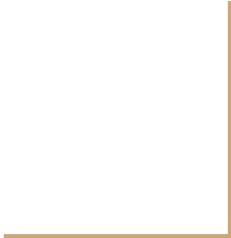




Git x GitHub

版本控制系統 x 原始碼代管服務平台



目錄

1. 基本介紹

- 版本控制
- Git
- GitHub

2. Git 下載/安裝

- Windows 安裝
- MacOS 安裝

3. Linux 指令

4. Git 初始設定與概念

- 使用者設定 global & local config
- 工作區域
- 檔案狀態的生命週期

5. Git 基礎指令 - 1

- 建立數據庫
- 檔案狀態檢視
- 加入/移出暫存區
- 檢視歷史紀錄

6. Git 基礎指令 - 2

- 修改檔案
- 檔案還原修改
- 檔案差異比較
- 檔案復原/切換版本

7. Git 團隊開發

- 開發工作流程
- 分支管理

8. 遠端數據庫

- 遠端數據庫介紹
- 遠端數據庫連線管理
- 本地端與遠端數據庫資料同步

9. 編輯器整合應用 /GUI 介面

- VS Code
- Visual Studio
- GitHub Desktop

10. 附錄

基本介紹

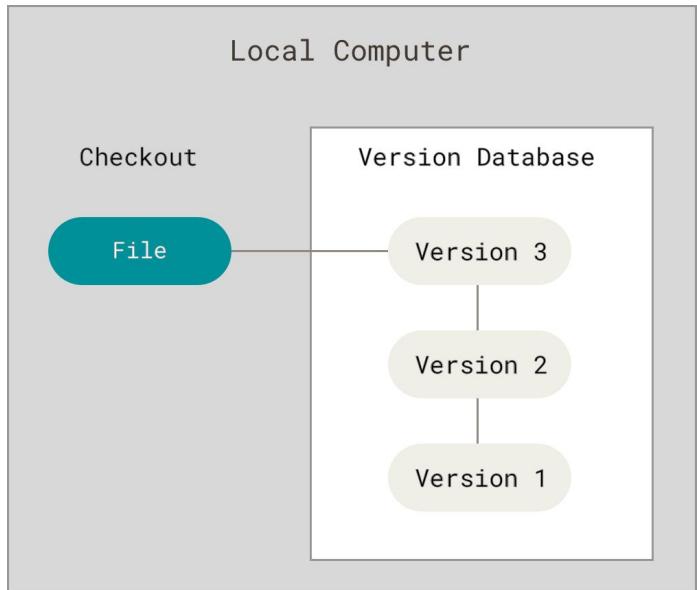
什麼是版本控制?

用來紀錄檔案的變更，當你未來想回到某一個版本時就可以簡單呼叫。

大致上分為**【本地端】**、**【集中化】**、**【分散式】**三種版本控制。

本地端版本控制

做法簡單，但容易覆蓋錯誤

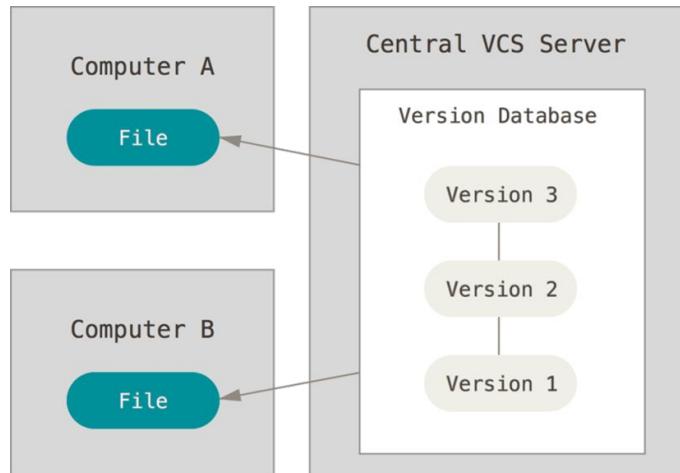


課程企劃案	課程企劃案-20251018
課程企劃案-2	課程企劃案-20251020
課程企劃案-20250711	課程企劃案-20251020-final-version
課程企劃案-20250711-最新版	課程企劃案-final-version

- 運用了「備份」和「檔案名稱」讓開發者能查找不同版本的檔案。
- 這種作法有以下缺點：
 - 無法得知檔案版本之間的差異
 - 無法得知備份的原因
 - 無法追蹤修改者和修改的內容

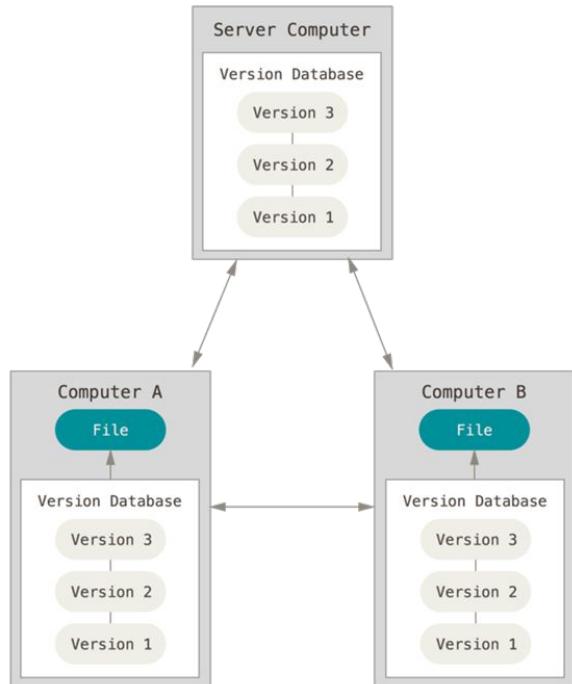
集中化版本控制

中央伺服器故障時，所有人都無法存取檔案



- 資料統一由伺服器管理，開發者連到伺服器讀取資料。
- 解決本地端版本控制無法與其他開發者共同合作的問題。
- 當伺服器損壞時，資料可能會跟著遺失。

分散式版本控制



- Distributed Version Control Systems, 簡稱 DVCSs
- 開發者不但可以取出最新的檔案資訊，還可以將整個檔案夾備份。
- 假如有任何一個伺服器損壞，就可以採用開發者的資料來進行還原。

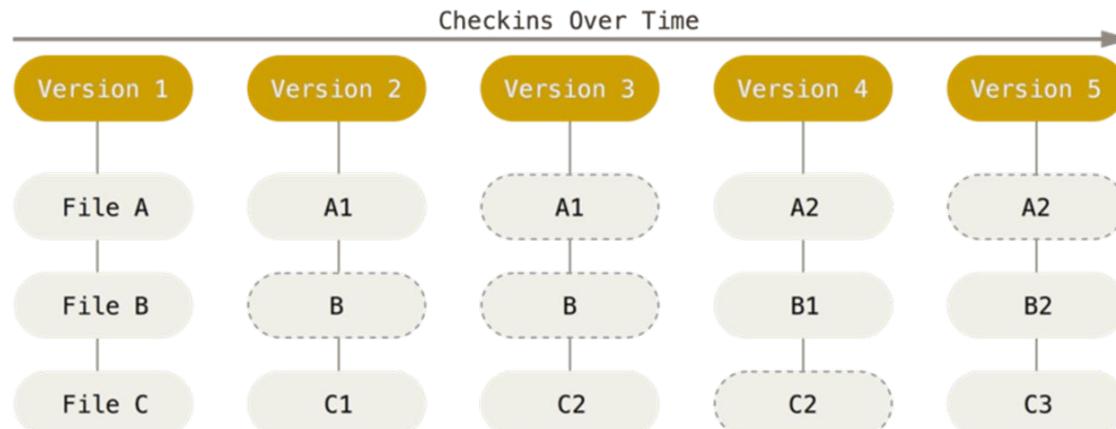
什麼是Git？

Git的介紹

- Git 是由 Linux 之父 Linus Torvalds 在 2005 年親自開發的一套版本控制系統。
- 他當時為了替代原本無法再使用的版本工具(BitKeeper)，在短短 10 天 就打造出了 Git 的核心架構，奠定今日開源世界的標準。
- Git 特色與優點
 - 完全免費、開源
 - 採用「快照(Snapshot)」模型，而非版本差異(Diff)
 - 版本切換超快(因為是快照，而不是重貼差異)。
 - 資料完整性高(不會因為 diff 損壞導致整版出錯)。
 - 分散式系統
 - 每個人 clone 下來的不是副本，而是完整的版本庫。
 - 離線也能建立版本、查版本、開分支。
 - 不依賴中央伺服器。

Git 處理資料的方式 (Snapshot)

- Git 除理資料的方式是將其視為小型檔案系統的一組快照 (Snapshot)。
- 每當提交 commit 時, Git 會紀錄下你所有目前檔案的樣子, 並且參照到這次快照中。
- 為了講求效率, 只要檔案沒有變更, Git 不會再度儲存該檔案, 而是直接將上一次相同的檔案參照到這次快照中。Git 把它的資料視為一連串的快照。



什麼是 GitHub?

GitHub的介紹

- GitHub 是全球最大的線上程式碼雲端託管平台 (Git Server)。
- GitHub 提供以下功能：
 - 程式碼倉庫 (Git Remote Repository)
 - 提供原始程式碼的雲端儲存空間，每個倉庫 (數據庫)都可以設定權限。
 - 版本控制 (Version Control)
 - 開發者可以透過 Git Commit 的功能，備份在特定時間點的程式碼資料，並可以輕鬆地 查看和還原以前的版本。
 - 問題追蹤 (Issue Tracking)
 - 提供問題追蹤系統的功能，包含回報 Bug、討論與規劃、工作事項 等。
 - 程式碼審查 (Code Review)
 - 團隊成員可以在 GitHub 中透過提交評論和建議來進行程式碼審查。
 - 持續整合 (Continuous Integration)
 - 在強調 DevOps 敏捷開發的階段，GitHub 支援持續整合 (CI) 的工具
 - GitHub Copilot

Git 下載 / 安裝

Windows 安裝

Windows 下載安裝方式

1. 在官網首頁中點擊『Install』。
2. 選擇 Standalone Installer 的 Git for Windows Setup 版本。(依照自己的裝置選擇 x64 或 ARM64)

Latest version: 2.51.2 ([Release Notes](#))

Windows macOS Linux Build from Source

[Click here to download](#) the latest (2.51.2) x64 version of **Git for Windows**. This is the most recent [maintained build](#). It was released **16 days ago**, on 2025-10-28.

Other Git for Windows downloads

[Standalone Installer](#)
[Git for Windows/x64 Setup.](#)
[Git for Windows/ARM64 Setup.](#)

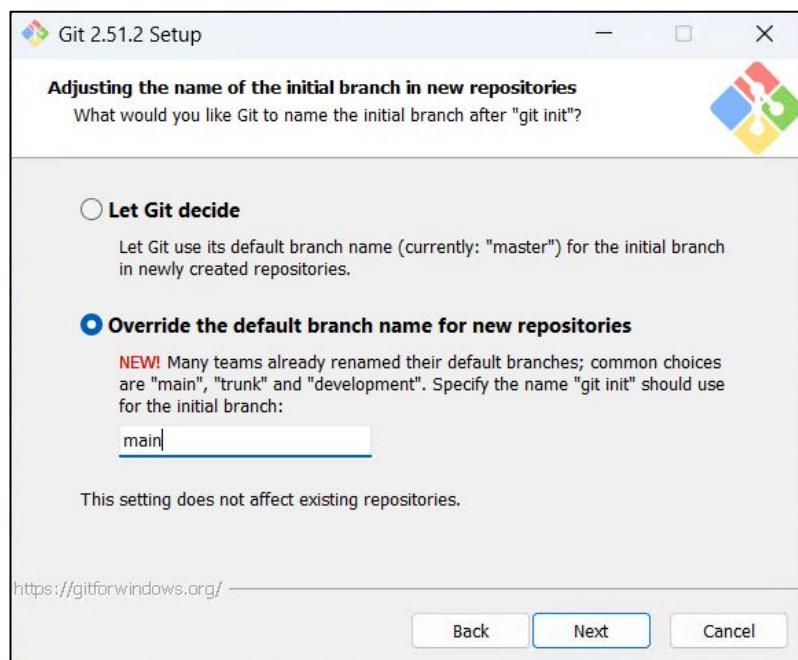
[Portable \("thumbdrive edition"\)](#)
[Git for Windows/x64 Portable.](#)
[Git for Windows/ARM64 Portable.](#)

Using winget tool
Install [winget tool](#) if you don't already have it, then type this command in command prompt or Powershell.
`winget install --id Git.Git -e --source winget`

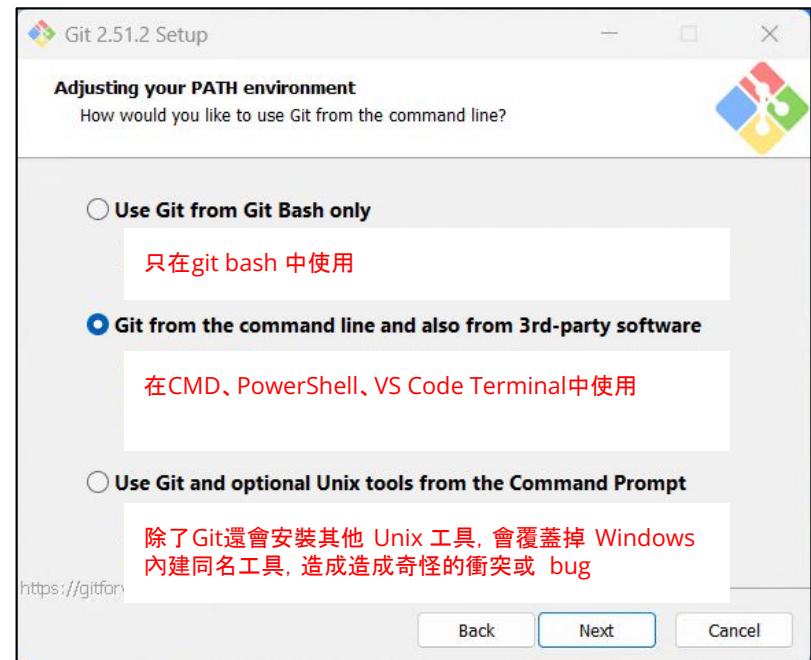
The current source code release is version **2.51.2**. If you want the newer version, you can build it from [the source code](#).

Windows 安裝設定

將預設分支名稱設定『 main 』



設定Git指令的使用地方



macOS 安裝

macOS 下載安裝方式

1. homebrew 方式安裝

- 先安裝 [homebrew](#)
- 輸入指令「`brew install git`」

2. MacPorts 方式安裝

- 先安裝 [MacPorts](#)
- 輸入指令「`sudo port install git`」

Install

Latest version: 2.51.2 ([Release Notes](#))

[Windows](#) [macOS](#) [Linux](#) [Build from Source](#)

There are several options for installing Git on macOS. Note that any non-source distributions are provided by third parties, and may not be up to date with the latest source release.

Choose one of the following options for installing Git on macOS:

Homebrew

Install [homebrew](#) if you don't already have it, then:

```
$ brew install git
```

MacPorts

Install [MacPorts](#) if you don't already have it, then:

```
$ sudo port install git
```

Linux 指令

Linux 基本指令

MacOS / Linux 指令	說明
pwd	顯示目前所在路徑
cd <目的地路徑>	改變檔案目錄 change directory
mkdir <資料夾名稱>	建立新的資料夾 make directory
ls	列出目前資料夾中的檔案 (不包含隱藏檔) list
touch <檔案名稱.副檔名>	建立檔案
cat <檔案名稱.副檔名>	檢視檔案內容
cp <被複製的檔案> <新檔名>	複製檔案 copy
mv </原本路徑/檔名> </新的路徑/檔名>	移動檔案 move
mv <舊檔名> <新檔名>	更改檔名
rm <檔案名稱.副檔名>	刪除檔案 remove
rm -r <檔案夾名稱>	刪除資料夾
clear	清空畫面
logout	關閉終端機指令

檢查 git 版本

```
git --version || git -v
```

```
MINGW64:/c/Users/WDAGUtilityAccount
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~
$ git --version
git version 2.51.2.windows.1

WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~
$ git -v
git version 2.51.2.windows.1
```

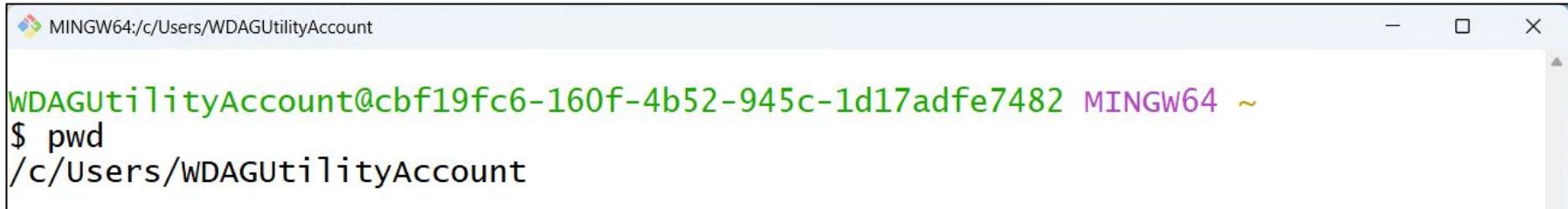
使用者以及哪台電腦

使用中的 Shell/環境類型
Minimal GNU for Windows 64-bit

目前所在位置

查看目前所在位置

```
pwd
```



```
MINGW64:/c/Users/WDAGUtilityAccount
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~
$ pwd
/c/Users/WDAGUtilityAccount
```

~:代表「家目錄」(home directory)

Windows

- 通常是 C:\Users\<使用者帳號>

Linux/macOS:

- 通常是 /home/<使用者帳號>

移動目前位置

cd <目的路徑>

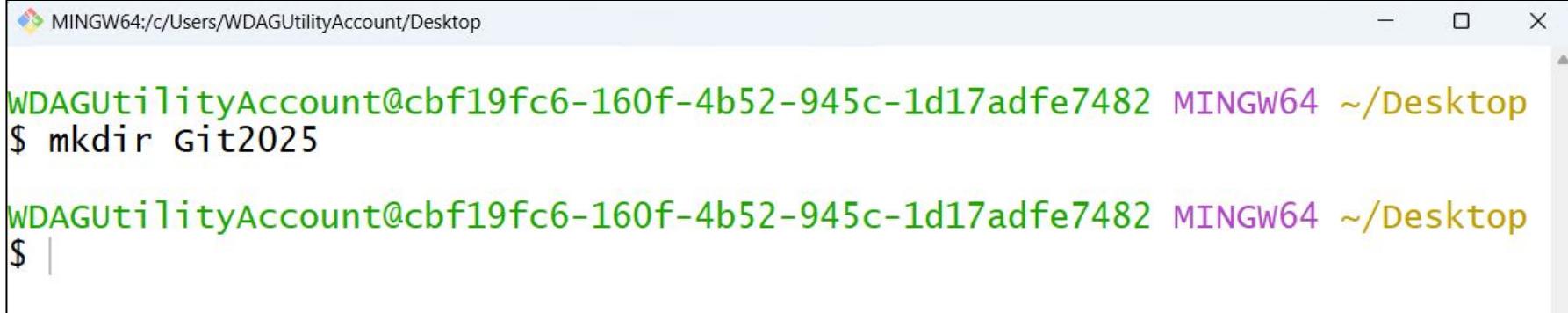
- 「.」 ⇒ 現在的檔案夾位置
- 「..」 ⇒ 回上一層檔案夾
- 「~」 ⇒ 目前使用者的目錄



```
MINGW64:/c/Users/WDAGUtilityAccount/Desktop
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 ~
$ cd .
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 ~
$ pwd
/c/Users/WDAGUTILITYACCOUNT
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 ~
$ cd ..
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 /c/Users
$ pwd
/c/Users
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 /c/Users
$ cd ~
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 ~
$ cd ./Desktop/
WDAGUTILITYACCOUNT@CBF19FC6-160F-4B52-945C-1D17ADFE7482 MINGW64 ~/Desktop
$ |
```

建立新資料夾

```
mkdir <資料夾名稱>
```



The screenshot shows a terminal window titled 'MINGW64:c/Users/WDAGUtilityAccount/Desktop'. The command '\$ mkdir Git2025' is entered and executed successfully, creating a new directory named 'Git2025'. The terminal prompt '\$' is visible at the bottom.

```
MINGW64:c/Users/WDAGUtilityAccount/Desktop
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop
$ mkdir Git2025
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop
$ |
```

檢視目前資料夾中的檔案

ls

- 顯示詳細資訊 (long format)

ls -l

- 顯示所有隱藏檔(含 . 開頭的檔案)

ls -a

- 顯示詳細資訊(含隱藏檔)

ls -al

```
MINGW64:/c/Users/WDAGUtilityAccount/Desktop
$ ls
Git2025/ desktop.ini

MINGW64:/c/Users/WDAGUtilityAccount/Desktop
$ ls -l
total 1
drwxr-xr-x 1 WDAGUtilityAccount 197121 0 Nov 13 12:30 Git2025/
-rw-r--r-- 1 WDAGUtilityAccount 197121 282 Nov 12 04:04 desktop.ini

MINGW64:/c/Users/WDAGUtilityAccount/Desktop
$ ls -a
./ ../ Git2025/ desktop.ini

MINGW64:/c/Users/WDAGUtilityAccount/Desktop
$ ls -al
total 5
drwxr-xr-x 1 WDAGUtilityAccount 197121 0 Nov 13 12:30 ../
drwxr-xr-x 1 WDAGUtilityAccount 197121 0 Nov 13 12:16 ../
drwxr-xr-x 1 WDAGUtilityAccount 197121 0 Nov 13 12:30 Git2025/
-rw-r--r-- 1 WDAGUtilityAccount 197121 282 Nov 12 04:04 desktop.ini
```

Git 初始設定與概念

使用者設定

使用者設定方式

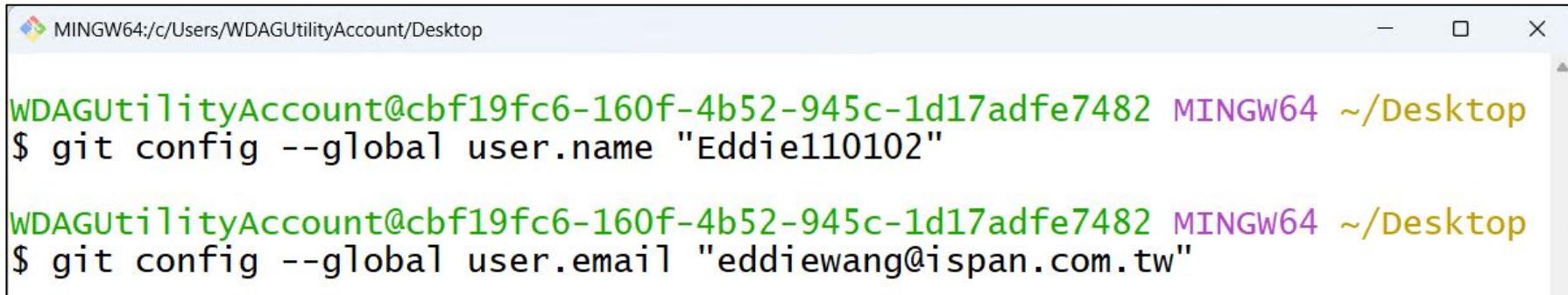
- 當你開始使用 Git, 第一件事就是「**設定使用者身分**」。
- 透過此身份可以辨識各版本 commit 是由誰製作。
- 設定分為三種層級：
 - 優先順序：local > global > system

層級	範圍	檔案位置
system	整台電腦共用	C:\ProgramData\Git\config
global	當前使用者	C:\Users\<使用者名稱>\.gitconfig
local	單一 Git 專案	<專案資料夾路徑>\.git\config

使用者設定方式

```
git config <層級> user.name "你的GitHub暱稱"  
git config <層級> user.email "你的GitHub信箱"
```

- 如果是設定 local 層級，先將終端機路徑設定至專案再輸入此指令即可(<層級>可以忽略輸入)。

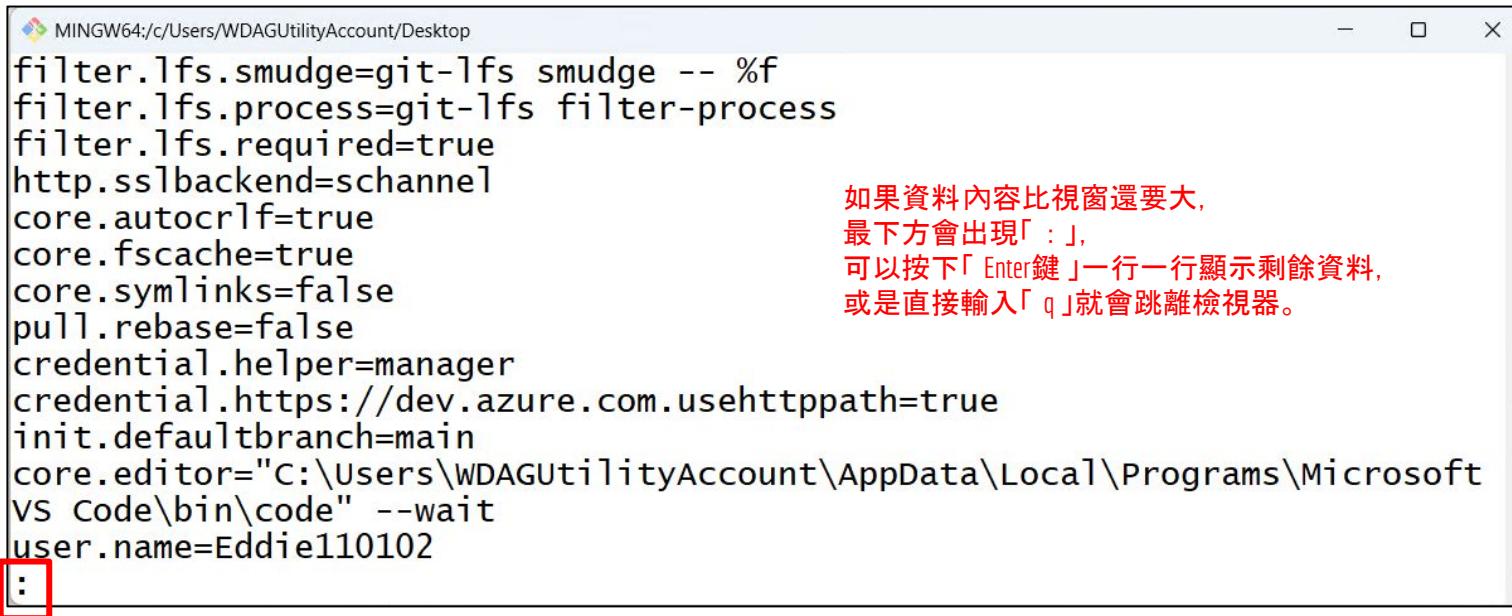


A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/WDAGUtilityAccount/Desktop'. The window contains two command-line entries:

```
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop  
$ git config --global user.name "Eddie110102"  
  
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop  
$ git config --global user.email "eddiewang@ispan.com.tw"
```

檢視 config 設定

```
git config --list
```

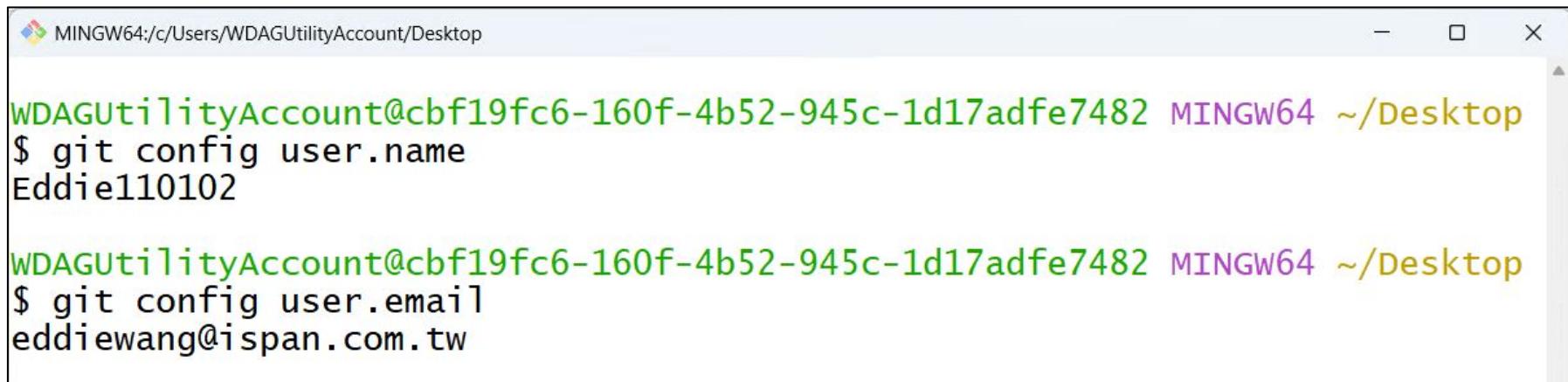


```
MINGW64:/c/Users/WDAGUtilityAccount/Desktop
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=schannel
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
core.editor="C:\Users\WDAGUtilityAccount\AppData\Local\Programs\Microsoft
VS Code\bin\code" --wait
user.name=Eddie110102
:
```

如果資料內容比視窗還要大,
最下方會出現「:」,
可以按下「Enter鍵」一行一行顯示剩餘資料,
或是直接輸入「q」就會跳離檢視器。

檢視單一config設定

```
git config <選項>
```



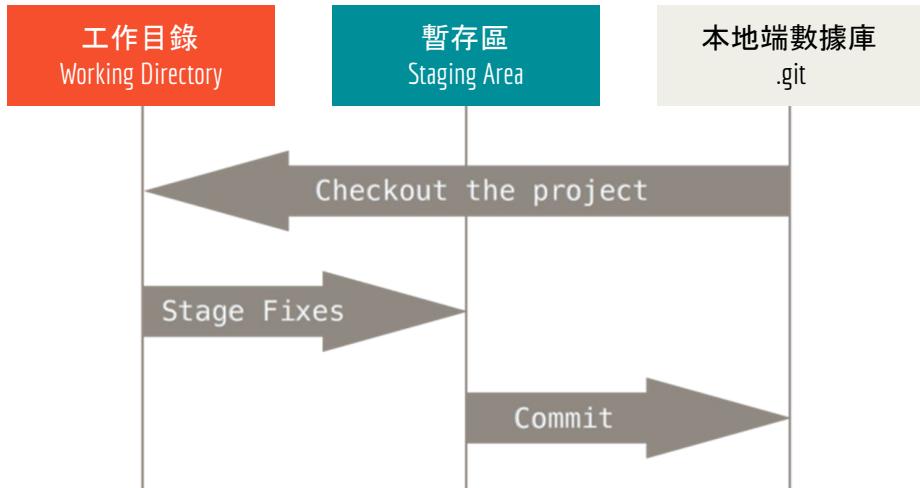
```
MINGW64:/c/Users/WDAGUtilityAccount/Desktop
WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop
$ git config user.name
Eddie110102

WDAGUtilityAccount@cbf19fc6-160f-4b52-945c-1d17adfe7482 MINGW64 ~/Desktop
$ git config user.email
eddiewang@ispan.com.tw
```

Git 工作區

Git工作區域

- Git的版本控制流程中有三個區域：
 - **工作目錄 (Working Directory)**
 - 專案資料夾。
 - 可以在此新增/讀取/修改/刪除檔案。
 - 透過文字編輯器操作。(VS Code)
 - **暫存區 (Staging Area / Index)**
 - 將存成一個版本(Commit)並且有變更的檔案加入此區。
 - 透過Git指令將檔案加入/移除暫存區。
 - **本地端數據庫 (Repository / .git)**
 - 所有的版本(Commit)、分支(Branch)、標籤(Tag)都存在這邊。

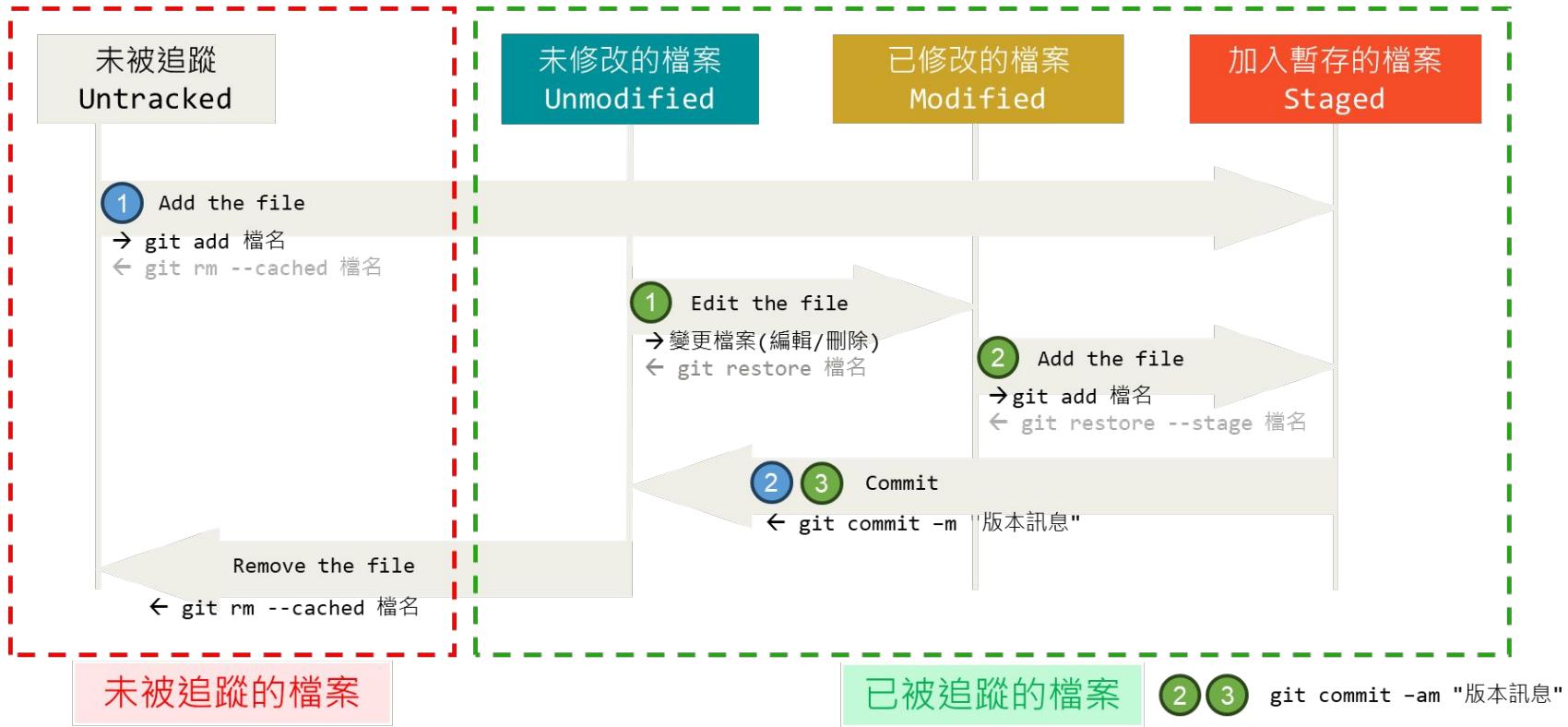


工作目錄(Working Directory)檔案狀態

- 所有檔案會分為三大狀態
 - 忽略檔案
 - 需要使用 `.gitignore` 將要忽略的檔案名稱加入。
 - `.gitignore` 要放在專案資料夾最外層。
 - 被追蹤的檔案(tracked files) 已經被 Git 追蹤(控管)的檔案，又分為三種狀態
 - Staged (Changes to be committed) 檔案已加入暫存區
 - Modified (Changes not staged for commit) 經過最新一次 commit 紀錄後，有被修改過的檔案
 - Unmodified 經過最新一次 commit 紀錄後，還未修改的檔案(不會顯示)
 - 未被追蹤的檔案(Untracked files) 已存在在工作目錄，尚未被 Git 追蹤(控管)的檔案

檔案狀態的生命週期

檔案狀態的生命週期



Git基礎指令 -1

建立 Git 數據庫 (Git Repository)

```
git init
```

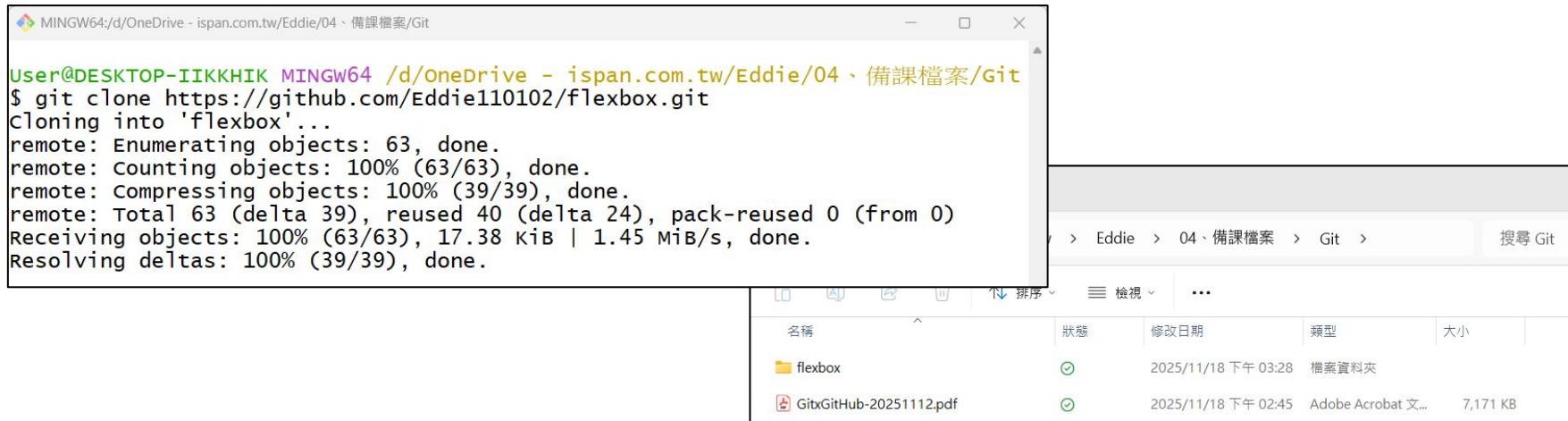
- 輸入此指令後，會在專案資料夾中自動建立「.git」隱藏資料夾。



建立 Git 數據庫 (Git Repository)

```
git clone <remote repository url>
```

- 遠端數據庫已有專案，要複製一份到本地端電腦(預設下載 main / master 分支)。
- 如要直接複製其他分支可以加上 -b <branch name>。



```
MINGW64:/d/OneDrive - ispan.com.tw/Eddie/04、備課檔案/Git
User@DESKTOP-IIKKHIK MINGW64 /d/oneDrive - ispan.com.tw/Eddie/04、備課檔案/Git
$ git clone https://github.com/Eddie110102/flexbox.git
cloning into 'flexbox'...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 63 (delta 39), reused 40 (delta 24), pack-reused 0 (from 0)
Receiving objects: 100% (63/63), 17.38 KiB | 1.45 MiB/s, done.
Resolving deltas: 100% (39/39), done.
```

右側文件資源管理器顯示：

名稱	狀態	修改日期	類型	大小
flexbox	正常	2025/11/18 下午 03:28	檔案資料夾	
GitxGitHub-20251112.pdf	正常	2025/11/18 下午 02:45	Adobe Acrobat 文...	7,171 KB

檢視資料夾中的檔案狀態

git status

- 輸入此指令後，會在專案資料夾中自動建立「.git」隱藏資料夾。

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git status
On branch main

No commits yet
```

nothing to commit (create/copy files and use "git add" to track) 沒有新的檔案可以被追蹤

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git status
On branch main

No commits yet
```

Untracked files:
(use "git add <file>..." to include in what will be committed)
page1.txt

未被追蹤的檔案

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

將檔案加入暫存區

- 新增單一檔案

```
git add <檔案名稱.副檔名>
```

- 新增相同副檔名的檔案

```
git add *.副檔名
```

- 新增所有檔案

```
git add . || git add --all || git add *
```

「.」加入目前資料夾底下的所有檔案

「--all」加入專案資料夾中所有檔案

「*」加入專案資料夾中所有檔案, 不包含隱藏檔(.env / .gitignore)

將檔案移出暫存區

```
git rm --cached <檔案名稱.副檔名>
```

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git status
on branch main
```

```
No commits yet
```

```
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:   page1.txt
```

綠色為已加入暫存區的檔案

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git rm --cached page1.txt
rm 'page1.txt'
```

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git status
On branch main
```

```
No commits yet
```

```
Untracked files:
(use "git add <file>..." to include in what will be committed)
  page1.txt
```

紅色為未被追蹤/已修改的檔案

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

將暫存區內的檔案(們)存成版本

- 暫存區中的檔案可以透過指令，變成一個commit並儲存到Local Repository

```
git commit -m "本次新增/修改的描述(自行撰寫)"
```

```
WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   page1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    page2.txt
    page3.txt

WDAGUtilityAccount@65dcda2-fcf1-4f00-8083-acf29650b99a MINGW64 ~/Desktop/git-lesson (main)
$ git commit -m " note what you do, both chinese or english are fine"
[main (root-commit) afd17/89] note what you do, both chinese or english are fine
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 page1.txt
```

版本訊息

- 版本訊息的註記對於未來找尋資訊有極大幫助，團隊也可以定義版本的格式。

<type>:<message>

類型 <type>	意義	例子
feat	新功能	feat:新增優惠券檢視
fix	修Bug	fix:修正登入跳轉頁面錯誤
docs	文件修改	docs:修改README
style	排版/格式, 程式碼不變	style:調整input元件與input.module.css排版
refactor	重構程式	refactor:優化API結構
chore	建構、工具、套件更新	chore:更新plyr.js套件

檢視commit歷史

- 使用log指令，顯示新到舊的版本紀錄

```
git log
```

- 顯示單行版本紀錄

```
git log --oneline
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git log
commit 1c77073031c19ec4fc55a09bd74ccdef4fa385d6 (HEAD -> main)
Author: Eddie110102 <eddiewang@ispan.com.tw>
Date:   Wed Jul 23 10:28:11 2025 +0800
```

note what you done, both Chinese or english are fine.

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git log --oneline
1c77073 (HEAD -> main) note what you done, both Chinese or english are fine.
```

檢視所有動作歷史

- 可以查看所有HEAD移動紀錄，包含reset...等。

```
git reflog
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git reflog
146de76 (HEAD -> main) HEAD@{0}: reset: moving to 146de76
9bb732c HEAD@{1}: reset: moving to 9bb732c
9bb732c HEAD@{2}: reset: moving to 9bb732c
146de76 (HEAD -> main) HEAD@{3}: commit: add two sentences
9bb732c HEAD@{4}: commit: add one sentence
1c77073 HEAD@{5}: commit (initial): note what you done, both Chinese or english are fine.
```

修改最新一個版本訊息

(盡量少用)

```
git commit --amend -m "重新撰寫版本訊息"
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git log --oneline
f6e6855 (HEAD -> main) not what you do, both chinese or english are fine

User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git commit --amend -m "feat:新增會員登入登出功能"
[main 7392c22] feat:新增會員登入登出功能
Date: Thu Nov 13 16:25:27 2025 +0800
6 files changed, 1 insertion(+)
create mode 100644 .gitignore
create mode 100644 images/image1.txt
create mode 100644 images/image2.txt
create mode 100644 page1.txt
create mode 100644 page2.txt
create mode 100644 page3.txt

User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git log --oneline
7392c22 (HEAD -> main) feat:新增會員登入登出功能
```

多 久 要 commit 一 次？什 麼 情 況 下 要 commit？

- **合理的 commit 時機**

- ✓ 功能完成一小段(例如完成一個按鈕事件、完成一個 API 呼叫)
- ✓ 修正一個 bug
- ✓ 重構程式碼(但不改功能)
- ✓ 修改 UI 樣式或文案
- ✓ 加上註解或整理格式(可單獨 commit)
- ✓ 完成單一任務清單中的一項

一個好的 commit 是「小而完整，清楚描述變更」，
讓未來的自己或其他協作者可以輕鬆還原、理解與除錯。

- **不建議的情況**

- ✗ 完成一大段功能才 commit(不好分辨)
- ✗ 加了很多東西但只 commit 一次(不利追蹤錯誤)
- ✗ 把測試中未完成的半成品 commit

Git基礎指令 -2

修改檔案

- 修改已commit過的檔案後，可以透過「git status」檢查檔案狀態
- 使用「git add」指令、「git commit」指令再存成一個版本紀錄。
- 這兩個動作可以合併成一次

```
git commit -am "本次版本新增/修改的描述(自行撰寫)"
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git commit -am " add one sentence "
[main 9bb732c] add one sentence
 2 files changed, 1 insertion(+)
 create mode 100644 page2.txt
```

還原修改

- 檔案修改後，尚未加入到暫存區(Staging Area)，要還原成修改前的版本

```
git restore <檔案名稱.副檔名>
```

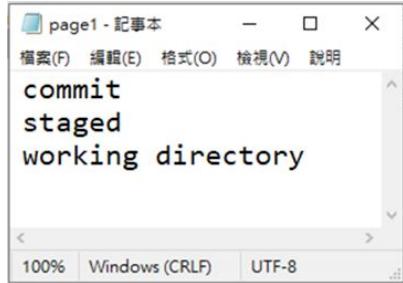


```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git restore page1.txt
```



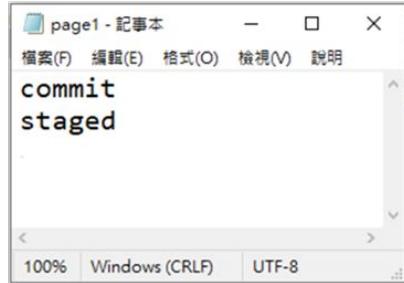
檔案差異比較

工作目錄
Working Directory



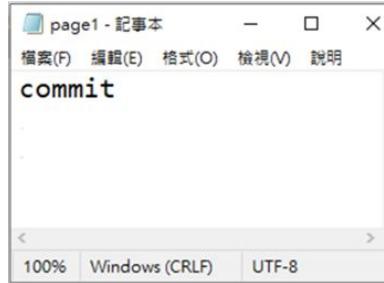
```
page1 - 記事本
commit
staged
working directory
```

暫存區
Staging Area



```
page1 - 記事本
commit
staged
```

Git 工作目錄
.git Directory (Repo)



```
page1 - 記事本
commit
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   page1.txt

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:   page1.txt
```

檔案差異比較 - 情境1

- 對比工作目錄(Working Directory)與暫存區(Staging Area)檔案之間的差異

git diff

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git diff
diff --git a/page1.txt b/page1.txt
index 204be94..10daf4f 100644
--- a/page1.txt          --- 舊版檔案名稱    +++
+++ b/page1.txt          +++ 新版檔案名稱
@@ -1,2 +1,3 @@
第一次commit後又撰寫的文字。
-第二次commit後又增加的文字，這行文字已經加進暫存。
\ No newline at end of file
+第三次commit後又增加的文字，這行文字已經加進暫存。
+在檔案加進status後再進行編輯。
\ No newline at end of file
```

紅色(-)=在舊版本存在、現在被刪除
綠色(+)=在新版本加入

檔案差異比較 - 情境2

- 對比工作目錄(Working Directory)與Git工作目錄(Local Repository / .git)檔案之間的差異

git diff HEAD

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git diff HEAD
diff --git a/page1.txt b/page1.txt
index 765d762..10daf4f 100644
--- a/page1.txt
+++ b/page1.txt
@@ -1 +1,3 @@
-第一次commit後又撰寫的文字。
\ No newline at end of file
+第一次commit後又撰寫的文字。
+第二次commit後又增加的文字，這行文字已經加進暫存。
+在檔案加進status後再進行編輯。
\ No newline at end of file
```

檔案差異比較 - 情境3

- 對比暫存區(Staging Area)與Git工作目錄(Local Repository / .git)檔案之間的差異

git diff --cached HEAD

git diff --staged HEAD

git diff --cached

git diff --staged

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64
$ git diff --cached HEAD
diff --git a/page1.txt b/page1.txt
index 765d762..542b9ca 100644
--- a/page1.txt
+++ b/page1.txt
@@ -1 +1,2 @@
-第一次commit後又撰寫的文字。
\ No newline at end of file
+第一次commit後又撰寫的文字。
+第二次commit後又增加的文字，這行文字已經加進暫存。
```

檔案差異比較 - 情境4

- Git工作目錄(Local Repository / .git)中任意兩個版本(7碼)檔案之間的差異
 - - 為commit1版本中的資料, +為commit2版本中的資料

```
git diff <commit1> <commit2> <檔案名稱.副檔名>
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git diff 1c77073 9bb732c page1.txt
diff --git a/page1.txt b/page1.txt
index e69de29..765d762 100644
--- a/page1.txt
+++ b/page1.txt
@@ -0,0 +1 @@
+第一次commit後又撰寫的文字。
\ No newline at end of file
```

切換到過去的版本 - 情境1

- 保留工作目錄(Working Directory)中的內容, Git 工作目錄(Repository)回到指定的版本
- 指定版本的幾種寫法
 - HEAD 代表目前的版本
 - ^ 代表前一個版本
 - ~數字 代表回到前幾個版本

```
git reset HEAD^^
```

```
git reset HEAD~2
```

```
git reset <commit>
```

切換到過去的版本 - 情境2

- 工作目錄(Working Directory)中的內容與 Git 工作目錄(Repository)回到指定的版本

```
git reset --hard HEAD^^
```

```
git reset --hard HEAD~2
```

```
git reset --hard <commit>
```

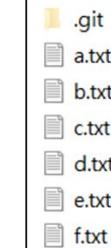
Reset 比較

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git log --oneline
61d5a7a (HEAD -> main) add files ( e.txt & f.txt )
a060f0a add files ( c.txt & d.txt )
8736cf2 add files ( a.txt & b.txt )
```

git reset a060f0a

```
WDAGUtilityAccount@dc3ef9c
$ git reset --soft a060f0a

WDAGUtilityAccount@dc3ef9c
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged")
    new file:   e.txt
    new file:   f.txt
```



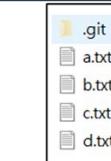
git reset --soft a060f0a

```
WDAGUtilityAccount@dc3ef9c
$ git reset a060f0a

WDAGUtilityAccount@dc3ef9c
$ git status
On branch main
Untracked files:
  (use "git add <file>...")
    e.txt
    f.txt
```

git reset --hard a060f0a

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6
$ git reset --hard a060f0a
HEAD is now at a060f0a add files ( c.txt & d.txt )
```



切換到過去的版本 - 情境3

- 將單一檔案切換到某一個版本的內容

```
git checkout <commit> <檔案名稱.副檔名>
```

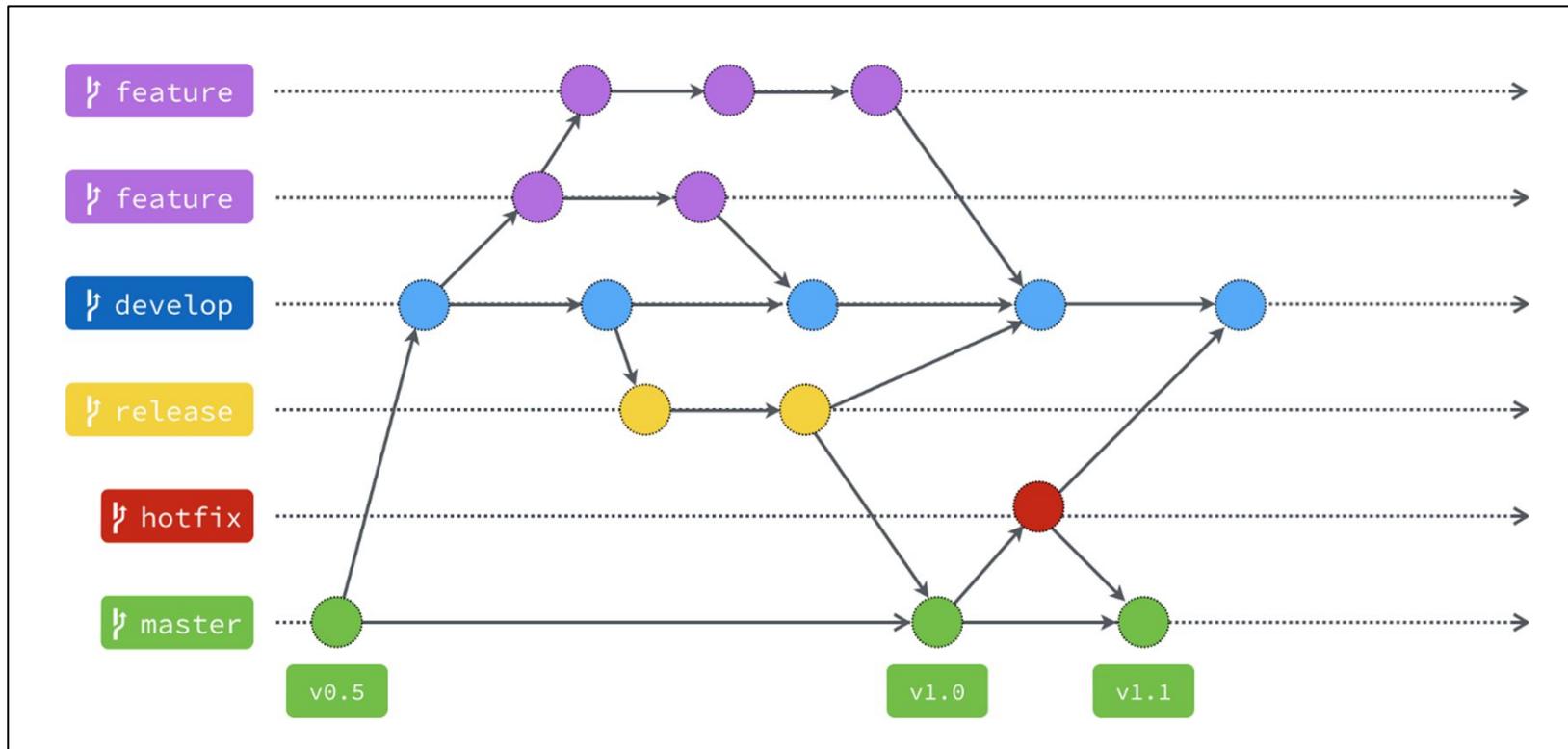
```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git log --oneline
12e456e (HEAD -> main) add one sentence
146de76 add two sentences
9bb732c add one sentence
1c77073 note what you done, both Chinese or english are fine.
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git checkout 146de763 page2.txt
Updated 1 path from 7aaa61c
```

Git 團隊開發

開發工作流程

Git Flow



分支 Branch

分支 Branch

- 將程式開發從開發主線上分離開
 - 不同的版本就使用不同的分支, ex: 1.0、1.0.1、1.1、2.0、....
 - 不同的軟體版本週期使用不同的分支, ex: Alpha、Beta、RC、RTM、.....
 - 單一功能的開發使用不同的分支
 - 不同的開發人員使用不同的分支
 - 為了修復問題也可以使用不同的分支
- 預設的分支名稱: 以前是master, 現在是main
- 透過HEAD指標, 指定目前使用中的branch

檢視目前存在的分支

```
git branch
```

- 「*」代表目前所在分支(分支名稱綠色)

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git branch
* main
```

新增分支

```
git branch <branch name>
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git branch develop
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git branch
  develop
* main
```

切換分支

```
git switch <branch name>
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git switch develop
Switched to branch 'develop'

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (develop)
$ |
```

同時建立並切換分支

```
git switch -c <branch name>
```

- 「-c」: create

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (develop)
$ git switch -c hotfix
Switched to a new branch 'hotfix'
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (hotfix)
$ git branch
  develop
* hotfix
  main
```

修改分支名稱

```
git branch -m <目前branch name> <新的branch name>
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (hotfix)
$ git branch
  develop
* hotfix
  main

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (hotfix)
$ git branch -m hotfix eddie
$ git branch -m hotfix eddie

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (eddie)
$ git branch
  develop
* eddie
  main
```

刪除分支

```
git branch -d <branch name>
```

- 欲刪除正在使用的分支，要先切換到其他分支

- d：會先檢查檔案是否有合併到其他分支，已合併的分支可安全刪除
- D：不檢查檔案，確定不要再保留該分支，強制刪除

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (eddie)
$ git branch
  develop
* eddie
  main

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (eddie)
$ git branch -d develop
Deleted branch develop (was 12e456e).

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (eddie)
$ git branch
* eddie
  main
```

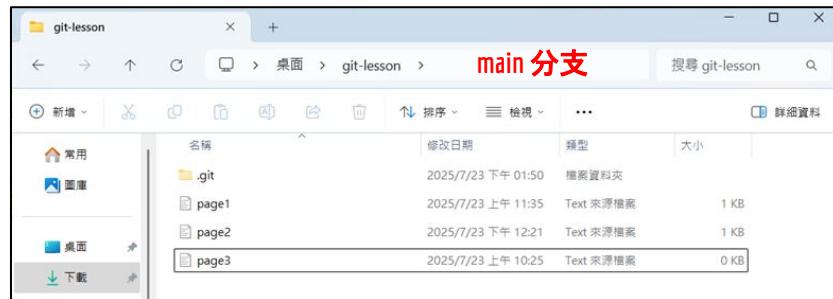
合併分支

```
git merge <branch name> -m "合併訊息"
```

- 如果要把其它分支的內容和並進main，就要先切換到main分支

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (eddie)
$ git switch main
Switched to branch 'main'

WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git merge eddie -m " eddie.txt merge into main "
Updating 12e456e..5bb3daf
Fast-forward (no commit created; -m option ignored)
 eddie.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 eddie.txt
```



遠端數據庫

遠端數據庫介紹

遠端數據庫

- Git數據庫 (Git Repository)
 - 本地端數據庫 (Local Repository) : 專案資料夾中的 Git 工作目錄 (.git)
 - 遠端數據庫 (Remote Repository) : 放在伺服器上的 Git 版本庫, 用來讓開發者之間同步、協作與集中備份。
- 可以自行建置 Git Server 或使用坊間的 Git Repository 託管服務
 - Git Server : GitHub、GitLab、Bitbucket



建立遠端數據庫

1. 數據庫名稱 (Repository Name)

- 使用英文/數字/符號 (- / _ / .)
- 可以使用專案資料夾名稱

2. 數據庫描述 (Description)

- 不限中/英文的數據庫相關描述

3. 設定數據庫狀態 (Visibility)

- 公開 - 所有人皆可檢視
- 私人 - 需要邀請協作者

4. 介紹文件 (README)

- 介紹專案、安裝、使用、功能等等

5. 忽略檔案 (.gitignore)

- 列出不要被 Git 版本控制追蹤的檔案或資料夾

6. 授權 (license)

- 告訴其他工程師你的程式碼如何引用、修改等等

The screenshot shows the GitHub 'Create a new repository' form. The 'General' section is labeled '1'. The 'Repository name' field (marked with an asterisk) is highlighted with a red box and contains the value '1'. The 'Description' field is highlighted with a red box and contains the value '2'. The 'Choose visibility' dropdown is highlighted with a red box and contains the value '3'. The 'Add README' toggle switch is highlighted with a red box and contains the value '4'. The '.gitignore' dropdown is highlighted with a red box and contains the value '5'. The 'Add license' dropdown is highlighted with a red box and contains the value '6'. At the bottom right is a green 'Create repository' button.

建立遠端數據庫

1. 數據庫地址 (Repository URL)

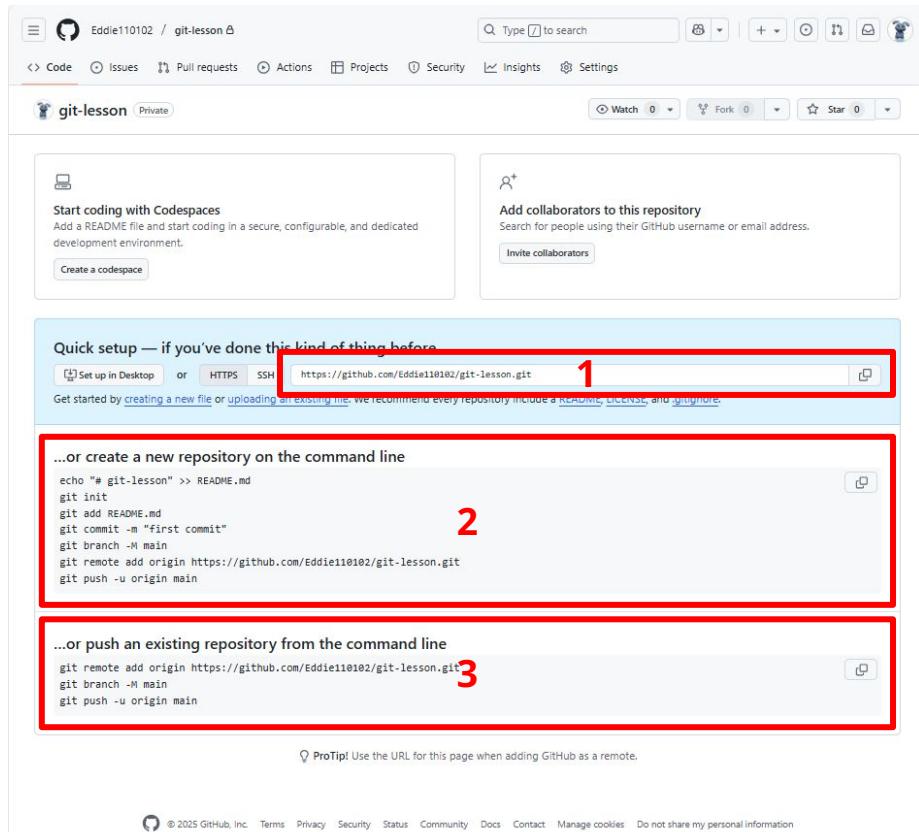
- 要檢視數據庫 / 分享數據庫給使用者
- Clone 數據庫

2. 無本地端數據庫 (No Local Repository)

- 本地端 (電腦) 沒有資料
- 需要建立新的數據庫並連線

3. 已有本地端數據庫 (Existing Repository)

- 本地端 (電腦) 已有專案要連線遠端數據庫



遠端數據庫連線管理

新增遠端數據庫連線

```
git remote add <remote name> <remote repository URL>
```

- <remote name>: 自取遠端數據庫代號。(習慣叫做origin)
- <remote repository URL>: 遠端數據庫地址。(參考講義P73頁)

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
$ git remote add origin https://github.com/Eddie110102/git-lesson.git
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
$
```

檢視遠端數據庫連線

git remote

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git remote
origin
```

- 檢視遠端數據庫(含地址)

git remote -v

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git remote -v
origin  https://github.com/Eddie110102/git-lesson.git (fetch)
origin  https://github.com/Eddie110102/git-lesson.git (push)
```

檢視單一遠端數據庫連線

```
git remote show <remote name>
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/Eddie110102/git-lesson.git
  Push URL: https://github.com/Eddie110102/git-lesson.git
  HEAD branch: (unknown)
```

修改遠端數據庫名稱

```
git remote rename <old remote name> <new remote name>
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
```

```
$ git remote  
origin
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
```

```
$ git remote rename origin eddie
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
```

```
$ git remote  
eddie
```

修改遠端數據庫URL

```
git remote set-url <remote name> <new remote url>
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git remote -v
eddie  https://github.com/Eddie110102/git-lesson.git (fetch)
eddie  https://github.com/Eddie110102/git-lesson.git (push)
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git remote set-url eddie https://github.com/Eddie110102/git-newrepo.git
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git lesson (main)
$ git remote -v
eddie  https://github.com/Eddie110102/git-newrepo.git (fetch)
eddie  https://github.com/Eddie110102/git-newrepo.git (push)
```

刪除遠端數據庫連線

```
git remote remove <remote name>
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
```

```
$ git remote  
eddie
```

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
```

```
$ git remote remove eddie
```

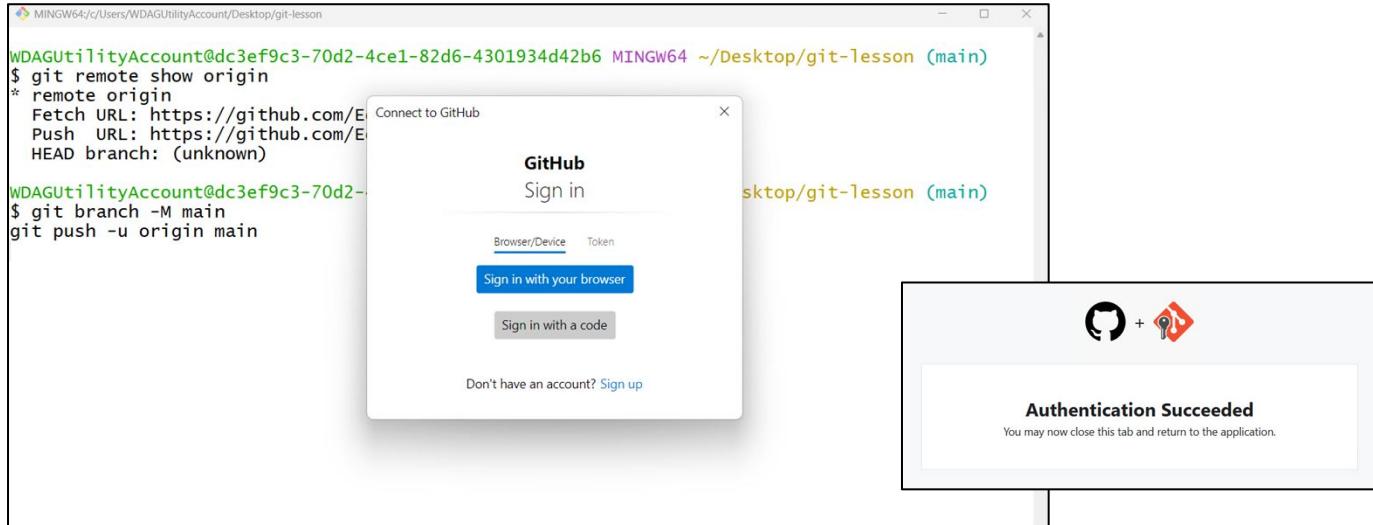
```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/Git Lesson (main)
```

```
$ git remote
```

本地端與遠端 數據庫資料同步

push (推送)

- 將本地端的commit紀錄上傳到遠端數據庫(GitHub Repository)
- 該台電腦第一次連線時會需要登入GitHub，可以使用瀏覽器/Token登入。



push (推送)

- 將本地端數據庫(local)的資料上傳到遠端數據庫(remote)

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

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git push -u origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.45 KiB | 296.00 KiB/s, done.
Total 15 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Eddie110102/firstRepo.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

- u (--set upstream branch): 外來在使用 push & pull 時會自動連結遠端數據庫的這個分支。

pull (提取)

- 將遠端數據庫(remote)的資料同步到本地端數據庫(local)
- pull = fetch + merge

```
git pull <remote name> <branch name>
```

```
WDAGUtilityAccount@dc3ef9c3-70d2-4ce1-82d6-4301934d42b6 MINGW64 ~/Desktop/git-lesson (main)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 1.05 KiB | 154.00 KiB/s, done.
From https://github.com/Eddie110102/firstRepo
  5bb3daf..7b1e677 main      -> origin/main
Updating 5bb3daf..7b1e677
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 README.md
```

- 如果之前已有先加上參數 -u，這邊就可以省略 <remote name><branch name>。

fetch (擷取)

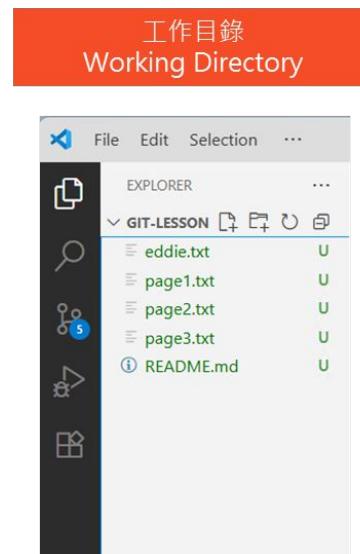
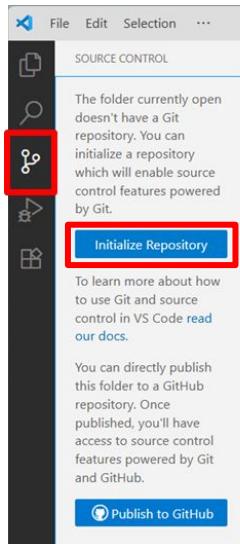
- 僅檢視遠端數據庫最新資料與紀錄
- 不會同步修改本地端專案

編輯器整合應用 GUI介面

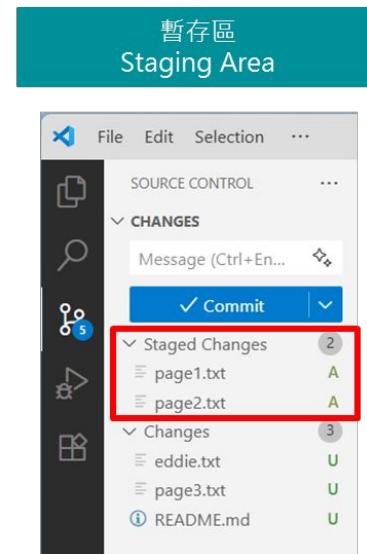
Visual Studio Code

VS Code + Git

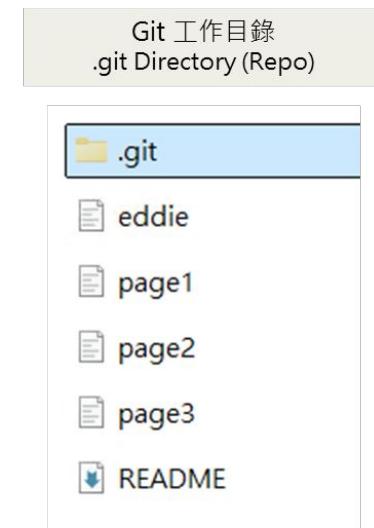
- git init: 開啟專案資料夾 -> 原始檔控制 -> 將存放庫初始化
- git status: 使用 GUI 介面



git init



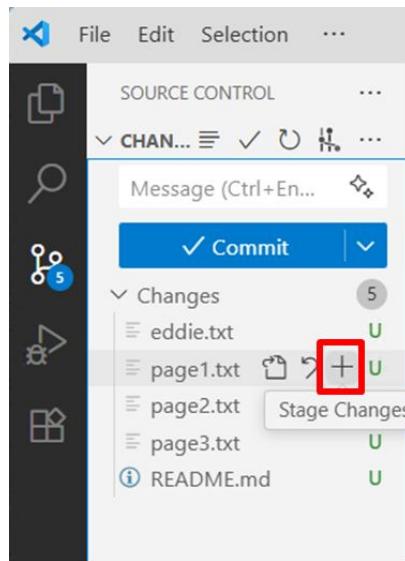
git status



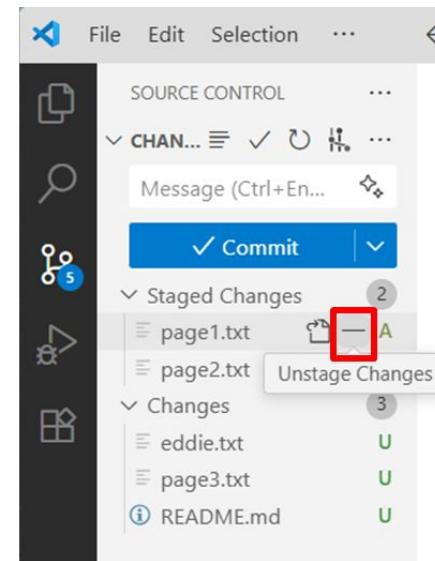
VS Code + Git

- `git add ...` 檔案加入 / `git rm --cached ...` 移出暫存區
- 檔案狀態檢視

- U : Untracked
- A : Added
- M : Modified



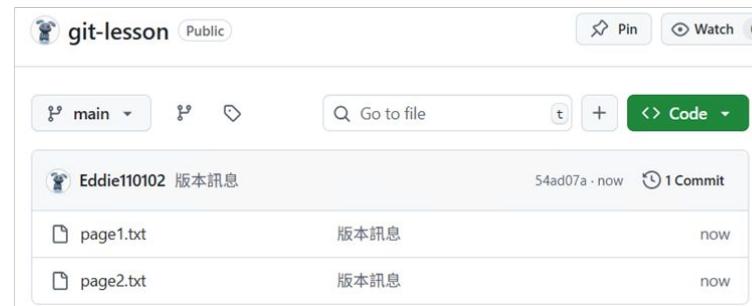
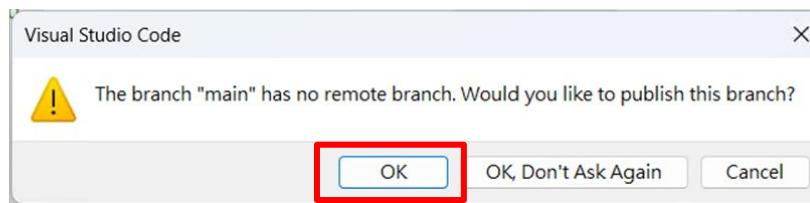
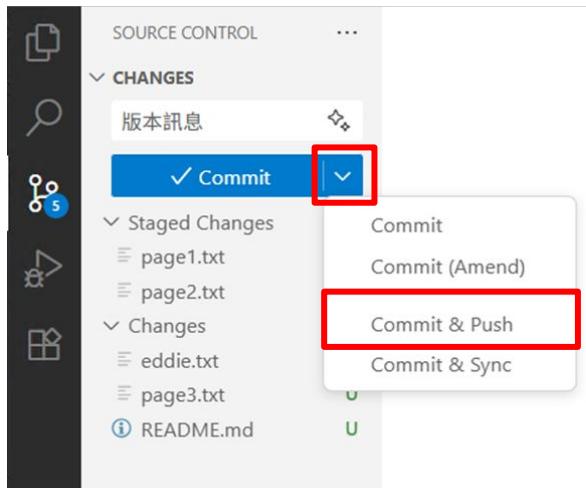
將檔案加入暫存區



將檔案移出暫存區

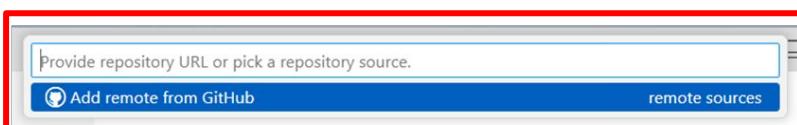
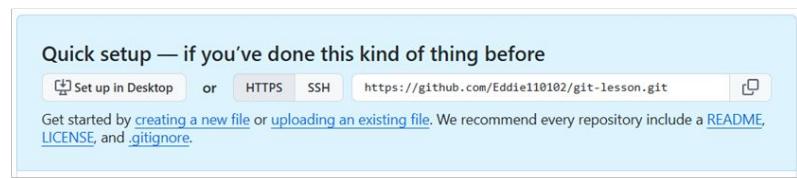
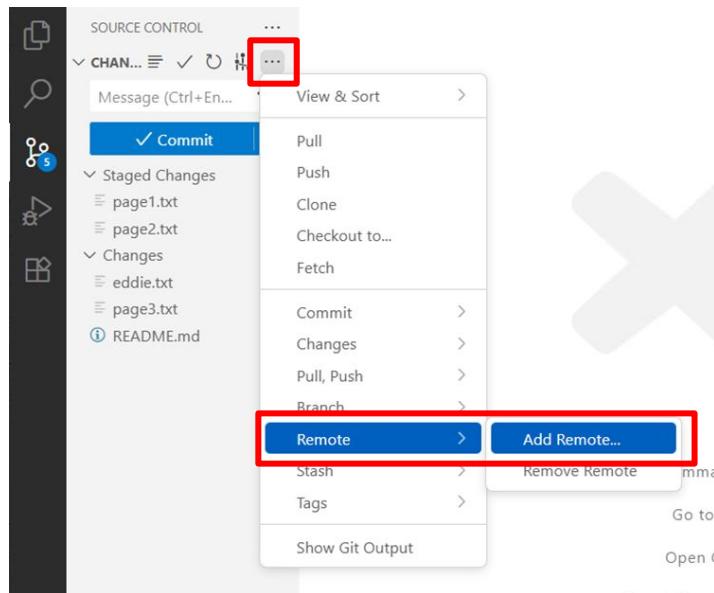
VS Code + Git

- git commit & git push : Commit & Push → 建立分支 → 確定

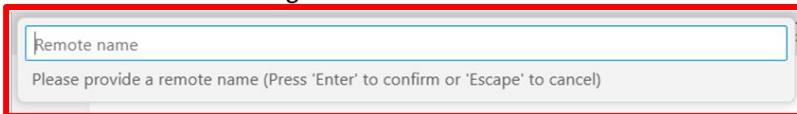


VS Code + Git

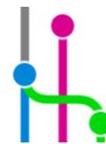
- git remote add : 在 GitHub 中建立資料庫 → 複製 url → 將資料庫命名



第一個通常命名為 origin



Git Graph 套件



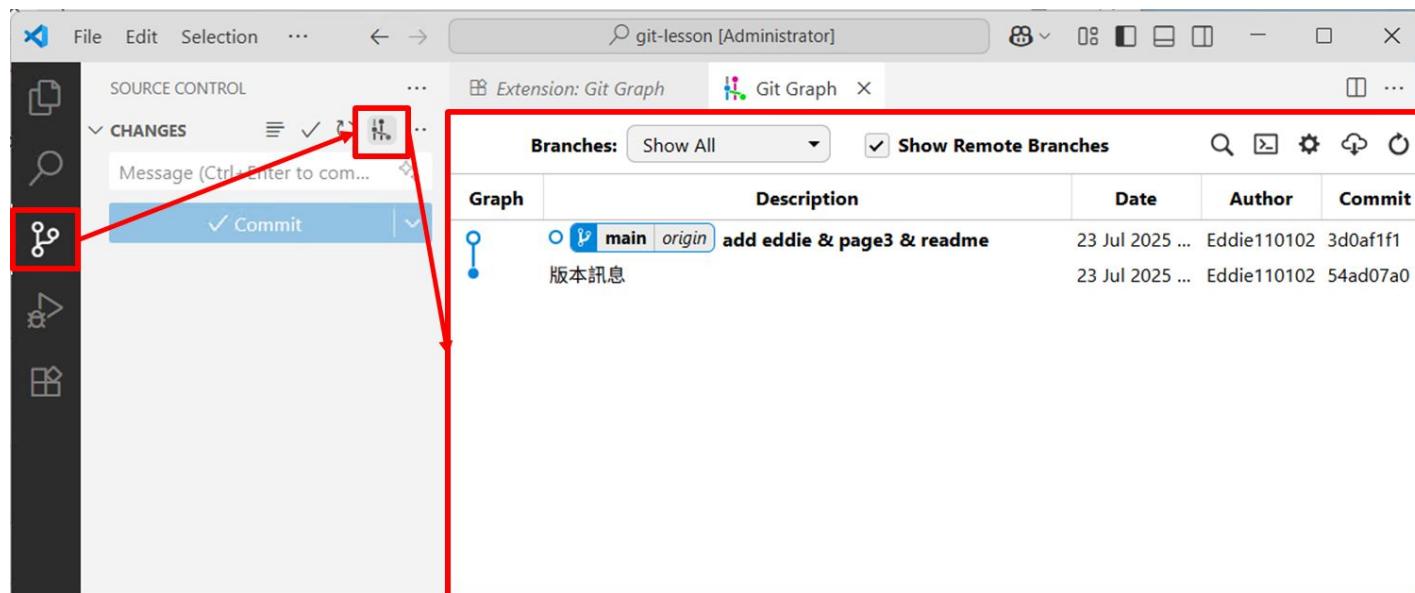
Git Graph

mhutchie | ⌂ 11,787,888 | ★★★★★ (649)

View a Git Graph of your repository, and perform Gi...

[Disable](#) [Uninstall](#) Auto Update

- Git Graph 可以圖形化檢視版本變化及演進。



Visual Studio

開啟 Git 視窗

看版本、操作分支、合併



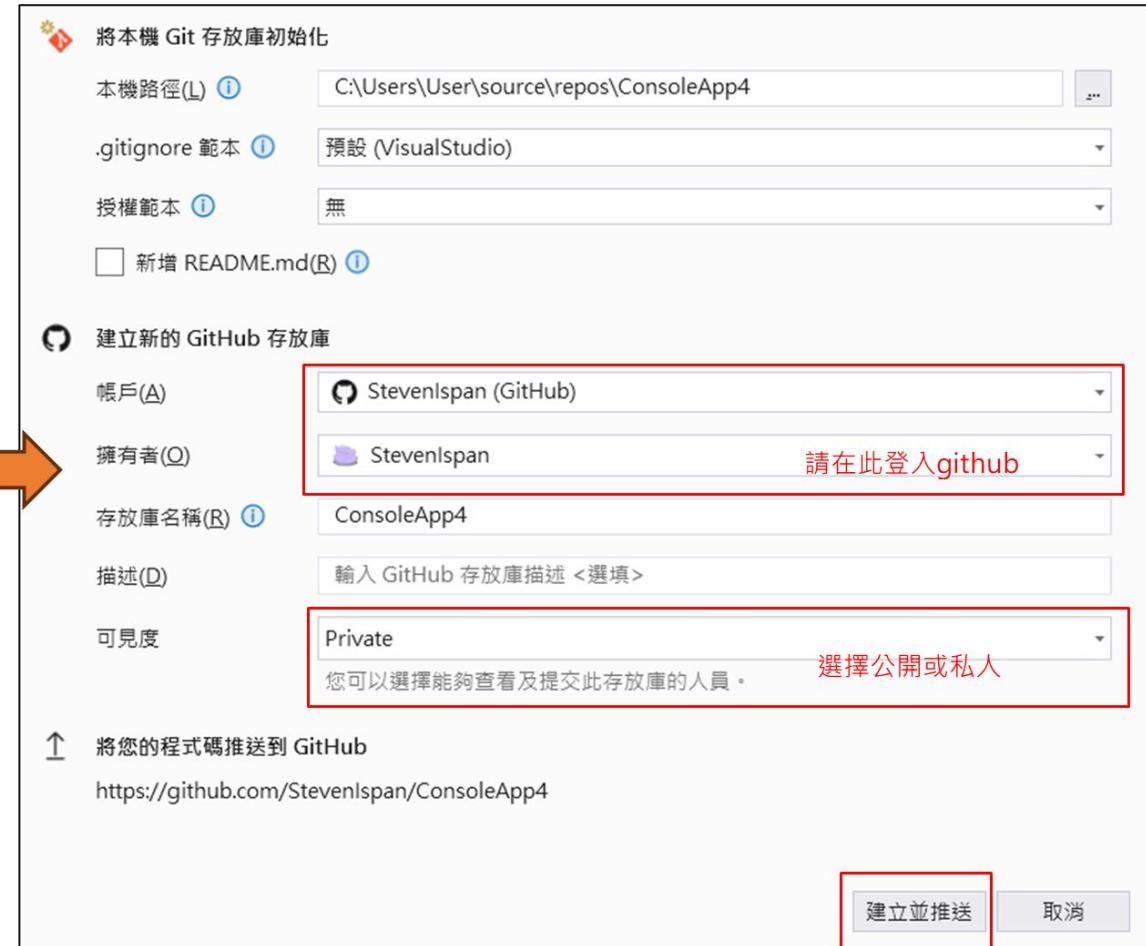
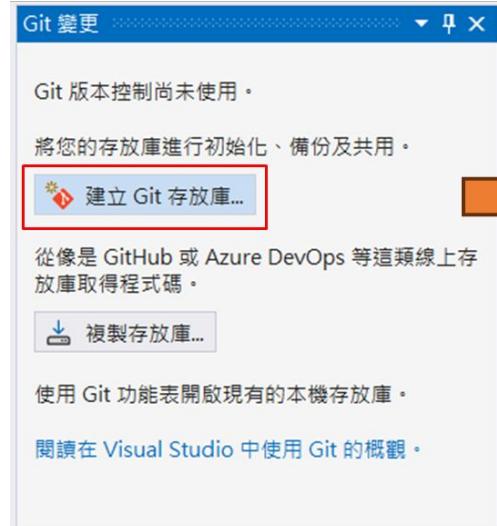
Program.cs ✘ X Git 存放庫 - C...eApp4 (main) ✘ X

ConsoleApp4 (main) < 篩選 篩選記錄

分支 / 標籤: main,

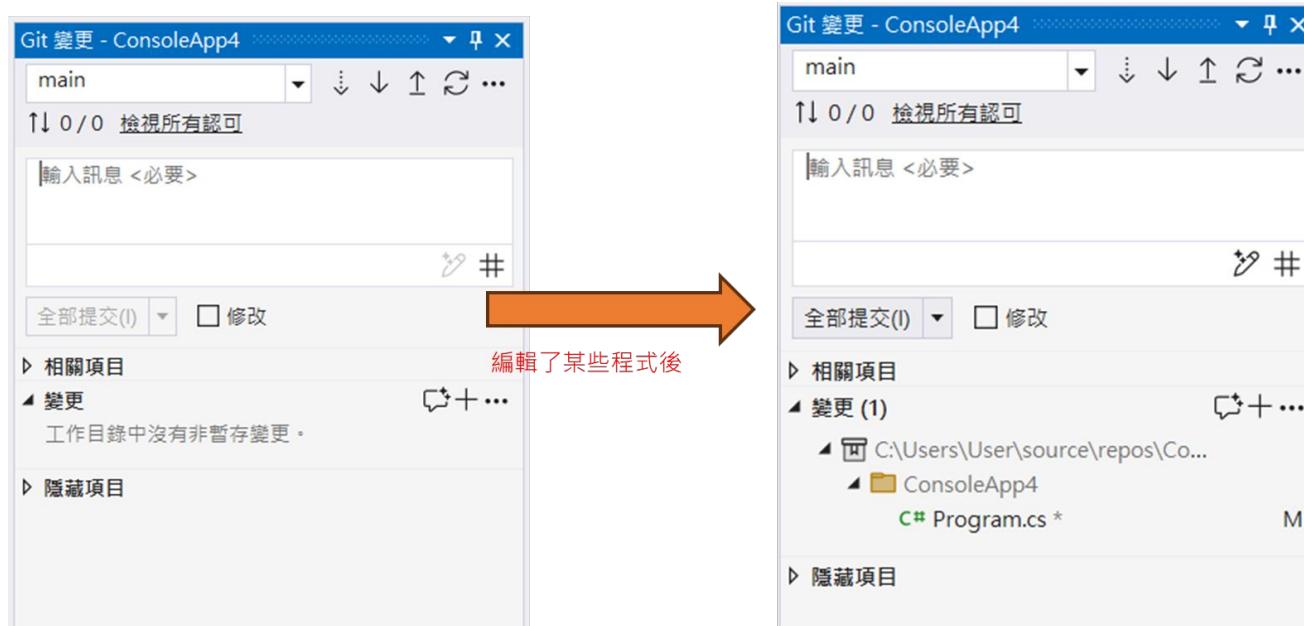
分支/標籤	圖形	訊息	作者	日期	ID
▶ 傳入的 (0) 摘取 提取					
▶ 傳出 (0) 推送 同步					
▶ 本機記錄					
main	加入專案檔案。 新增 .gitattributes 和 .gitignore 。		Steven	2025/5/16...	a68691b5
			Steven	2025/5/16...	a3add482

建立數據庫

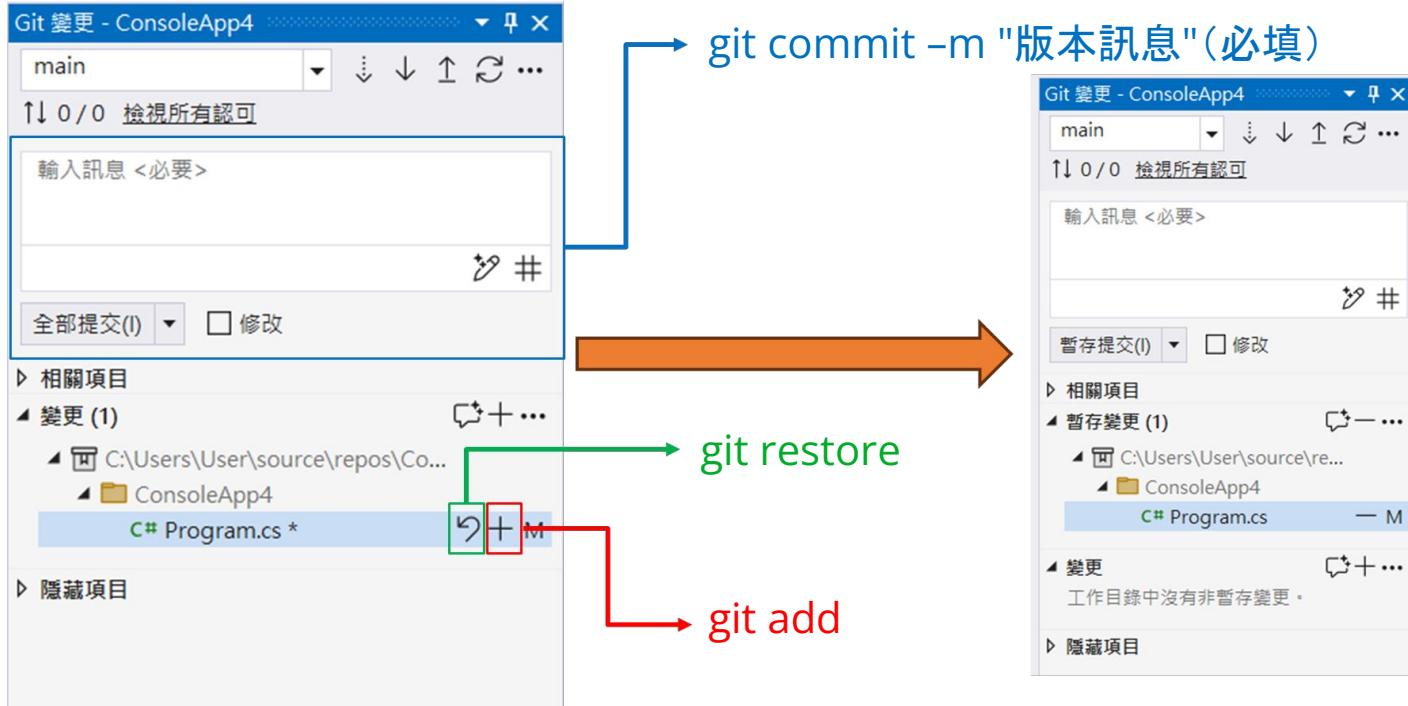


Git 變更視窗

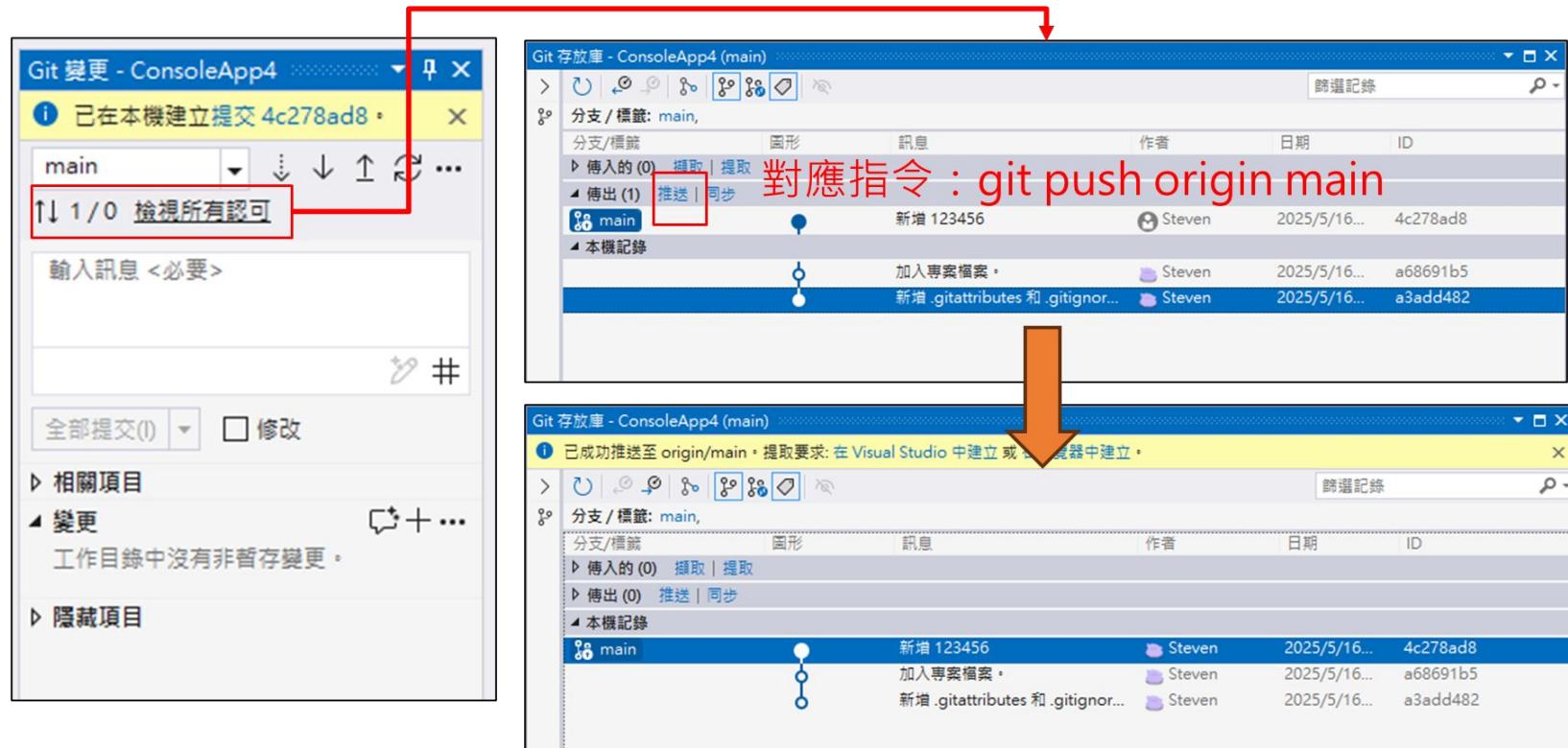
- 當檔案有變更時會自動執行 git status

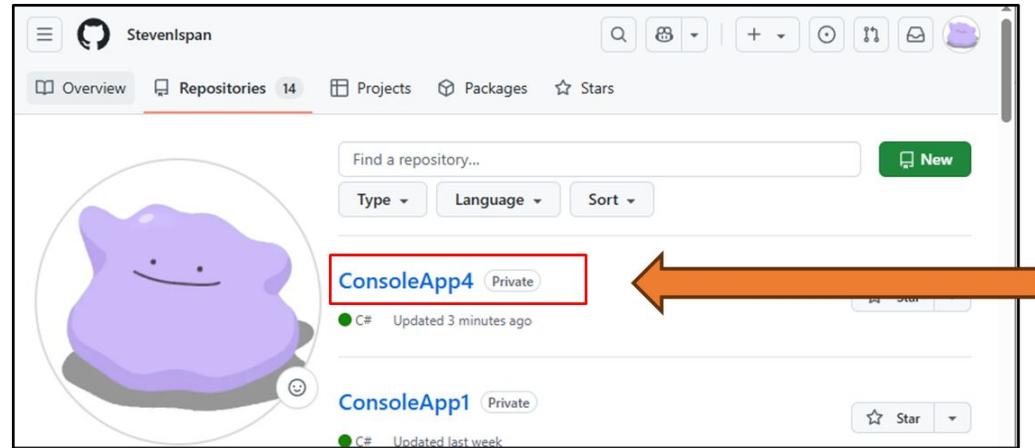


Git 變更視窗

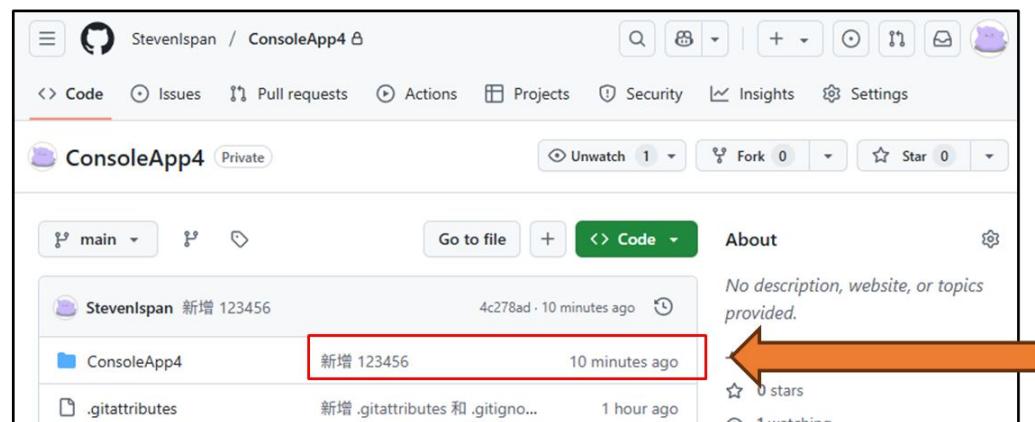


commit後推送檔案





登入後會發現
專案已上傳至GitHub



剛才建立的commit
也顯示於此

提取與推送

- 按鈕控制項由左至右，包括 Fetch、Pull、Push 和 Sync
- Pull 提取 (拉下/下載) - 在推送之前一律提取(Fetch)。當您第一次提取時，可以防止上游 合併衝突
- Push 推送 (推上/上傳) - 使用 push 將認可推送至 GitHub，您可以在其中將認可儲存為備份，或與其他人共用您的程式碼。



工作流程

1. 先提取 Pull 下載:看看團隊是否有新的變更要提取
2. 確認變更:commit 認可到 Local Repository 儲存區
3. Push 推送上去 GitHub



解決合併衝突

- 在 [合併編輯器] 中，使用下列任何方法來開始解決衝突，(如編號螢幕擷取畫面) 所示：
 1. 依序移至您的衝突行，然後選取核取方塊來選擇保留右邊或左邊的位置。
 2. 保留或忽略所有衝突的變更。
 3. 在 [結果] 視窗中手動編輯您的程式碼。

解決合併衝突

The screenshot shows a Visual Studio interface with two code editors and a Git Changes tool window.

Code Editors:

- Left Editor:** Shows code from a temporary merge branch. Lines 19 and 20 are identical (">//00000000 //aaaaaaaaaaaaaaa). Line 21 is marked as a conflict (indicated by a red background). Line 22 starts with "://>>>>> Temporary merge branch 2". Line 24 contains the code "public void Test()".
- Right Editor:** Shows the same code after merging. Lines 19 and 20 are identical. Line 21 is marked as a conflict (indicated by a red background). Line 22 starts with "://>>>>> Temporary merge branch 2". Line 24 contains the code "public void Test()".

Git Changes Tool (Right Side):

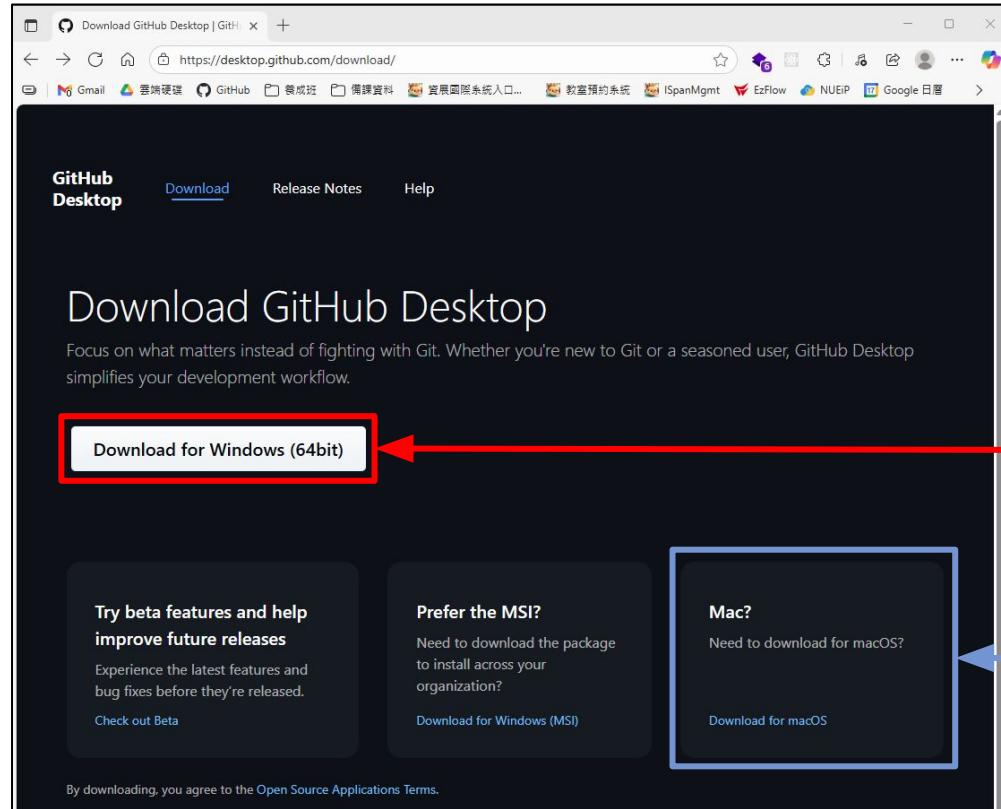
- Top Status:** Displays a warning: "提取完成，但 'WpfApp1' 存放庫中發生衝突。請解決衝突，並認可結果。"
- Branch:** master
- Conflict Resolution Buttons:** 全部認可(I) and 中止
- Unmerged Changes (1):** A list containing "Class1.cs [兩者都已...]" with a red circle around it.
- Changes (1):** A list containing "Git_2022.pptx" with a red circle around it.

Output Window (Bottom):

```
顯示輸出來源(S): 原始檔控制 - Git
在存放庫 D:\temp\WpfApp1 中於本機建立了認可 9c8cd9d1
正在推送 master
To https://github.com/0711ffWang/WpfApp1.git
Error: failed to push some refs to 'https://github.com/0711ffWang/WpfApp1.git'
Error: hint: Updates were rejected because the remote contains work that you do
```

GitHub Desktop

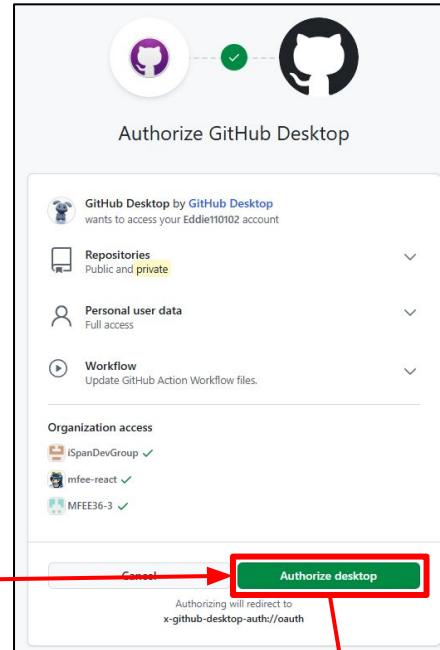
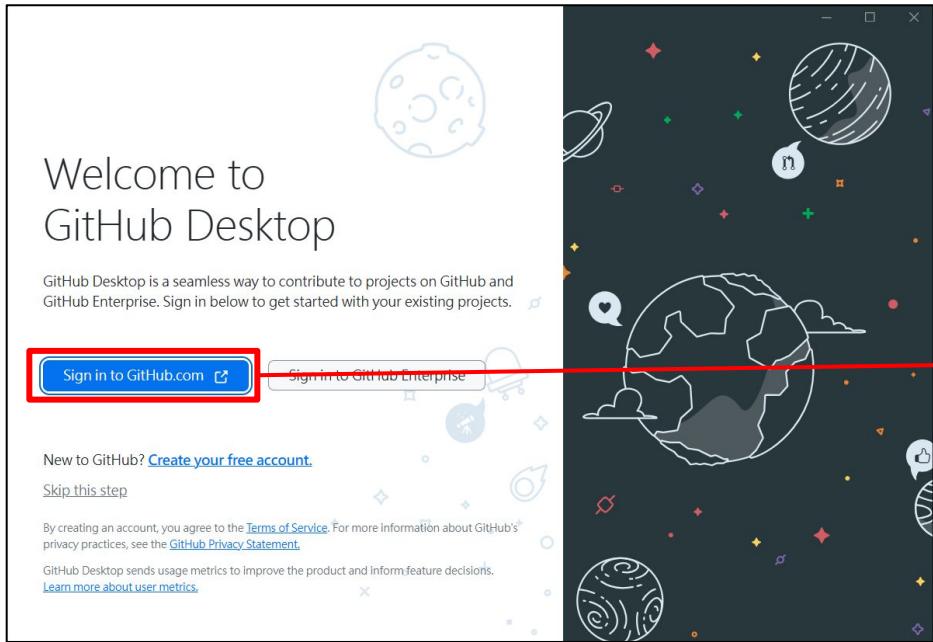
下載GitHub Desktop



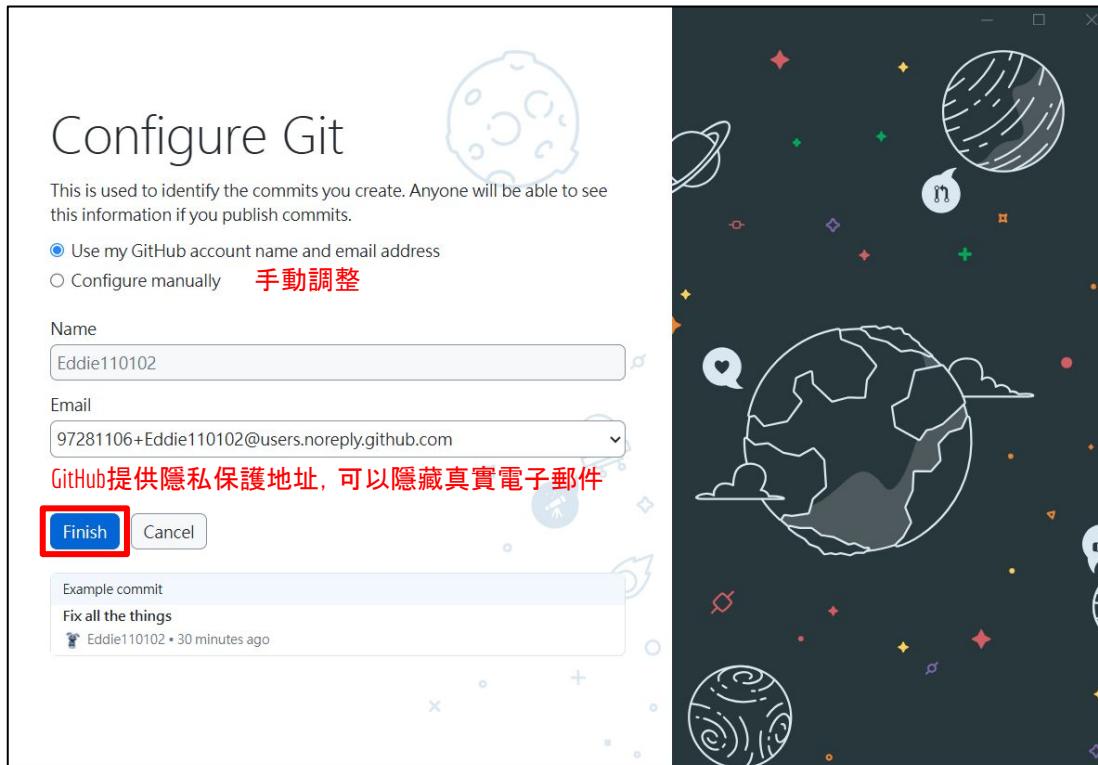
Windows 64位元下載連結

macOS下載連結

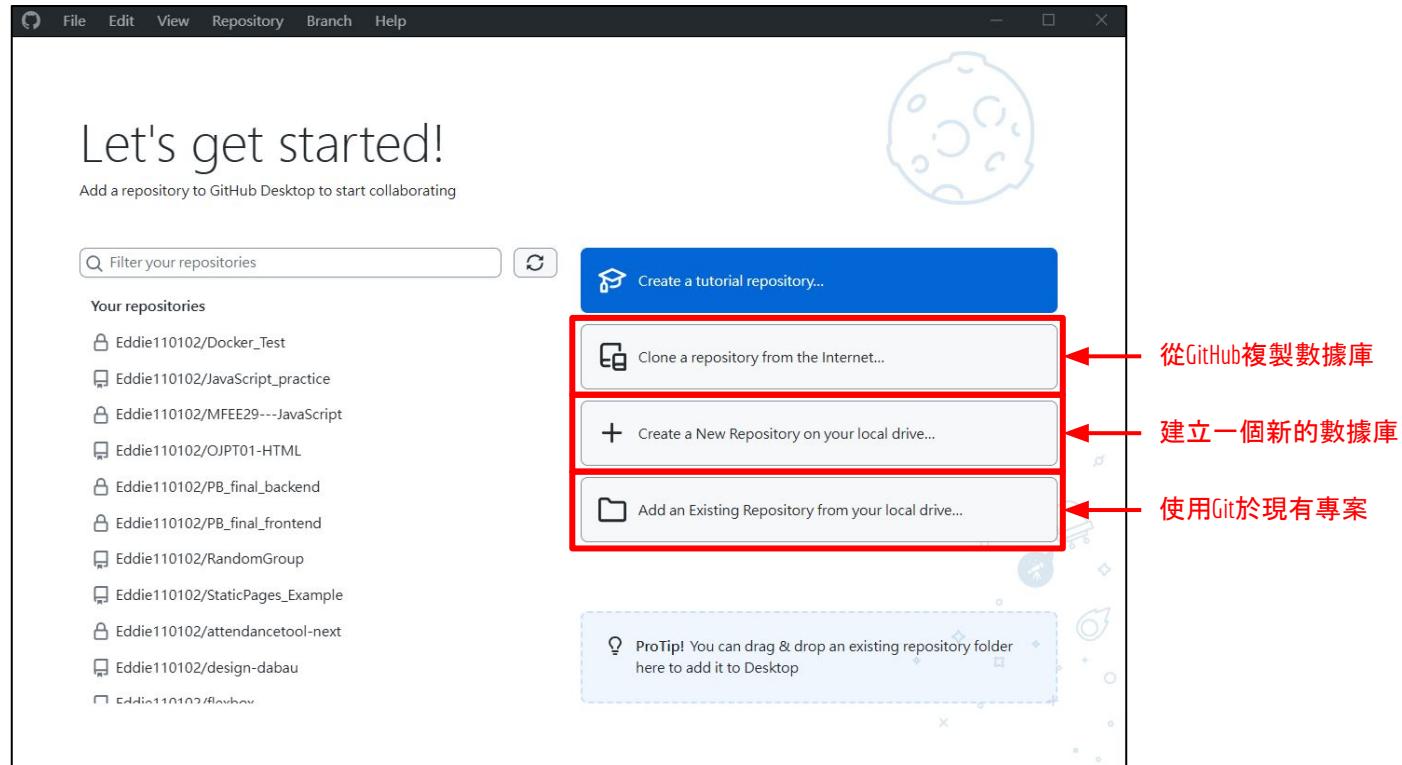
登入及授權



設定使用者名稱與電子郵件



完成頁面

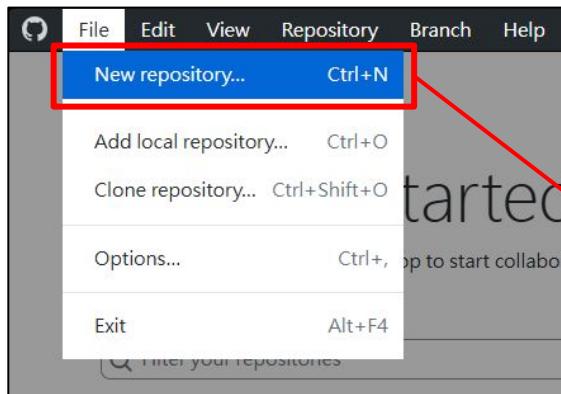


建立本地端專案 發布到遠端 repo

完全沒有資料，從零開始。

建立新的本地工作目錄

- 點擊 File -> New repository

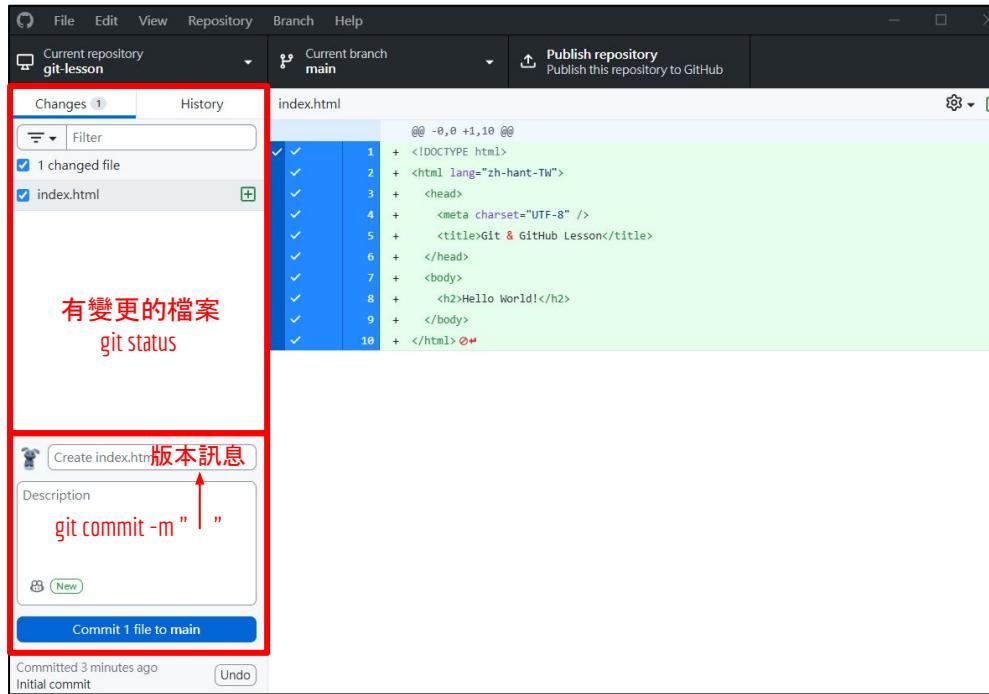


Create a new repository

Name	repository name	專案名稱
Description	專案描述	
Local path	C:\Users\User\Desktop	專案路徑
<input type="checkbox"/> Initialize this repository with a README 建立 README 檔		
Git ignore	None 建立 .gitignore 檔	
License	None 設定 授權	
Create repository		Cancel

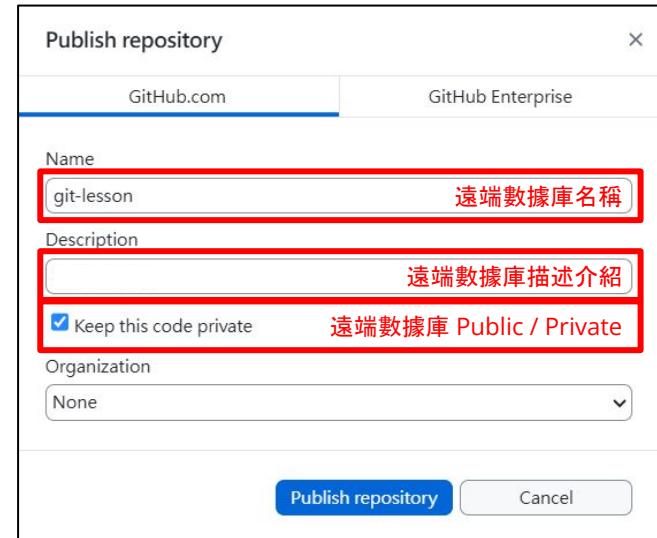
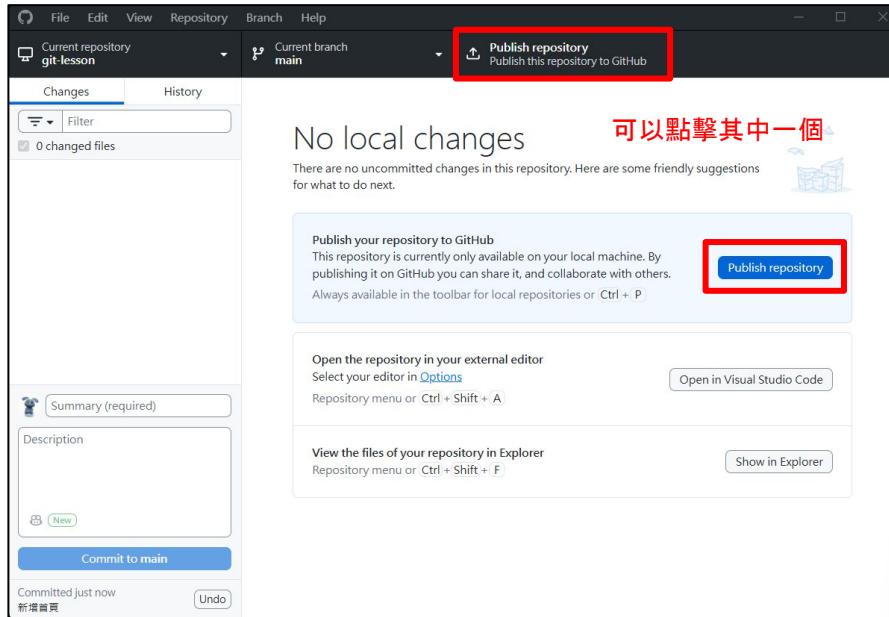
編輯工作目錄中的檔案

- 新增 / 刪除 / 編輯檔案後，在Desktop中即顯示該檔案，即可填寫message並commit。



同步到遠端數據庫(GitHub)

- 點擊 Publish repository -> 發布到遠端 repo -> 填寫遠端repo名稱
- Keep this code private : 將repo設為private

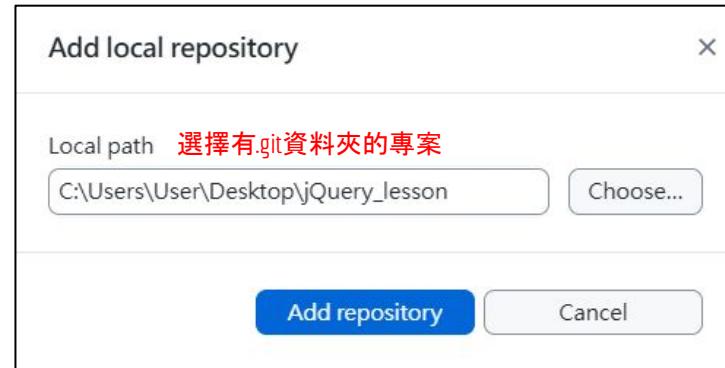
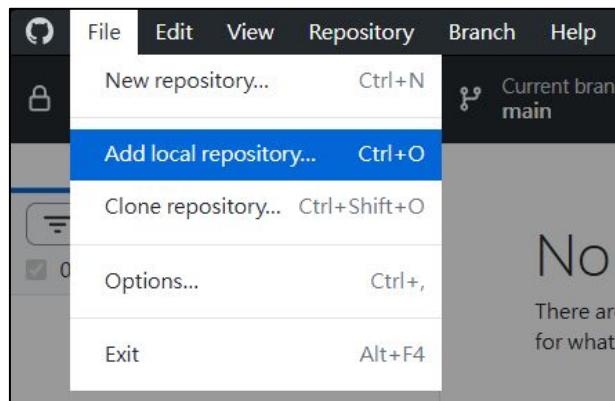


已有本地端專案 發佈到遠端Repo

電腦中有專案資料夾，已有本地端數據庫 / 未有本地端數據庫。

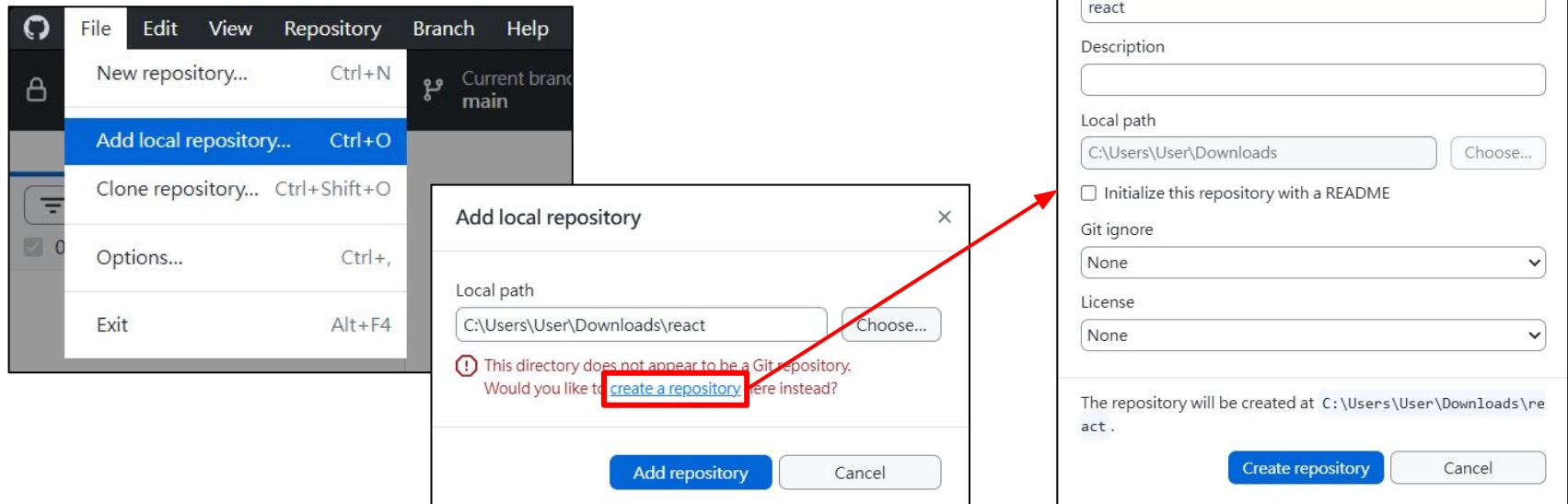
將本地端專案加入GitHub Desktop

- 專案中已使用Git版本控制(有.git)
- 點擊 File -> Add local repository -> publish repository



將本地端專案加入GitHub Desktop

- 專案中尚未開始使用Git版本控制
- 點擊 File -> Add local repository -> create a repository -> Create repository

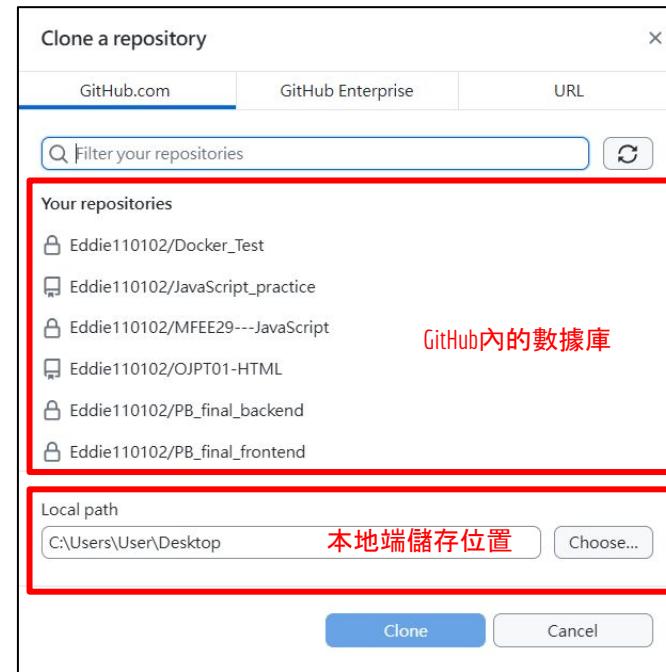
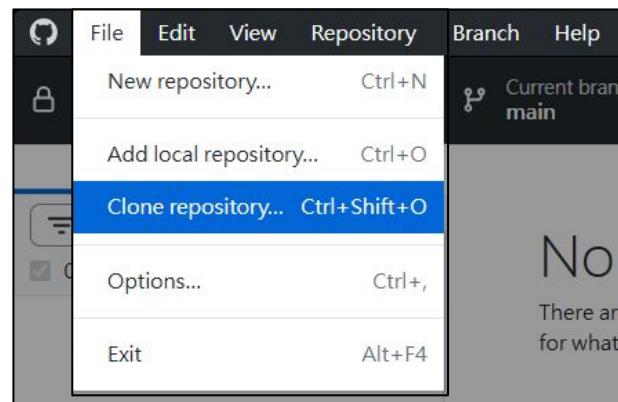


從遠端數據庫 複製一份專案到本地端

使用Clone指令。

從遠端複製一份專案到本地端

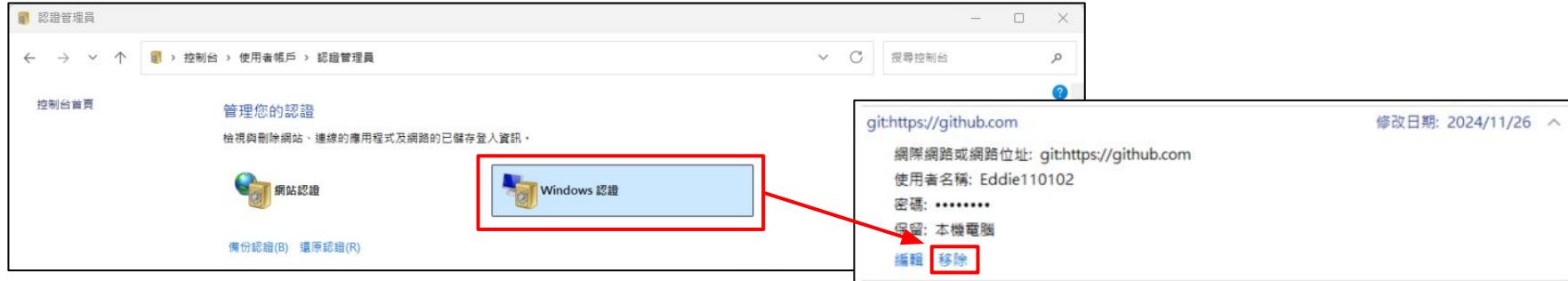
- 點擊 File -> Clone repository -> 選擇遠端資料庫 -> 選擇儲存路徑



附錄

Push時顯示錯誤找不到對應Repository

- 先確認 repo 的 URL 是否正確。
- 如果是共同協作的 repo, 是否有權限使用。
- 如果有多重帳號問題, 在本機的控制台 -> 使用者帳戶 -> 認證管理員 -> windows 認證 中尋找 github 相關帳號並移除。
- 重新 push 時會需要再驗證一次帳戶即可。



Git Bash 無法正常顯示中文字

- 當我們輸入 `git status` 的時候，如果檔名或路徑中有中文字，而且畫面顯示亂碼
- 在 Git bash 中直接輸入「`git config --global core.quotepath false`」。

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/jquery_lesson (master)
$ git status
on branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    "\346\226\260\345\242\236 \346\226\207\345\255\227\346\226\207
\344\273\266.txt"

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

```
User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/jquery_lesson (master)
$ git config --global core.quotepath false

User@DESKTOP-IIKKHIK MINGW64 ~/Desktop/jquery_lesson (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

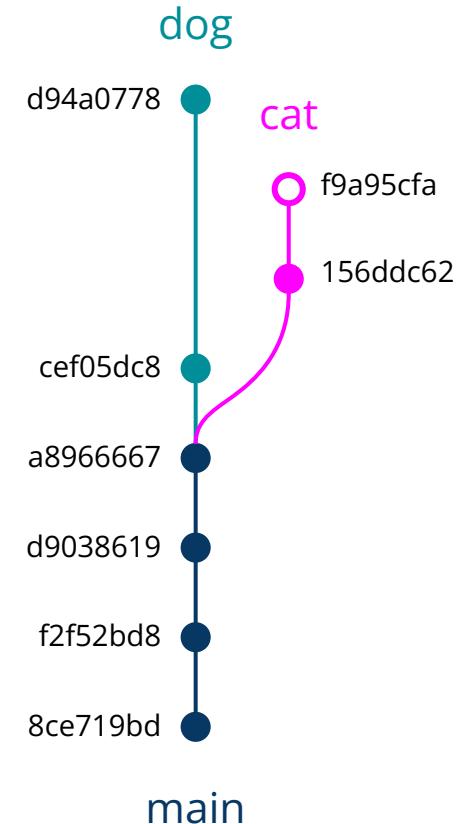
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    新增 文字文件.txt

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

合併分支 Rebase

```
git rebase <branch name>
```

- 合併分支的第二種方式 rebase 是直接改寫根基，並取得新的版本代碼。
- **非必要不要用在已經 push 過的 commit 上**，可能會影響到其他人。
- 範例：
 - 目前在 cat 分支，下指令 git rebase dog。



合併分支 Rebase

```
git rebase <branch name>
```

- 合併分支的第二種方式 rebase 是直接改寫根基，並取得新的版本代碼。
- **非必要不要用在已經 push 過的 commit 上**，可能會影響到其他人。
- 範例：
 - 目前在 cat 分支，下指令 git rebase dog。
 - 結果會是把 cat 分支的 commit 版本**複製一份**內容，接到 dog 分支生成的 commit 之後，取得新的 commit 代碼。

