假設現在雲端比本地領先一個 commit (可能是其他開發者 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'
```

左圖：遠端，右圖：本地

如果此時本地提交了一個新的變更，並嘗試推送：

```
$ 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.
```

推送至遠端倉庫

當 push 發生衝突時

假設現在雲端比本地領先一個 commit (可能是其他開發者 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'
```

左圖：遠端，右圖：本地

如果此時本地提交了一個新的變更，並嘗試推送：

```
$ 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.
```

遠端有本地不存在的 commit，需要先拉取遠端更新才可以進行 push

拉取遠端更新

git pull

拉取遠端的更新，並將其合併至本地倉庫中。無參數時 merge 遠端，加 --rebase 時進行 rebase 的動作

- 如果遠端倉庫比本地新（本地沒有遠端不存在的 commit 可以直接合併）
- 如果遠端倉庫和本地都有對方不存在的 commit，則需要手動 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..b6bb6ae  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.
```

拉取遠端更新

```
$ 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
```

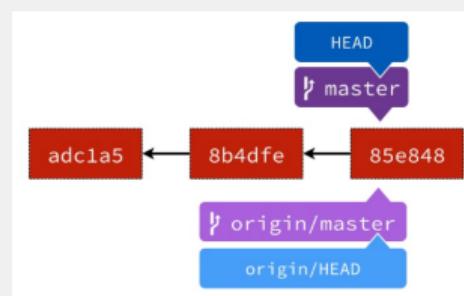
有衝突時需要手動合併

拉取遠端更新

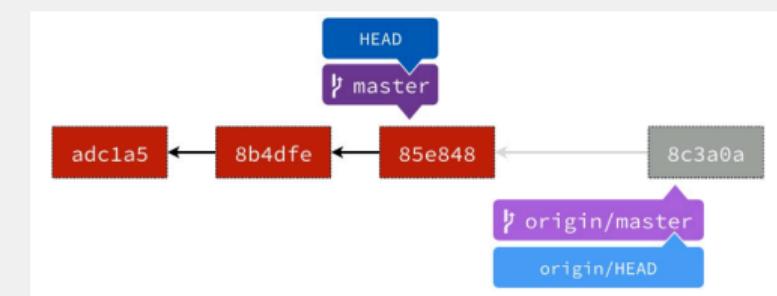
git fetch

僅拉取遠端倉庫的代碼，但是不進行更新。

- 「git pull」實際上是「git fetch」與「git merge」或者「git rebase」的組合



git fetch 之前



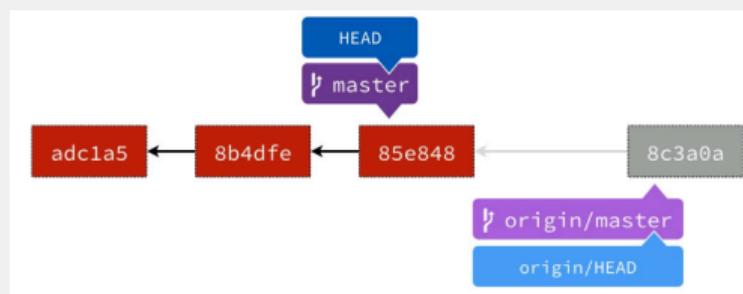
git fetch 之後

拉取遠端更新

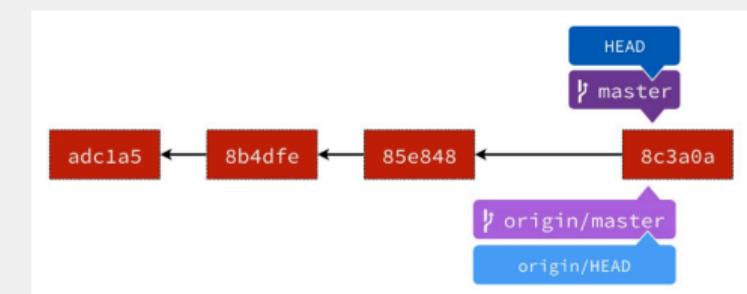
git fetch

僅拉取遠端倉庫的代碼，但是不進行更新。

- 「git pull」實際上是「git fetch」與「git merge」或者「git rebase」的組合



git fetch 之後



git merge 之後

克隆倉庫

git clone

將遠端倉庫克隆至本地。

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

- path如果不加，預設存儲到倉庫名字相同的目錄
- -b 將本地分支切換至 <branch>
- --single-branch 僅克隆單一分支（有 -b 同步 <branch>，沒有預設同步主分支）
- --depth 僅同步最近 n 個 commit（預設將同步所有的 commit）

--single-branch 和--depth 指令在同步大倉庫時尤為有用。

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

3.1 GitHub 簡介

3.2 Git 遠端連線

3.3 向他人的倉庫貢獻

4. 練習

5. 附錄

提交至其他人的倉庫

預設情況下，每一個帳戶僅可以對自己的倉庫直接進行 push 的操作，如果對於他人擁有的倉庫進行交互，則：

- 成為其他人倉庫具有直接存取（Direct access）許可權的合作者
- 克隆倉庫到自己帳戶，然後做出修改後提交回原始倉庫

新增倉庫合作者

倉庫合作者可以在項目頁面「Settings」 - 「Collaborators」 - 「Manage access」 中新增倉庫具有直接存取權限的人。

The screenshot shows the GitHub repository settings page for managing access. The left sidebar lists various settings categories like General, Access, Collaborators, and Security. The 'Collaborators' tab is selected. The main area is titled 'Who has access' and shows two sections: 'Public repository' (which is public and visible to anyone) and 'DIRECT ACCESS' (which shows 0 collaborators have access). A large button at the bottom says 'Add people'.

Actions Projects Wiki Security Insights Settings

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security

Deploy keys

Secrets and variables

Who has access

Public repository

This repository is public and visible to anyone

Manage

PUBLIC REPOSITORY

This repository is public and visible to anyone.

Manage

DIRECT ACCESS

0 collaborators have access to this repository. Only you can contribute to this repository.

Manage access

You haven't invited any collaborators yet

Add people

新增倉庫合作者

組織所擁有的倉庫可在「Settings」 - 「Collaborators and teams」 中新增倉庫具有特定權限的人或團隊。

The screenshot shows the GitHub repository settings page for a public repository. The left sidebar lists various settings categories: General, Access (selected), Collaborators (highlighted in blue), Moderation options, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), Security (Code security, Deploy keys, Secrets and variables). The main content area is titled 'Who has access'. It shows that the repository is a 'Public repository' visible to anyone. Below this, there are 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. At the bottom, under 'Manage access', it says 'You haven't invited any collaborators yet' and features a 'Add people' button.

克隆、更改和提交

將其他人所有的倉庫克隆後修改後提交，就算沒有對原始倉庫的直接訪問許可權，也可以使原倉庫獲得批准後合併自己的修改。

- 1 在原倉庫使用「Fork」功能複製一份原始倉庫到自己帳戶下
- 2 對自己帳戶下的倉庫進行修改
- 3 在自己倉庫下使用「Pull requests」功能向原倉庫發起合併申請
- 4 原倉庫擁有者通過申請並處理可能的衝突之後，克隆的倉庫會合併至原始倉庫

自動合併

若克隆倉庫與原始倉庫之間沒有衝突，在許可之後可以自動合併。

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 | bc52ade | <>

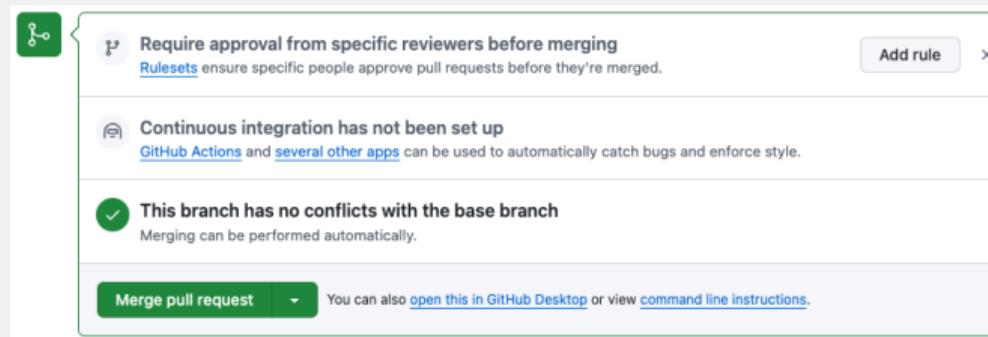
Showing 1 changed file with 1 addition and 0 deletions. Split Unified

EULB.txt

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

自動合併

若克隆倉庫與原始倉庫之間沒有衝突，在許可之後可以自動合併。



手動合併

若克隆倉庫與原始倉庫之間有衝突：

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 [View commit](#) [Copy commit URL](#)

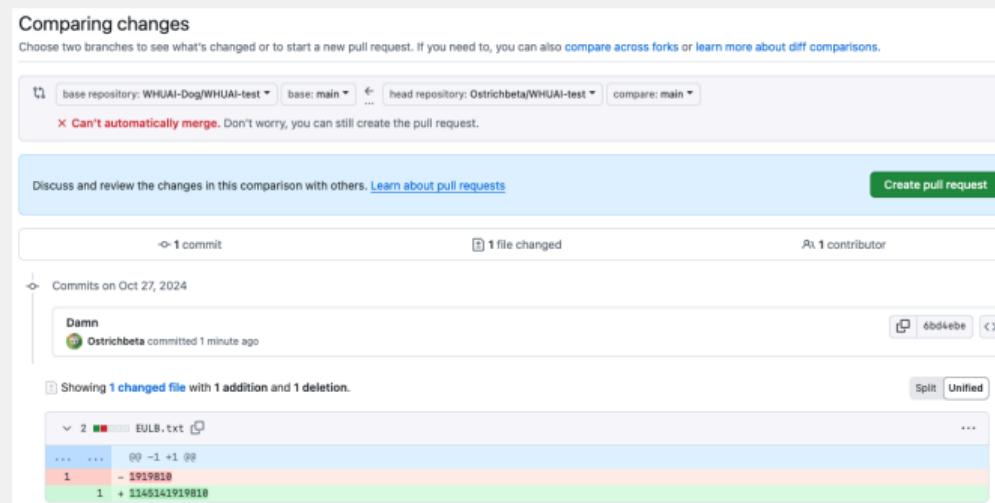
Showing 1 changed file with 1 addition and 1 deletion.

EULB.txt

Line	Change Type	Content
1	-	1919810
1	+	1148141919810

手動合併

若克隆倉庫與原始倉庫之間有衝突：



- 1 撤回合併申請，在克隆倉庫中 pull 原始倉庫解決衝突後再提交申請
- 2 由原始倉庫的維護者解決衝突

手動合併

若衝突有原始倉庫的維護者解決：

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

- 在網頁中按「Resolve conflicts」手動編輯衝突部分解決
- 將衝突處移到本地進行處理
 - 1 原分支複製出一個合併用分支
 - 2 在複製的分支中解決衝突後合併
 - 3 在原分支中合併複製分支
 - 4 提交正常合併後的原分支

回退合併

原倉庫擁有者如果在合併之後發現原倉庫出現了重大問題，或只是單純想撤回合併，可以在合併的頁面按「Revert」將倉庫回退到合併之前的狀態。

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 Verified 733a49f
- Merge branch 'main' into main Verified d2ac536

Ostrichbeta merged commit e1d3218 into WHUAI-Dog:main 1 minute ago

Revert

Create a new pull request to revert these changes

No milestone

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

4. 練習

5. 附錄

小練習

本倉庫有一個 zip 包的練習倉庫供練習 Git 指令。內含有 A, B, C 三個資料夾。

- 1 A 資料夾是一個未被 Git 管控的資料夾，初始化並完成一個 commit。
- 2 B 資料夾的最新一個 commit 中，token.txt 被加入了奇怪的內容。使用 git log 查詢，並使用 git reset 或其他指令對其進行退回更新。
- 3 C 資料夾中，合併 hk 分支到 main 分支，解決可能出現的合併衝突。

若有不明之處，可以通過各種方式詢問搜尋引擎或者 AI 解決問題。

1. Git 簡介

2. Git 基本操作

3. Git 遠端聯動

4. 練習

5. 附錄

Git 輔助資料

除了命令行的介面外，還可以使用一些 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>

參考資料

- 「為你自己學 Git」高見龍 - <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/>