GitHub & Git

助教：余嘉袖、蘇育琳、謝名泙

Outline

1 認識 Git / GitHub 與基本環境設定

2 Git 版本控制基礎指令與檔案管理實務

3 同步儲存庫

4 協作

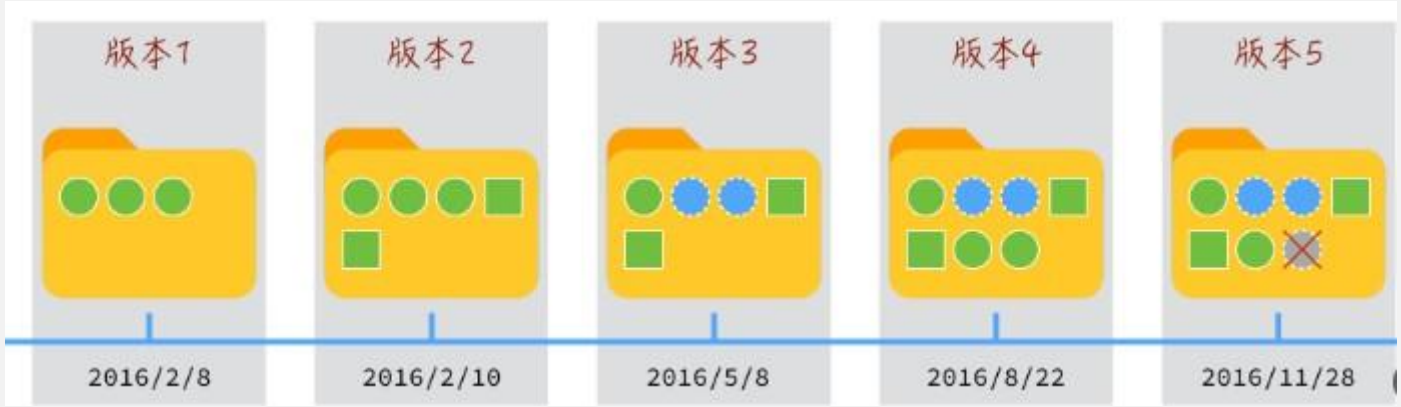
5 分支與合併

6 Git GUI (GitHub Desktop)

1-1 何謂Git

- Git 是一種分散式版本的版本控制系統
- e.g.:

假設你有一個 resume 的目錄(存放面試者的資料)



1-1 何謂Git

- **分散式系統**:

每台使用者的電腦都擁有完整的版本庫，因此即使沒有伺服器或沒有網路，也能正常做版本控制。

- **版本**:

這每一個「resume 目錄的狀態變化」，不管是新增或刪除檔案，亦或是修改檔案內容，都稱之為一個「版本」，例如上一張投影片圖例的版本 1 ~ 5。

- **版本控制系統**:

指會幫你記錄這些所有的狀態變化，並且可以像搭乘時光機一樣，隨時切換到過去某個「版本」時候的狀態。

1-2 Git處理檔案方式

- e.g.:

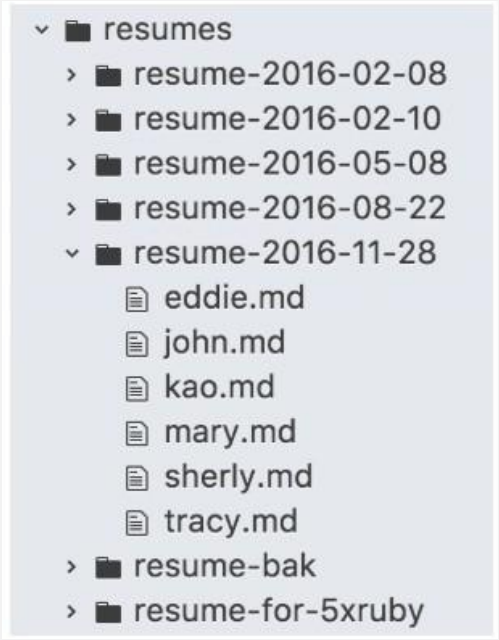
Git:

像拍照（snapshot）一樣，在每次版本變化的時候，Git 會更新並記錄整個目錄跟檔案的樹狀結構。

1-3 學習Git的理由

- 透過版本控制系統，可以清楚的記錄每個檔案是誰在什麼時候加進來、什麼時候被修改或刪除。Git 就是一種版本控制系統，也是目前業界最流行的版本控制系統
- e.g.:

整理或備份檔案範例



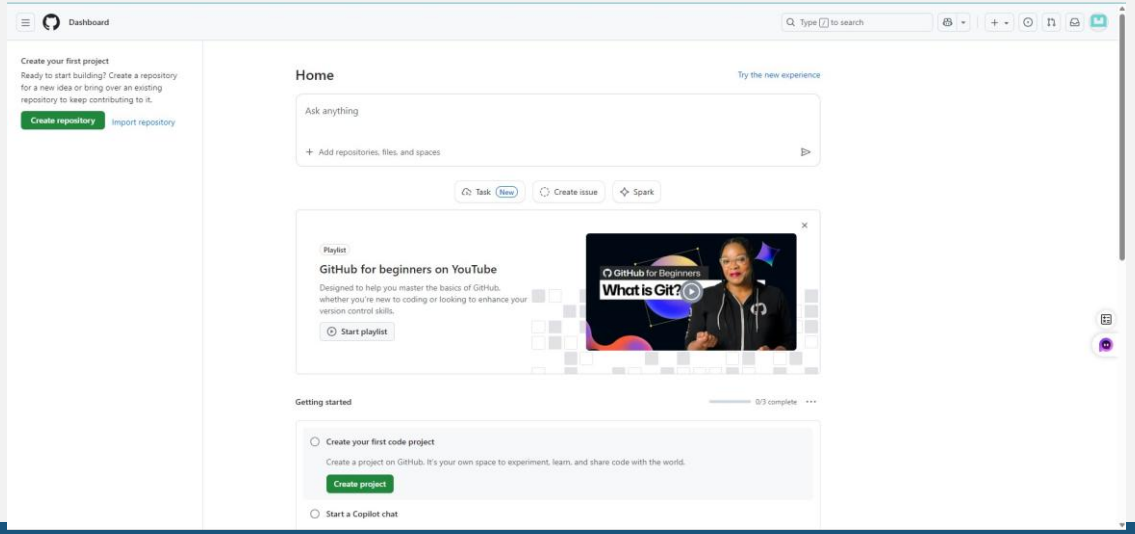
1-4 何謂GitHub&GitHub註冊

- GitHub 是一個基於 Git 的「線上程式碼託管平台」，用來存放、分享、管理程式碼。簡單說，就是 把你的 Git 專案放到雲端，方便協作、備份、與別人一起開發。
- **GitHub註冊**:

進入 GitHub 官方網站，點擊「Sign up」註冊帳號；已註冊過者，點擊「Sign in」登入帳號

- GitHub官方網站畫面：

(已註冊過的)



1-5 Git VS GitHub

項目	Git	GitHub
是什麼?	一套 **版本控制工具**（在你的電腦本地端運作）	一個 **雲端程式碼託管平台**（用來放 Git 專案）
主要用途	管理程式碼版本、歷史、分支、合併	上傳 Git 專案、分享、備份、協作、多人開發
是否需要網路?	不需要（離線也能完整運作）	需要網路才能存取倉庫
安裝位置	安裝在本機電腦	網站平台（github.com）

1-5 Git VS GitHub

項目	Git	GitHub
存放方式	本地端保存完整版本庫	雲端保存遠端版本庫
核心特色	分散式：每人電腦都有完整備份	集中協作：大家一起連同一個雲端倉庫
開發者支援功能	版本控制、快照、回溯、合併	Issue、Pull Request、Code Review、Wiki、CI/CD（GitHub Actions）

1-6 Git ＆ GitHub協作關係

Git × GitHub 協作流程:

I. 在本地（Local）使用 Git 進行開發

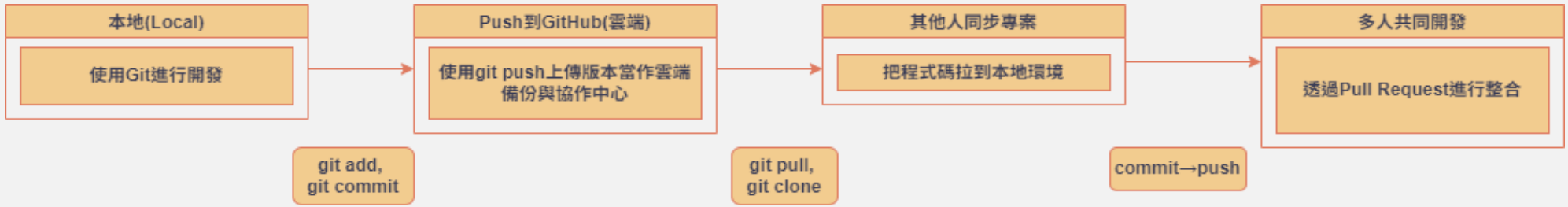
II. Push 到 GitHub（雲端)

III.其他人從 GitHub 同步專案

IV.多人共同開發

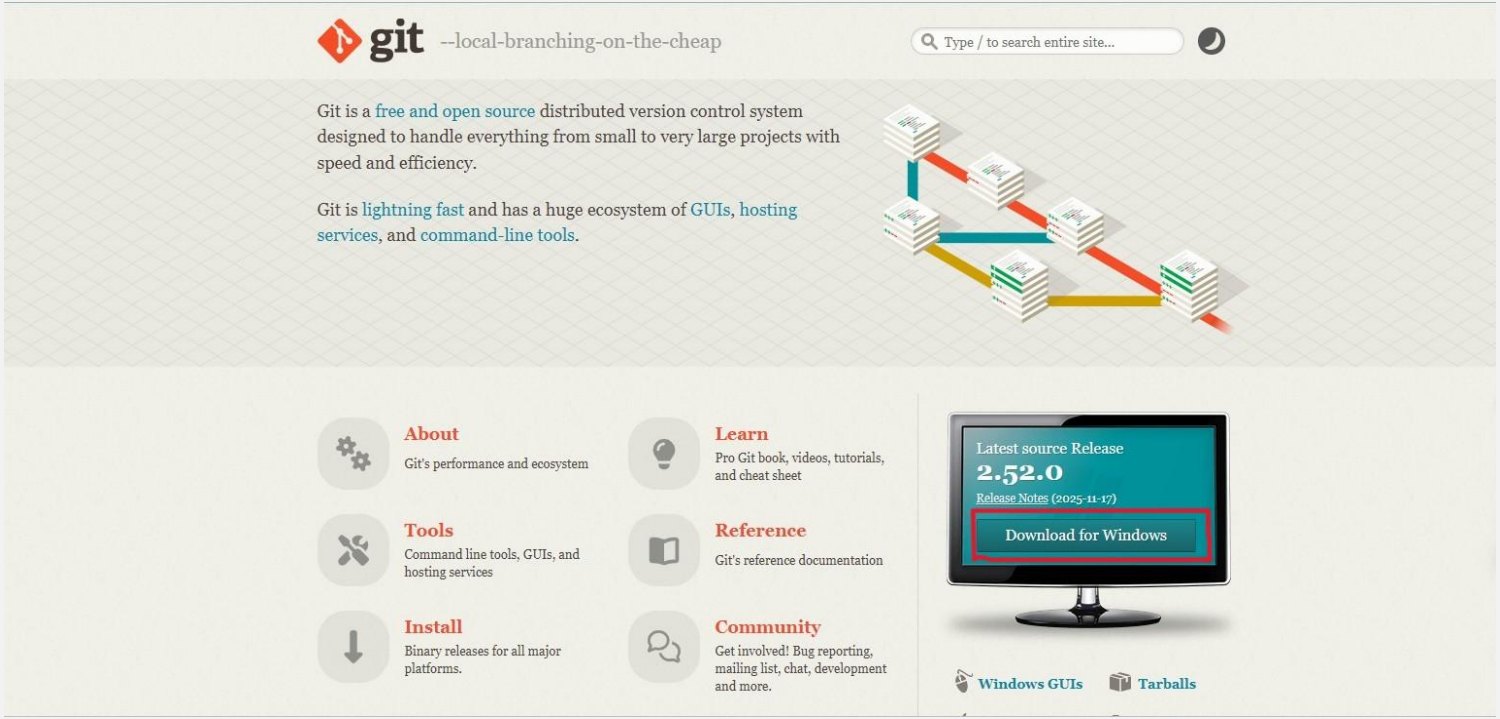
1-6 Git ＆ GitHub協作關係

Git × GitHub 協作流程圖：



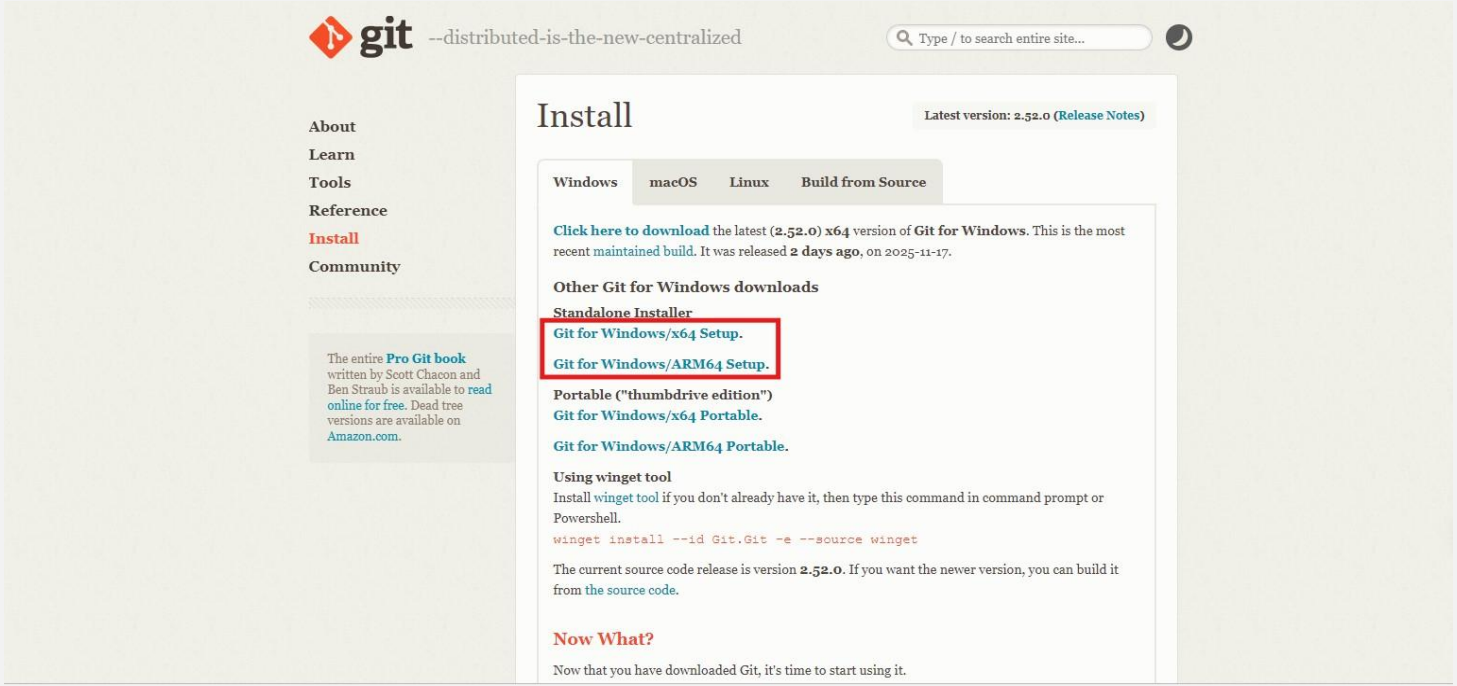
1-7 Git安裝-Windows

Step1.請到 Git官方網站 點擊 “Download for Windows”



1-7 Git安裝-Windows

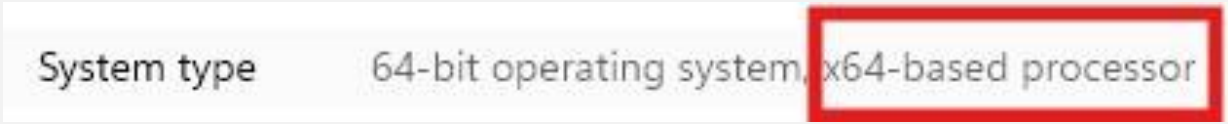
Step2.請在Install頁面的Windows區塊點擊“Git for Windows/x64 Setup”或“Git for Windows/ARM64 Setup”進行安裝



1-7 Git安裝-Windows

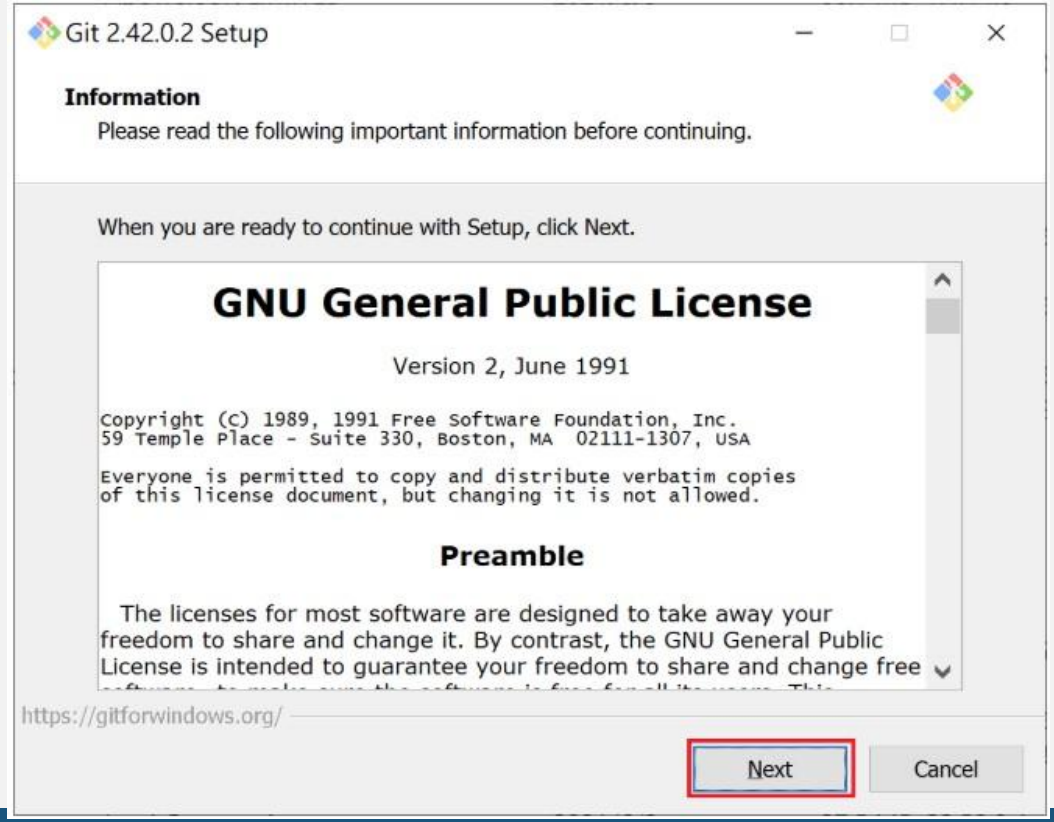
- “Git for Windows/x64 Setup”或“Git for Windows/ARM64 Setup”選擇:

先確認自己PC/筆電的CPU架構：設定 → 系統 → 關於 → 裝置規格 → 系統類型（System Type）(大多數是x64版)



1-7 Git安裝-Windows

Step3.安裝過程中原則上維持預設值，按“Next”即可

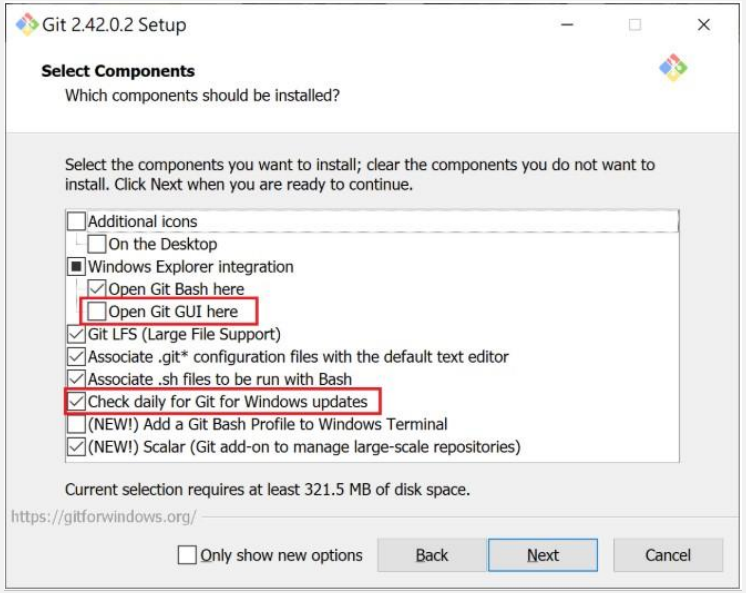


1-7 Git安裝-Windows

- 安裝過程中有幾個需要注意的部分：

(1)取消勾選 Git GUI 工具

(2)勾選 `check daily for git for windows updates`

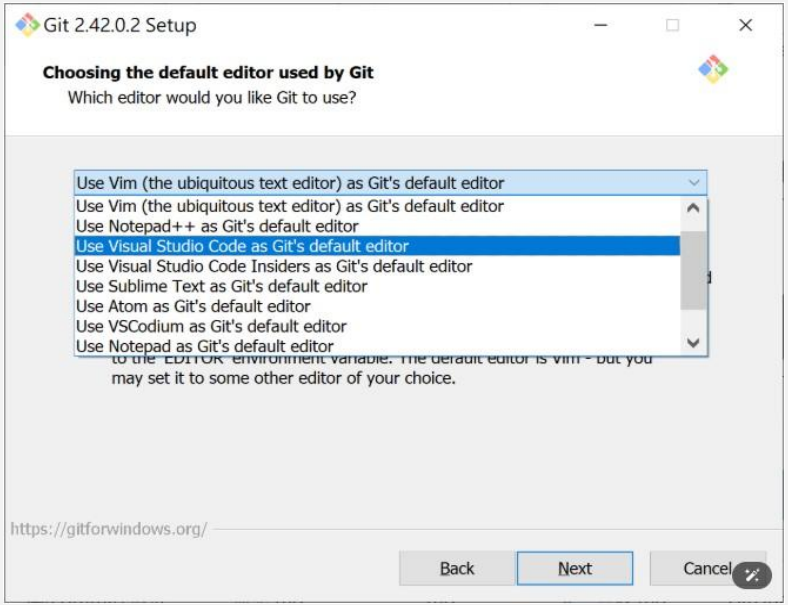


1-7 Git安裝-Windows

- 安裝過程中有幾個需要注意的部分：

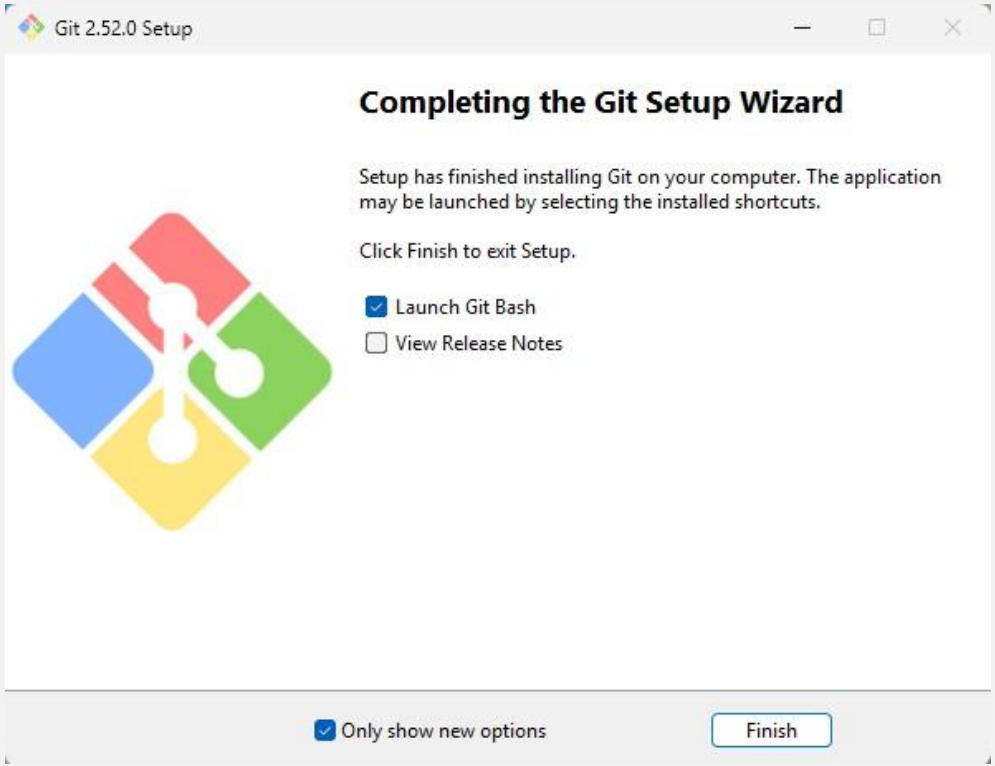
選擇 Git 預設的編輯器，Git 預設使用的編輯器是 Vim =Visual Improved

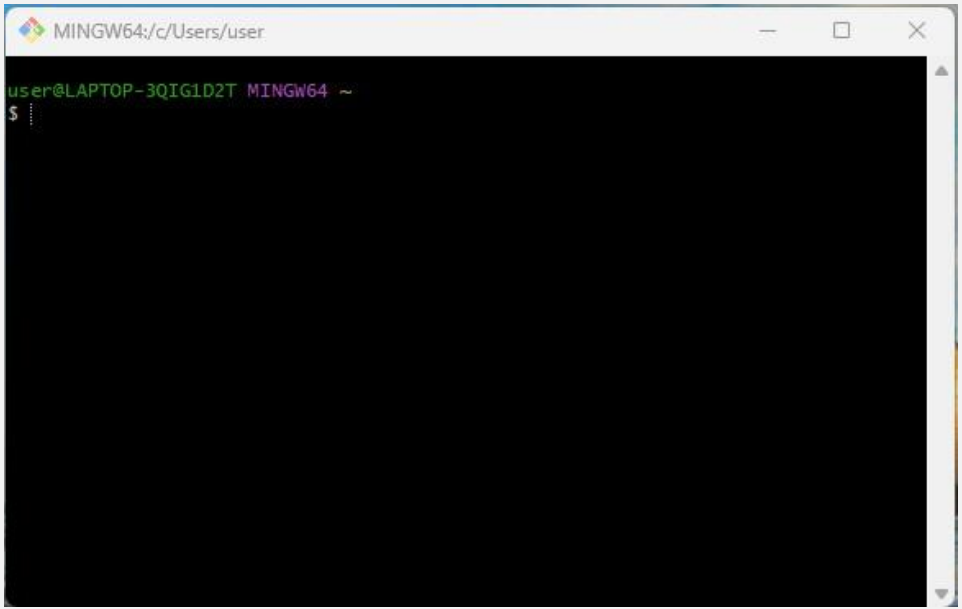
e.g.可更換為Use Visual Stuido Code as Git's default editor



1-7 Git安裝-Windows

Step4.安裝完成，啟動Git Bash(預設是Vim編輯器)





1-7 Git安裝-Windows

Step5.確認Git 是不是有安裝起來(在VSCode編輯器)

- 在 VS Code 裡新增終端:

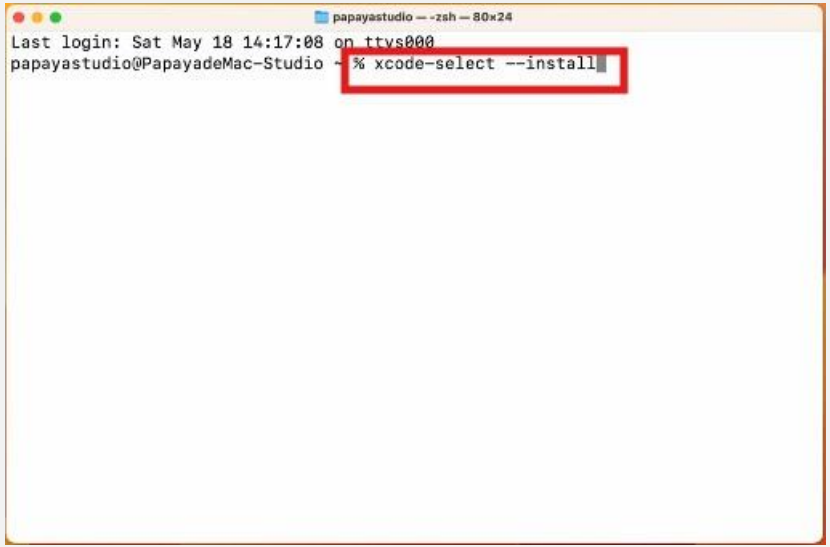
git --version

1-8 Git安裝-macOS

Step1.請按cmd+空白鍵→呼叫Spotlight→輸入終端機打開它

Step2.安裝Git：

Xcode-select –install

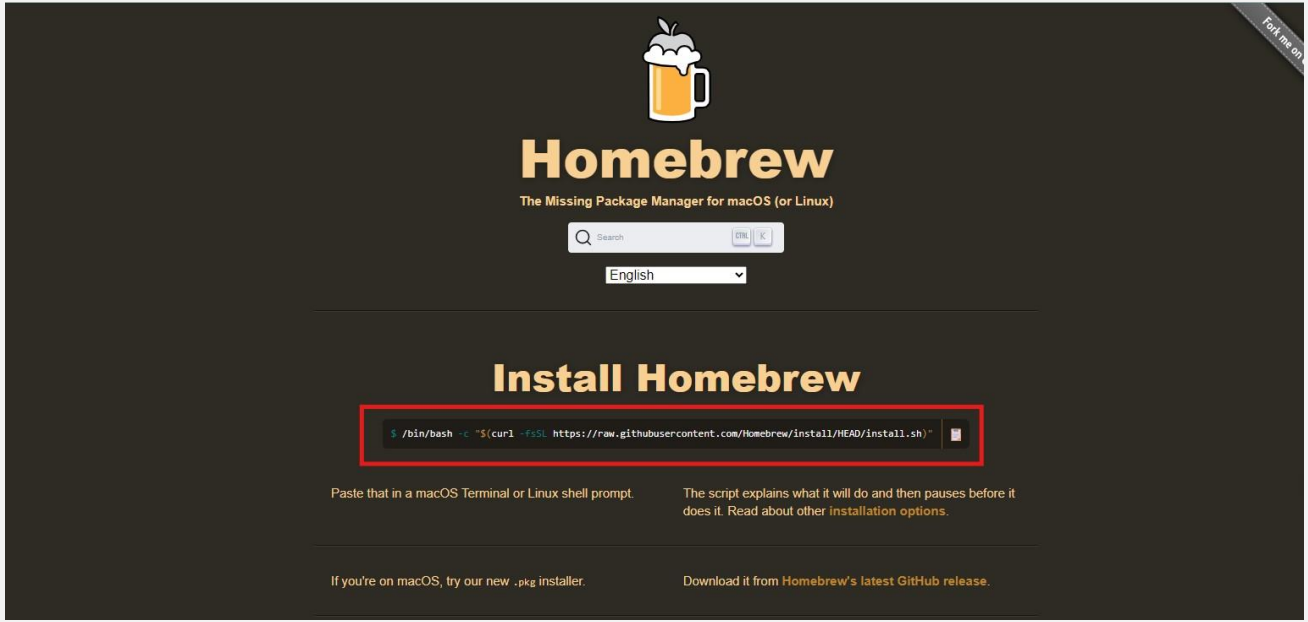


1-8 Git安裝-macOS

Step1.請到 Homebrew官方網站 複製Install Homebrew那一行:

/bin/bash -c "$(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"



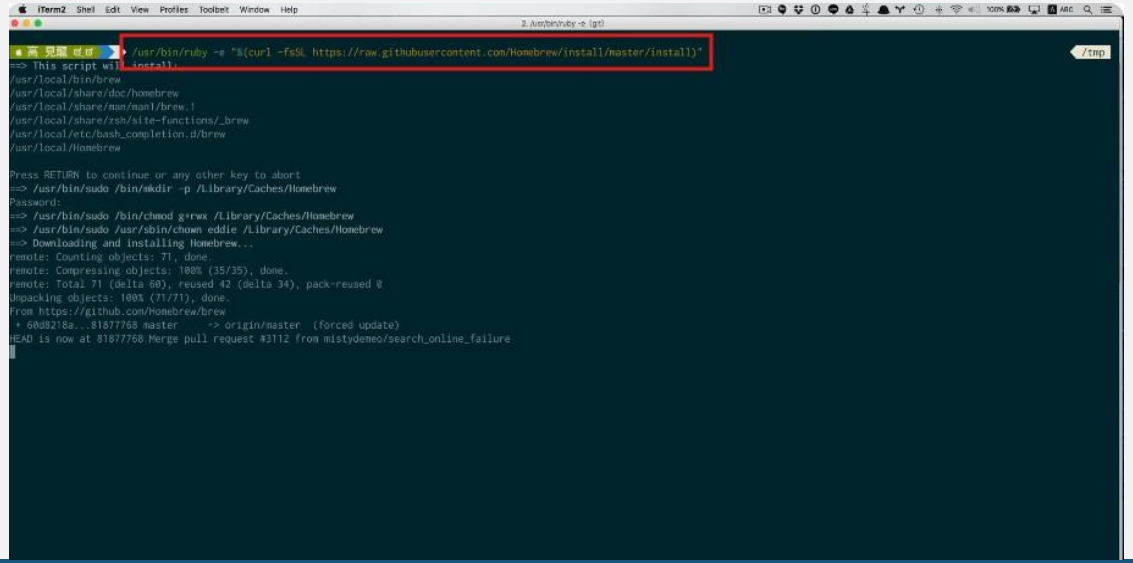
1-8 Git安裝-macOS

Step2.請在終端機視窗安裝Homebrew：

- 貼上剛剛在Homebrew官網上複製的那一行並執行：

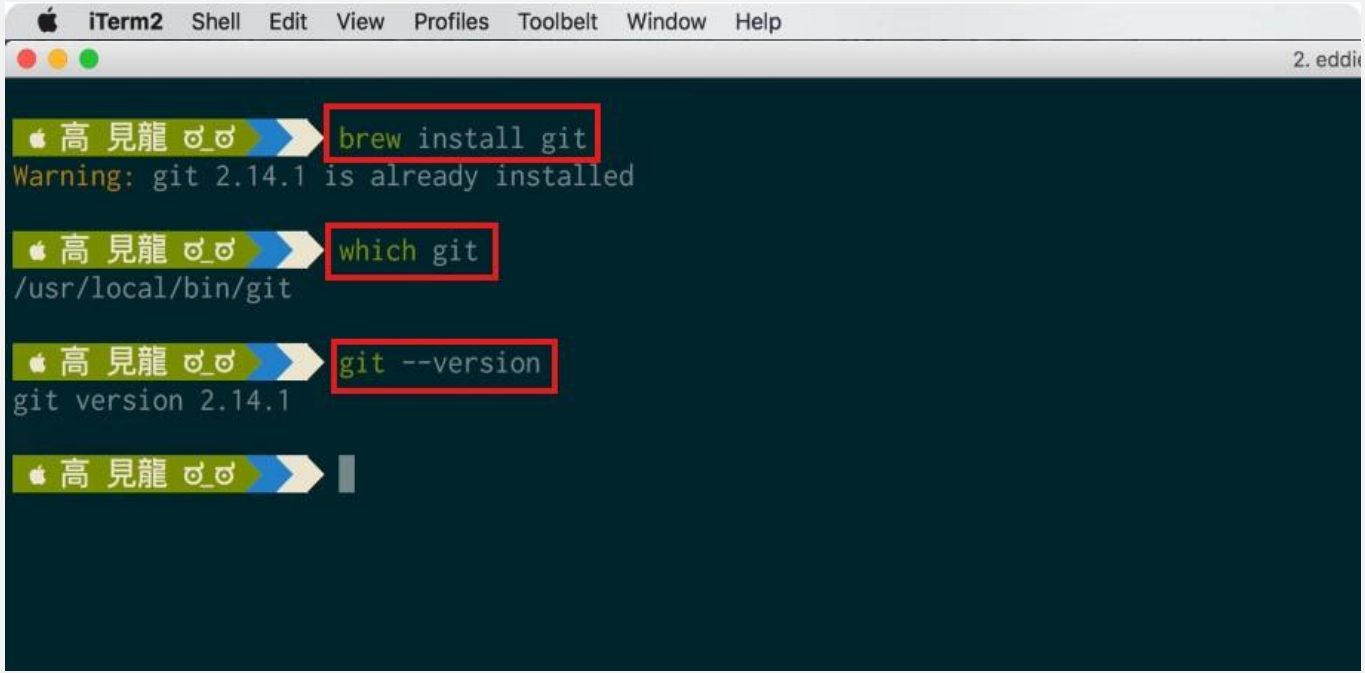
/bin/bash -c "$(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"



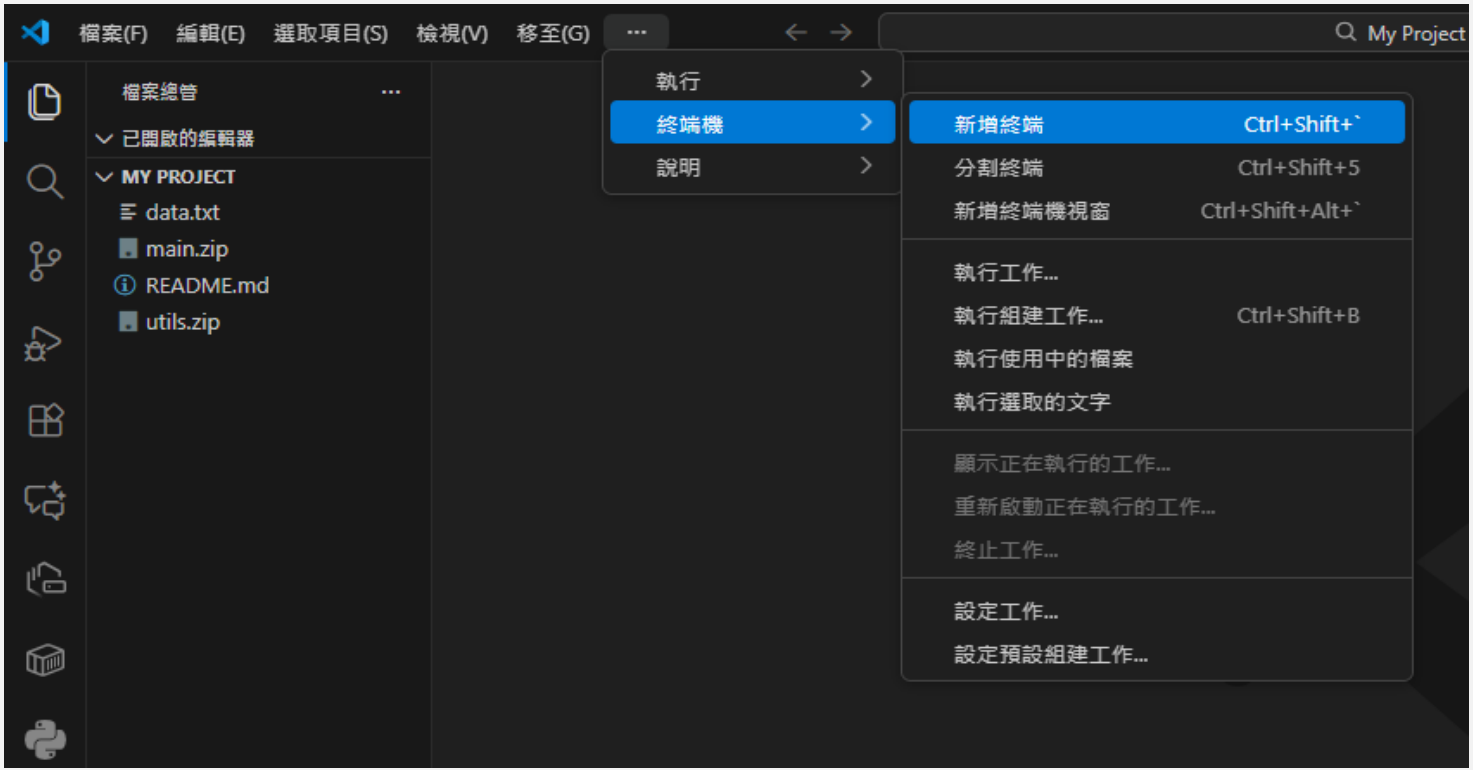
1-8 Git安裝-macOS

Step3.請在終端機視窗呼叫Homebrew 安裝 Git，並確認Git 是否已安裝起來，以及核對版本資訊



1-9 Git基本設定與初始化

Step1.在VSCode新增終端



1-9 Git基本設定與初始化

Step2.在VSCode終端機設定UserName&Email:

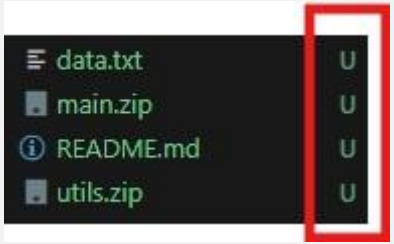
```
git config --global user.name "xxxxxx"
git config --global user.email "xxxxxx"
```

Step3.在VSCode終端機初始化檔案狀態:

```
git init
```

U:Untracked(未追蹤)

1-9 Git基本設定與初始化




```
PS D:\網路資料庫程式設計(大學部助教課程)碩一上\第十四週Git&GitHub教材(帶大三學弟妹上課)\My Project> git config --global user.na
me "[redacted]"
PS D:\網路資料庫程式設計(大學部助教課程)碩一上\第十四週Git&GitHub教材(帶大三學弟妹上課)\My Project> git config --global user.em
ail "[redacted]"
PS D:\網路資料庫程式設計(大學部助教課程)碩一上\第十四週Git&GitHub教材(帶大三學弟妹上課)\My Project> git init
Reinitialized existing Git repository in D:/網路資料庫程式設計(大學部助教課程)碩一上/第十四週Git&GitHub教材(帶大三學弟妹上課)/M
y Project/.git/
```

2-1 Git 基本觀念：Git 如何看你的檔案

（1）Git 的三個工作區域

區域	說明
Working Directory（工作區）	你正在編輯的檔案實體
Staging Area（暫存區）	你「選擇」要放進下一次 commit 的變化
Repository（版本庫）	你真正提交後永久保存的版本紀錄

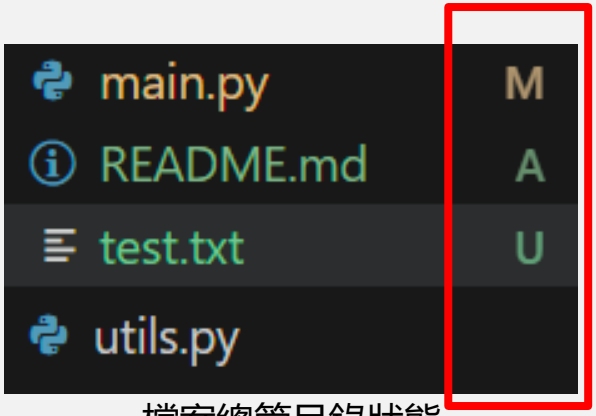
（2）Git 追蹤的是「檔案變化」，不是檔案本身

- 包括新增、修改、刪除
- 刪除檔案也會變成一種「變化」，需要 commit 記錄

2-2 Git 常用指令與操作流程

（1）查看檔案狀態指令：git status

狀態	意義
Untracked（未追蹤）	新增檔案，Git 尚未開始追蹤
Modified（已修改）	修改後尚未加入暫存
Staged（已暫存）	已準備加入下一次 commit
Committed（已提交）	已寫入版本歷史



檔案總管目錄狀態

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        data.txt
        main.py
        utils.py
```

未追蹤狀態

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
        new file:   data.txt
        new file:   main.py
        new file:   utils.py
```

已暫存狀態

VS Code 終端機介面狀態

2-2 Git 常用指令與操作流程

(2) 暫存變化指令:

指令類型	說明
git add 檔案名稱	
git add .	(新增/修改全部加入暫存)
git add *.py	(萬用字元+檔案類型)

重點:

- 只有「加入 Staging」（暫存）的變化才會進入 commit
- 若你修改檔案，需要重新 add 才會記錄新的變化

```
Untracked files:
  (use "git add <file>
        README.md
        data.txt
        main.py
        utils.py
```

git add .

```
Changes to be committed:
  (use "git rm --cached <file>
        new file:   README.md
        new file:   data.txt
        new file:   main.py
        new file:   utils.py
```

VS Code 終端機介面狀態

2-2 Git 常用指令與操作流程

（3）提交版本指令：git commit -m "提交訊息"

重點：

- commit = 把暫存區的變化拍照寫進版本庫
- 提交訊息建議清楚描述更動內容（如「修正登入錯誤」）

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git commit -m "git教學範例"
[main (root-commit) 3eccb05] git教學範例
 4 files changed, 22 insertions(+)
 create mode 100644 README.md
 create mode 100644 data.txt
 create mode 100644 main.py
 create mode 100644 utils.py
PS C:\Users\ASUS\OneDrive\桌面\My Project> git status
On branch main
nothing to commit, working tree clean
```

終端機介面狀態

2-2 Git 常用指令與操作流程

(4) 查看歷史紀錄指令：git log / git log --oneline

重點：●每一筆 commit 都有一個唯一的 commit ID(黃色雜湊數)

●--oneline 版本簡潔，方便快速找到版本

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git log
commit 6854d4083b485cb61c2883e08dc15e6ea2f65871 (HEAD -> main)
Author: joyce <sia0123789@gmail.com>
Date:   Sat Nov 15 14:08:11 2025 +0800

    修改utils中的add函式、新增sub函式、修改data

commit 3f942f20a824d2f769df1ade9176cfed6e50338e
Author: joyce <sia0123789@gmail.com>
Date:   Sat Nov 15 06:31:33 2025 +0800

    main更改成hello world

commit 3eccb052c24276048c07e6742b4981630ca4fc99
Author: joyce <sia0123789@gmail.com>
Date:   Sat Nov 15 06:17:30 2025 +0800

    git教學範例
```

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git log --oneline
6854d40 (HEAD -> main) 修改utils中的add函式、新增sub函式、修改data
3f942f2 main更改成hello world
3eccb05 git教學範例
```

VS Code 終端機介面狀態

2-2 Git 常用指令與操作流程

(5) 比較新舊版本差異指令: git diff/ git diff <commitID> --檔案名稱

- 重點: ●綠色:新增 ●紅色:刪除 ●用來找 bug、確認變更、了解差異

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git diff
diff --git a/data.txt b/data.txt
index 4a8aef3..0306029 100644
--- a/data.txt
+++ b/data.txt
@@ -1,3 +1,4 @@
 sample data
-12345
+1234
+5678

diff --git a/utils.py b/utils.py
index 4693ad3..411cbcb 100644
--- a/utils.py
+++ b/utils.py
@@ -1,2 +1,4 @@
-def add(a, b):
+def add(a, b, c):
+    return a + b +c
+def sub(a, b):
     return a + b
```

看全部尚未暫存的修改

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git diff 3f942f2 -- utils.py
diff --git a/utils.py b/utils.py
index 4693ad3..6388db6 100644
--- a/utils.py
+++ b/utils.py
@@ -1,2 +1,6 @@
-def add(a, b):
+def add(a, b, c):
+    return a + b +c
+def sub(a, b):
     return a + b
+def mul(a, b):
+    return a * b
```

單個檔案-舊版本 vs 新版本

VS Code 終端機介面狀態

2-2 Git 常用指令與操作流程

(6) 還原檔案到過去版本指令：git checkout <commitID> -- 檔案名稱

重點：●把某個檔案還原到某個舊版本的狀態

●之後需要再 commit 才會正式記錄此還原

```
PS C:\Users\ASUS\OneDrive\桌面\My Project> git checkout 3f942f2 -- utils.py
PS C:\Users\ASUS\OneDrive\桌面\My Project> git commit -m "將utils改回最初狀態"
[main aee6be3] 將utils改回最初狀態
 1 file changed, 1 insertion(+), 5 deletions(-)
PS C:\Users\ASUS\OneDrive\桌面\My Project> git log --oneline
aee6be3 (HEAD -> main) 將utils改回最初狀態
5cda0ac 在utils中新增乘法函式
6854d40 修改utils中的add函式、新增sub函式、修改data
3f942f2 main更改成hello world
3eccb05 git教學範例
```

```
diff --git a/utils.py b/utils.py
index 6388db6..4693ad3 100644
--- a/utils.py
+++ b/utils.py
@@ -1,6 +1,2 @@
-def add(a, b, c):
-    return a + b +c
-def sub(a, b):
+def add(a, b):
     return a + b
-def mul(a, b):
-    return a * b
```

回到先前狀態

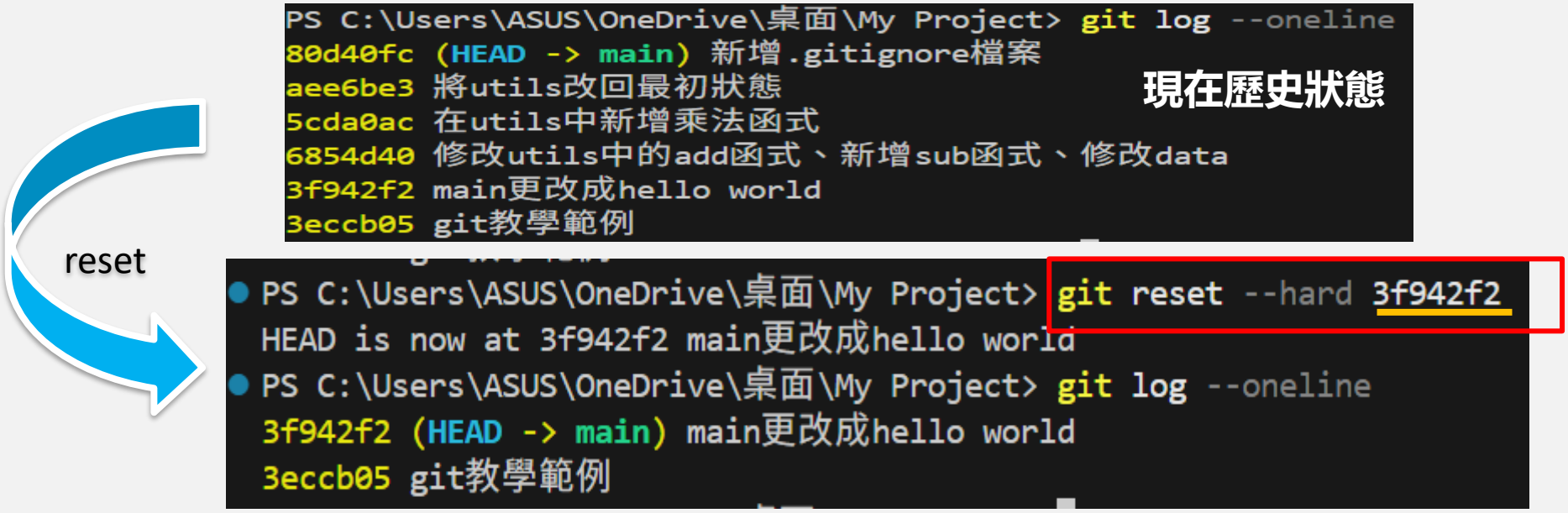
VS Code 終端機介面狀態

2-2 Git 常用指令與操作流程

（7）還原檔案到指定版本指令：git reset --hard <commitID>

重點：●這會將整個專案「完整回到」指定版本，後面所有 commit 都會消失。

●不可逆，需謹慎使用。



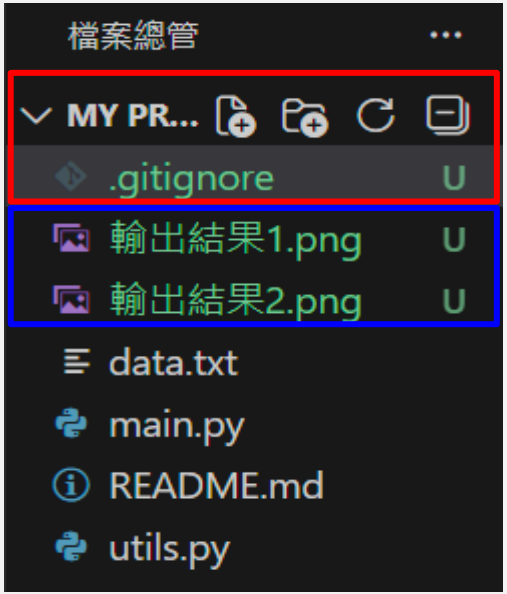
VS Code 終端機介面狀態

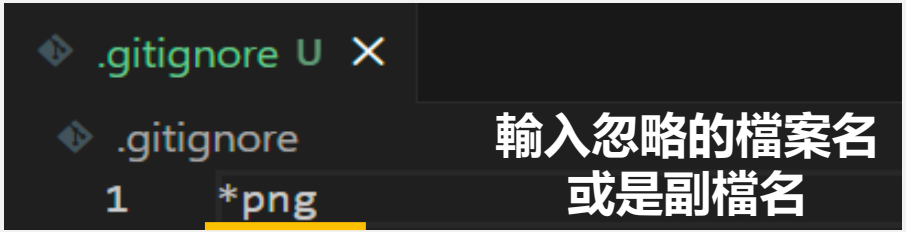
2-2 Git 常用指令與操作流程

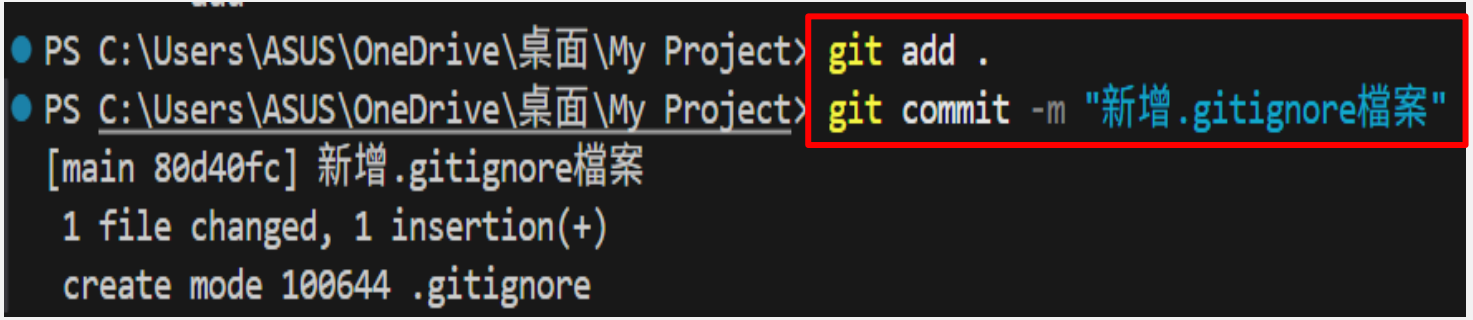
（8）忽略不必要追蹤的檔案：.gitignore

.gitignore 檔案用途：

- 避免追蹤不必要的檔案，讓版本庫保持乾淨
- 常用於：圖片、log、暫存檔、編譯產出、環境設定檔







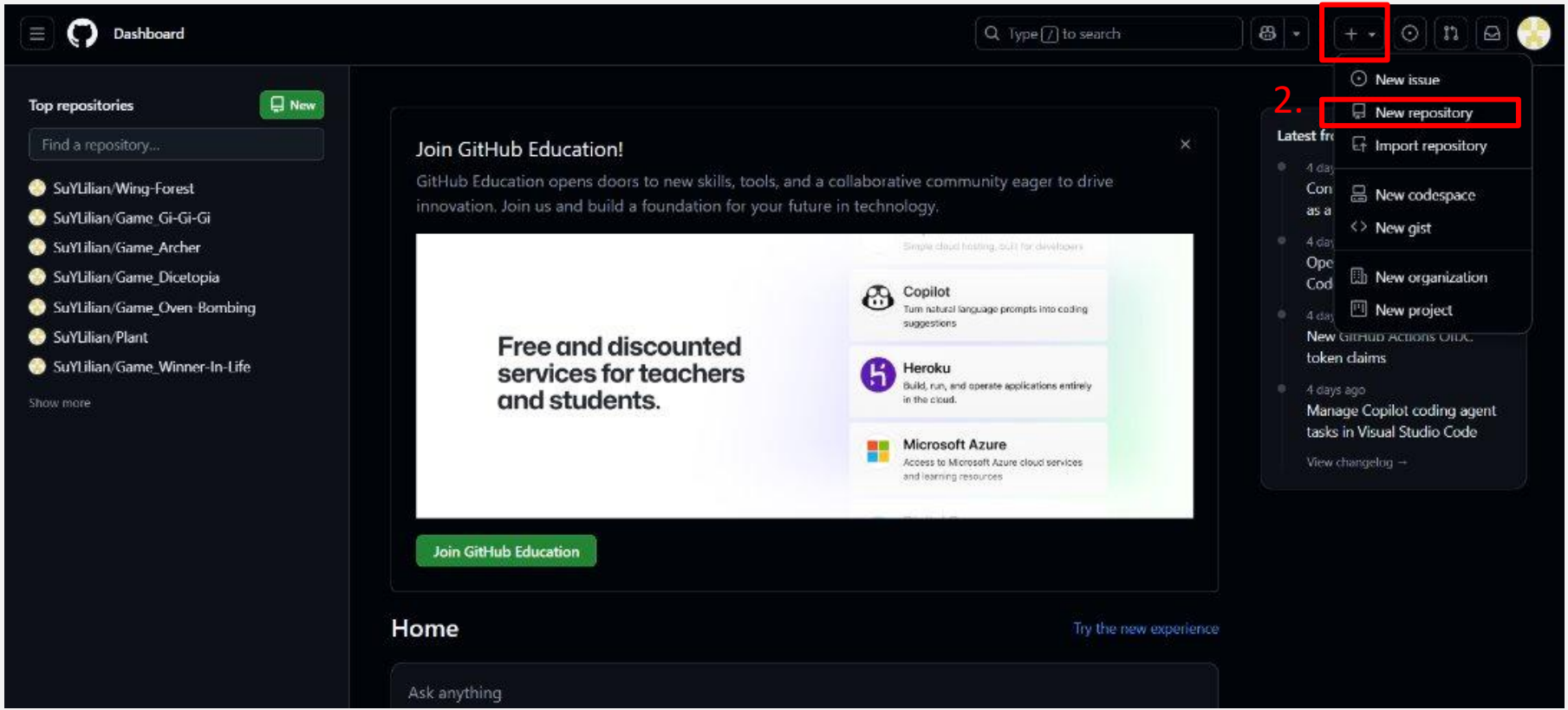
VS Code 終端機介面狀態

2-3 小結：Git 單人開發的基本工作流

功能	指令
看狀態	git status
暫存變化	git add 檔案、git add .、git add *.py
建立版本	git commit -m "訊息"
看歷史	git log、git log --oneline
比較差異	git diff、git diff <ID> -- 檔案
檔案還原	git checkout <ID> -- 檔案
專案回溯	git reset --hard <ID>（危險）
忽略檔案	.gitignore

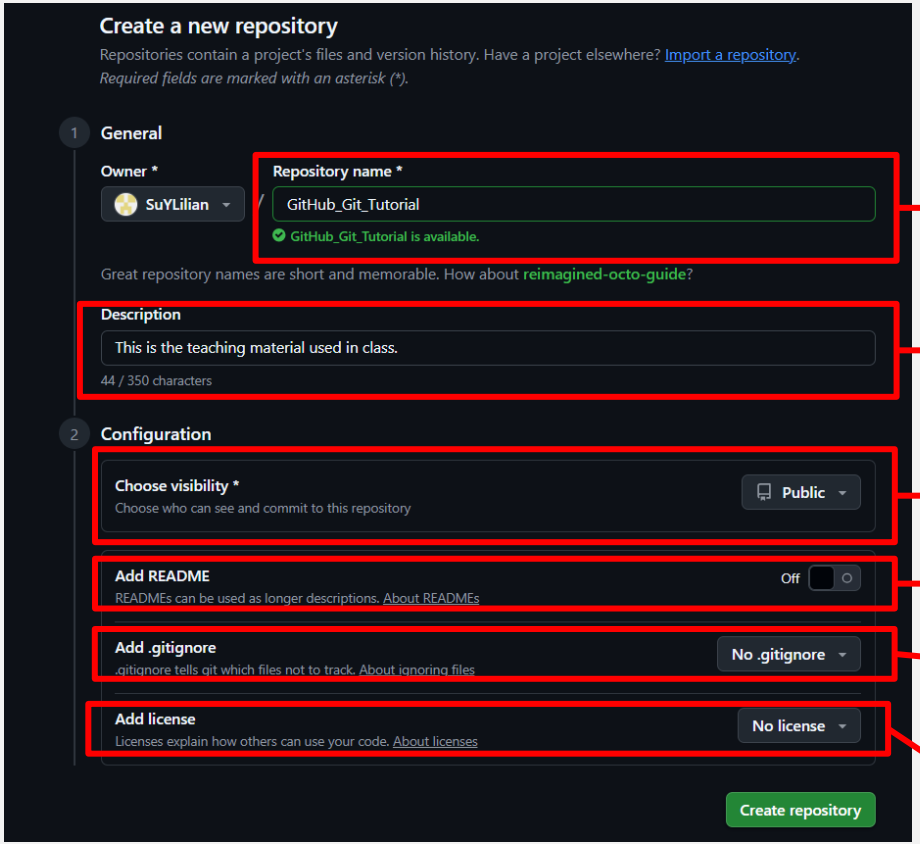
3 同步儲存庫

（1）新建儲存庫



3 同步儲存庫

（2）儲存庫基本設定



儲存庫**名稱**

儲存庫**簡介**

選擇可見性
- Public: 任何人都可以看到你的儲存庫
- Private: 只有你與你授權的人才能看到

加入說明文件

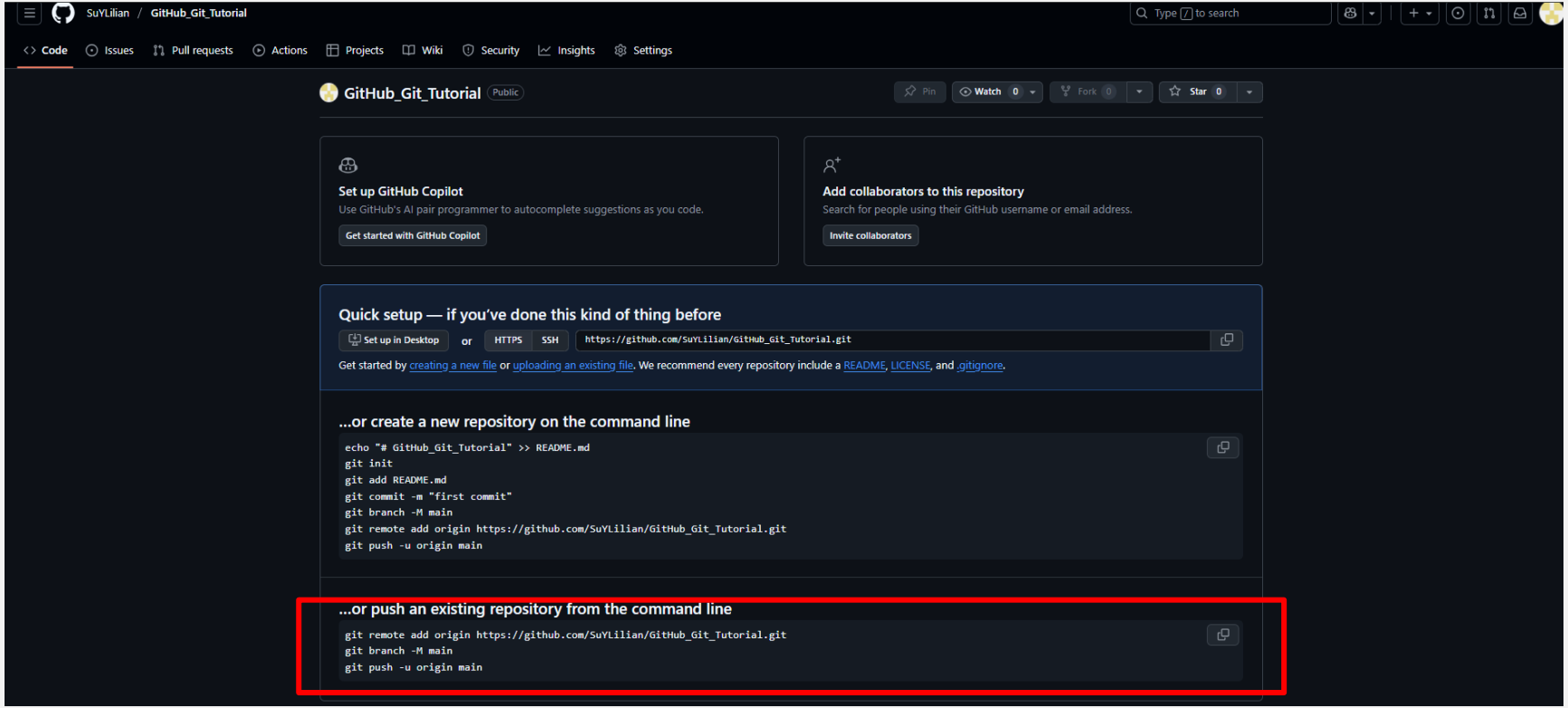
讓 Git 知道哪些檔案不應該被加入版本控制
(GitHub提供各語言/框架的範本)

加入授權條款

3 同步儲存庫

如果沒有加入README、.gitignore 或 license，會看到下面有提供現成的指令，

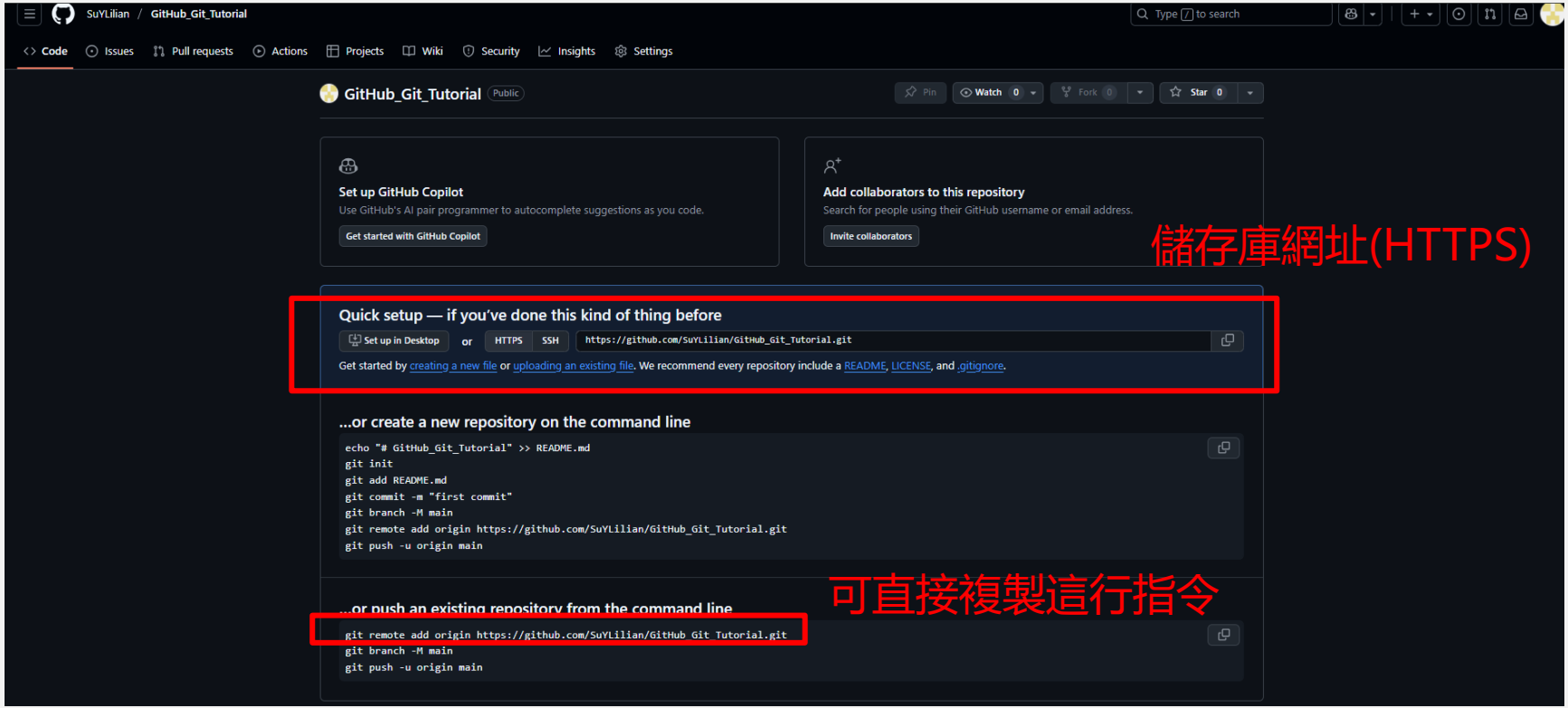
協助我們把本地的檔案推送到GitHub



3 同步儲存庫

(3) git remote add origin <儲存庫網址>

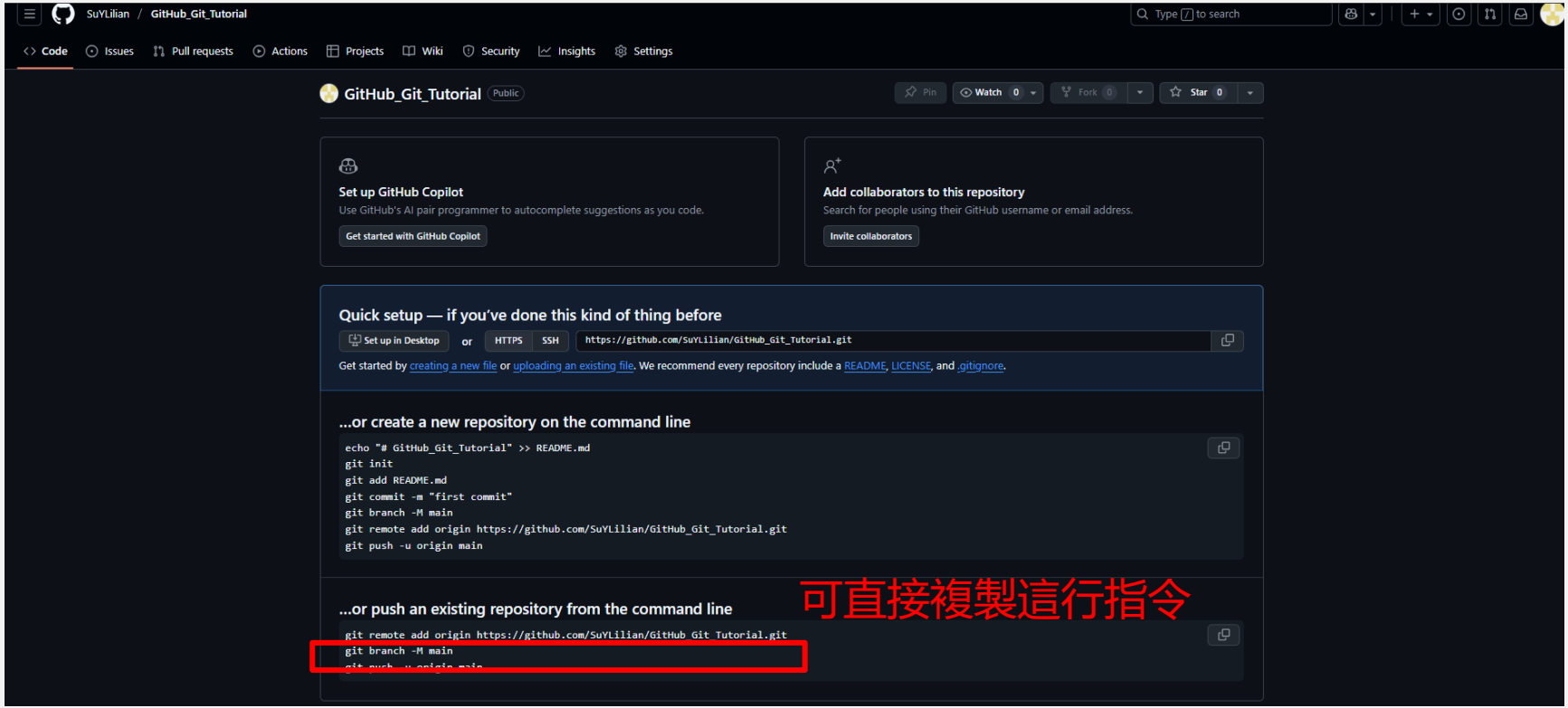
- 連接本地與遠端的儲存庫



3 同步儲存庫

(4) git branch –M main

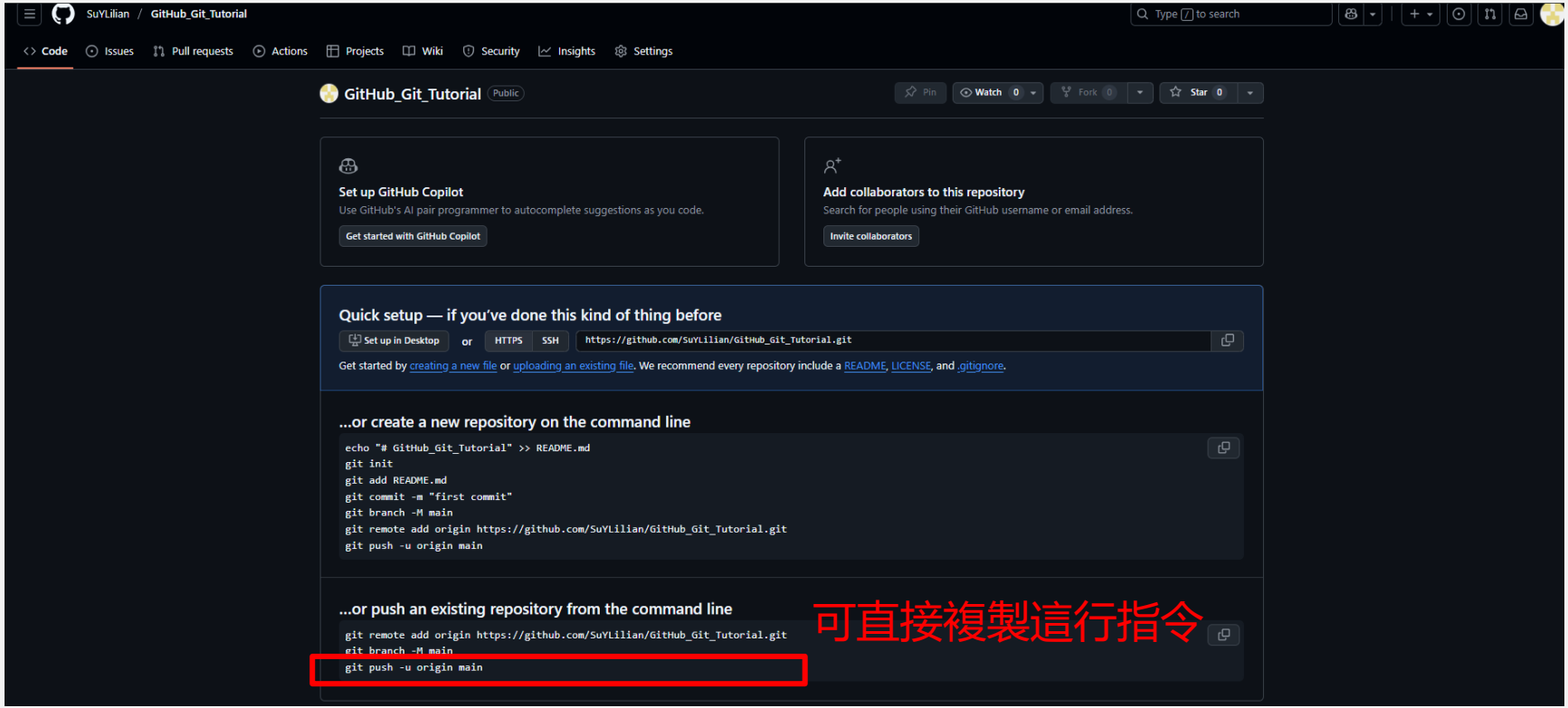
- 把目前的主線分支強制改名成 main，通常用來跟 GitHub 的預設分支名稱保持一致



3 同步儲存庫

(5) git push -u origin main

- 將本地檔案推送到GitHub



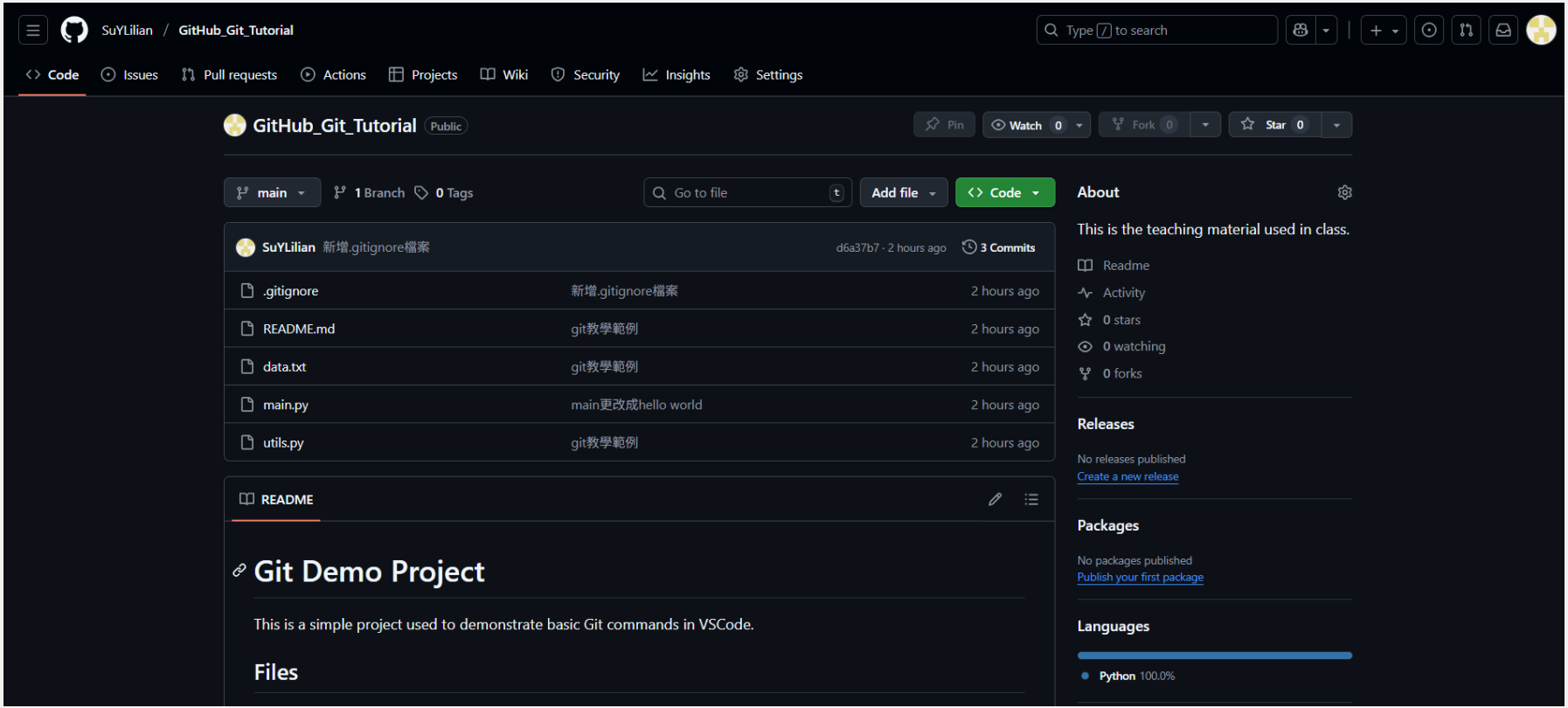
3 同步儲存庫

步驟	指令類型	說明
1	git remote add origin <儲存庫網址>	連接本地與遠端的儲存庫
2	git branch -M main	將主線分支改成main
3	git push -u origin main	推送到 GitHub

```
PS C:\Users\User\Desktop\My Project> git remote add origin https://github.com/SuYLilian/GitHub_Git_Tutorial.git
PS C:\Users\User\Desktop\My Project> git branch -M main
PS C:\Users\User\Desktop\My Project> git push -u origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.11 KiB | 1.11 MiB/s, done.
Total 12 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/SuYLilian/GitHub_Git_Tutorial.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\User\Desktop\My Project>
```

3 同步儲存庫

- 可以到GitHub看到從本地推送上去的檔案



3 同步儲存庫

- 若儲存庫在一開始新建時不是空的 (有README、License 或 .gitignore)，會出現以下錯誤
- 原因：GitHub 有某個檔案，但本地端沒有這個檔案，所以不讓你覆蓋

```
PS C:\Users\User\Desktop\MyProject_2> git remote add origin https://github.com/SuYLilian/RepoNotEmpty.git
PS C:\Users\User\Desktop\MyProject_2> git branch -M main
PS C:\Users\User\Desktop\MyProject_2> git push -u origin main
To https://github.com/SuYLilian/RepoNotEmpty.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/SuYLilian/RepoNotEmpty.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
PS C:\Users\User\Desktop\MyProject_2>
```

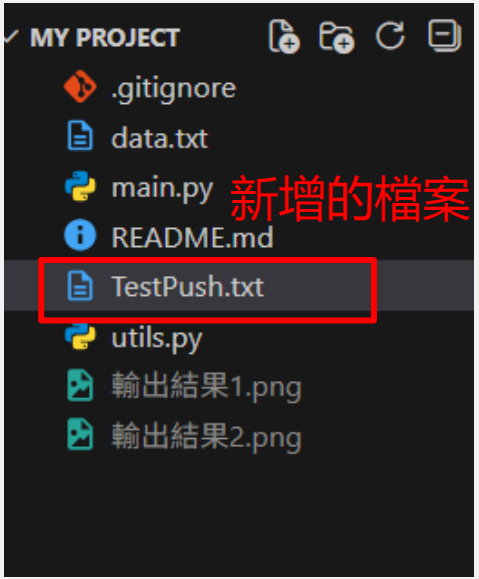
3 同步儲存庫

- 解決方式：在推送檔案到儲存庫之前，先把儲存庫原有的內容拉下來
- git pull origin main --allow-unrelated-histories

步驟	指令類型	說明
1	git remote add origin <儲存庫網址>	連接本地與遠端的儲存庫
2	git branch -M main	將主線分支改成main
3	git pull origin main --allow-unrelated-histories	把儲存庫原有的內容拉下來
4	git push -u origin main	推送到 GitHub

3 同步儲存庫

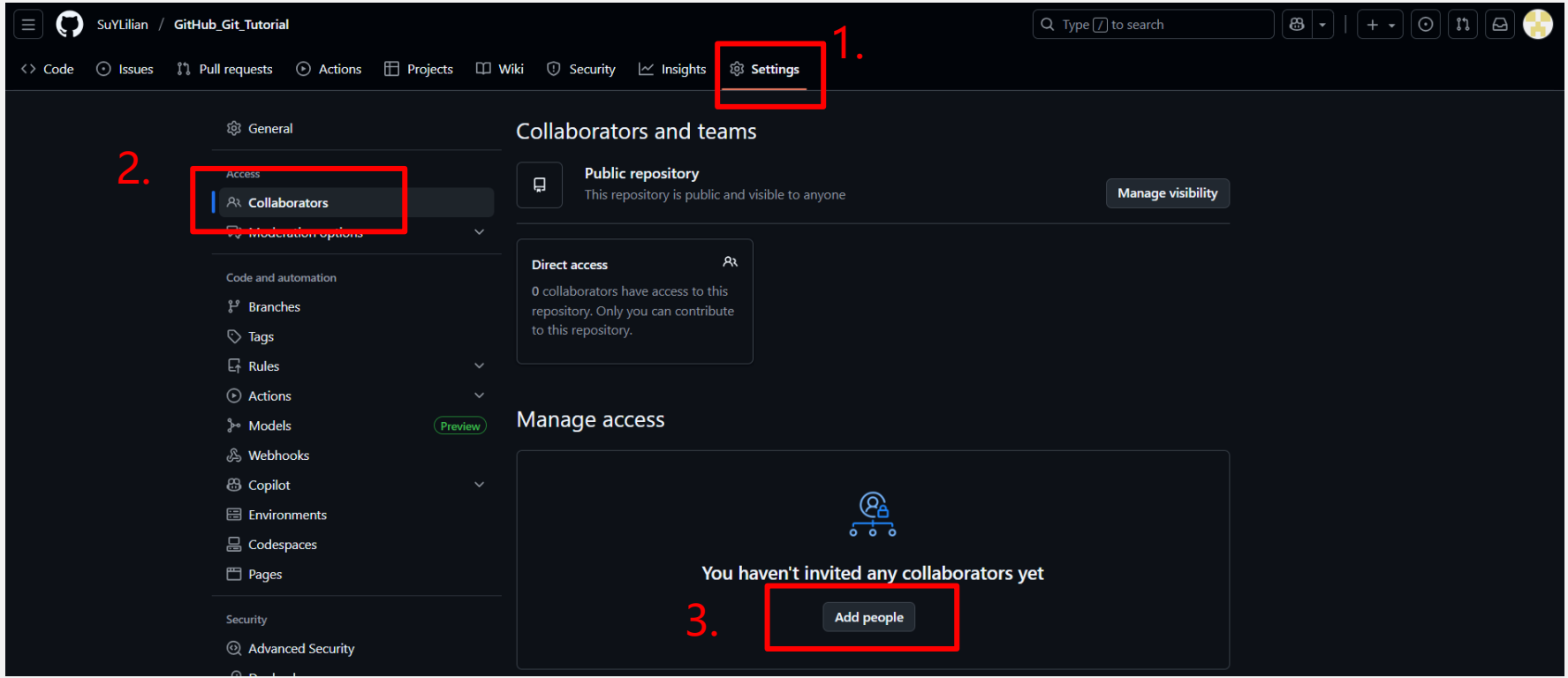
- 之後若有更改檔案內容 (修改內容、新增檔案、刪除檔案)，一樣要將檔案加入暫存 -> commit -> 推送到 GitHub




```
git add .
git commit -m "提交訊息 “
git push
```

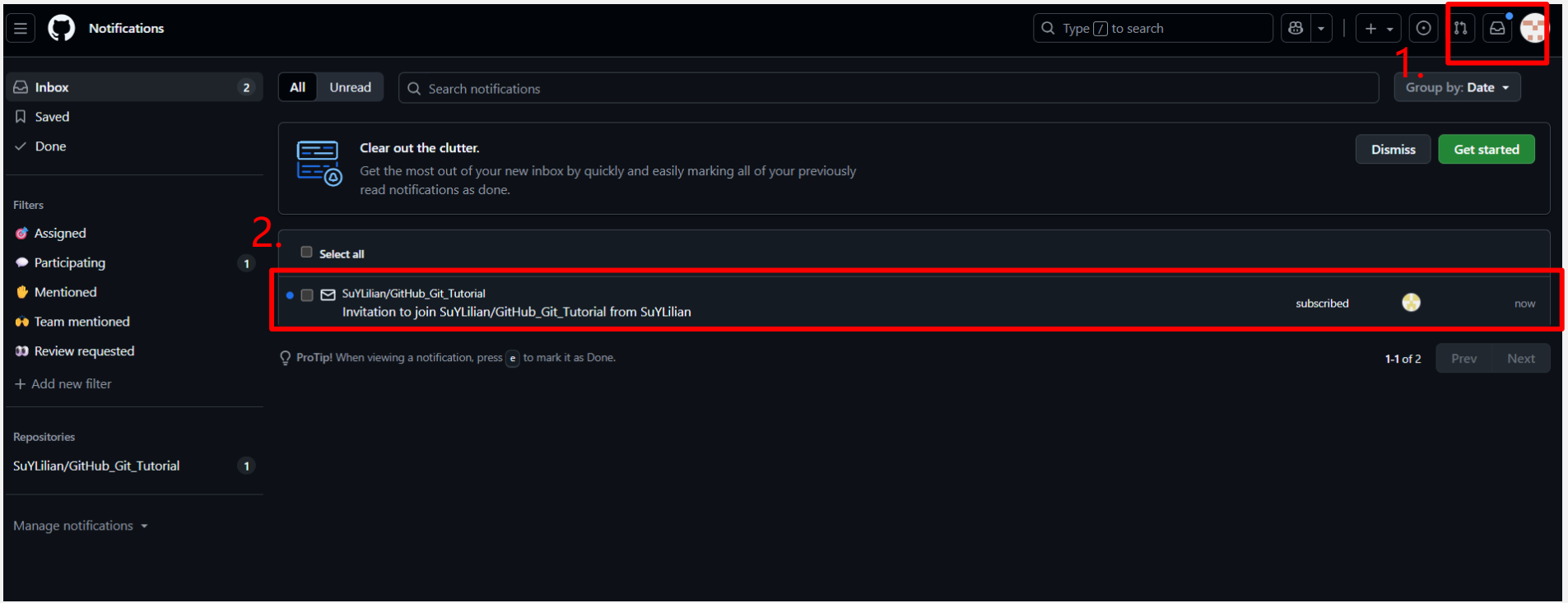
4 協作

- 輸入對方的名稱或email，可以加入其他人協作專案



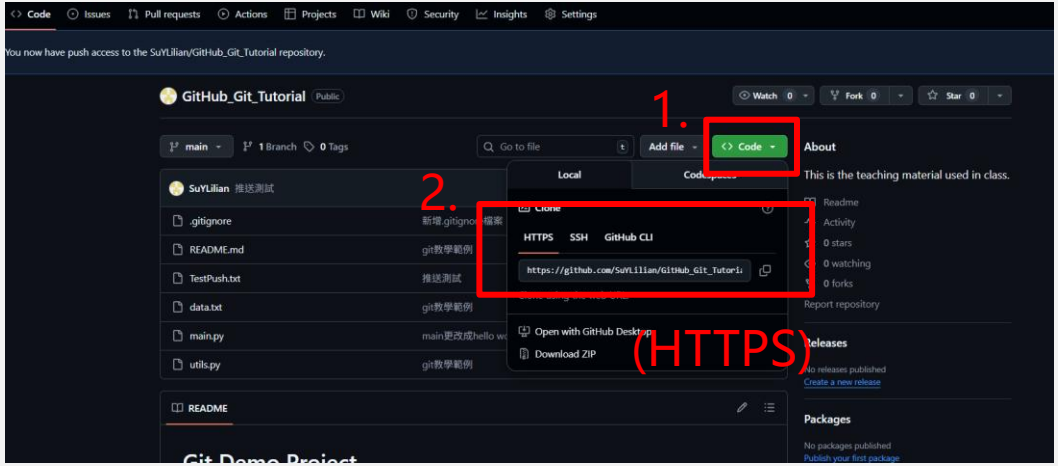
4 協作

- 被邀請者會收到訊息



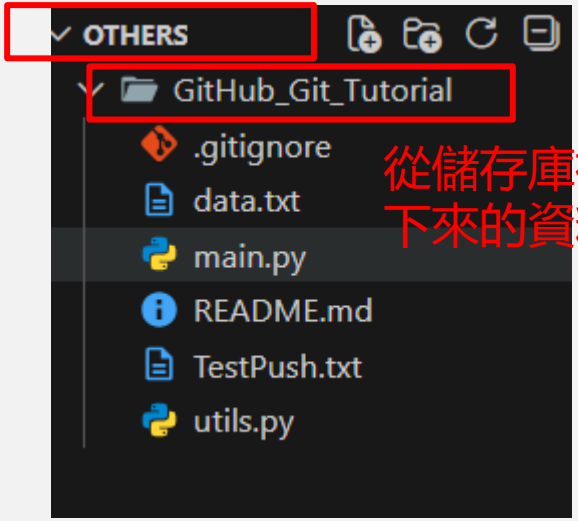
4 協作

- 被邀請者需新建一個資料夾，將儲存庫的內容複製到本地端
- git clone <儲存庫網址>




```
PS C:\Users\User\Desktop\Others> git clone https://github.com/SuYLilian/GitHub_Git_Tutorial.git
Cloning into 'GitHub_Git_Tutorial'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 15 (delta 4), reused 15 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (15/15), done.
Resolving deltas: 100% (4/4), done.
PS C:\Users\User\Desktop\Others>
```

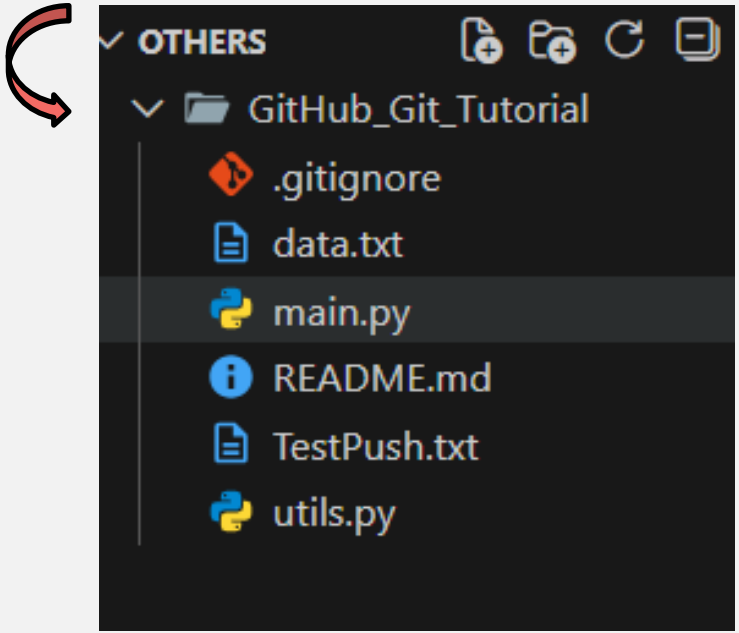
新建的資料夾



從儲存庫複製下來的資料夾

4 協作

- 若被邀請者要編輯儲存庫，需要將程式執行路徑更改至儲存庫的資料夾
- cd 資料夾名稱




```
PS C:\Users\User\Desktop\Others> cd GitHub_Git_Tutorial
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial>
```

維持同步

- 如果其他人有推送修改後的檔案至GitHub，我們也要更新自己的本地端，把新紀錄的檔案拉下來
- git pull

為什麼要建立分支（Branch）？

- 避免影響主線（main）程式

  → 在分支上開發新功能，不會破壞現有版本。

- 多人協作更安全

  → 每個人都有自己的分支，避免互相干擾。

- 方便測試與實驗

  → 想嘗試新想法？先在分支測試，失敗也不會影響主專案。

- 清楚管理不同功能與版本

  → 每個功能一條分支，最後再合併。

5 分支與合併

指令類型	說明
git switch -c 分支名稱	建立新分支並切換過去
git branch 分支名稱	建立新分支
git switch 分支名稱	切換分支
git branch	查看分支清單/目前所在分支

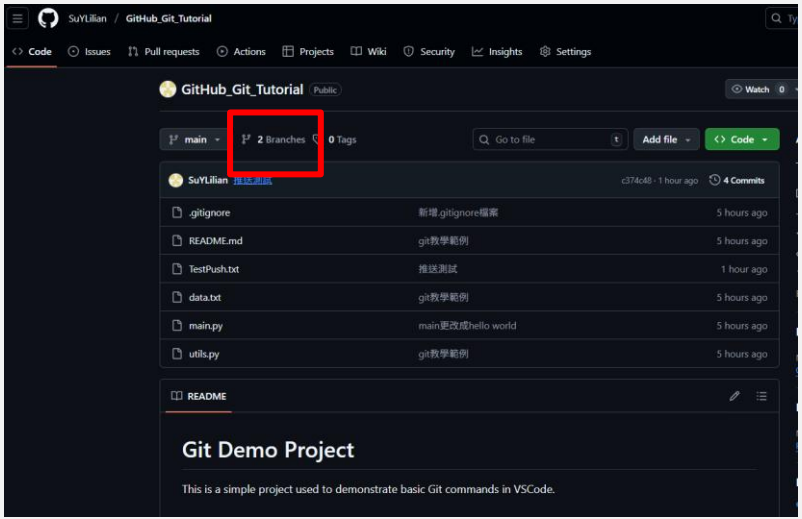
*代表目前所在分支

```
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git switch -c branch2
Switched to a new branch 'branch2'
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git branch
* branch2
  main
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git branch branch3
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git switch branch3
Switched to branch 'branch3'
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git branch
  branch2
* branch3
  main
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial>
```

5 分支與合併

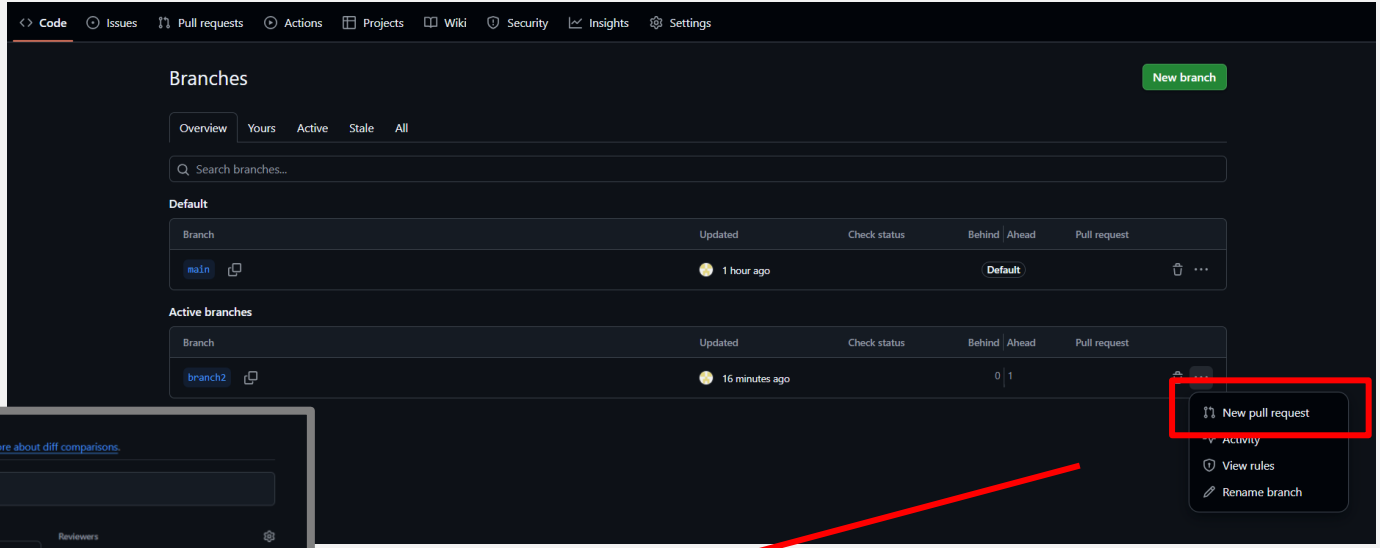
- 將當前分支所變更的紀錄推送到GitHub
- git push origin 分支名稱

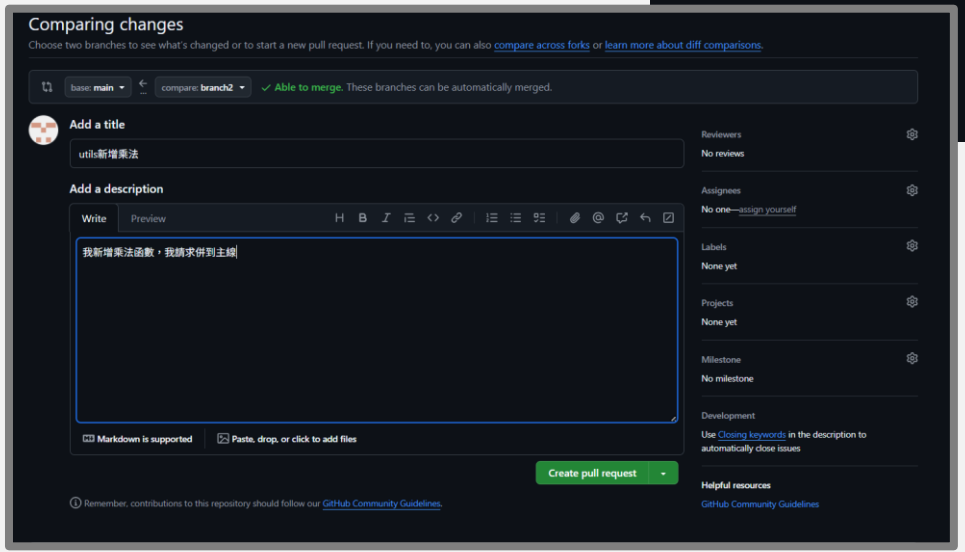
```
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git switch branch2
Switched to branch 'branch2'
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git add .
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git commit -m "utils新增乘法"
[branch2 e3171a7] utils新增乘法
 1 file changed, 3 insertions(+)
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git push origin branch2
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 311 bytes | 311.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'branch2' on GitHub by visiting:
remote:      https://github.com/SuYLilian/GitHub_Git_Tutorial/pull/new/branch2
remote:
To https://github.com/SuYLilian/GitHub_Git_Tutorial.git
 * [new branch]      branch2 -> branch2
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial>
```



5 分支與合併

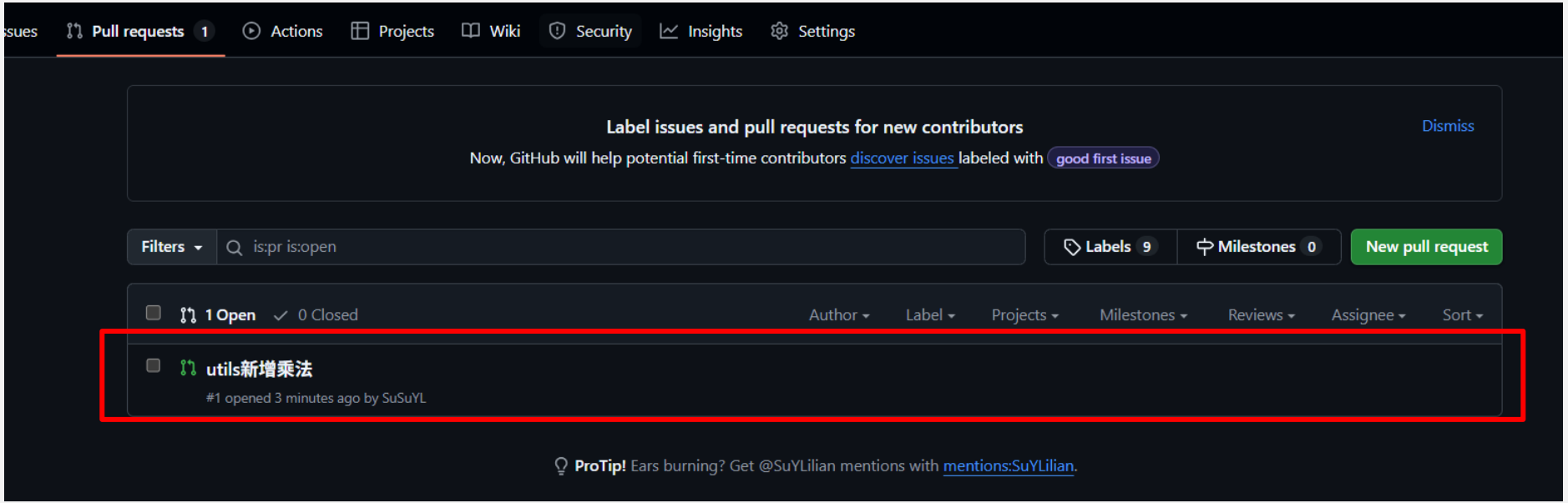
- 我們可以請求管理者將分支合併至主線
- Pull request



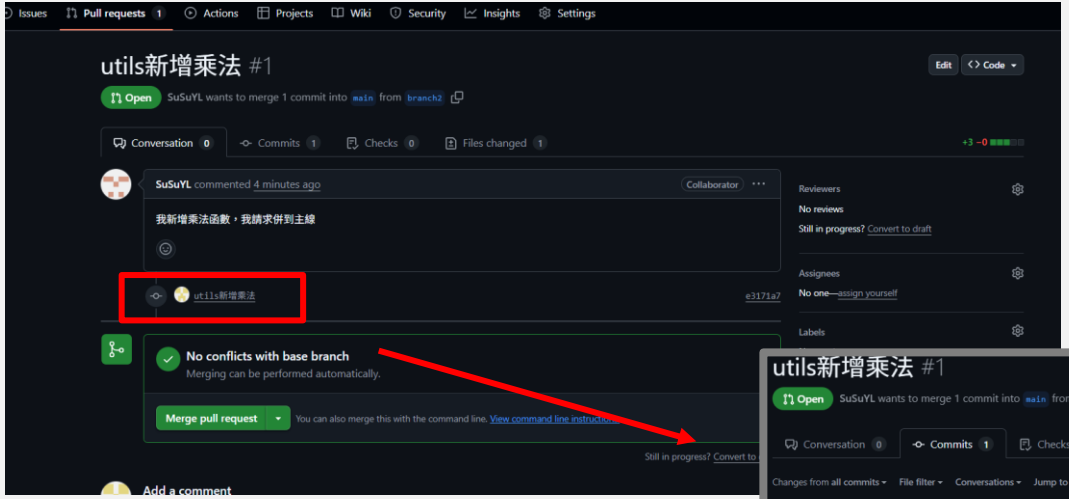


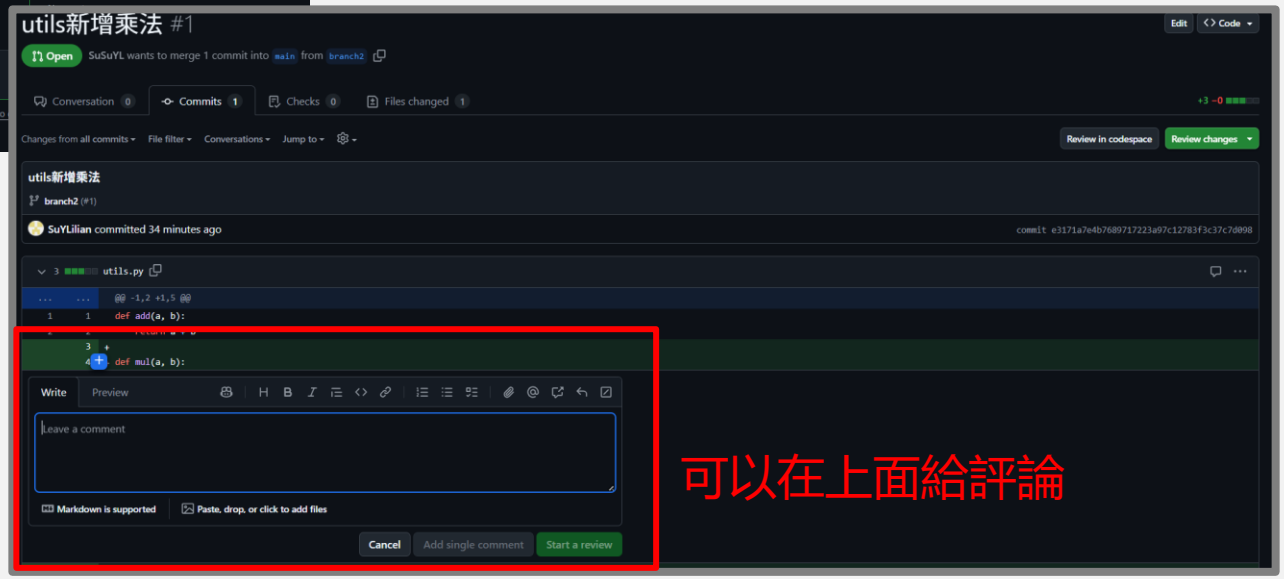
5 分支與合併

- 管理者可以看到請求



5 分支與合併



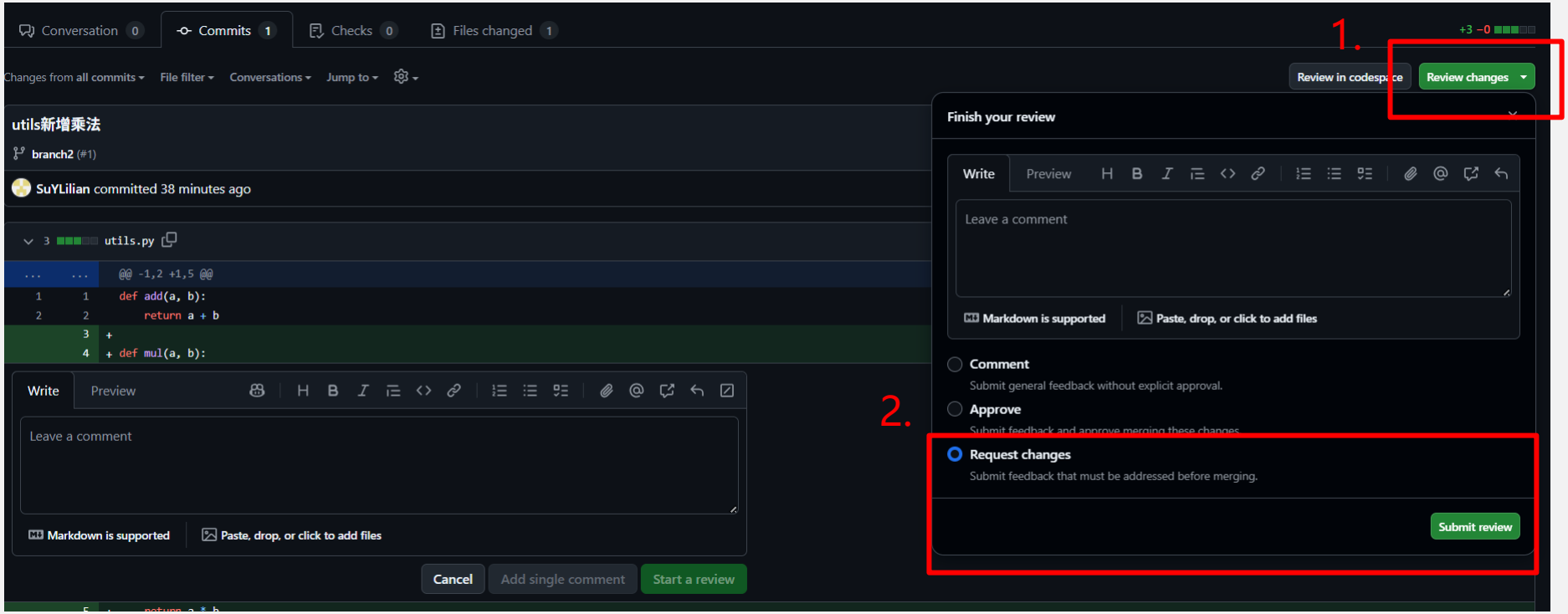


- 可以查看修改的內容，並在上面給評論

可以在上面給評論

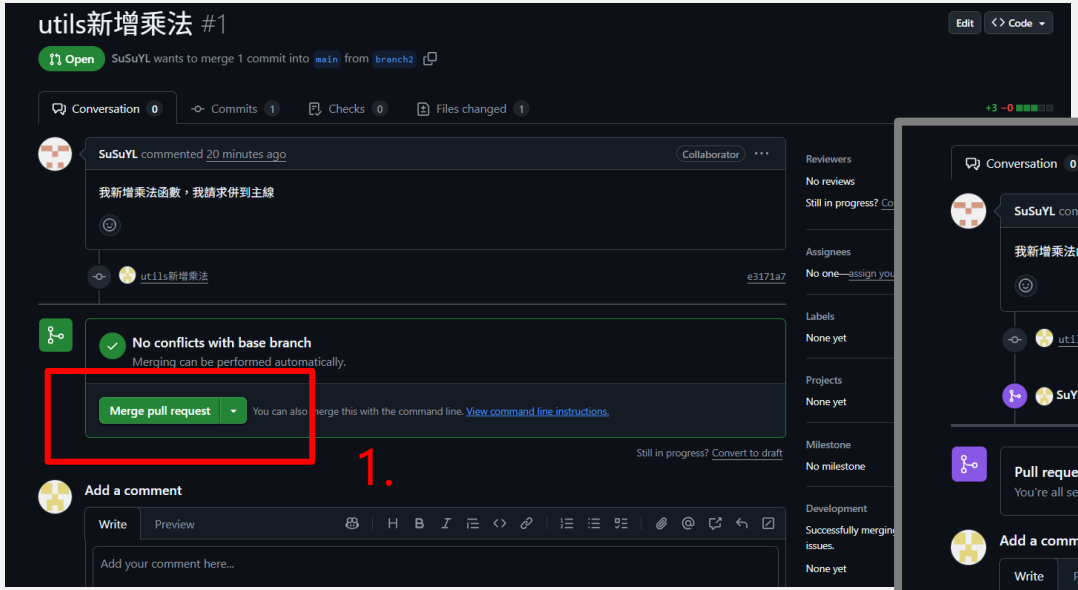
5 分支與合併

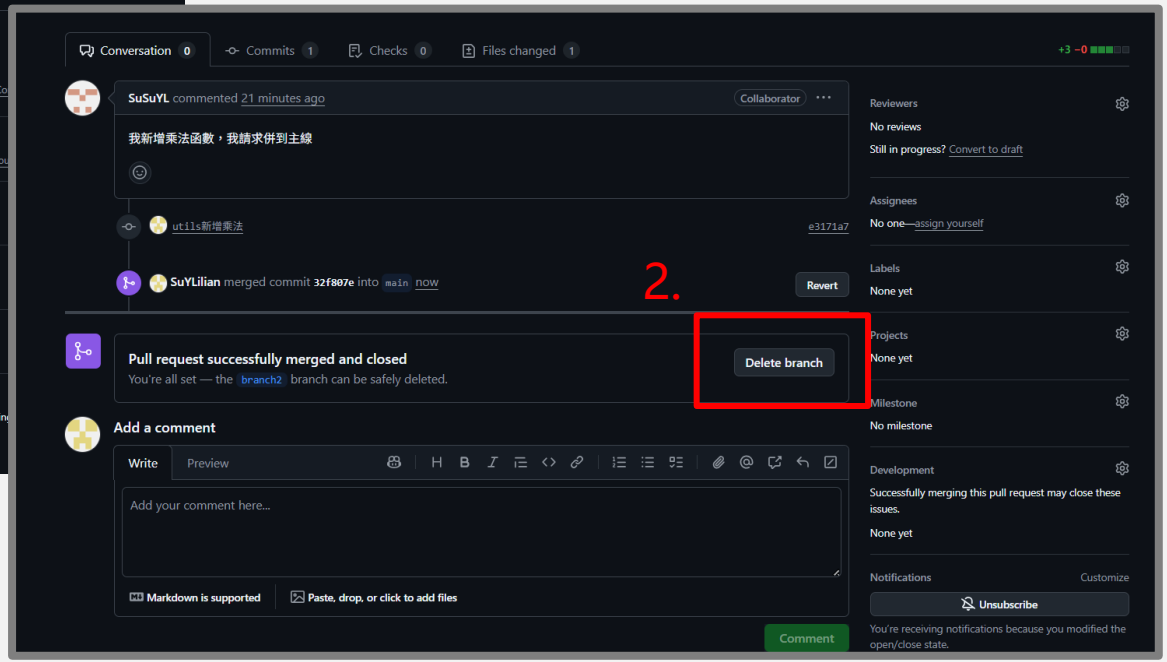
- 或是請求進行修改



5 分支與合併

- 如果覺得沒問題就可以合併分支，並刪除分支





5 分支與合併

- GitHubg上刪除分支後，本地端也需同步，需手動刪除

步驟	指令類型	說明
1	git switch main	切換至主線
2	git pull	下載GitHub上最新的變更
3	git branch -d 分支名稱	刪除分支
4	git fetch -p	清掉不存在的遠端分支記錄

5 分支與合併

```
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (1/1), 920 bytes | 306.00 KiB/s, done.
From https://github.com/SuYLilian/GitHub_Git_Tutorial
   c374c48..32f807e  main       -> origin/main
Updating c374c48..32f807e
Fast-forward
 utils.py | 3 +++
 1 file changed, 3 insertions(+)
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git branch -d branch2
Deleted branch branch2 (was e3171a7).
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git fetch -p
From https://github.com/SuYLilian/GitHub_Git_Tutorial
 - [deleted]         (none)     -> origin/branch2
PS C:\Users\User\Desktop\Others\GitHub_Git_Tutorial> git branch
  branch3
* main
```

指令整理

指令	說明
git status	看狀態
git add 檔案、git add .、git add *.py	暫存變化
git commit -m "訊息"	建立版本
git log、git log --oneline	看歷史
git diff、git diff <ID> -- 檔案	比較差異
git checkout <ID> -- 檔案	檔案還原
git reset --hard <ID>（危險）	專案回溯
git remote add origin <儲存庫網址>	連接本地與遠端的儲存庫
git branch -M main	將主線分支改成main
git push -u origin main	推送到 GitHub (第一次推送)
git pull origin main --allow-unrelated-histories	把儲存庫原有的內容拉下來
git push	推送更改記錄到 GitHub

指令	說明
git clone <儲存庫網址>	將儲存庫的內容複製到本地端
cd 資料夾名稱	切換到想操作的資料夾
git pull	將儲存庫的內容同步至本地端
git switch -c 分支名稱	建立新分支並切換過去
git branch 分支名稱	建立新分支
git switch 分支名稱	切換分支
git branch	查看分支請單/目前所在分支
git push origin 分支名稱	分支所變更的紀錄推送到GitHub
git branch -d 分支名稱	刪除分支
git fetch -p	清掉不存在的遠端分支記錄

什麼是Git 的圖形使用者介面（GUI）？

Git GUI 是用視覺化操作來管理版本控制的工具。使用者可以用滑鼠點選按鈕來完成 Git 的動作（如 commit、push），不需要輸入指令。

常見的 Git GUI 工具↓

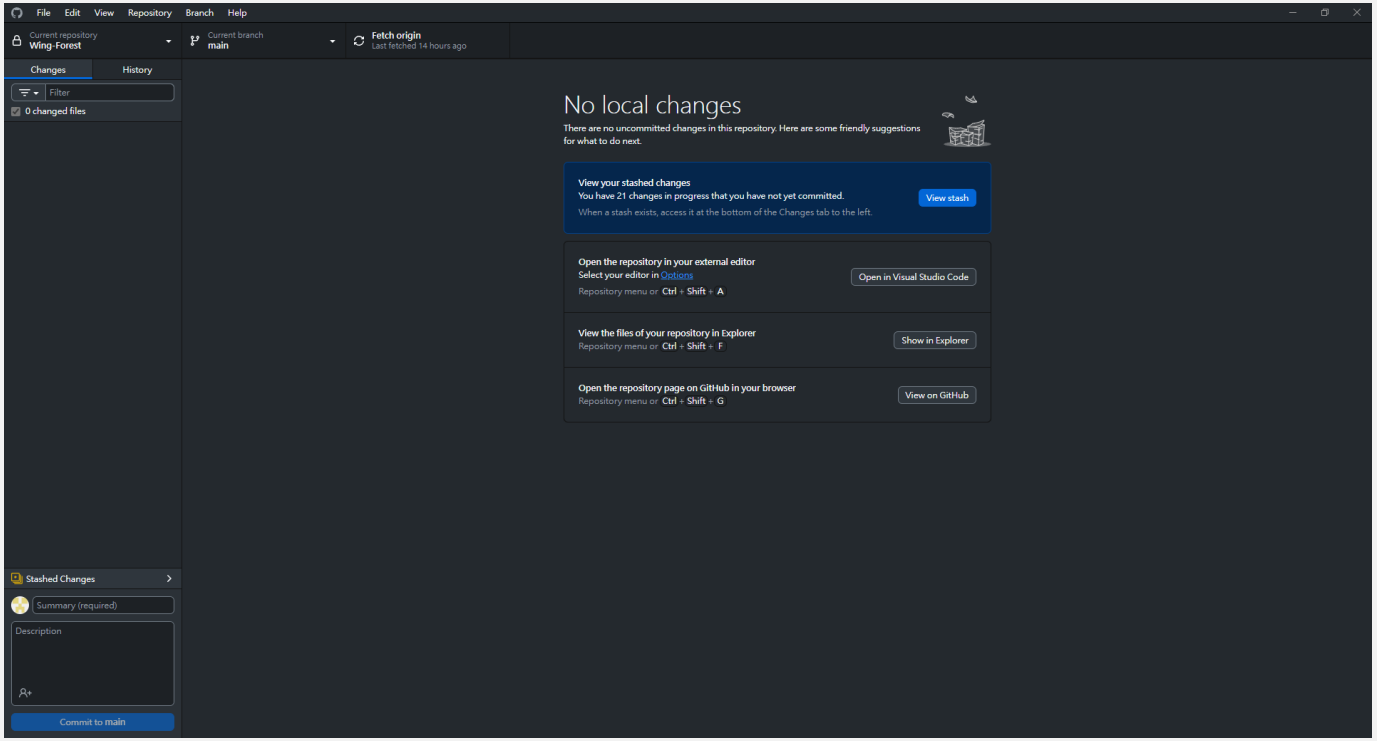
GitHub Desktop

Sourcetree

GitKraken

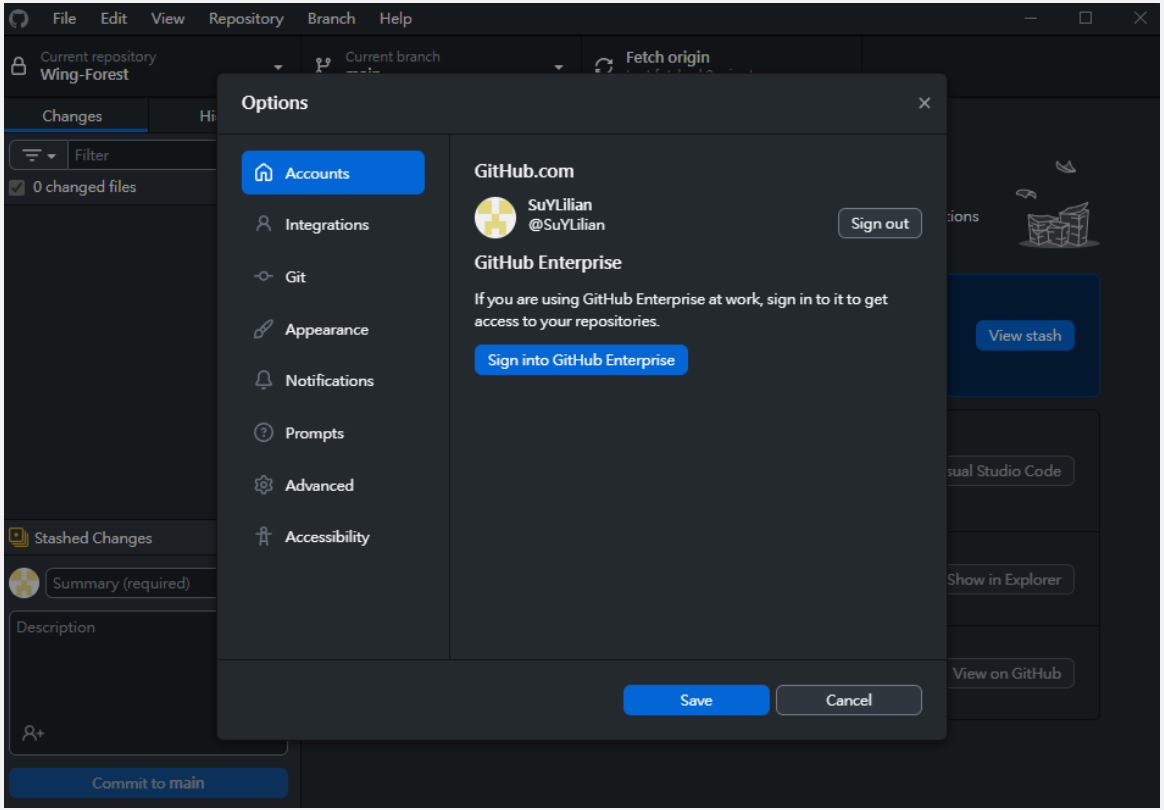
6 Git GUI (GitHub Desktop)

- GitHub Desktop
- 教學影片參考: https://youtu.be/8Dd7KRpKeaE?si=y72qpKdGI8Yx1jAj



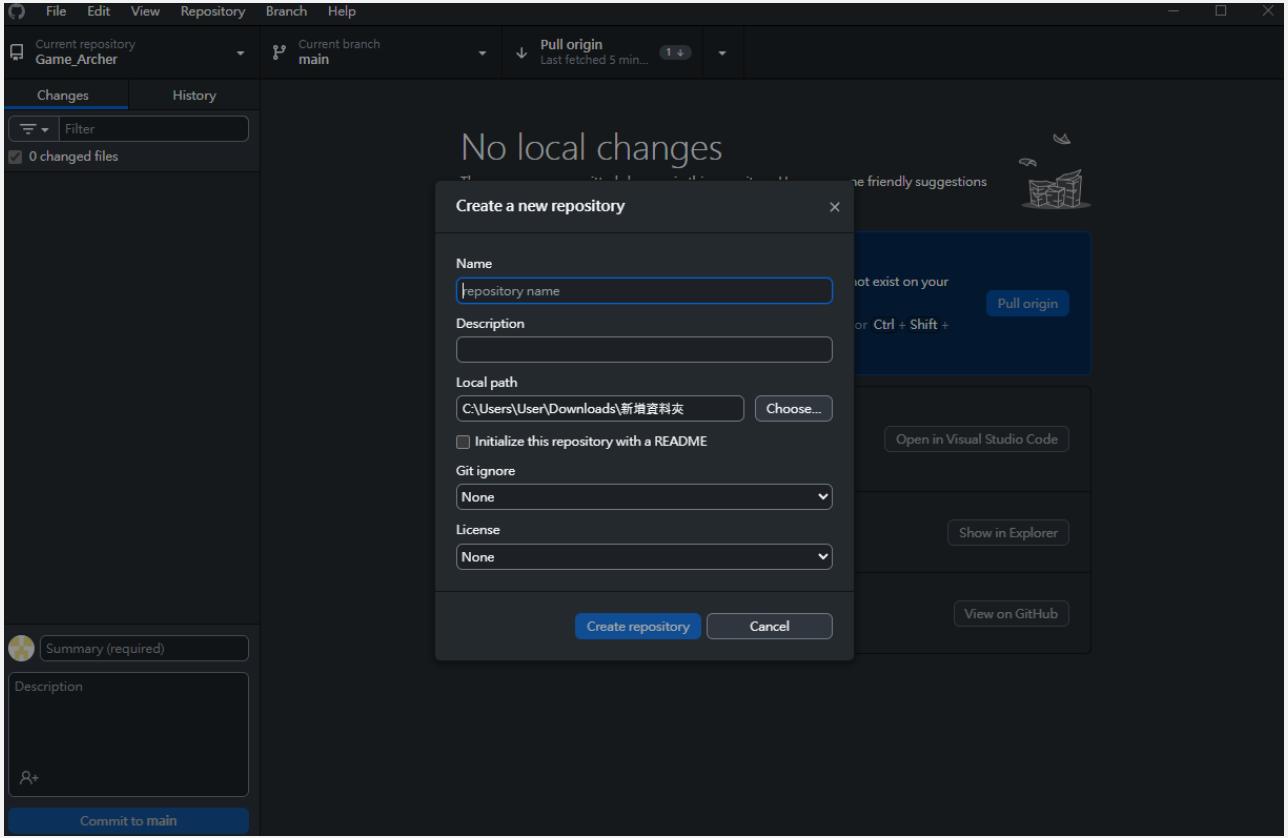
6 Git GUI (GitHub Desktop)

- 登入GitHub帳號，連結GitHub
- File → Options → Accounts



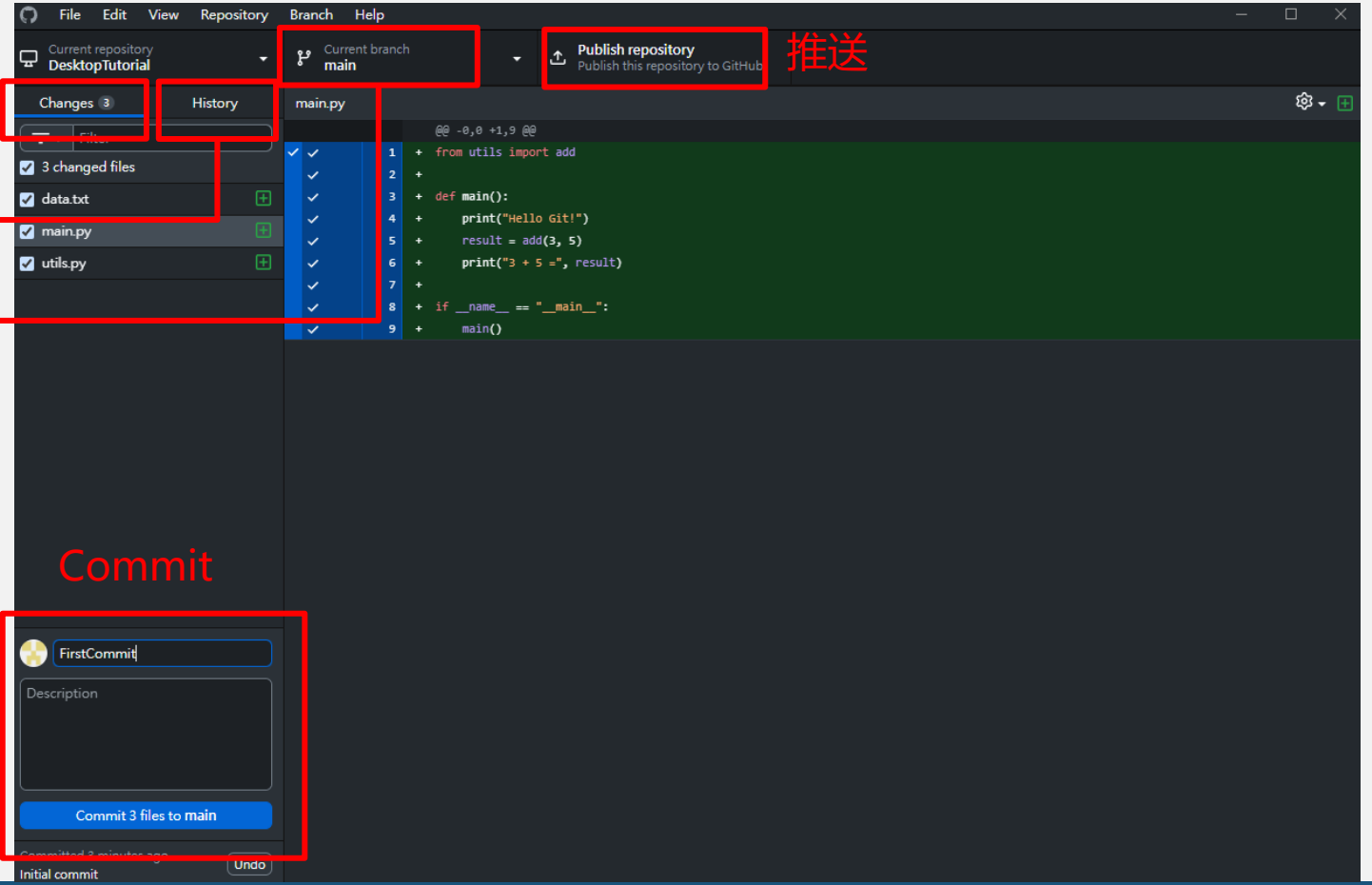
6 Git GUI (GitHub Desktop)

- 新建儲存庫
- File → New repository



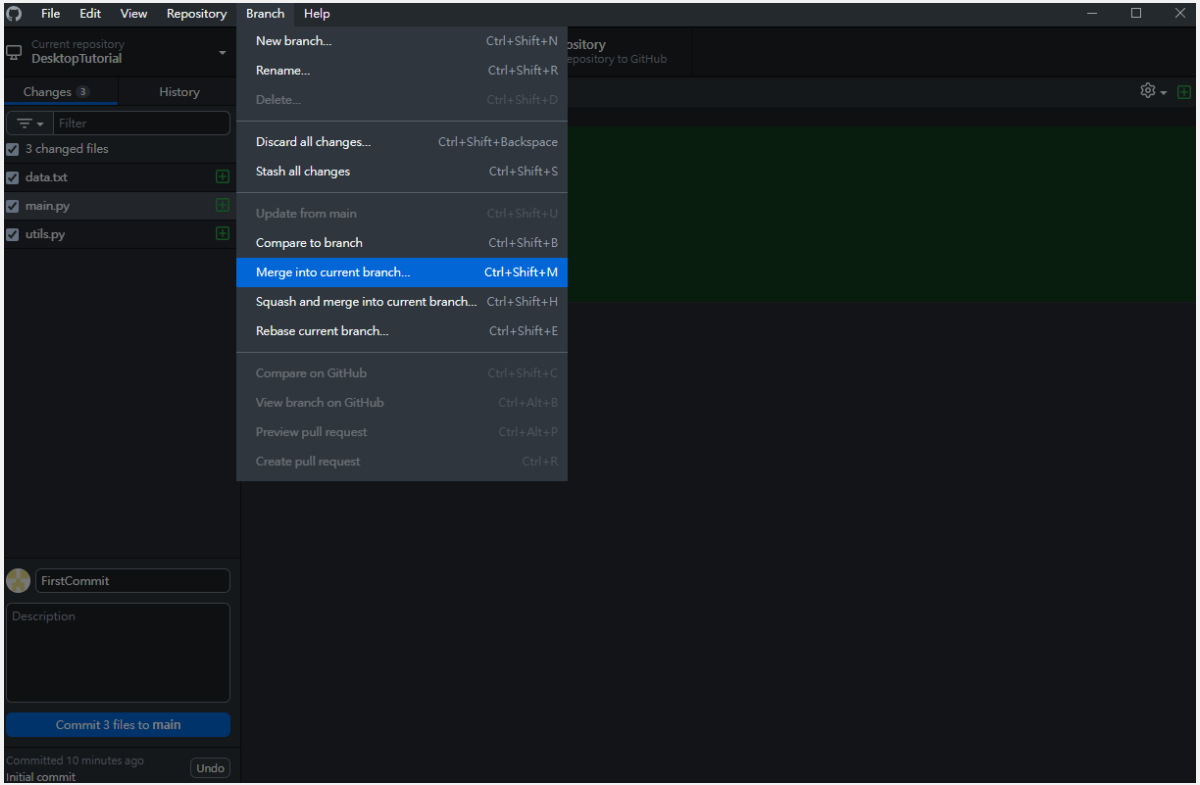
6 Git GUI (GitHub Desktop)

- 介面介紹



6 Git GUI (GitHub Desktop)

- 合併分支



Thank you for listening