Lab0
GitHub 版本控制

本節目的：

- 了解版本控制行為、版本控制系統。
- 用 Git 解決對於程式碼的版本控制的困擾。
- 了解常用 Git 的指令以及其功能。
- 實際練習 Git 與 GitHub 的基本使用情境。

0.1 Github 註冊及登入

0.1.1 註冊 GitHub 帳號

Step1 至 GitHub 官方網站 https://github.com/註冊帳號，如圖 1-1 所示。

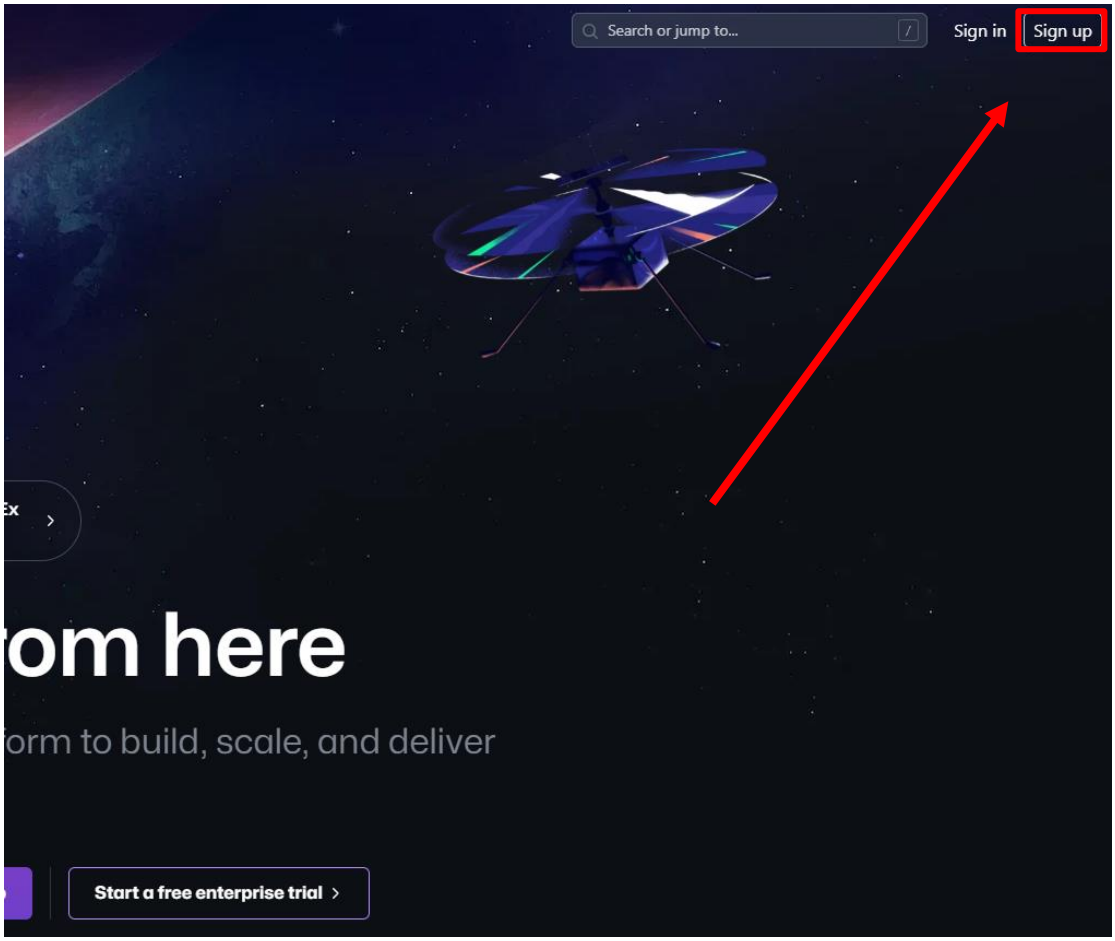


圖 1-1 GitHub 官網

Step2 填寫註冊資料並驗證，如圖 1-2 所示。

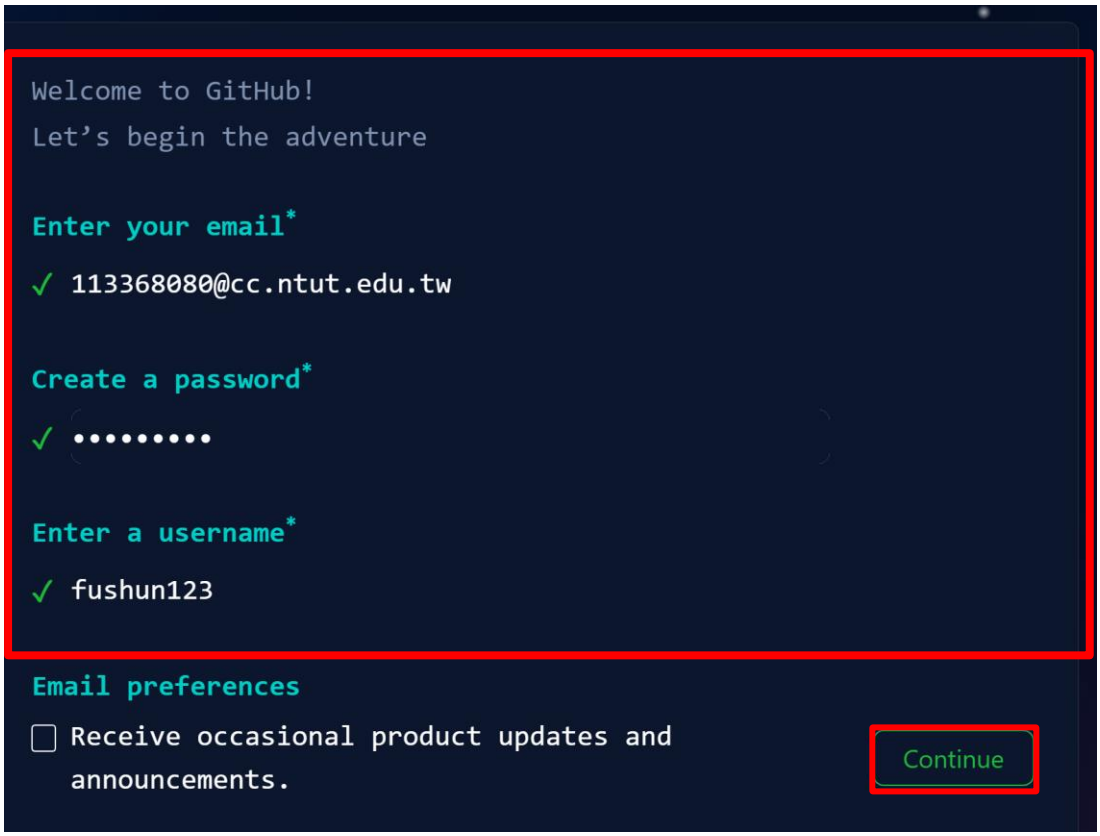


圖 1-2 註冊基本資料

Step3 選擇 GitHub 註冊帳號的方案，此處保持 Free 的方案，然後直接點擊「Continue for Free」即可完成註冊，如圖 1-3 所示。

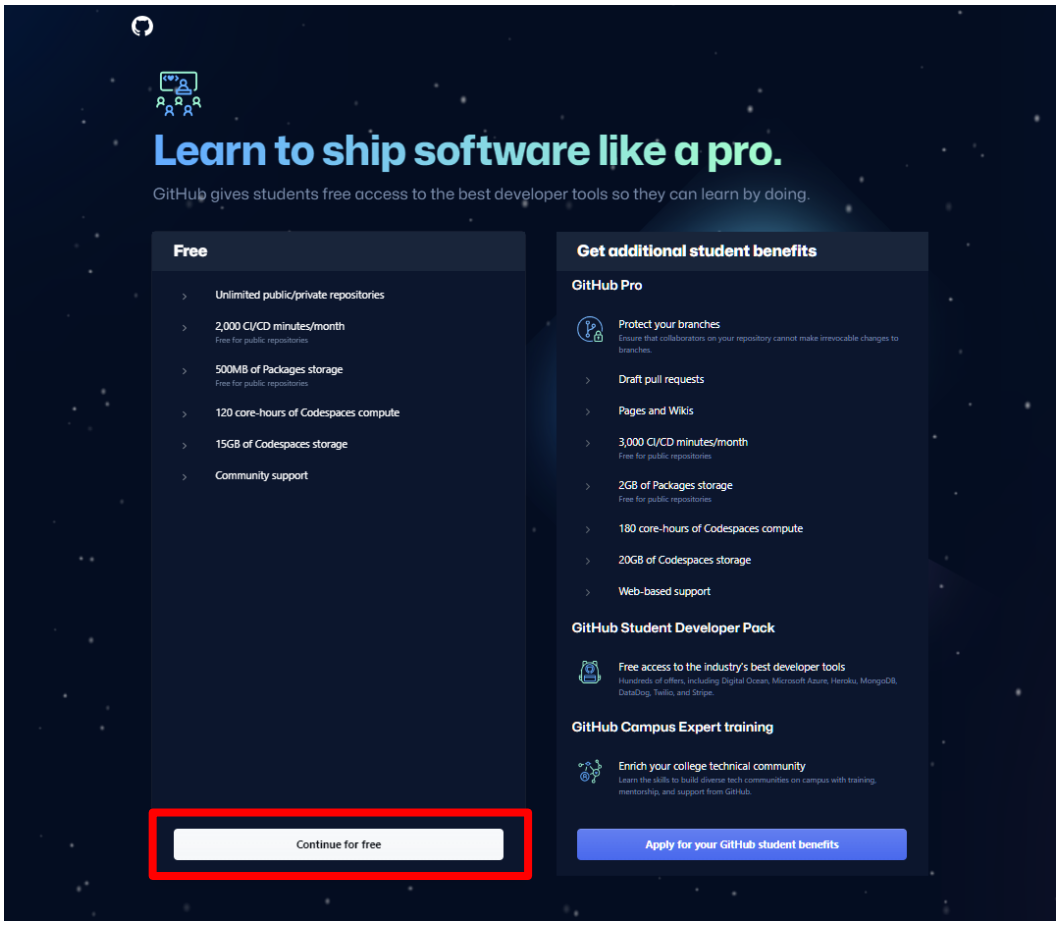


圖 1-3 選擇免費方案

Step4 完成後會看到此網頁，如圖 1-4 所示。

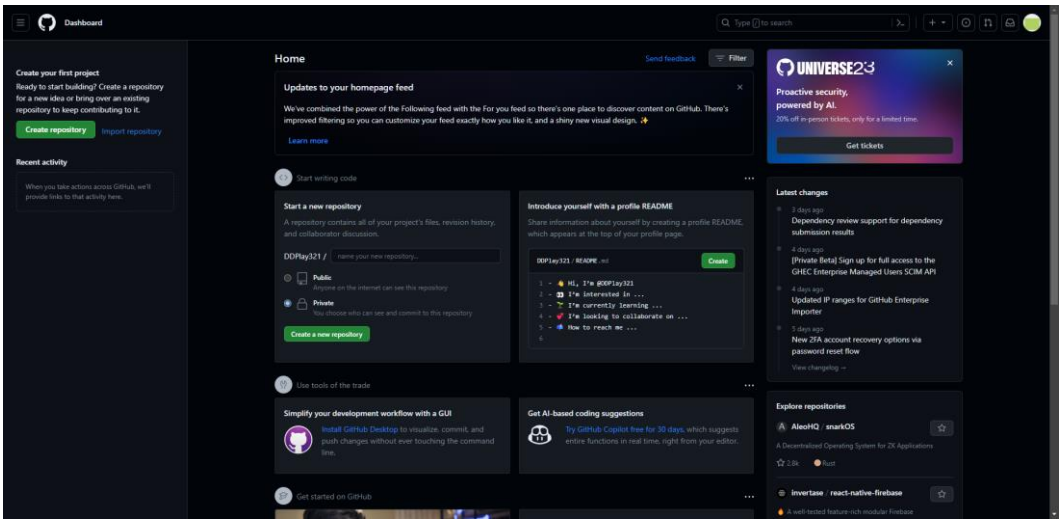


圖 1-4

0.1.2 Github 登入

Step1 在註冊完後並登入 Github，並點選右上角 Sign in，如圖 1-5 所示

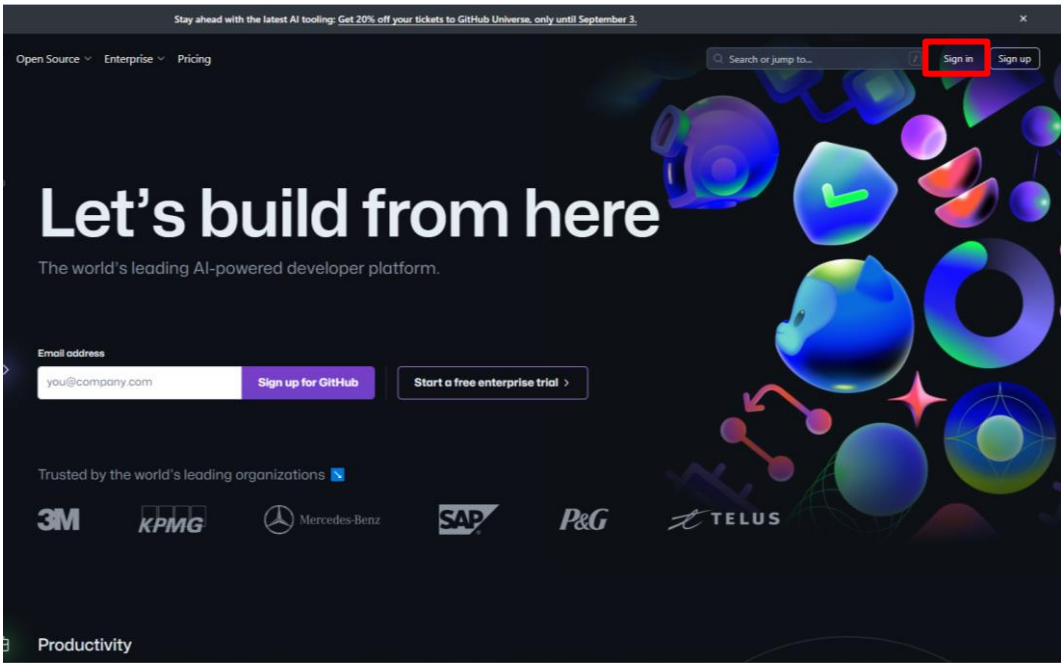


圖 1-5

Step2 輸入 Username or email address 及 Password，如圖 1-6 所示

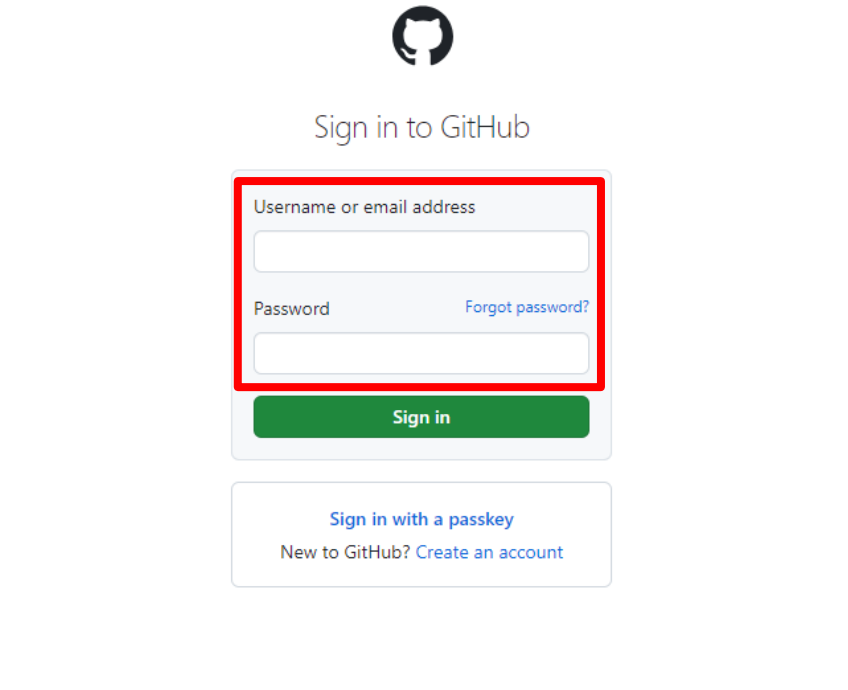


圖 1-6

0.1.3 安裝 Git 使用環境 Git Bash

Step1 至 https://git-scm.com/download/win 下載 Git 安裝檔，如圖 1-7 所示。

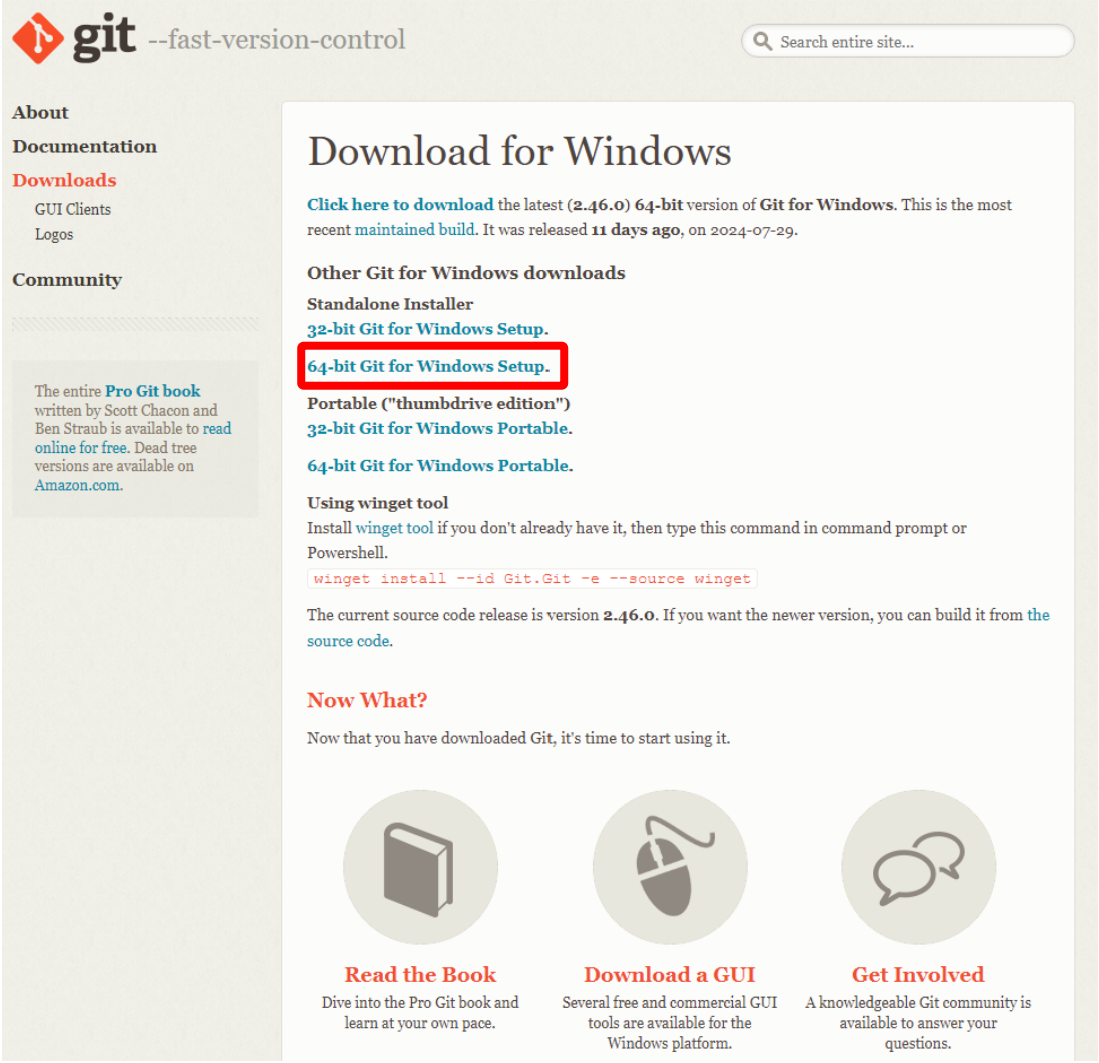


圖 1-7 下載 Git 安裝檔

Step2 開啟安裝檔，開始安裝 Git，如圖 1-8 所示。

圖 1-8 允許安裝 Git

Step3 閱讀並同意授權聲明，點擊「Next」，如圖 1-9 所示。

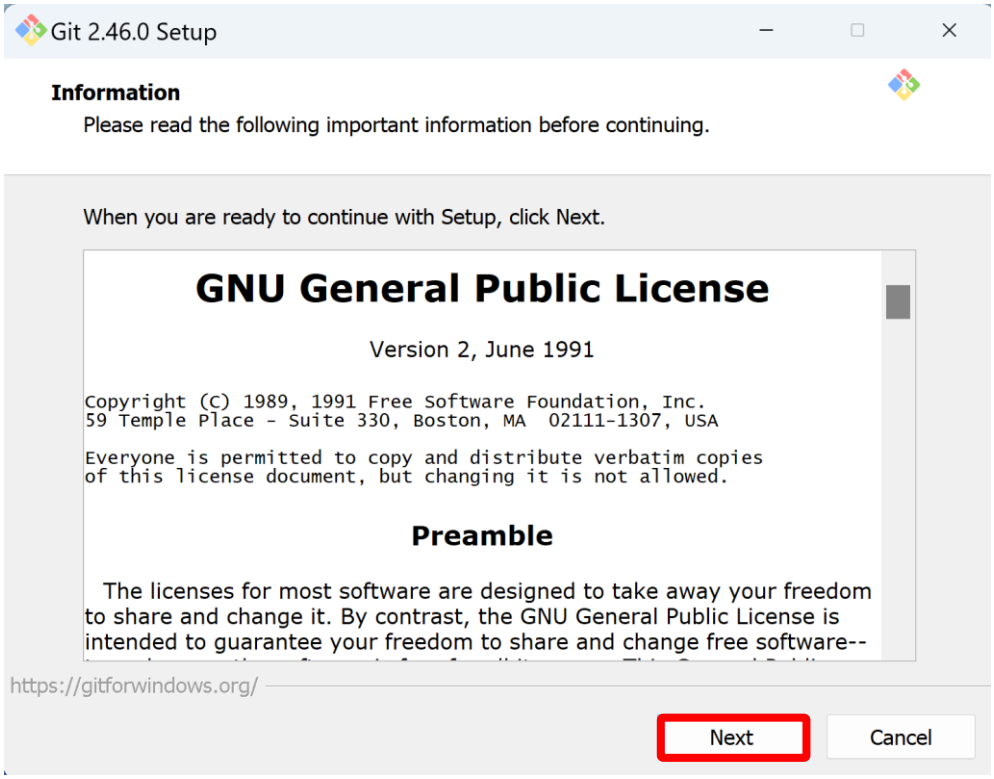


圖 1-9 Git 授權聲明

Step4 設定 Git 的安裝路徑，並點擊「Next」，如圖 1-10 所示。

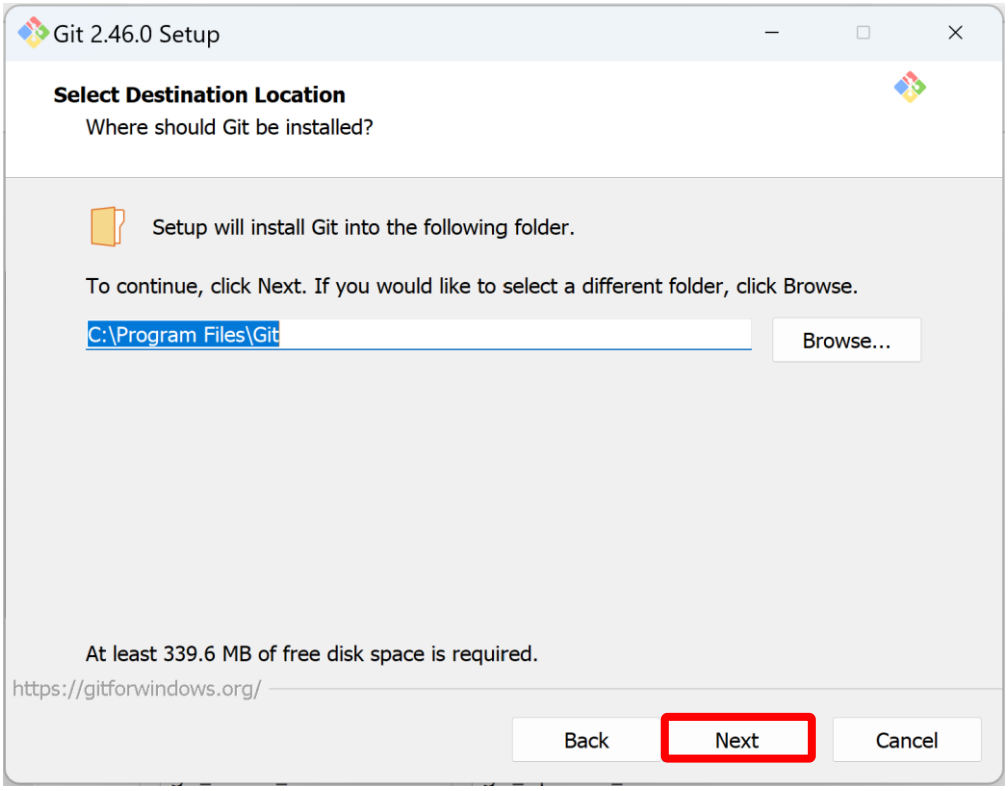


圖 1-10 選擇 Git 安裝路徑

Step5 勾選 On the Desktop，點擊「Next」，再點擊「Next」，如圖 1-11 所示。

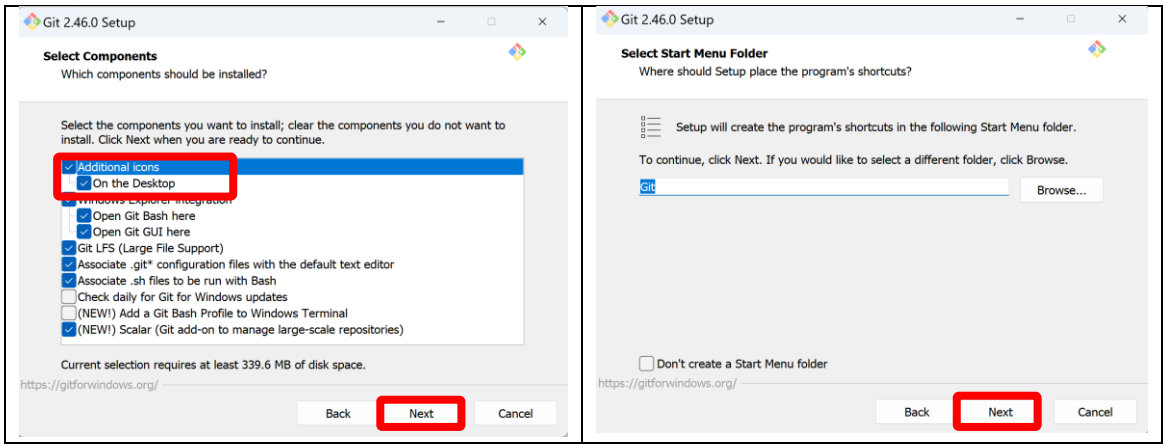


圖 1-11 勾選 On the Desktop（左）與設定開始目錄名稱（右）

Step6 設定 Git 編譯器，預設選擇 Use Vim（the ubiquitous text editor）as Git's default editor，點擊「Next」，如圖 1-12 所示。

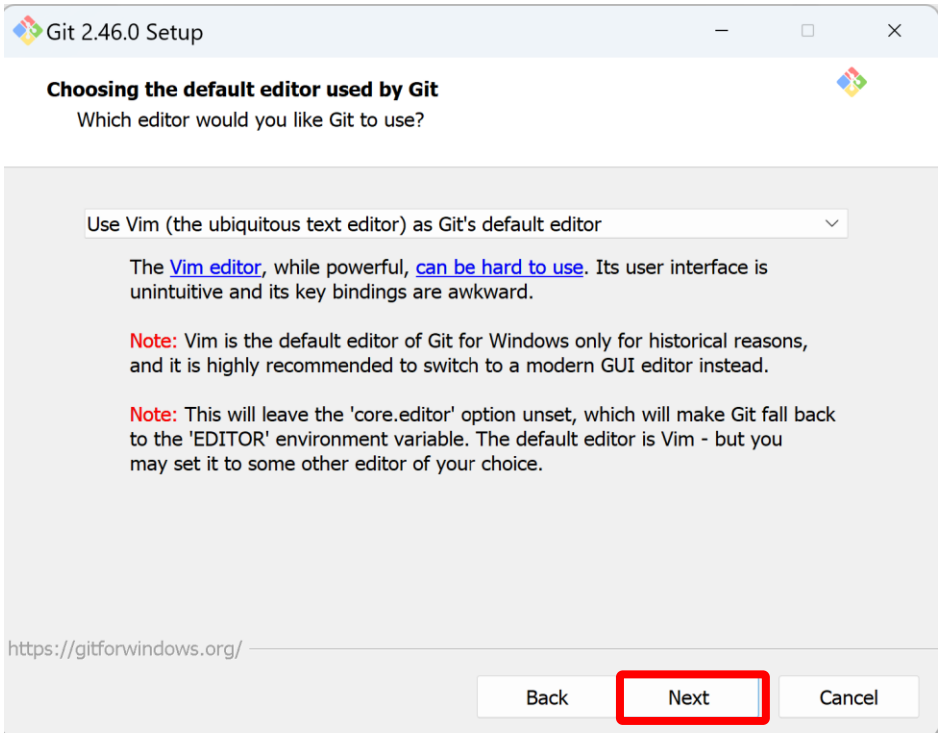


圖 1-12 選擇 Git 編輯器

Step7 設定預設分支名稱，選擇 Let Git decide 點擊「 Next」，如圖 1-13 所示。

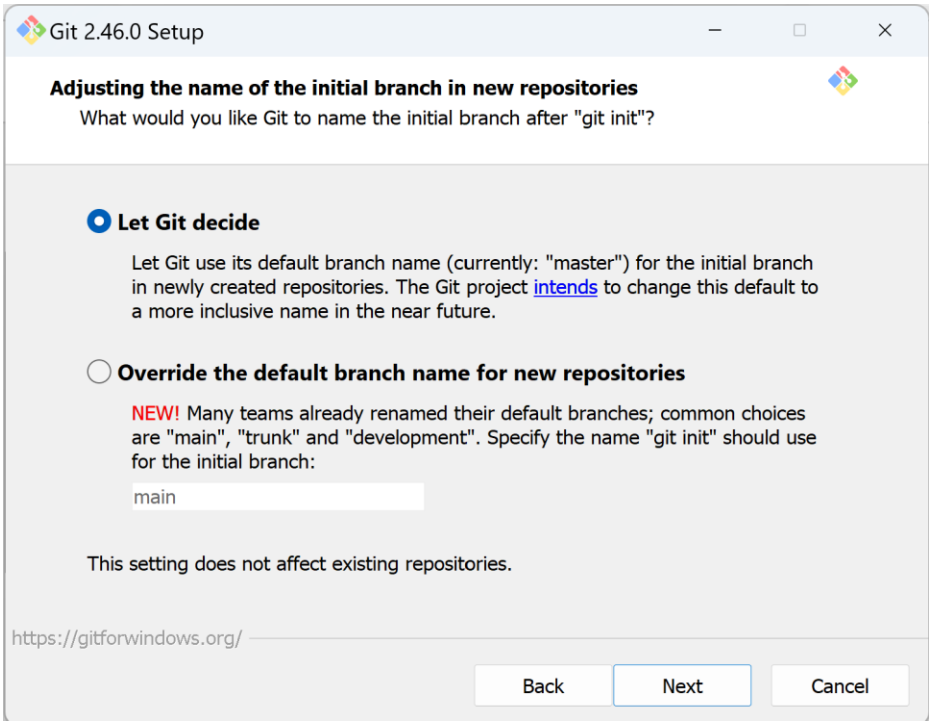


圖 1-13 設定預設分支名稱

Step8 選擇 Git from the command line and also from 3rd-party software，然後點擊「Next」，如圖 1-14 所示。

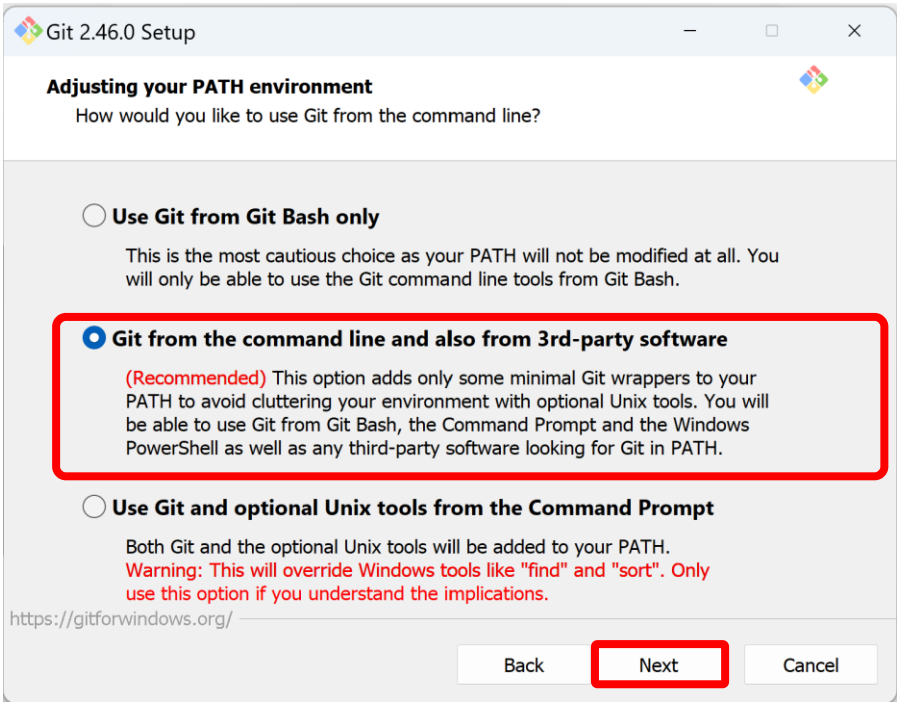


圖 1-14 設定 PATH 環境變數

Step9 選擇 Use bundled OpenSSH，點擊「Next」，如圖 1-15 所示。

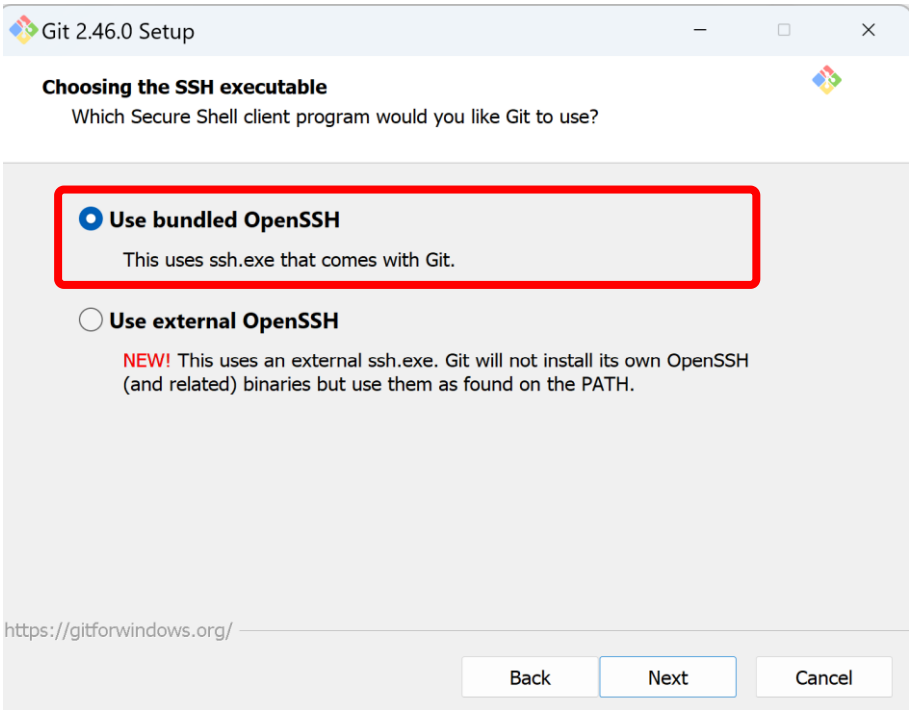


圖 1-15 設定 OpenSSH

Step10 選擇 Use the OpenSSL library，點擊「Next」，如圖 1-16 所示。

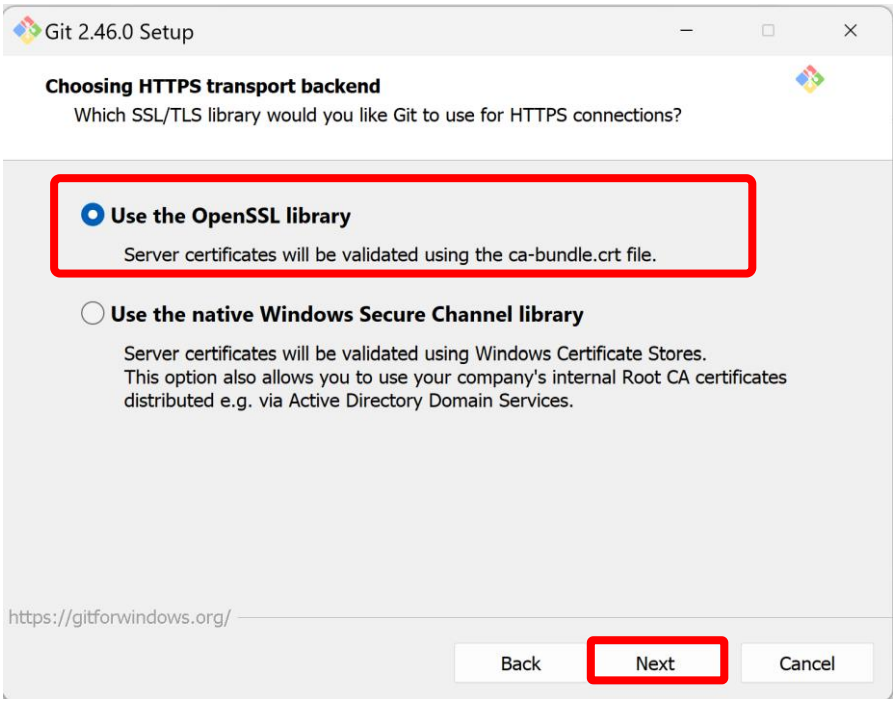


圖 1-16 選取 Use the OpenSSL library

Step11 設定檔案結束符號，預設選擇 Checkout Windows-style, commit Unix-style line endings，點擊「Next」，如圖 1-17 所示。

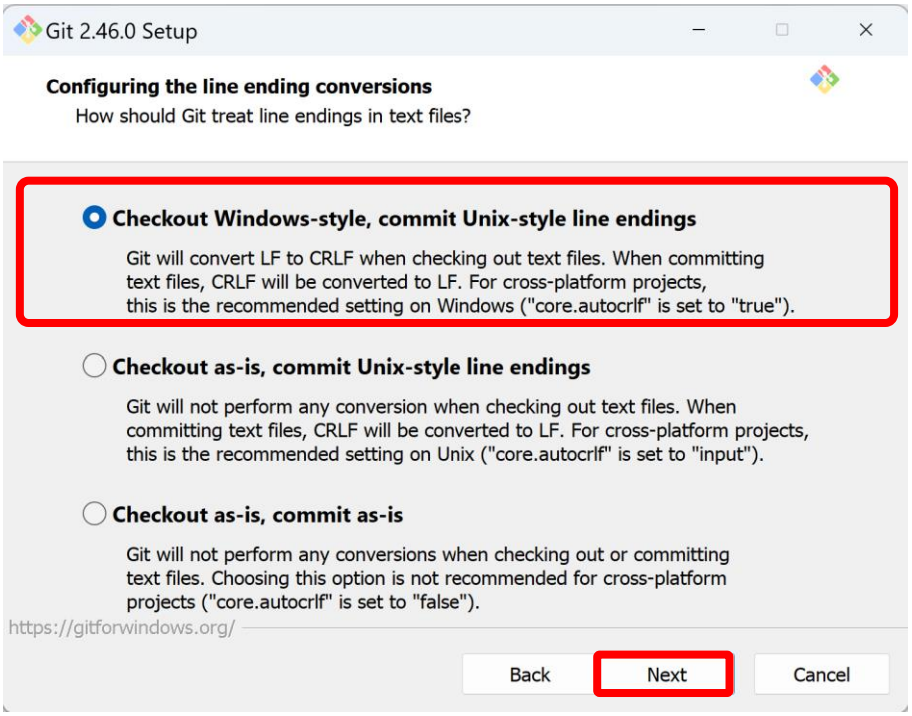


圖 1-17 設定結束符號

Step12 選擇 Use MinTTY，點擊「Next」，如圖 1-18 所示。

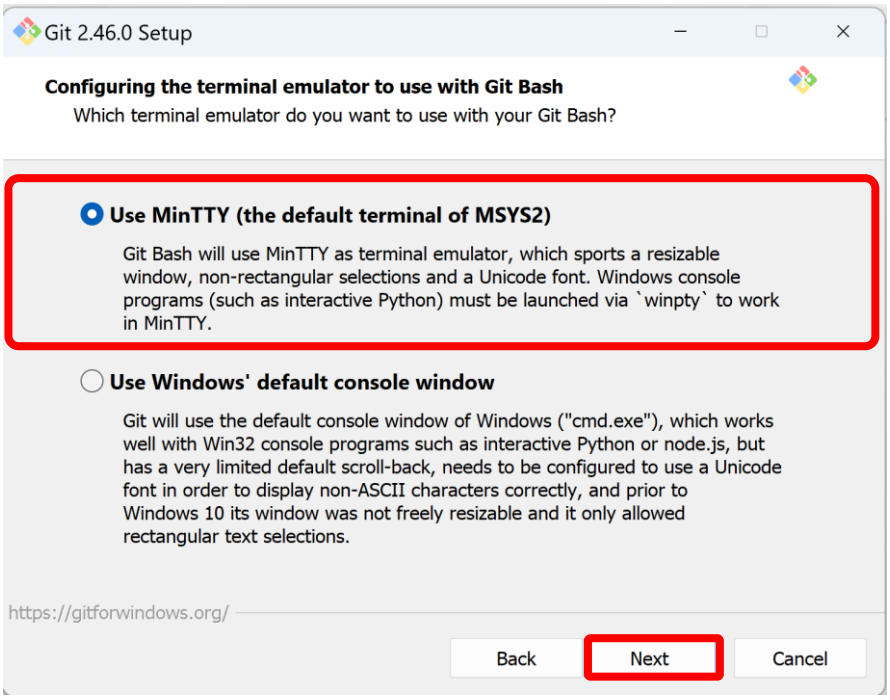


圖 1-18 選擇 Use MinTTY

Step13 選擇 Default (fast-forward or merge)，點擊「Next」，如圖 1-19 所示。

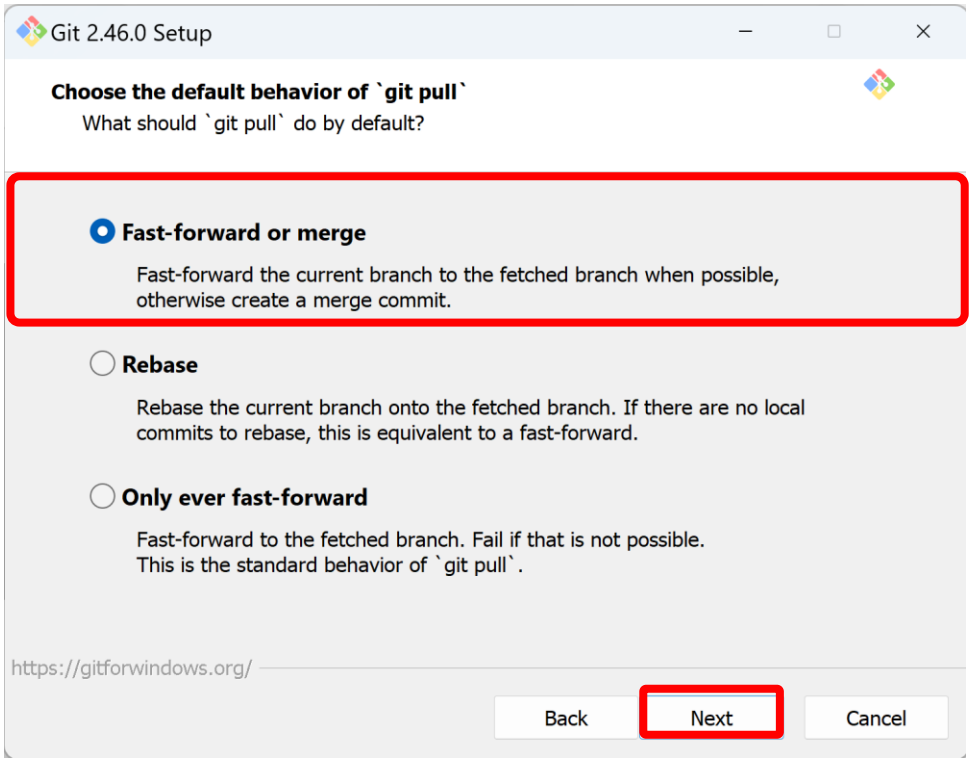


圖 1-19 選擇 Default (fast-forward or merge)

Step14 選擇 Git Credential Manager，點擊「Next」，如圖 1-20 所示。

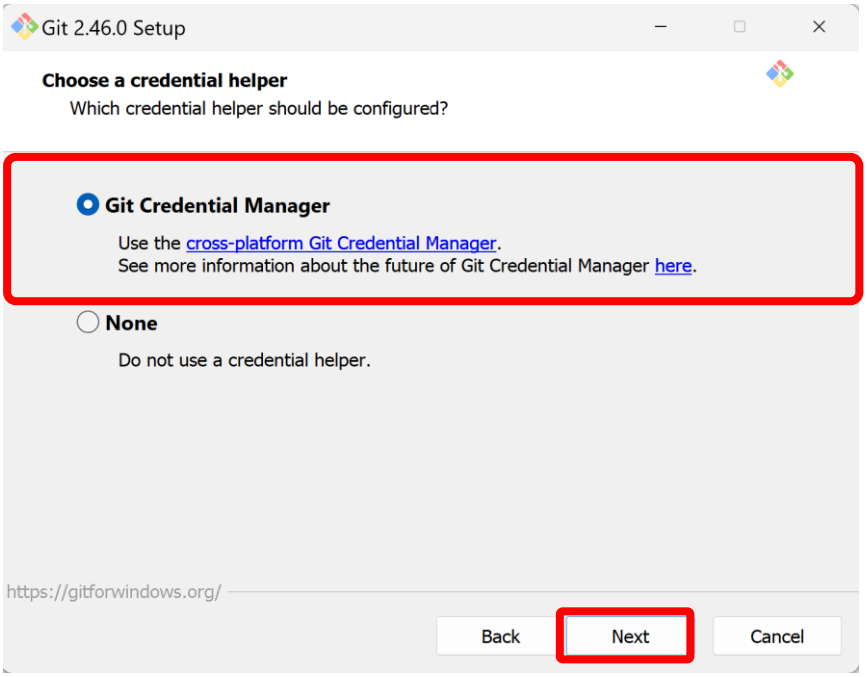


圖 1-20 選擇 Git Credential Manager

Step15 選擇 Enable file system caching 與 Enable symbolic links，點擊「Next」，如圖 1-21 所示。

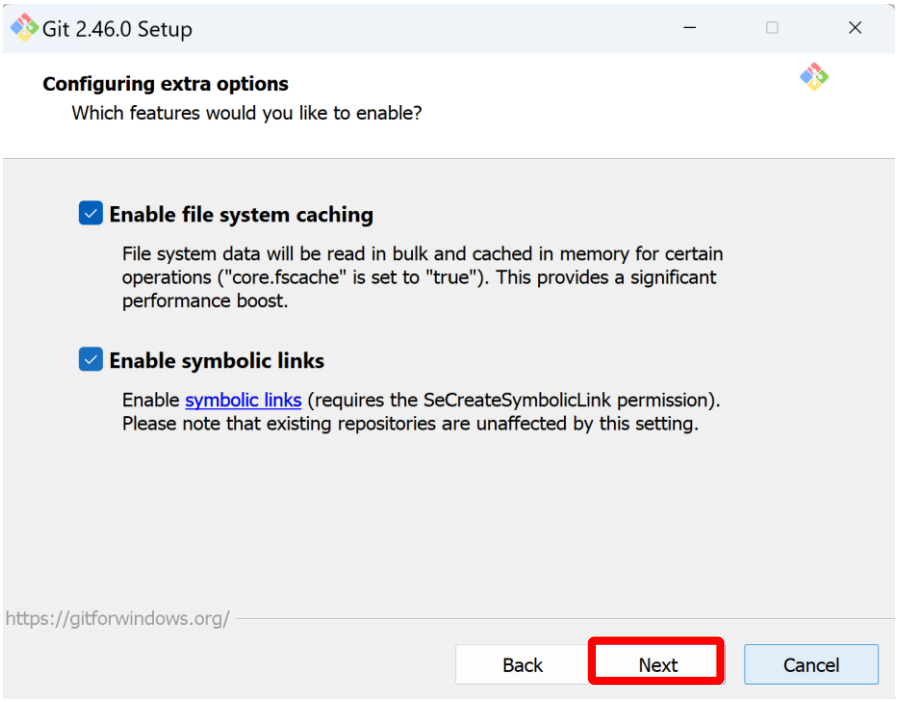


圖 1-21 選擇 Enable file system caching 與 Enable symbolic links

Step16 直接點擊「Install」，如圖 1-22 所示。

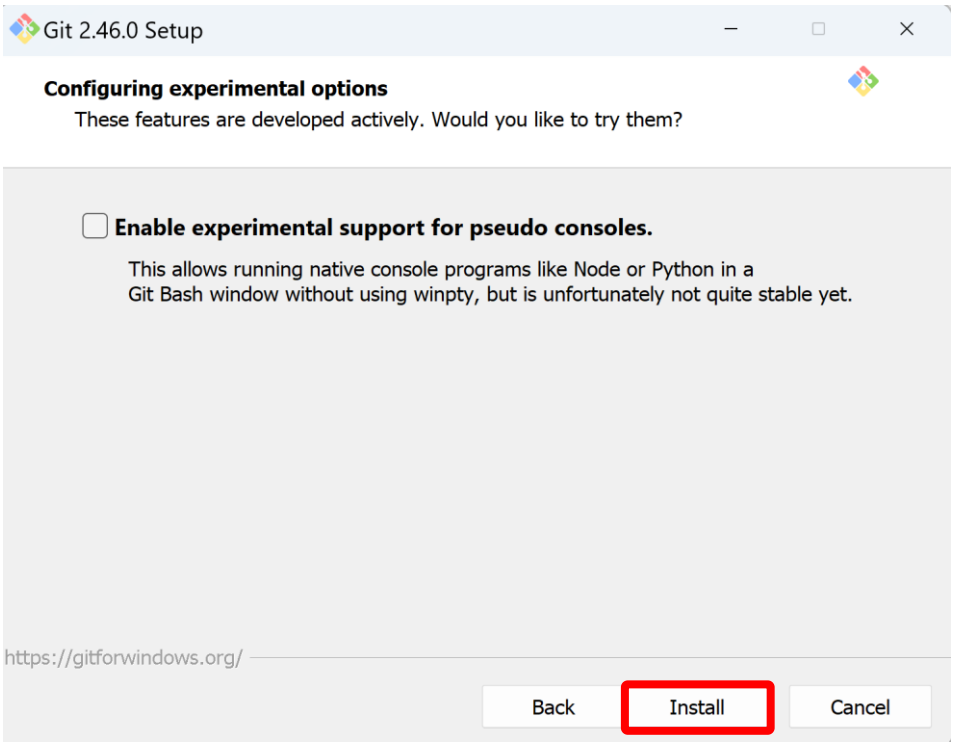


圖 1-22 開始安裝 Git

Step17 安裝完成後點擊「Finish」離開，如圖 1-23 所示。

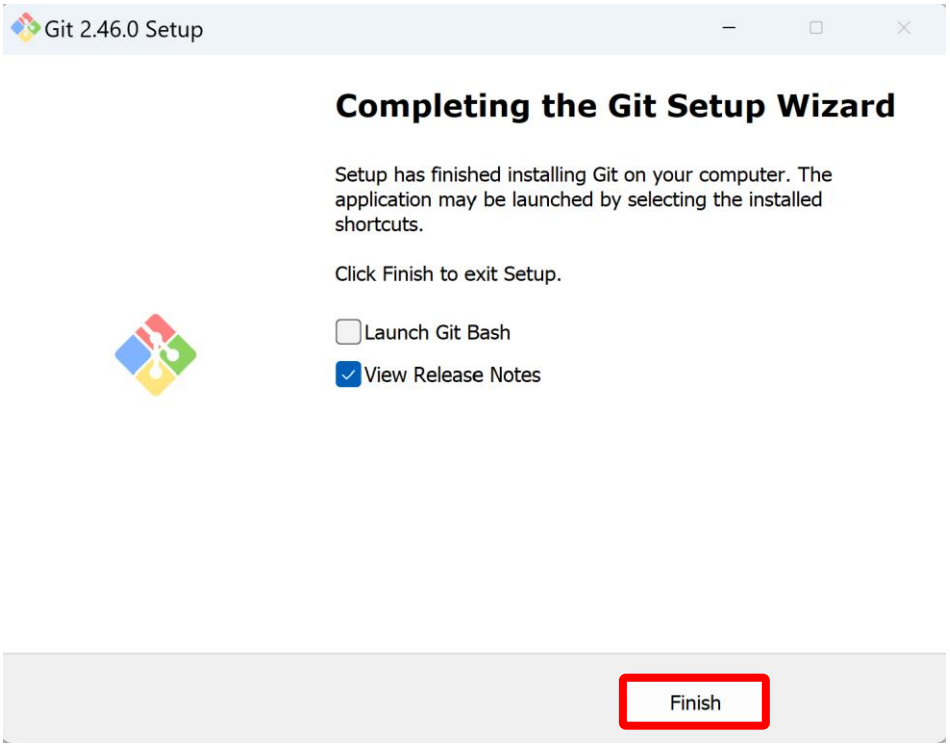


圖 1-23 安裝成功

Step18 到桌面點擊執行 Git Bash，畫面如圖 1-24 所示。

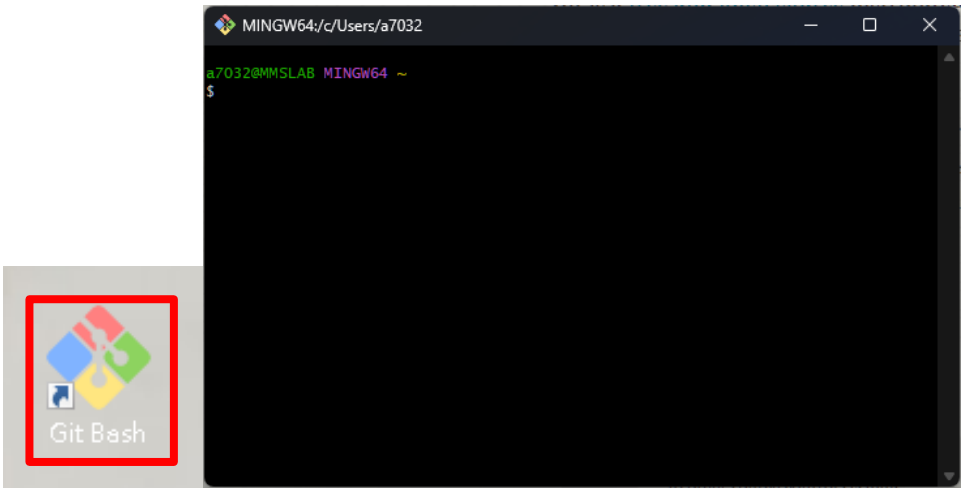


圖 1-24 開啟 Git Bash

Step19 在 git bash 內，設定自己的 user.email 和 user.name，如圖 1-25 所示。

```
$ git config --global user.email "xxx@gmail.com"

$ git config --global user.name "xxx"
```

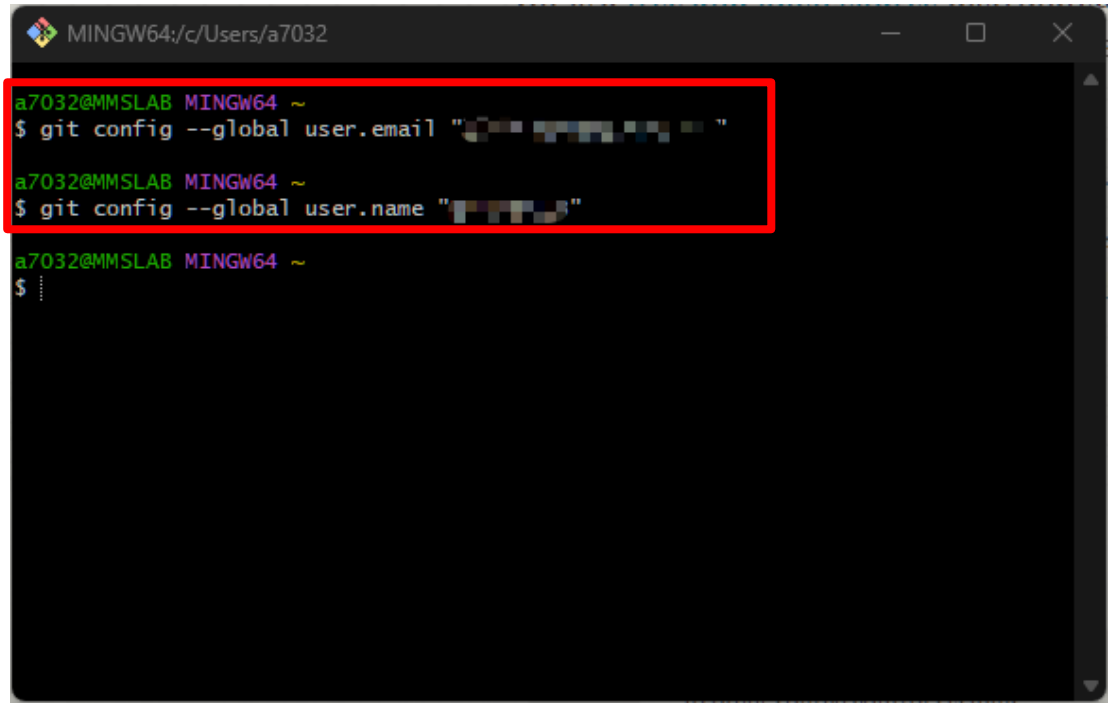


圖 1-25 設定使用者資料

0.2 Git 版本控制

在編輯檔案時，為了確保資料修改後能復原至更改前的狀態，有時會複製原檔案並以日期編號作為識別方式，如原始檔案為文檔.txt，經過這番操作後會變成下圖 2-1：

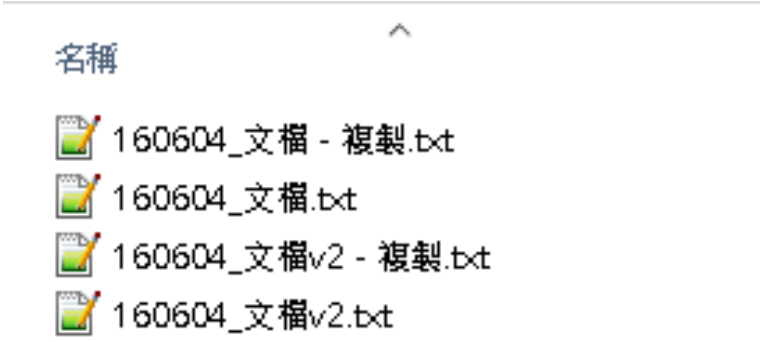


圖 2-1 雜亂的檔案管理

而要還原時才會意識到這種命名方式完全無益於找到需要回溯的版本，不僅麻煩，還無法識別修改的部分與內容。為了解決這個問題，我們需要控制版本的方案。如圖 2-2 所示，版本控制是系統開發的標準做法，透過系統化的管理備份資料，讓開發者可以從開始直到結案，完整的追蹤與記錄開發流程，甚至還能由此確保不同開發人員都能編輯同個專案進行操作，並同步更新彼此的進度。

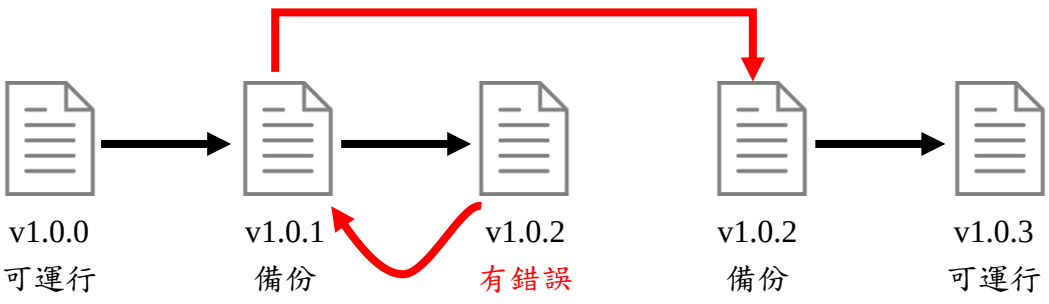


圖 2-2 透過版本控制保留檔案備份

0.2.1 Git

Git 為分散式的版本控制系統，可以把檔案每一次的狀態變更儲存為歷史紀錄，如圖 2-3所示。可以透過軟體把編輯過的檔案復原到指定的歷史紀錄，也可以顯示編輯前與編輯後的內容差異。個人開發中只需要使用 Git，就足以做到版本控制的目的，但是如果要與多人合作開發時，我們就會需要藉助到遠端資料庫來做管理。

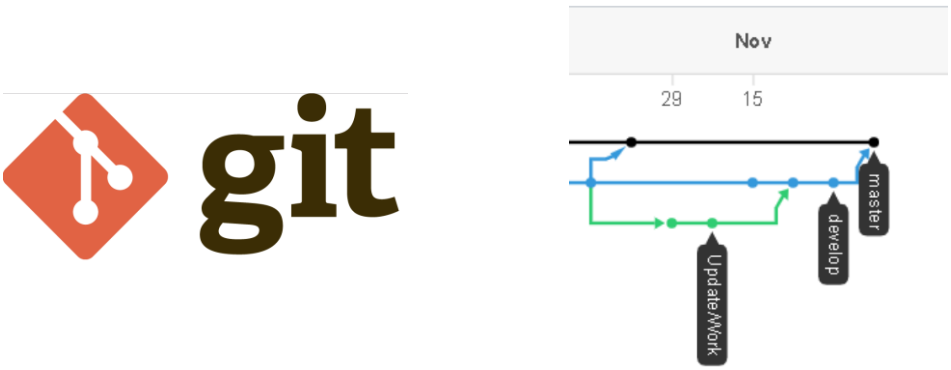


圖 2-3 Git（左）與版本控制（右）

0.2.2 GitHub

GitHub 是一個透過 Git 進行版本控制的遠端資料庫，用於軟體程式碼存放與共享的平台，由 GitHub 公司所開發，是目前世界上最大的程式碼存放平台。

如圖 2-4 所示，GitHub 最主要的功能是能將位於在電腦端經由 Git 操作後的歷史紀錄上傳至網路上，進行備份或者是分享，除了允許個人或是組織團體建立、存取資料庫外，也提供了圖形介面協助軟體開發，使用者可透過平台查看其他使用者的動態或是程式碼，也可以對其提出意見與評價。

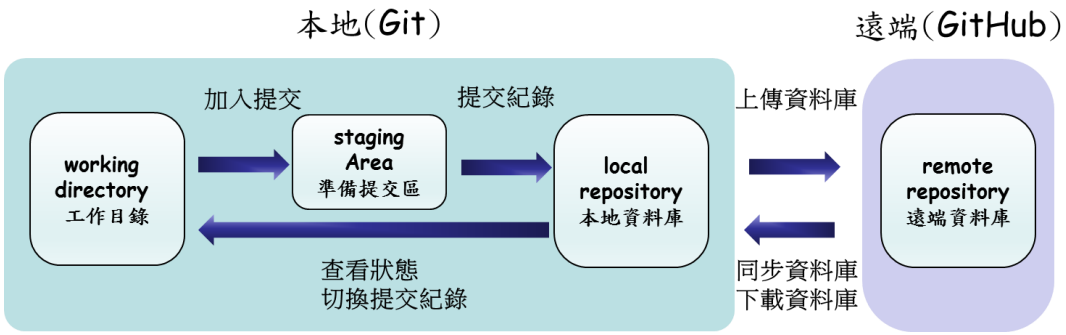


圖 2-4 使用 Git 備份資料到 GitHub

- **working directory（工作目錄）**：工作目錄主要是存放需要控制版本的檔案 資料夾。經由在資料夾內建立 git 的資料庫，此資料夾就能變成 git 的工作目錄。

- **staging area（準備提交區）**：準備提交區用於紀錄將要被提交的資料。若在工作目錄下的檔案有變動，且我們希望能提交這些更新過的資料，就將這些資料存入準備提交區，此狀態下只標記要被提交的資料，尚未將資料提交出去。

- **local repository（本地資料庫）**：即為自己電腦端上的資料庫。當確定好所有要提交的資料都加入到準備提交區之後，可以將準備提交區的資料做提交紀錄，提交後的資料會被紀錄成一個提交紀錄保存於資料庫中。

- **remote repository（遠端資料庫）**：即為遠端伺服器上的資料庫。當本地資料備齊後，我們可以透過上傳將資料保存到遠端資料庫，也可以反過來將遠端資料庫的紀錄同步下來。

1) 建立本地資料庫

要建立本地資料庫，首先要先選擇我們的工作目錄，然後在該工作目錄下使用 git init 指令來產生資料庫。

```
$ git init
```

指令執行後，目錄下會產生一個「.git」的目錄，即為本地資料庫，如圖 2-5 所示。若使用 git 的控制台查看此目錄可以看到圖 2-6 的路徑後增加了一個[master]的標籤，表示有偵測到資料庫，如圖 2-6 所示。

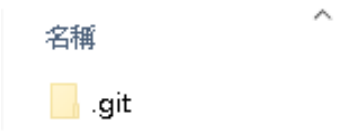


圖 2-5 目錄中的.git 目錄

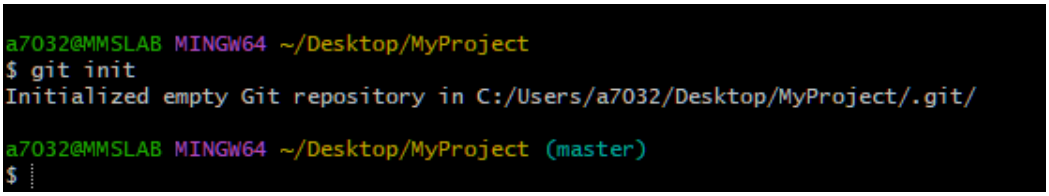


圖 2-6 [master]標籤

2) 查看狀態

建立好資料庫後，當工作目錄有更動，例如增加檔案，或是原本既有的檔案有更動，可以使用 git status 來查看更動過的資料，如圖 2-7 所示。

```
$ git status
```

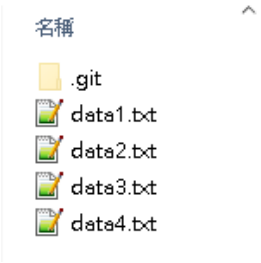


圖 2-7 工作目錄

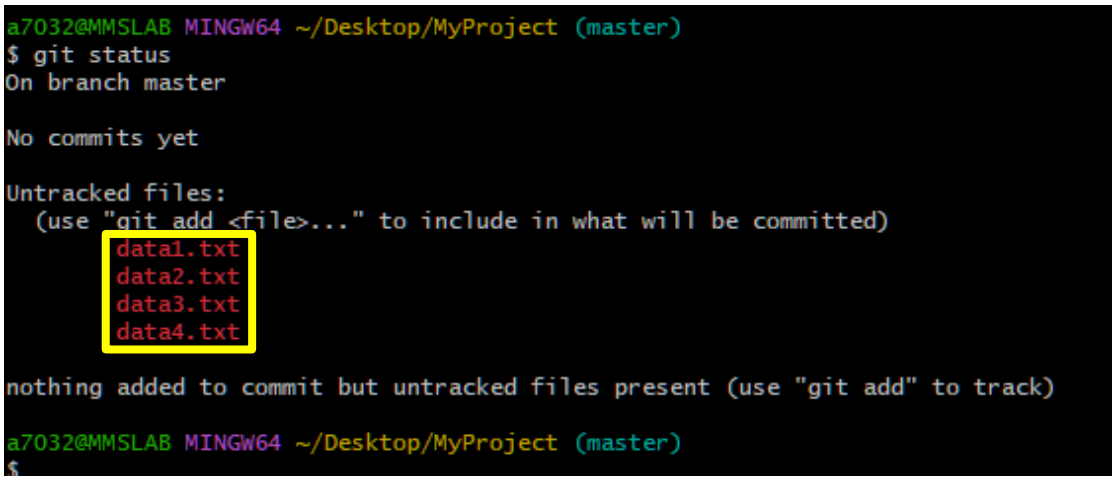


圖 2-8 查看目錄中更動過的檔案

在目錄中增加了 4 個檔案後執行指令，能發現控制台中以紅色字列出有更動過的檔案，如圖 2-8 所示。

3) 加入提交

在紀錄檔案版本之前，我們需要先將異動的資料放入準備提交區。這邊我們使用 git add 來將檔案加入到準備提交區，也可以用 git add .加入所有檔案。

```
$ git add 檔案名稱  或  git add .
```

圖 2-9 加入檔案變更提交

可以使用 git status 的指令觀察變化，可以發現原本的紅色標籤變為綠色，這表示先前的這些檔案已經有被加入到準備提交區，如圖 2-9 所示。

4)　登入

在提交紀錄之前，需進行設定 user.eamil 及 user.name，如圖 2-10 所示。

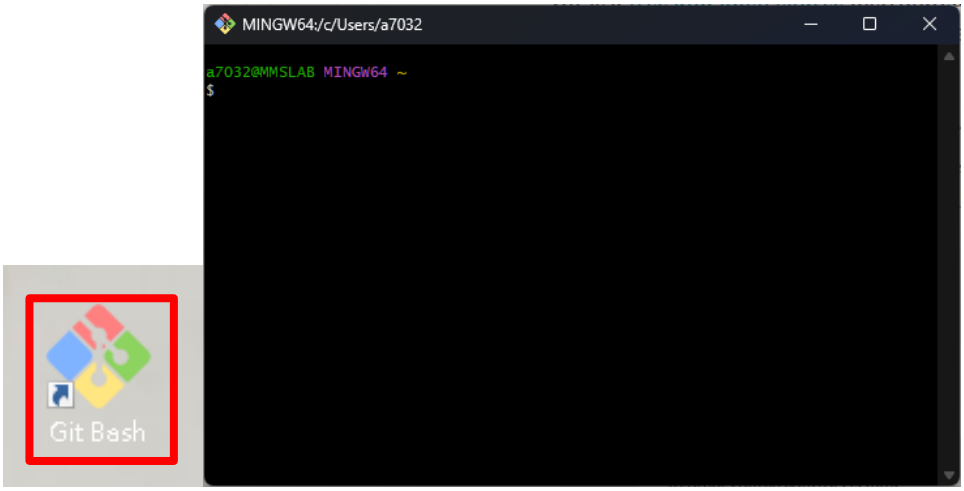


圖 2-10 開啟 Git Bash

在 git bash 內，設定自己的 user.email 和 user.name，如圖 2-11 所示。

```
$ git config --global user.email "xxx@gmail.com"

$ git config --global user.name "xxx"
```

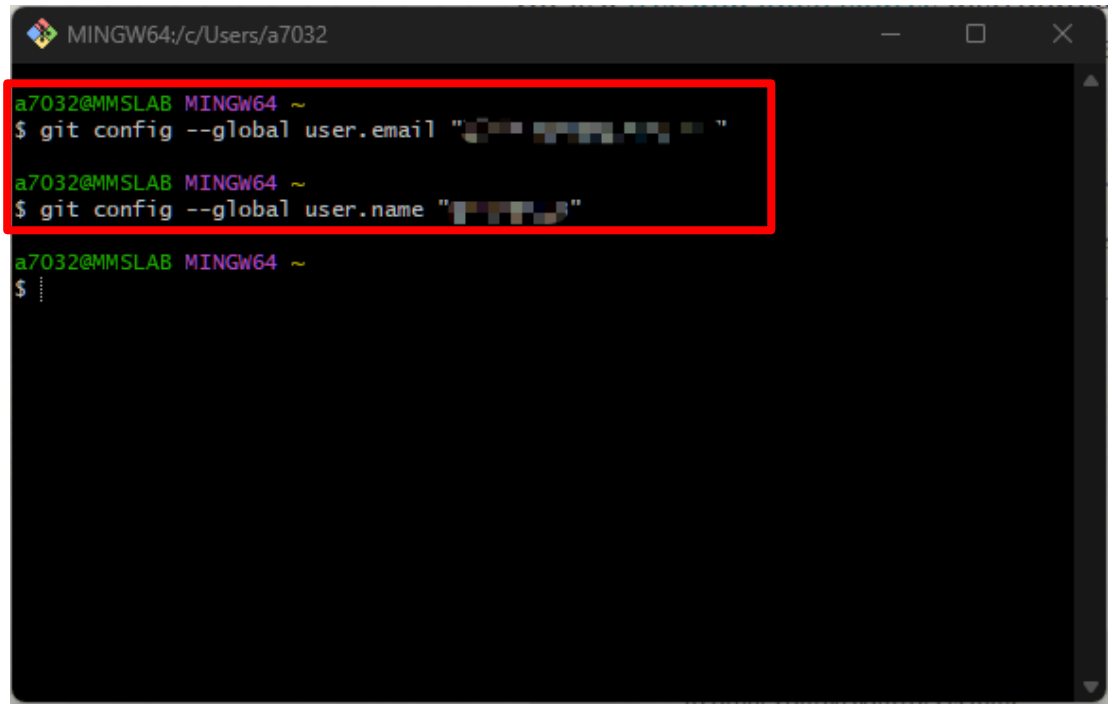


圖 2-11 設定使用者資料

5)　提交紀錄

若想把準備提交區的檔案儲存到資料庫中，需要執行提交（Commit）。這邊我們使用 git commit 將準備提交區的資料作提交。

執行提交的時候，需要附加提交訊息，提交訊息類似為該提交紀錄加上一般人能理解的標籤說明，如圖 2-12 所示，加上「這紀錄修改了 OO」的訊息來讓使用者辨識。如果沒有輸入提交訊息就直接執行提交，結果將會失敗的。

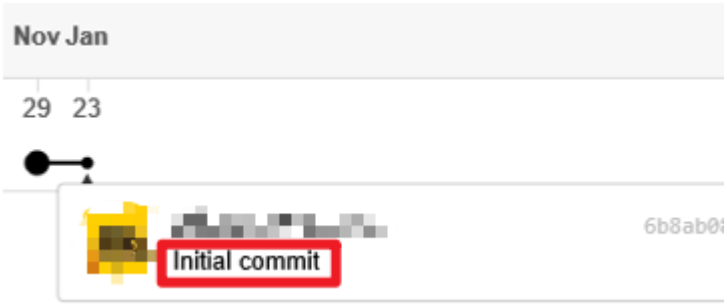


圖 2-12 GitHub 上的提交訊息

在 git commit 後面加上-m 的語法可以輸入說明文字。

```
$ git commit -m "說明文字"
```

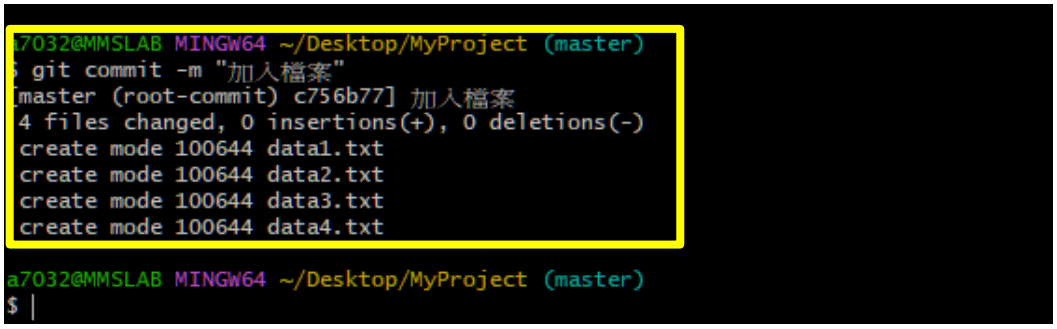


圖 2-13 加入提交紀錄

這步驟會將我們修改的內容做紀錄保存，如圖 2-13 所示。這時工作目錄與本地資料庫已經同步了。

6) 建立遠端資料庫

這邊事先要先註冊好 GitHub 帳號，上傳遠端資料庫之前，如果遠端沒有資料庫，就需要建立一個資料庫，如圖 2-14、2-15 所示。

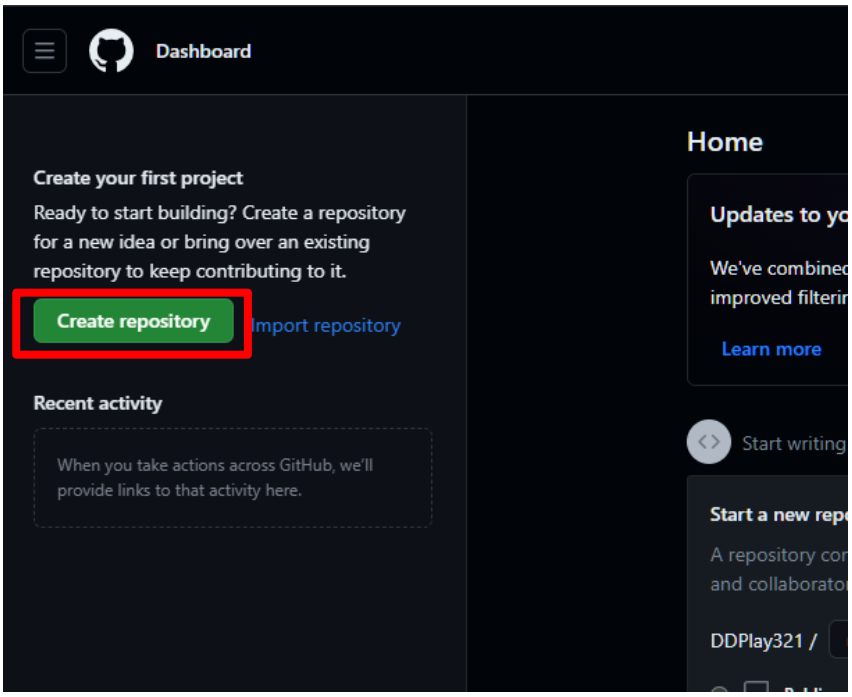


圖 2-14 點選 Start a project 建立新專案

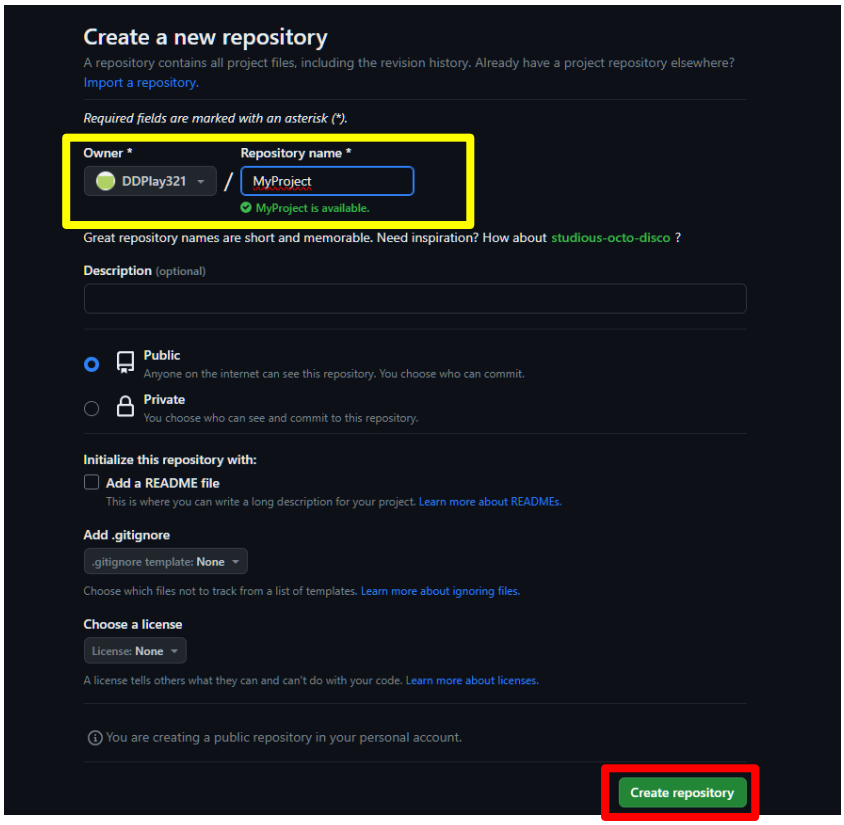


圖 2-15 輸入專案名稱並按下「Create repository」

建立資料庫後，如圖 2-16 所示，會產生一個連結，之後上傳資料庫時會需要。

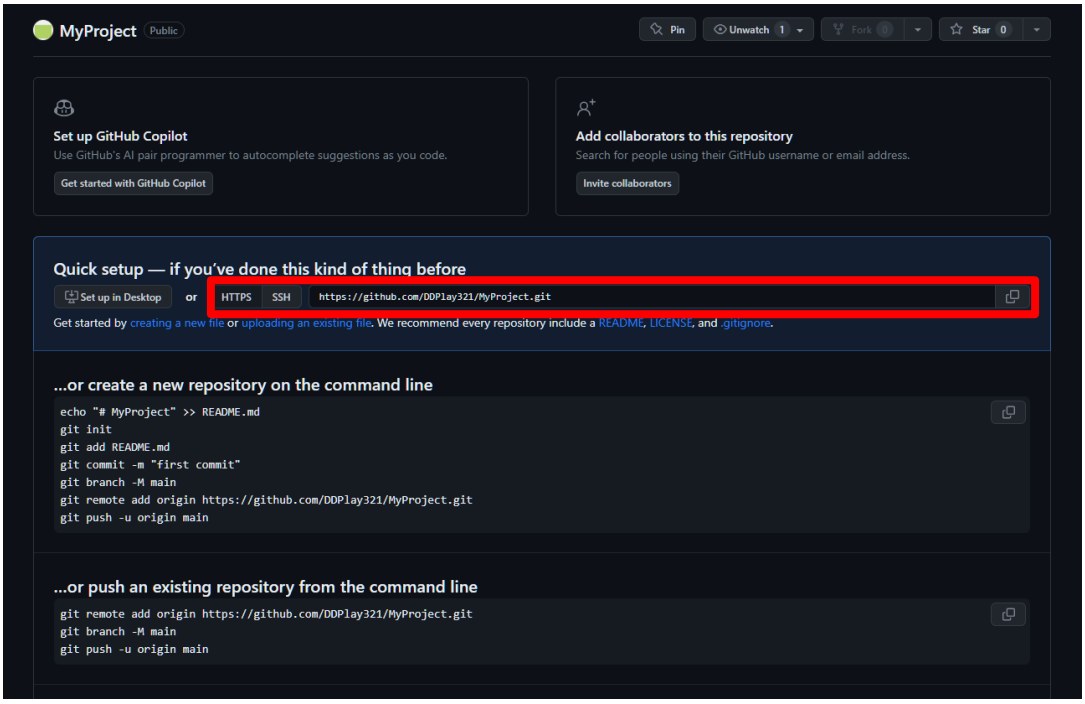


圖 2-16 GitHub 資料庫鏈結

7) 上傳到遠端資料庫

當我們需要與他人共享本地資料或是遠端備份時，我們就需要上傳資料庫。使用 git remote 指令連結遠端資料庫，這邊需要加上資料庫的連結，如圖 2-17 所示。

```
$ git remote add origin 資料庫連結
```

圖 2-17 連結遠端資料庫

然後，下 git push 指令，就可以將資料同步至遠端，如圖 2-18 所示。

```
$ git push origin 標籤名稱
```

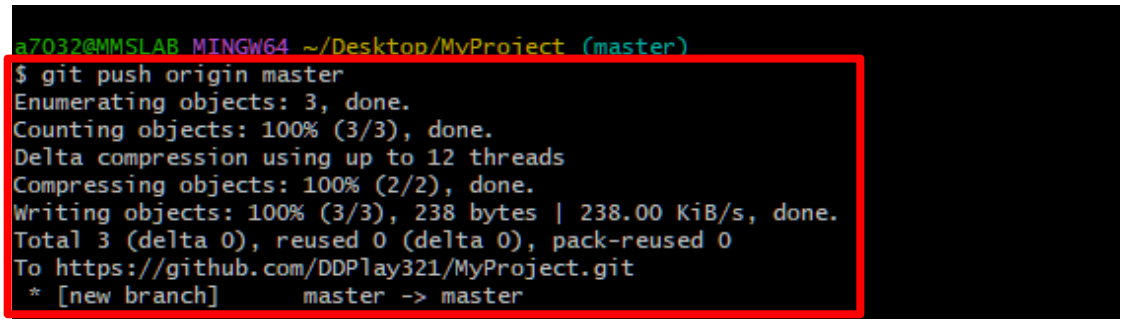


圖 2-18 同步資料到遠端資料庫

8) 同步遠端資料庫

在多人開發時，如果他人更新了 GitHub 上的專案，就會使本地與遠端的資料不同步，這時我們就需要將 GitHub 上的資料同步下來時，如果本地端已經有相對應的資料庫時，可以用 pull 同步資料庫到工作目錄，如圖 2-19 所示。

```
$ git pull origin 標籤名稱
```

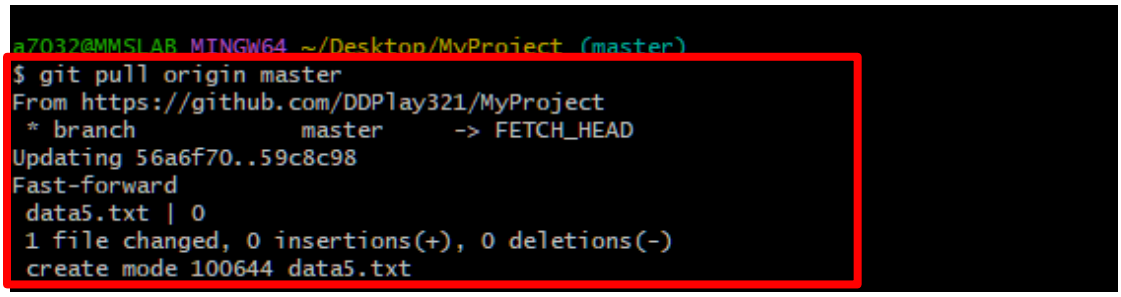


圖 2-19 從遠端資料庫同步資料到本地

9) 下載遠端資料庫

有時候，想要同步 GitHub 上的資料庫，但是本地端並沒有對應的資料庫，例如：希望使用他人開發的 GitHub 的專案，就無法透過同步的方式，而是要直接將遠端的資料庫複製到本地端，這時我們就可以用 Clone 複製資料庫下來，如圖 2-20 所示。

```
$ git clone 資料庫連結
```

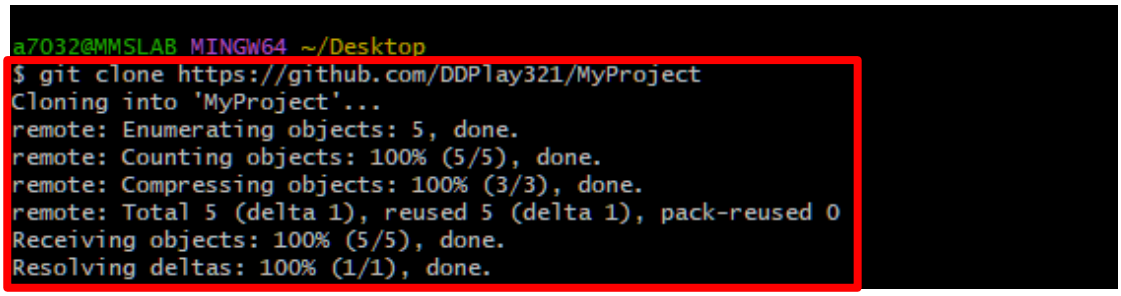


圖 2-20 複製遠端資料

10) 查看本地資料庫

任何時間點，我們都可以查看之前的本地資料庫中所有的提交紀錄，這邊我們使用 git 的圖形介面作查看。

要啟動圖形介面，我們需要使用 gitk 指令。

```
$ gitk
```

下達指令後便會出現圖 2-21 的圖形介面。

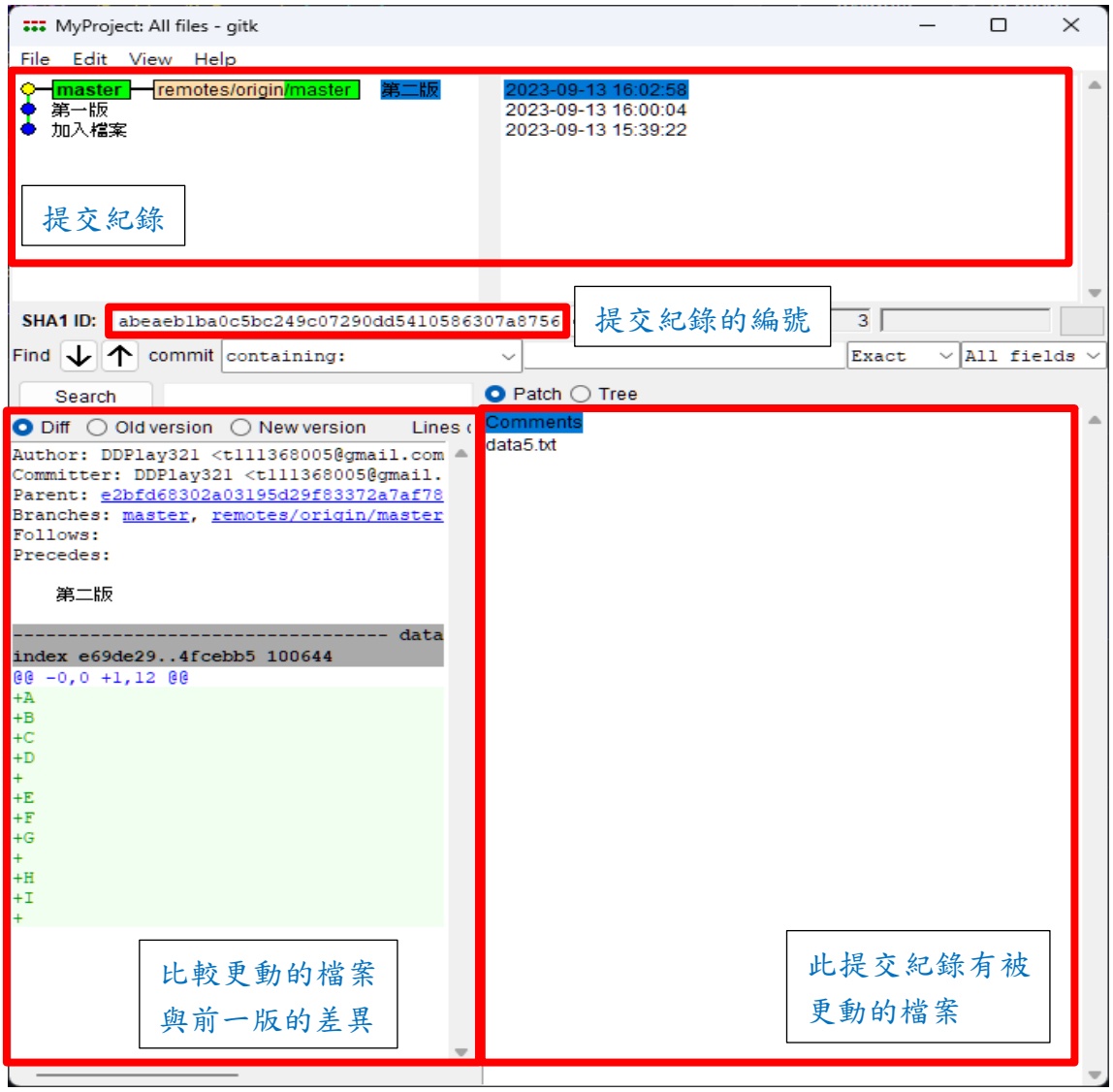


圖 2-21 Git 圖形化介面

版本紀錄是以時間先後順序來做儲存，於介面上方可以看到由下而上生長的樹狀圖，每次提交都會產生出新的節點，每一個節點都代表一次的版本紀錄。

為了區分每一個提交紀錄，系統會自動產生一組相對應的識別碼來給紀錄命名，識別碼會以不重複的 40 位英文數字做表示，如圖 2-22 所示。只要指定識別碼就可以在資料庫中找到相對應的提交紀錄。

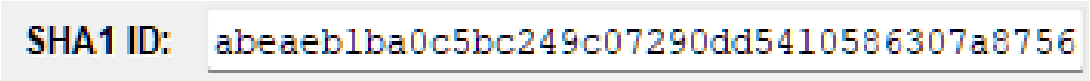


圖 2-22 SHA1 識別碼

執行提交之後，資料庫裡可以比較上次提交的紀錄與現在紀錄的差異。右下角會顯示此提交紀錄中有被更動的檔案，左下角會顯示比較該提交紀錄所更動的檔案與前一版的差異，呈現方式黑色是未更動的內容，紅色是移除掉的內容，綠色是新增的內容。

11) 切換提交紀錄

在前面的流程中，當發現目前版本出現不可修復的錯誤，或是希望能回到之前的版本，可以使用 git checkout 移動到指定的提交紀錄，在前面，我們知道每一個提交紀錄都有唯一識別碼，如圖 2-23 所示，我們可以透過識別碼移動到相對應的提交紀錄。

```
$ git checkout 識別碼
```

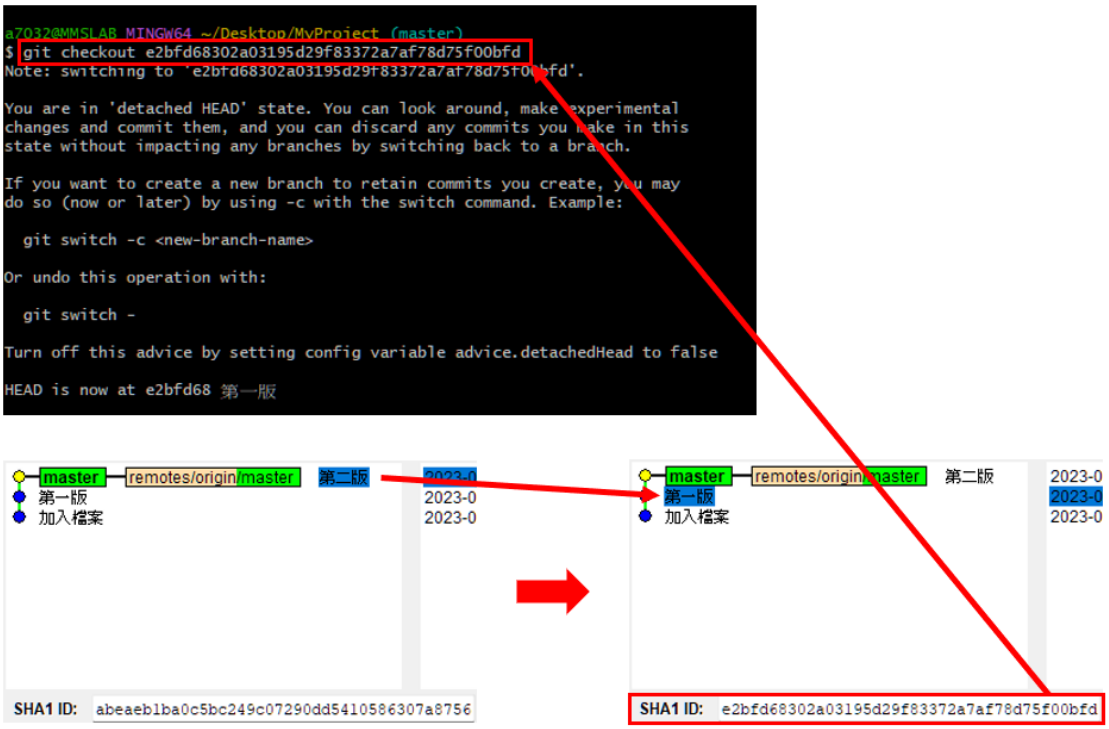


圖 2-23 切換提交紀錄

實作後，可以發現黃色的點從[master]移動到了「第一版」，控制台的標籤也變成了標籤起頭的亂碼，表示成功回到前面的提交點，工作目錄的資料也會自動回復到該提交紀錄的版本。如果有新版本，就可以從此提交紀錄繼續提交新的紀錄，進而產生新的版本分支，延續專案的開發。

0.3 GitHub 實戰演練

- 安裝 Git 使用環境 Git Bash
- 註冊 GitHub 帳號與建立一個遠端資料庫
- 將撰寫完成的專案推送到 GitHub 上

◆ 註冊 GitHub 帳號

Step1 至 GitHub 官方網站 https://github.com/註冊帳號，如圖 3-1 所示。

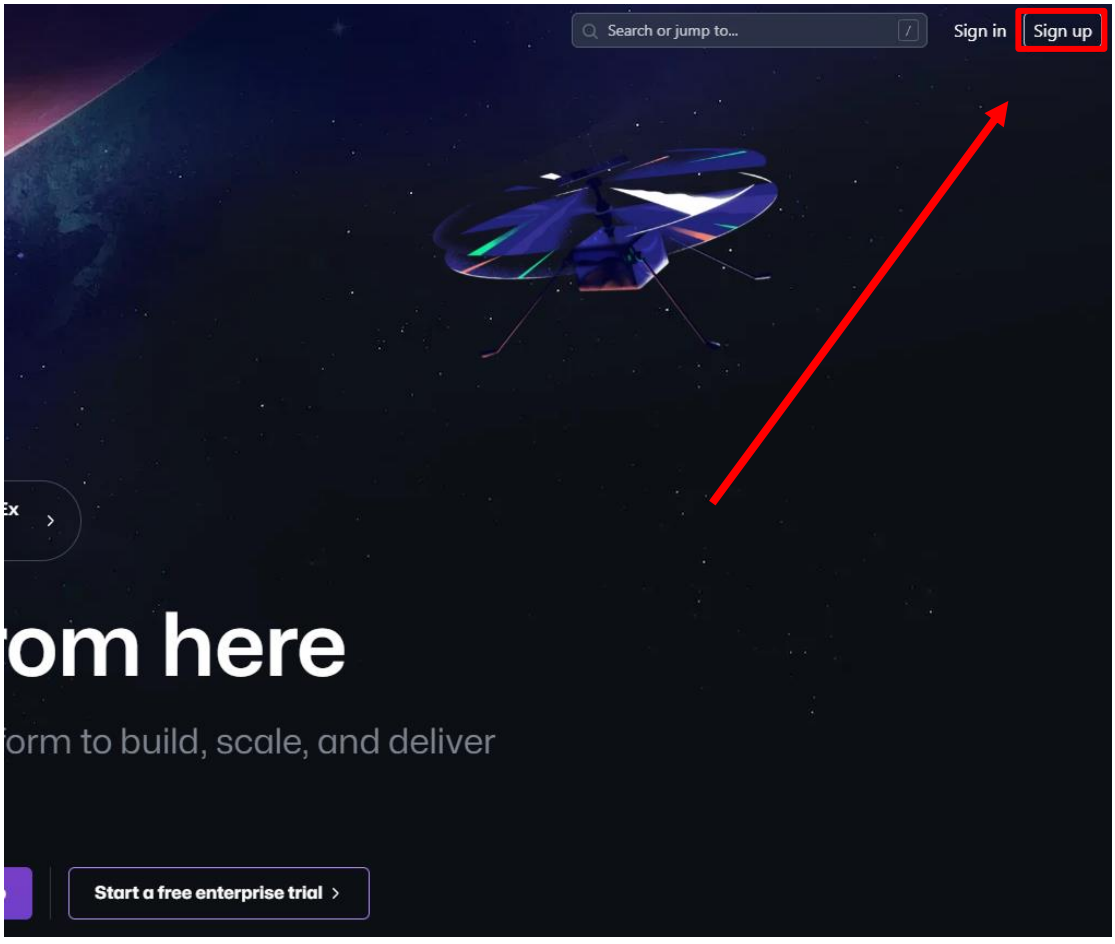


圖 3-1 GitHub 官網

Step2 填寫註冊資料並驗證，如圖 3-2 所示。

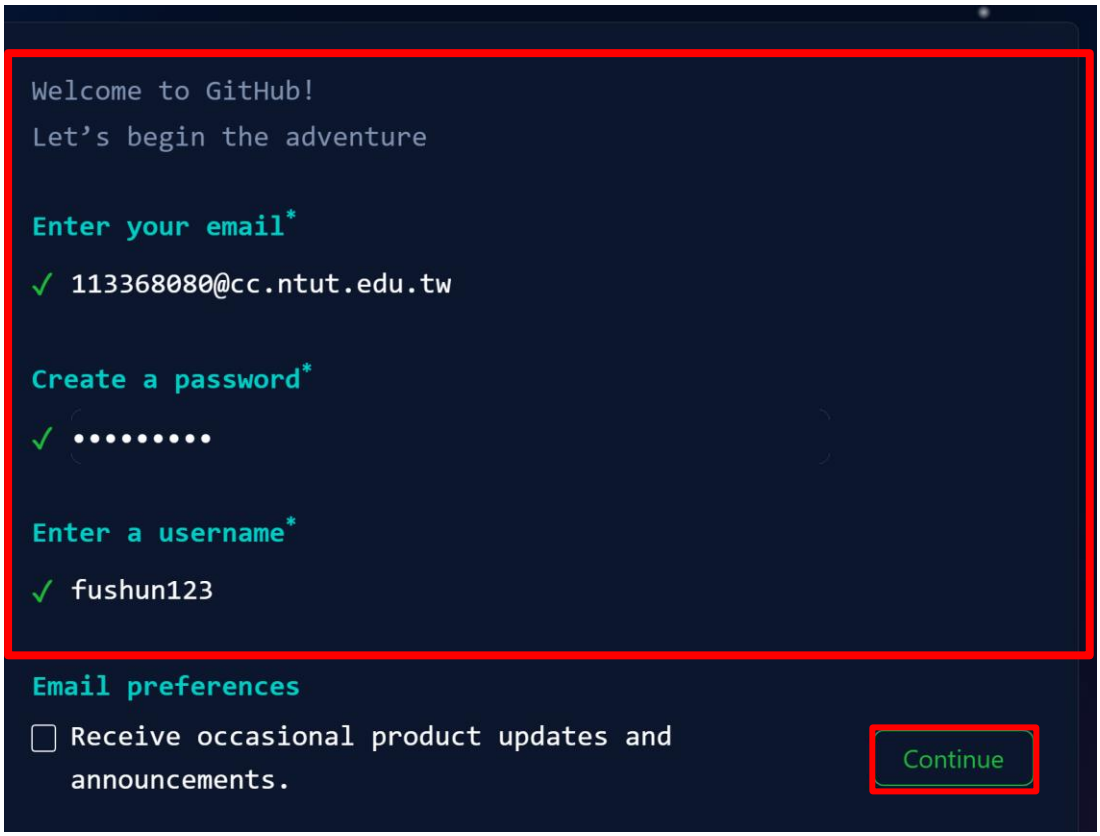


圖 3-2 註冊基本資料

Step3 選擇 GitHub 註冊帳號的方案，此處保持 Free 的方案，然後直接點擊「Continue for Free」即可完成註冊，如圖 3-3 所示。

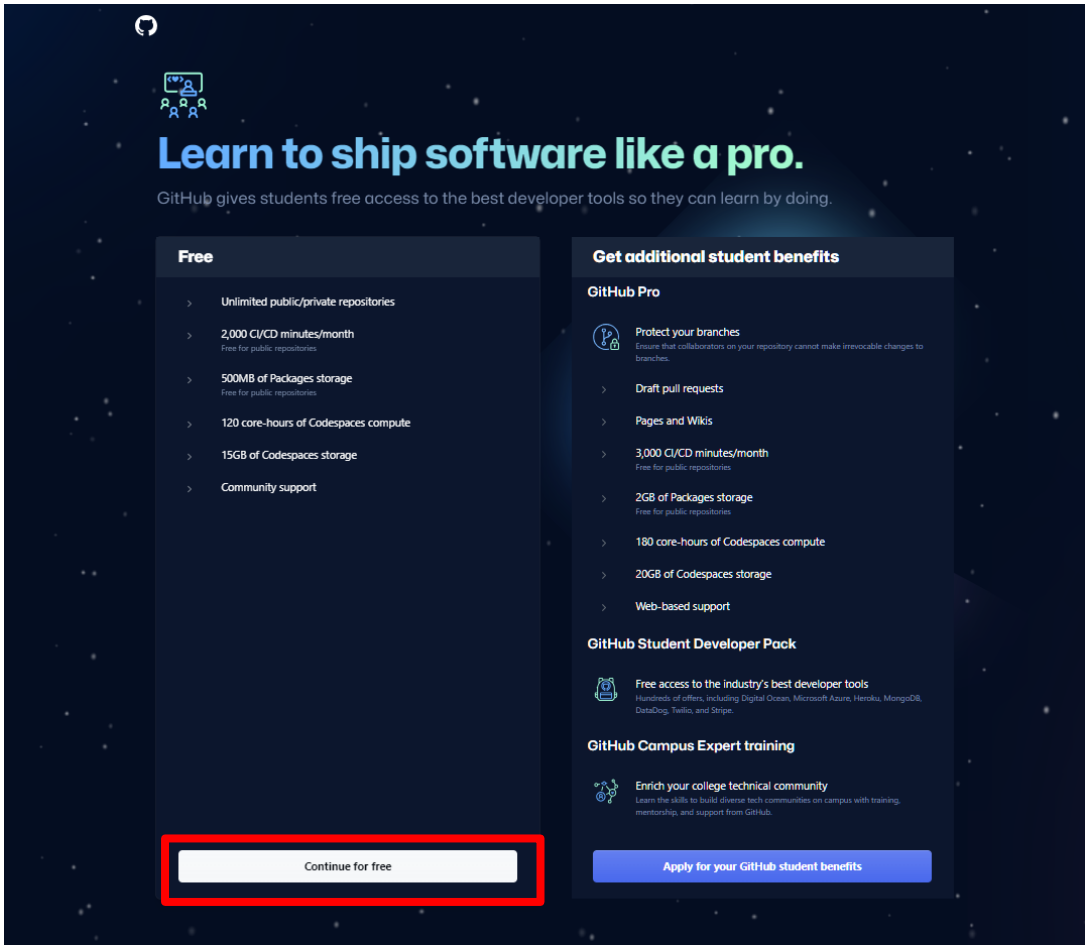


圖 3-3 選擇免費方案

Step4 完成後會看到此網頁，如圖 3-4 所示。

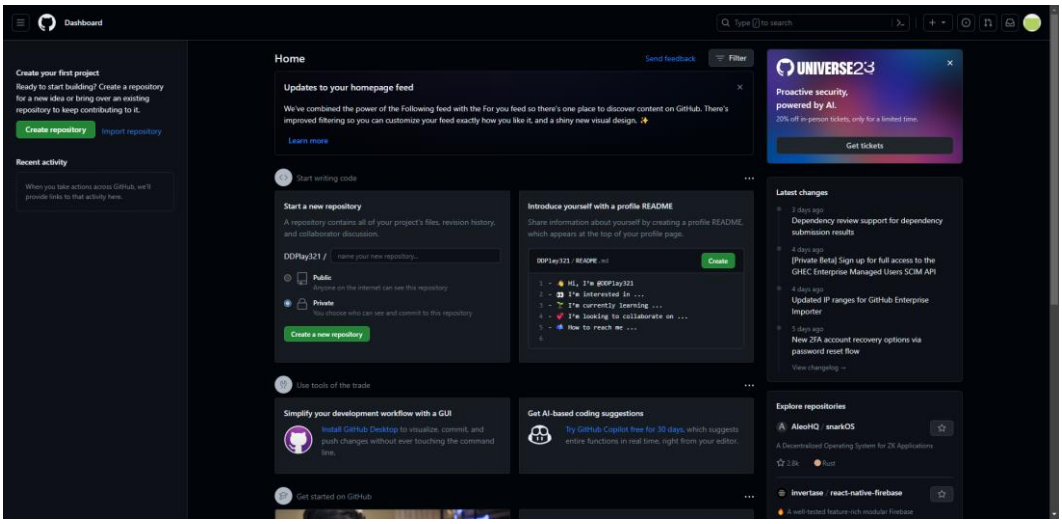


圖 3-4

0.3.1 安裝 Git 使用環境 Git Bash

Step1 至 https://git-scm.com/download/win 下載 Git 安裝檔，如圖 3-5 所示。

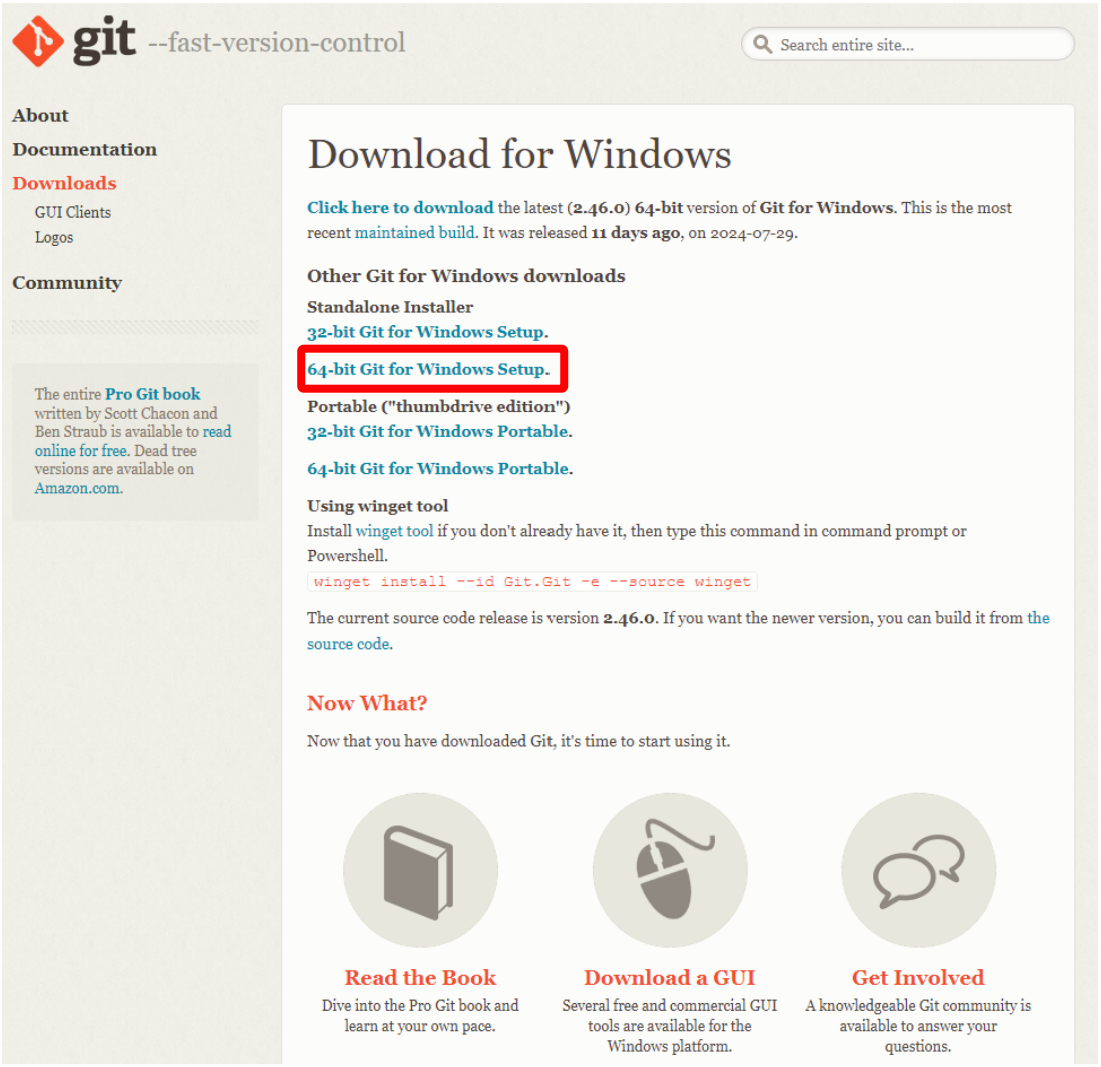


圖 3-5 下載 Git 安裝檔

Step2 開啟安裝檔，開始安裝 Git，如圖 3-6 所示。

圖 3-6 允許安裝 Git

Step3 閱讀並同意授權聲明，點擊「Next」，如圖 3-7 所示。

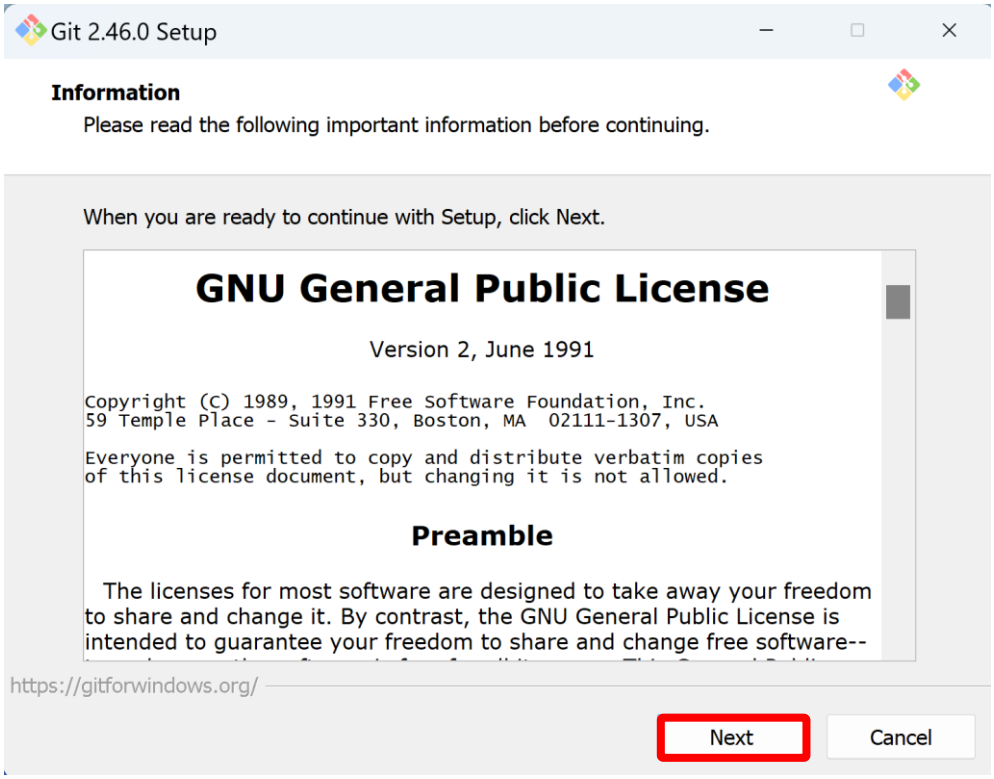


圖 3-7 Git 授權聲明

Step4 設定 Git 的安裝路徑，並點擊「Next」，如圖 3-8 所示。

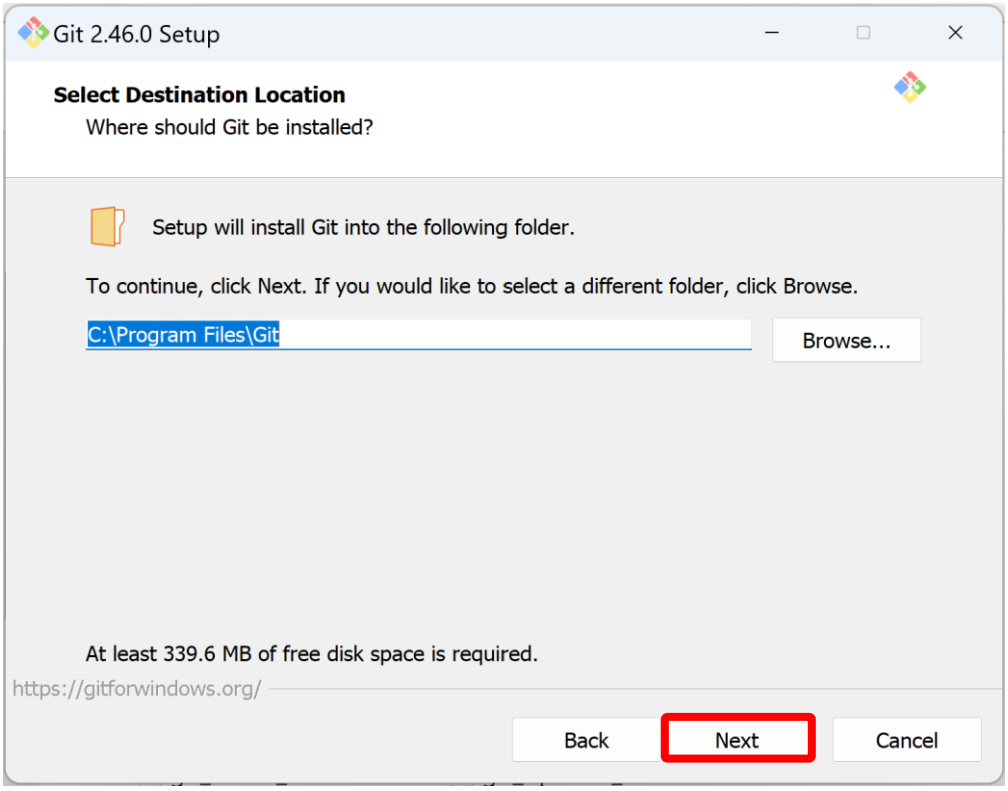


圖 3-8 選擇 Git 安裝路徑

Step5 勾選 On the Desktop，點擊「Next」，再點擊「Next」，如圖 3-9 所示。

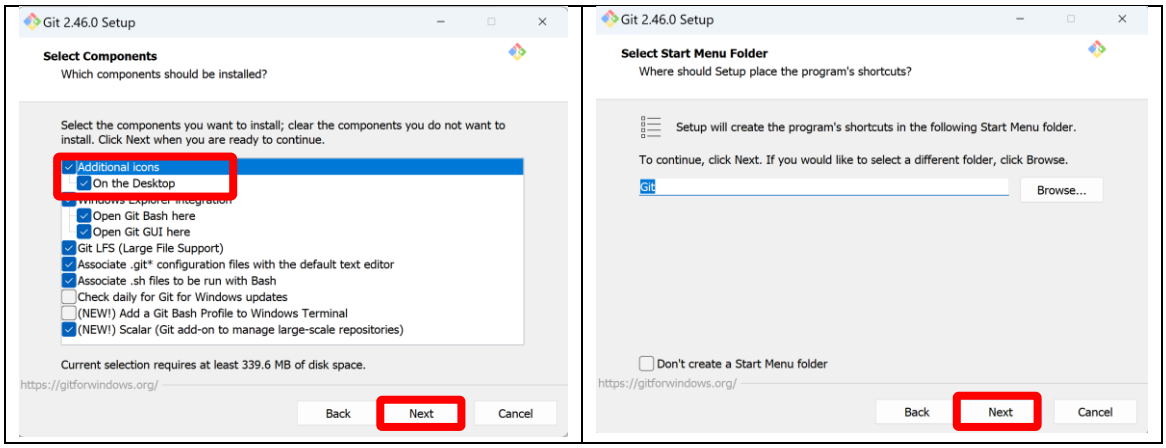


圖 3-9 勾選 On the Desktop（左）與設定開始目錄名稱（右）

Step6 設定 Git 編譯器，預設選擇 Use Vim（the ubiquitous text editor）as Git's default editor，點擊「Next」，如圖 3-10 所示。

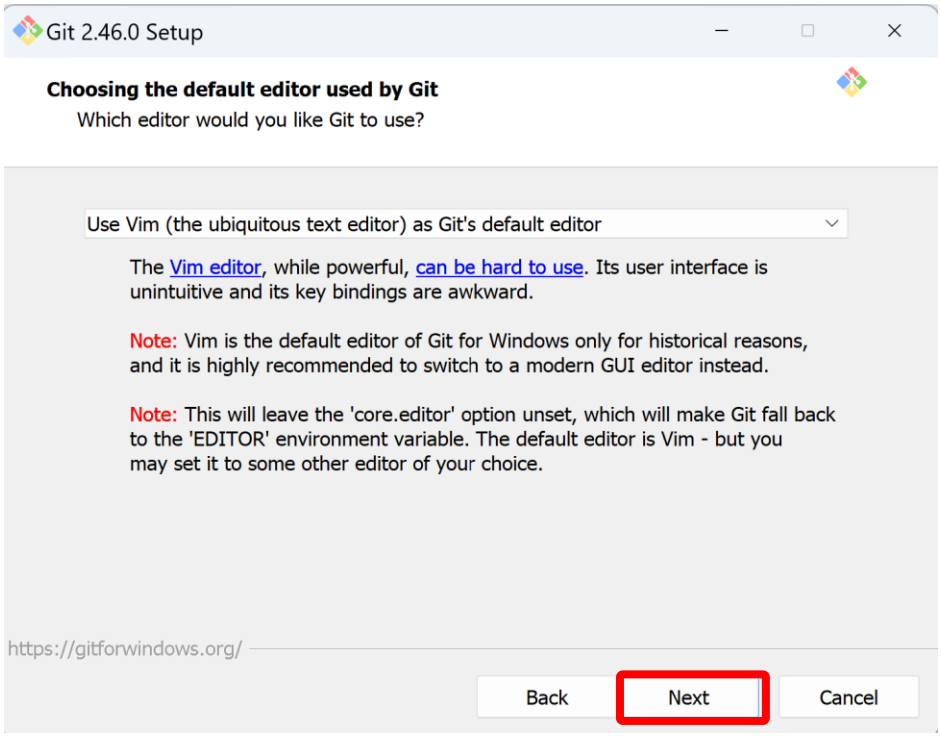


圖 3-10 選擇 Git 編輯器

Step7 設定預設分支名稱，選擇 Let Git decide 點擊「 Next」，如圖 3-11 所示。

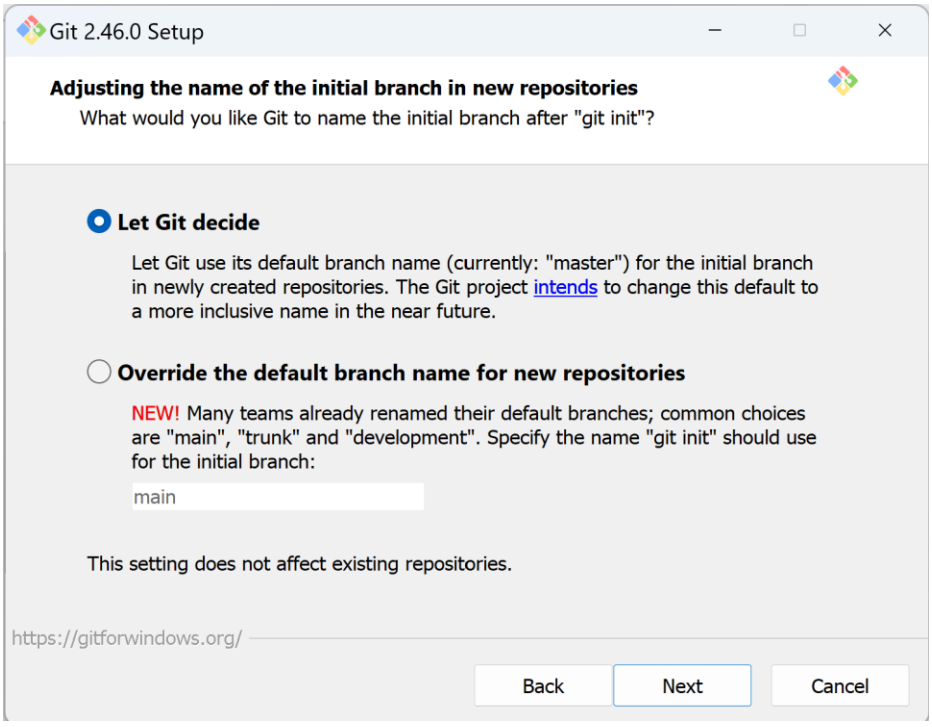


圖 3-11 設定預設分支名稱

Step8 選擇 Git from the command line and also from 3rd-party software，然後點擊「Next」，如圖 3-12 所示。

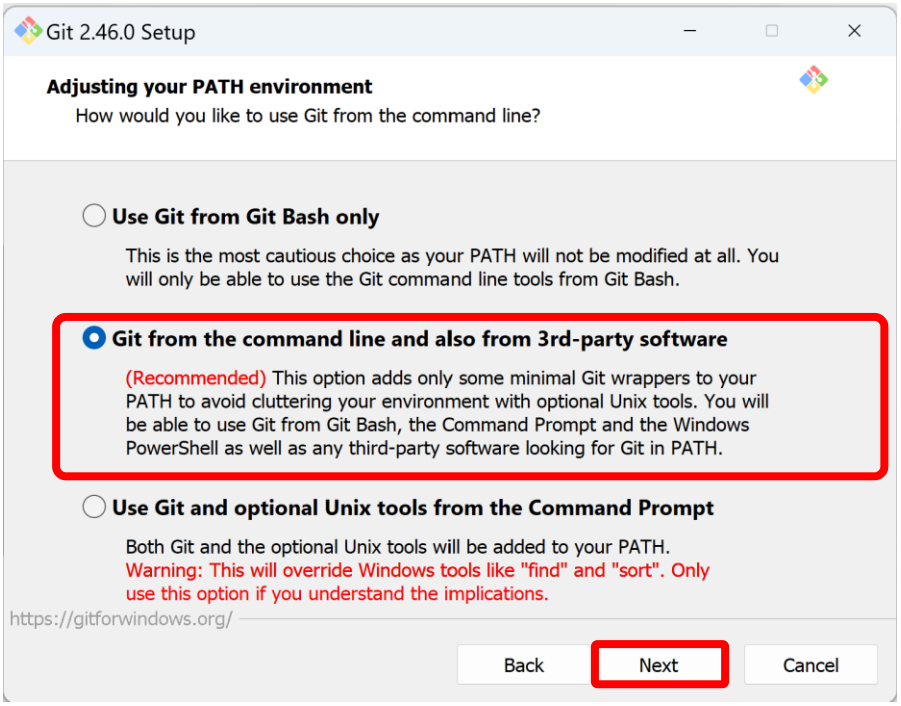


圖 3-12 設定 PATH 環境變數

Step9 選擇 Use bundled OpenSSH，點擊「Next」，如圖 3-13 所示。

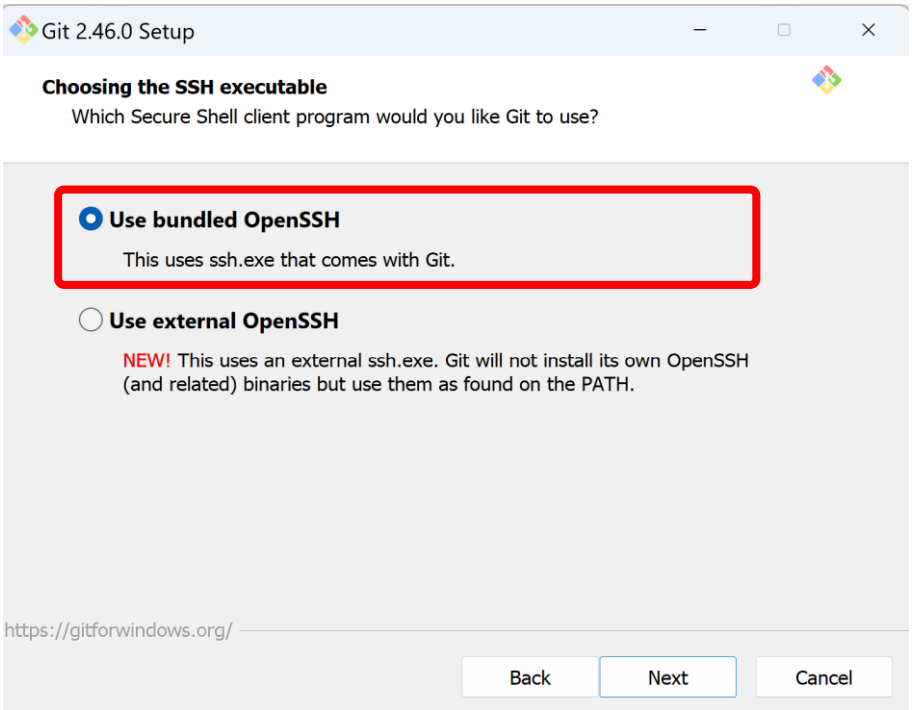


圖 3-13 設定 OpenSSH

Step10 選擇 Use the OpenSSL library，點擊「Next」，如圖 3-14 所示。

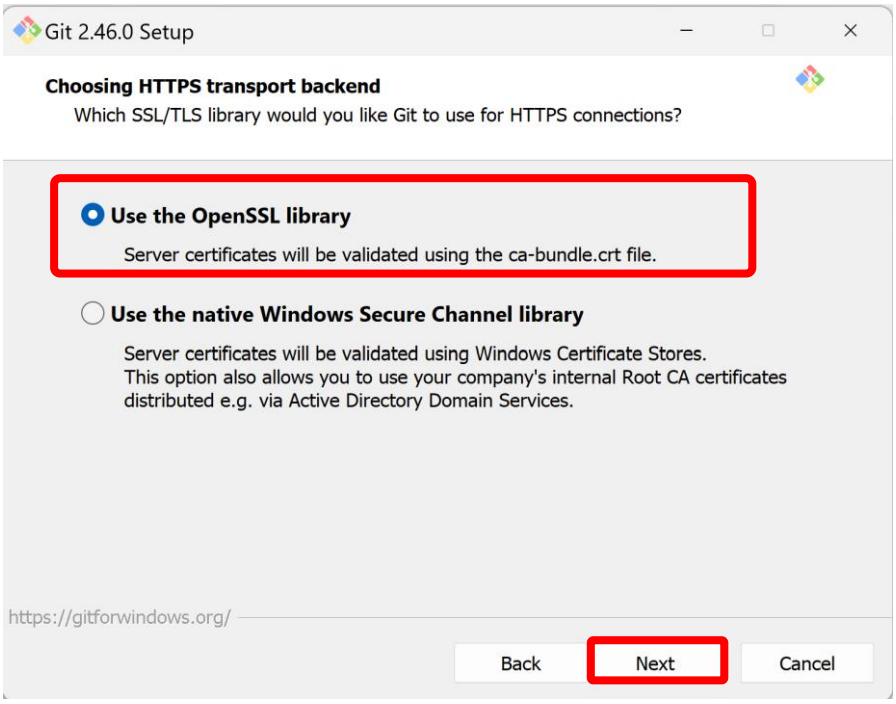


圖 3-14 選取 Use the OpenSSL library

Step11 設定檔案結束符號，預設選擇 Checkout Windows-style, commit Unix-style line endings，點擊「Next」，如圖 3-15 所示。

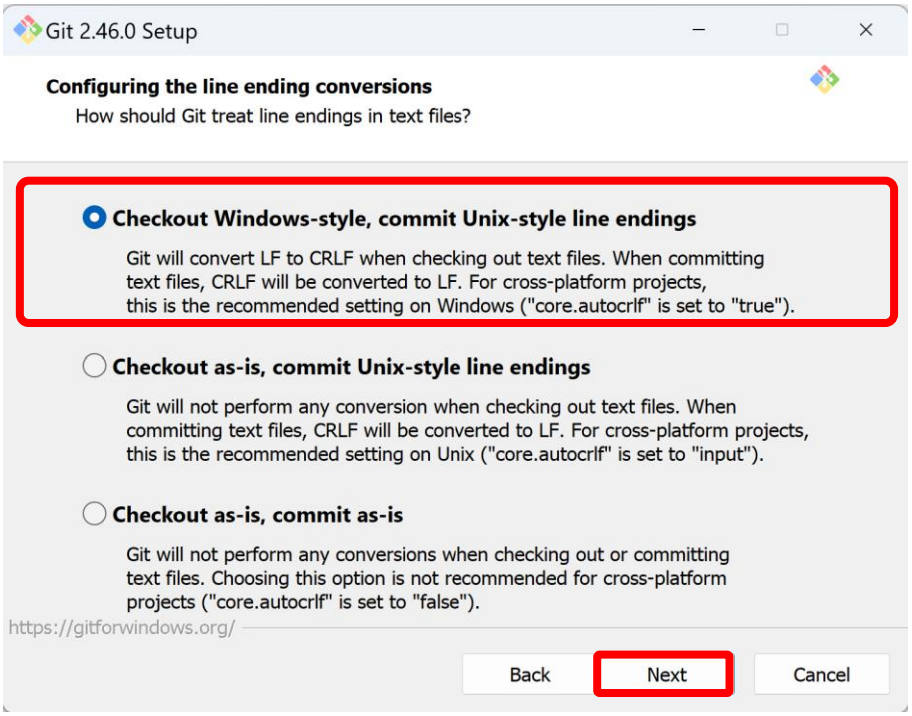


圖 3-15 設定結束符號

Step12 選擇 Use MinTTY，點擊「Next」，如圖 3-16 所示。

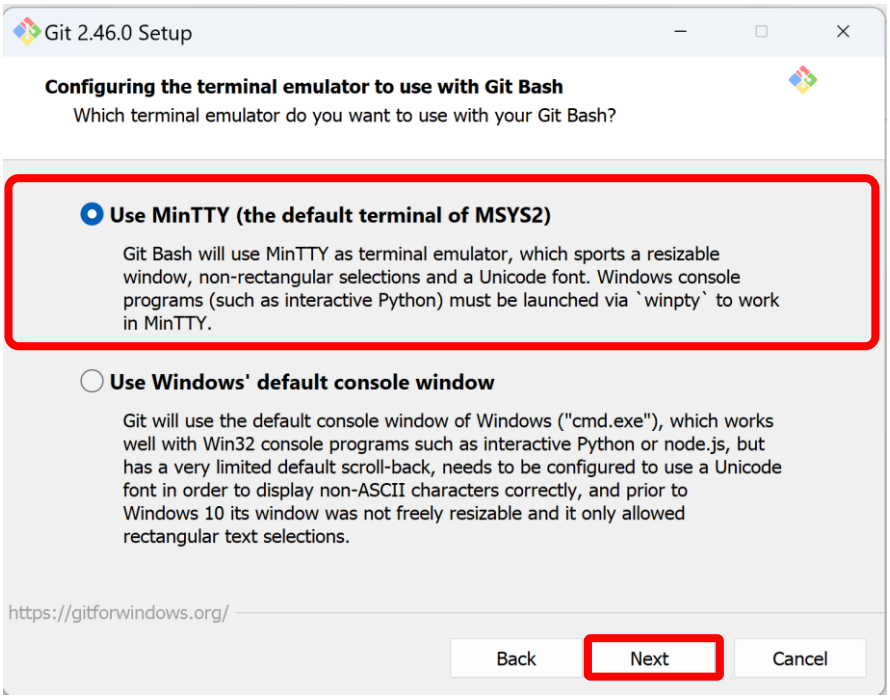


圖 3-16 選擇 Use MinTTY

Step13 選擇 Default (fast-forward or merge)，點擊「Next」，如圖 3-17 所示。

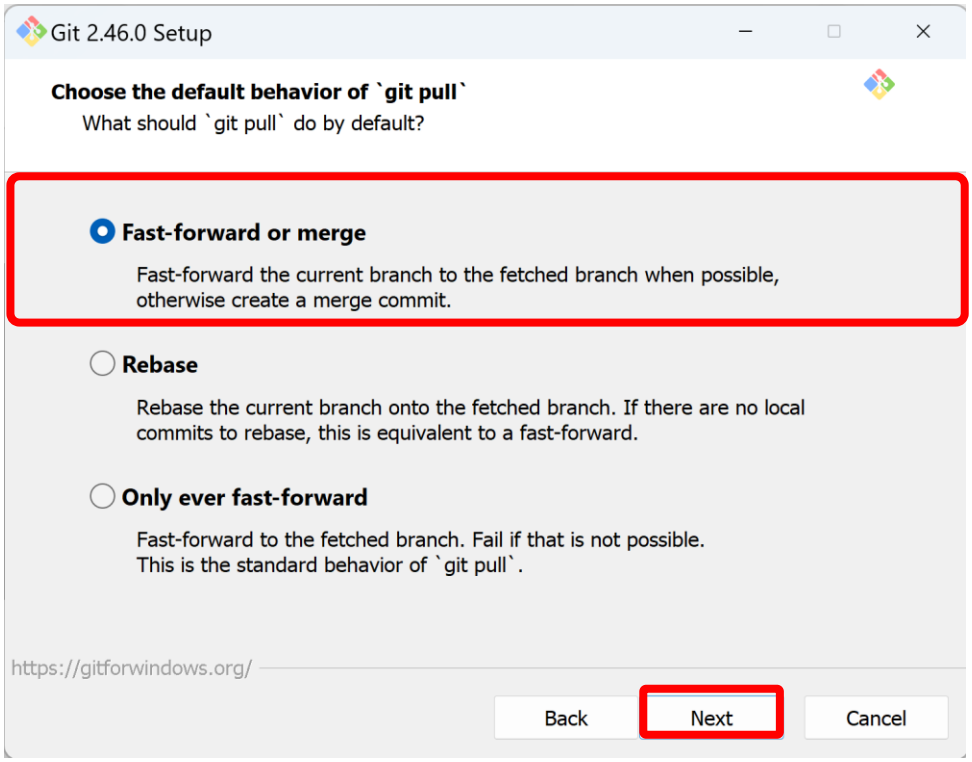


圖 3-17 選擇 Default (fast-forward or merge)

Step14 選擇 Git Credential Manager，點擊「Next」，如圖 3-18 所示。

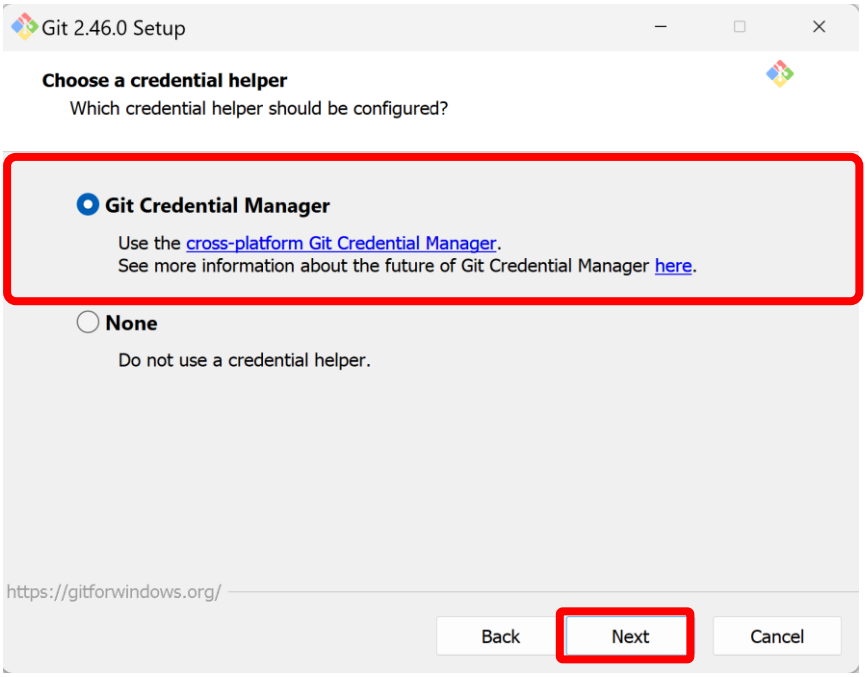


圖 3-18 選擇 Git Credential Manager

Step15 選擇 Enable file system caching 與 Enable symbolic links，點擊「Next」，如圖 3-19 所示。

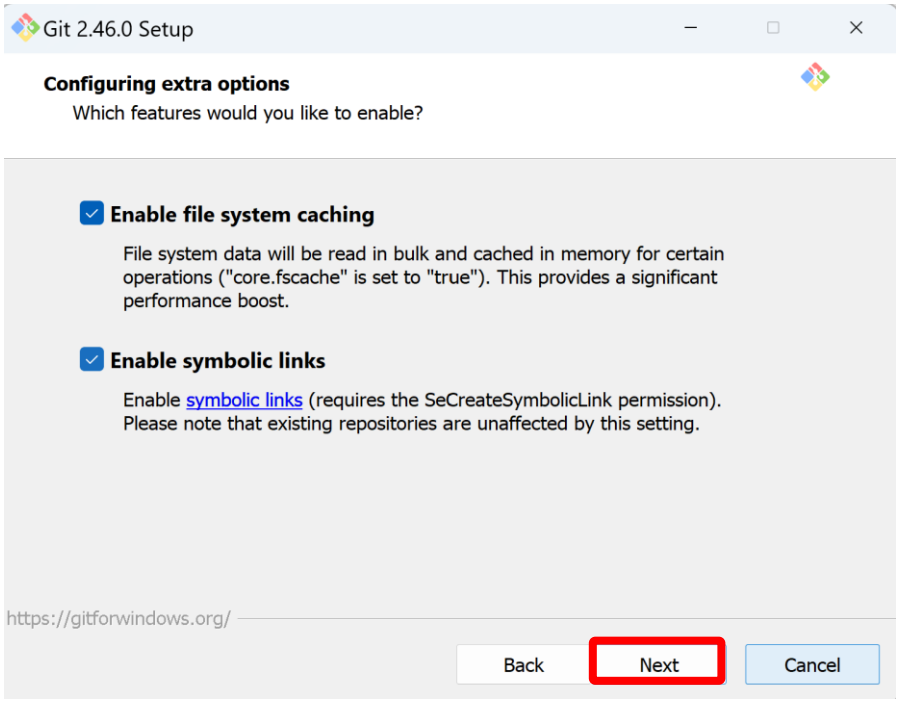


圖 3-19 選擇 Enable file system caching 與 Enable symbolic links

Step16 直接點擊「Install」，如圖 3-20 所示。

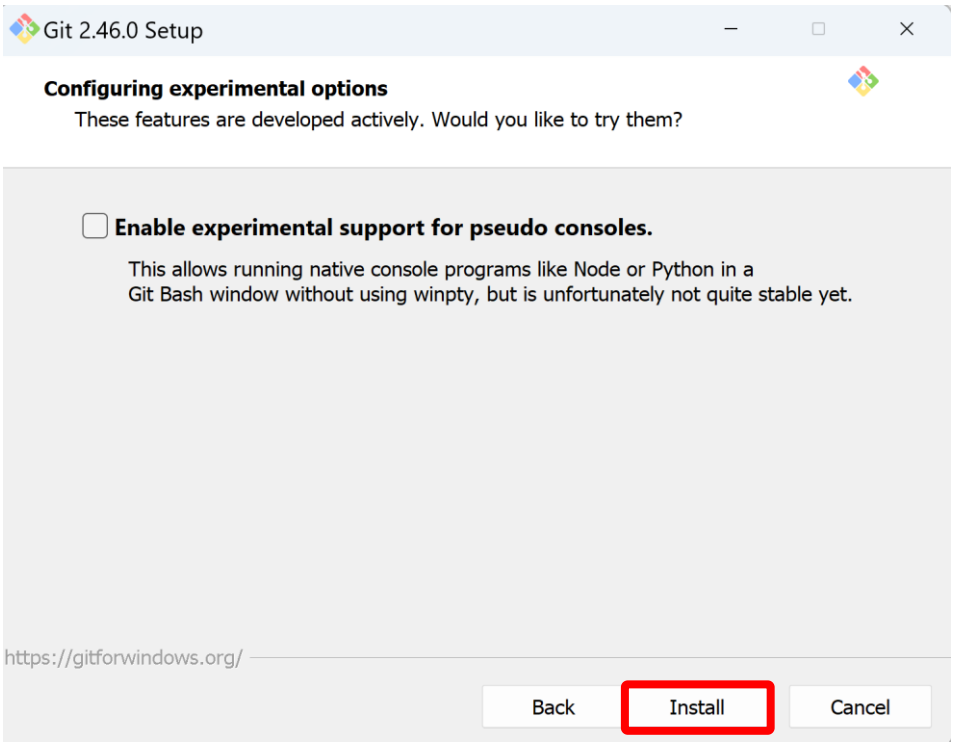


圖 3-20 開始安裝 Git

Step17 安裝完成後點擊「Finish」離開，如圖 3-21 所示。

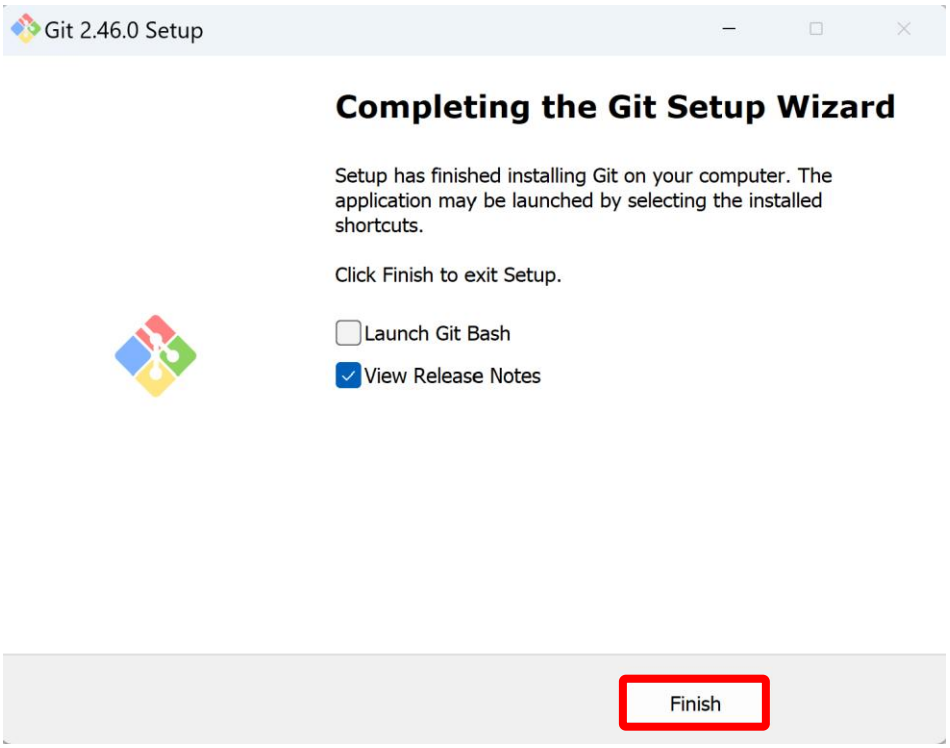


圖 3-21 安裝成功

Step18 到桌面點擊執行 Git Bash，畫面如圖 3-22 所示。

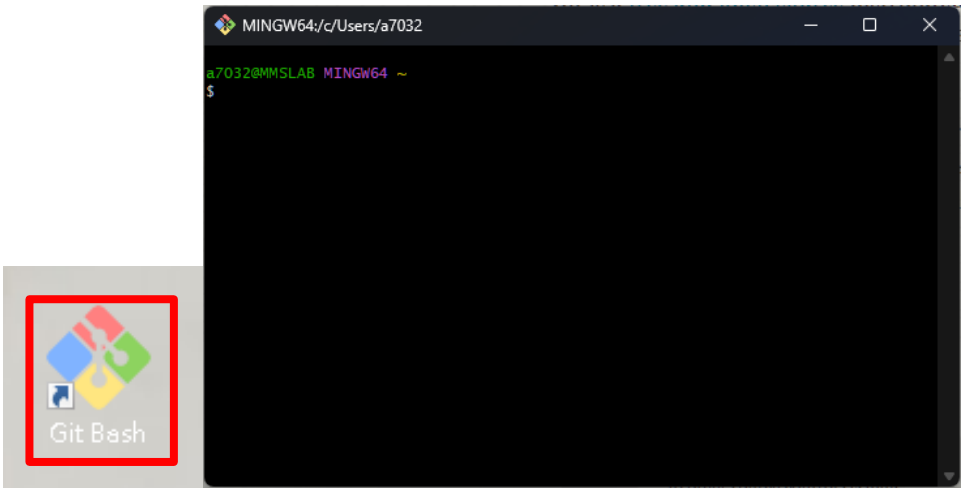


圖 3-22 開啟 Git Bash

Step19 在 git bash 內，設定自己的 user.email 和 user.name，如圖 3-23 所示。

```
$ git config --global user.email "xxx@gmail.com"

$ git config --global user.name "xxx"
```

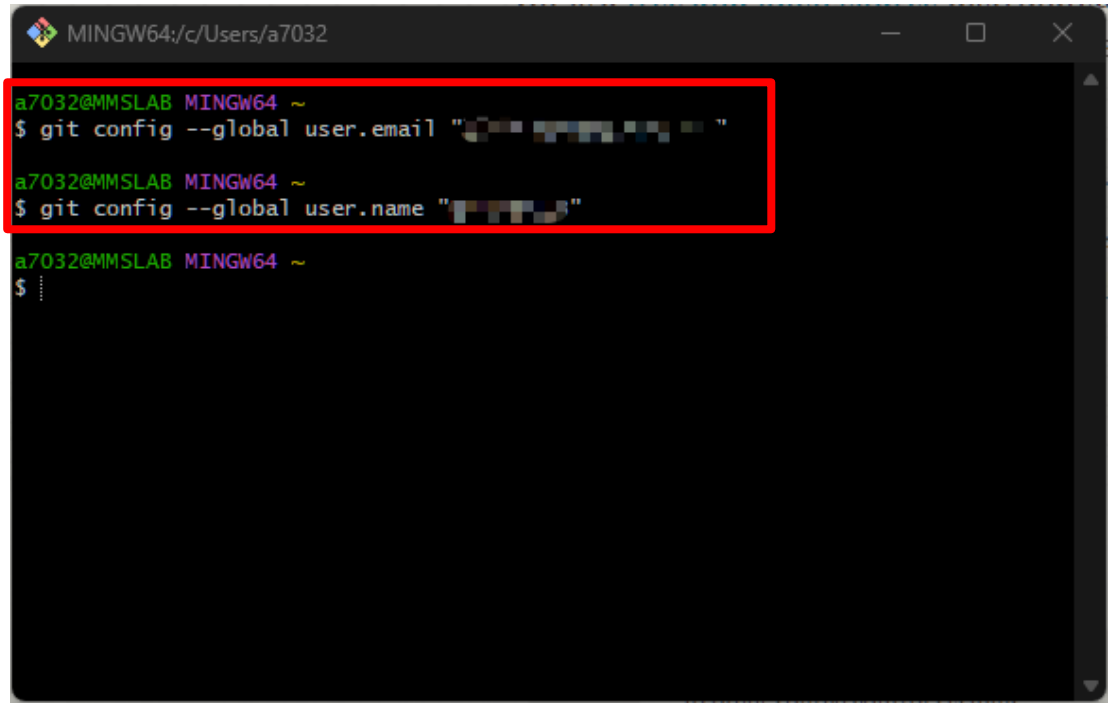


圖 3-23 設定使用者資料

0.3.2 加入課程的 Github Classroom

Step1 點擊實驗/作業繳交連結 (連結將在上課前發布)

Lab0-1 Github Classroom Assignment: 連結在上課前公布至課堂群組及 https://7333.italkutalk.com/

Step2 將自己的 Github 帳號與 Classroom 學生連結

同一門課程 Github Classroom 的作業或實驗僅需連結一次 (若曾經連結過，可以略過此步驟)。在學生清單中，會列出本門課程尚未被連結的學生，請找到並點擊自己的學號/姓名，如圖 3-24 所示。

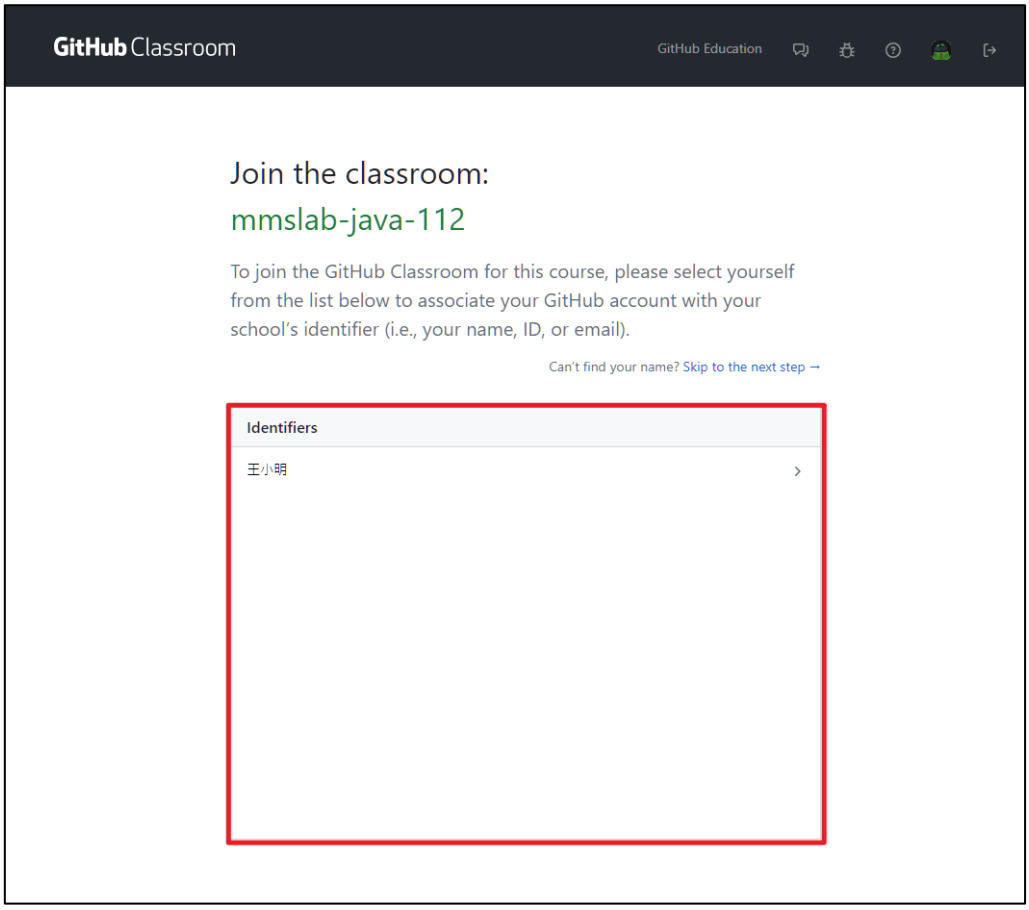


圖 3-24

點擊後會跳出確認提示，如圖 3-25 所示，確認無誤點擊確定。

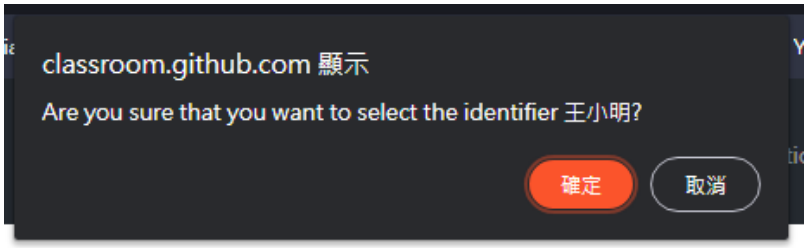


圖 3-25

如果遇到以下問題：

1. 名單中找不到自己的學號姓名
2. 選擇錯人
3. 學號姓名錯誤

請與課堂助教反應，助教將會協助處理。

Step3 接受 Assignment，點擊 Accept this assignment

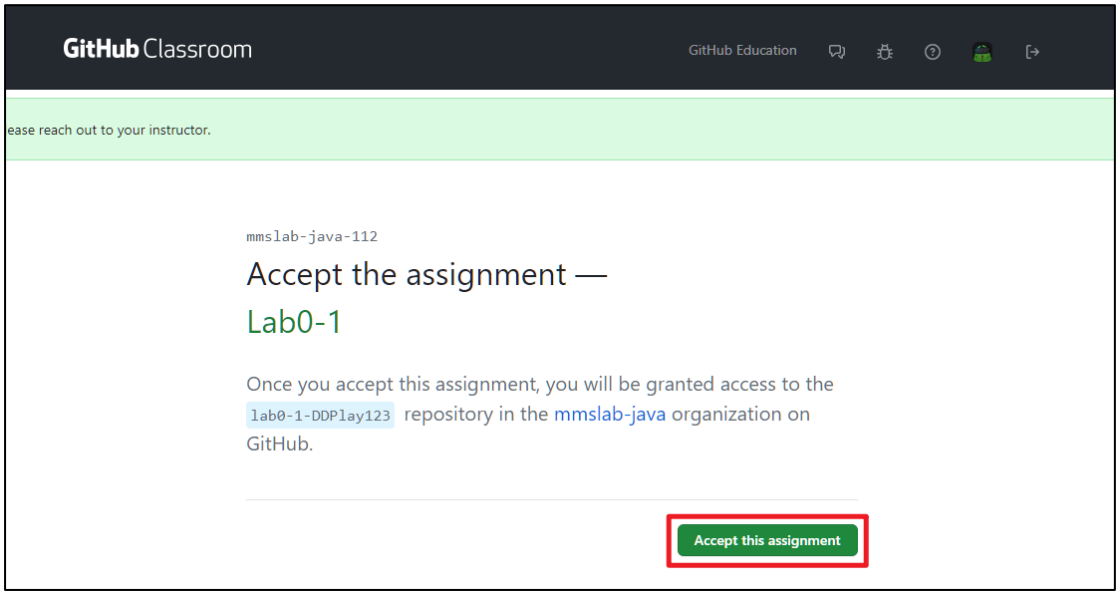


圖 3-26

接受 Assignment 後，Github classroom 會幫你建立專屬的 repository，而**建立專屬的 repository 需要一段時間，請等 10 秒左右刷新此頁面**

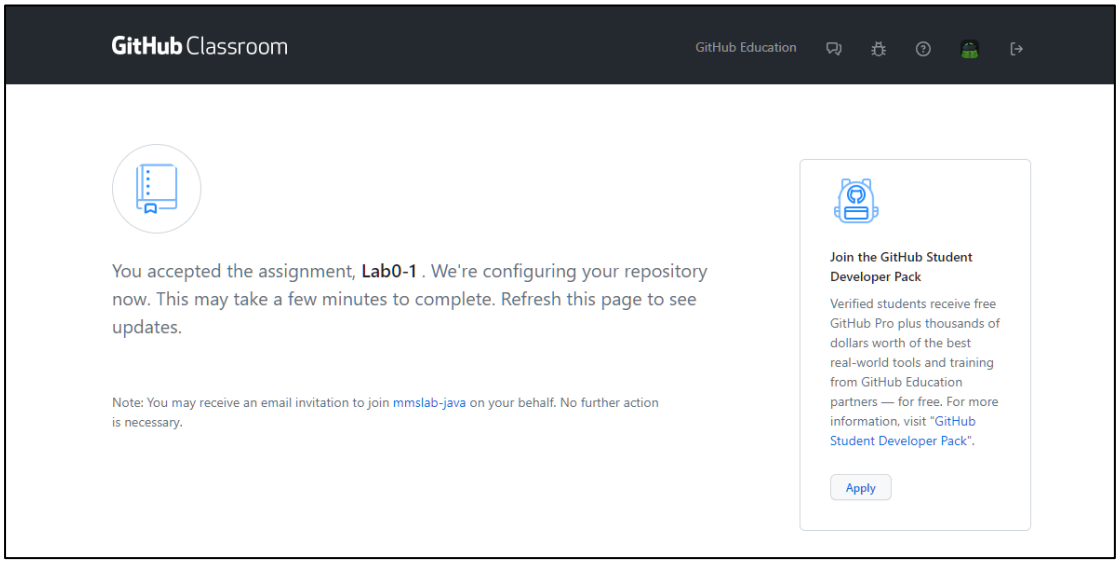


圖 3-27

刷新頁面後，將會看到屬於自己的 repository 連結，如圖 3-28 所示，並點擊該連結。

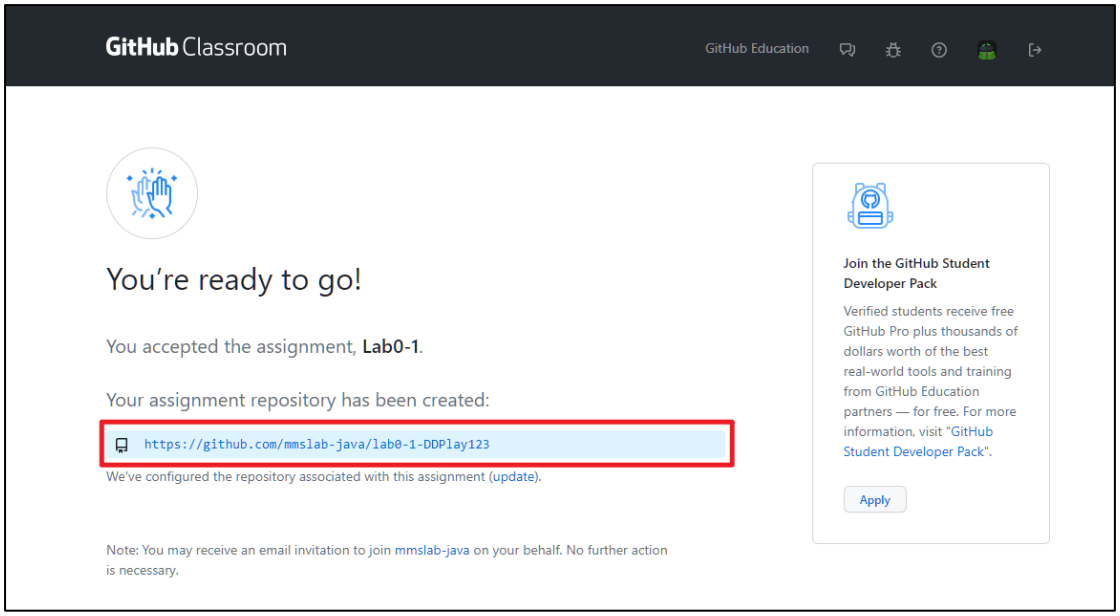


圖 3-28

Step4 將專案 Clone 到自己電腦，到桌面開啟 Git Bash

圖 3-29

複製專案在遠端資料庫的位置，如圖 3-30 所示，並在 Git bash 輸入指令進行下載專案(命令列點擊右鍵可以選貼上)

圖 3-30

```
$ git clone https://github.com/xxx/xxx.git (記得改成自己的資料庫連結網址)
```

輸入指令情況：

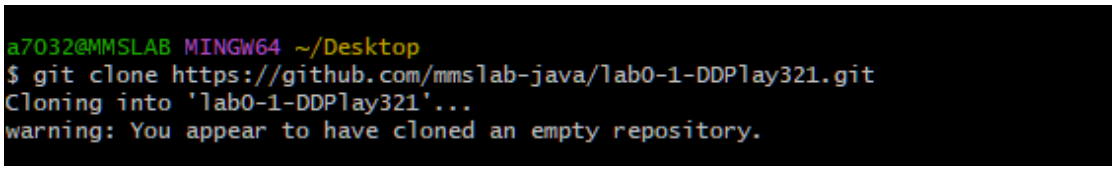


圖 3-31

若出現 SSL 錯誤，請輸入指令

```
$ git config --global http.sslVerify false
```

完成下載專案後請到桌面開啟剛下載的專案資料夾，再開啟一次 Git Bash 繼續下面演練。

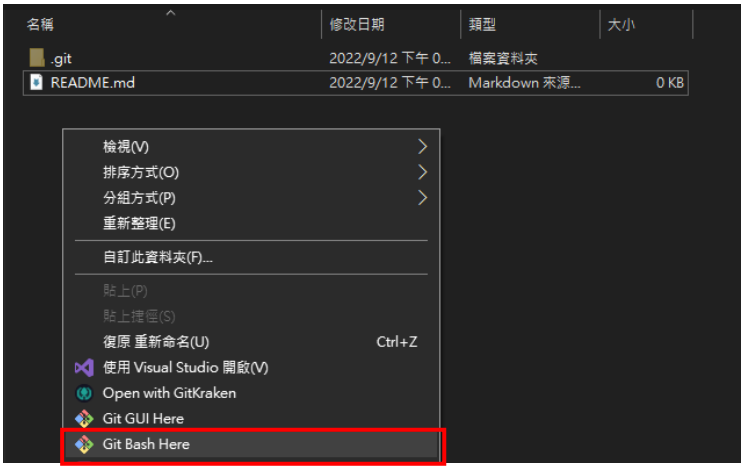


圖 3-32

0.3.3 將撰寫完成的專案推送到 GitHub 上

Step1 建立 README.md，如圖 3-33 所示，再打開資料夾檢查，如圖 3-33 所示。

```
$ echo "# Hello git" >> README.md
```

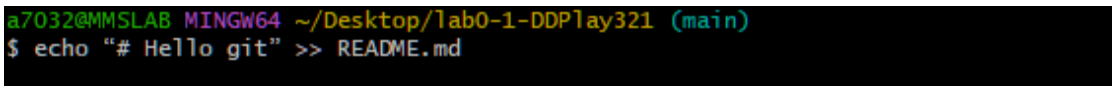


圖 3-33 建立 README.md

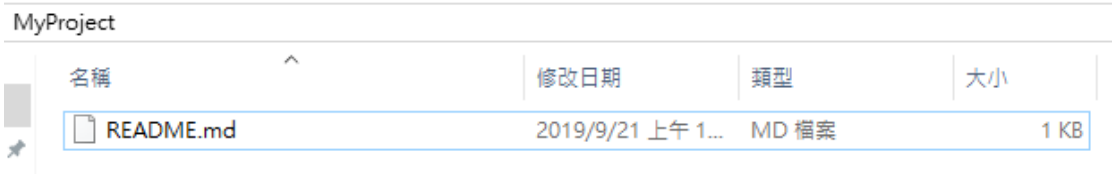


圖 3-34 檢查檔案

Step2 查看本地資料庫狀況，有紅色字體表示變更未被追蹤，如圖 3-35 所示。

```
$ git status
```

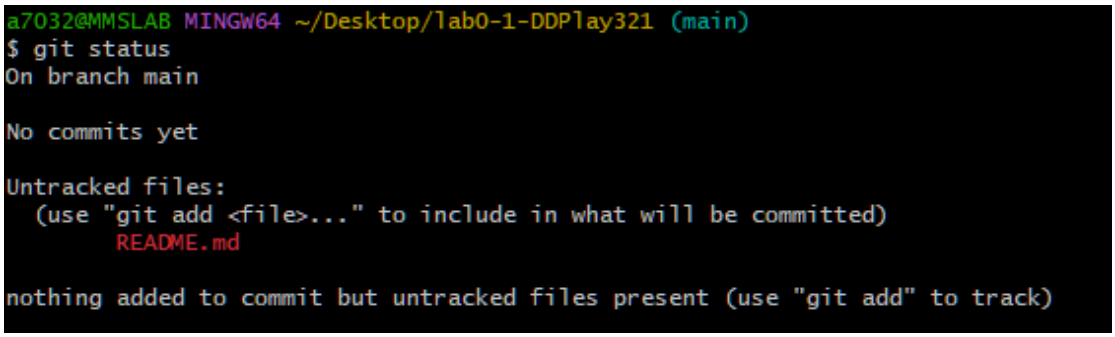


圖 3-35 查看變更

Step3 將檔案的變更動作加入至準備提交區，如圖 3-36 所示，指令如下：(提交後可以執行 git status 查看有哪些檔案被追蹤/新增了)

```
$ git add .
```

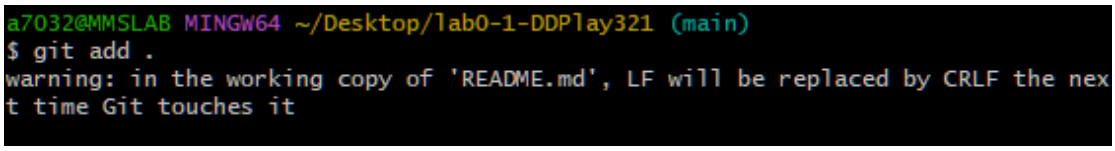


圖 3-36 加入變更的檔案

Step4 將提交區的檔案至本地資料庫，如圖 3-37 所示。

```
$ git commit -m "first"
```

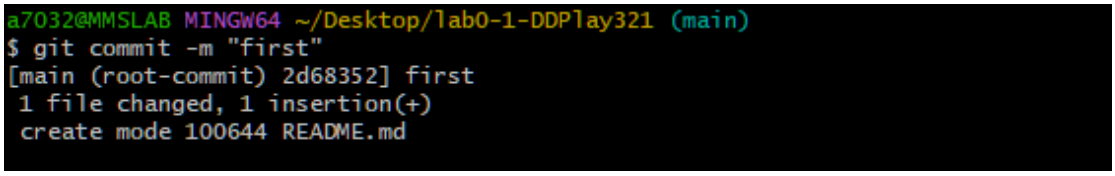


圖 3-37 提交檔案

Step5 將本地資料庫的紀錄提交到遠端資料庫（GitHub）上，此時須輸入 GitHub 的使用者名稱及使用者密碼，完成身分驗證，點選「Login」，如圖 3-38 所示。

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

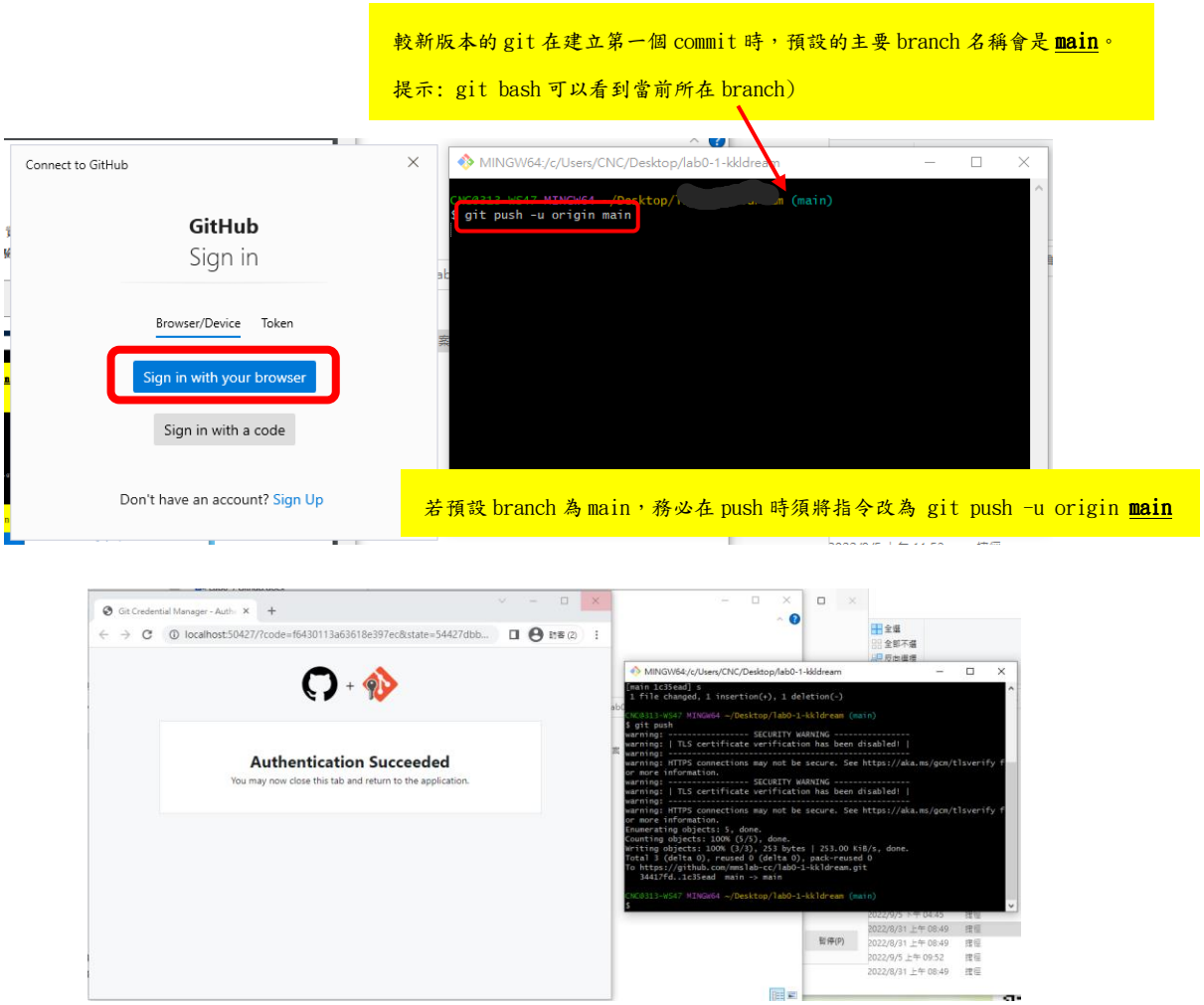


圖 3-38 提交檔案至遠端資料庫、身份驗證

Step6 檔案成功推至遠端的 GitHub，如圖 3-39 所示，若遇到 unable to access 的問題，請輸入以下指令，再執行一次 Step5。

```
$ git config --global http.sslVerify false
```

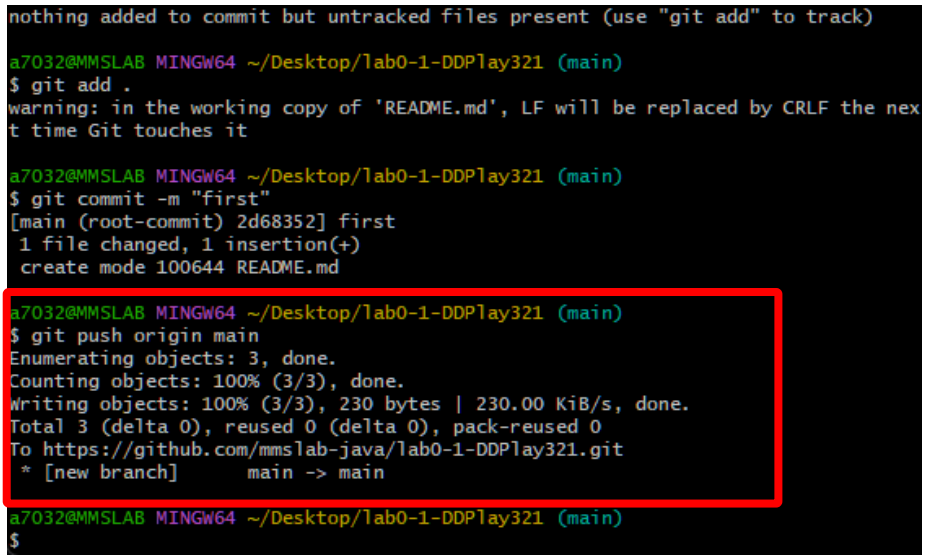


圖 3-39 檔案上傳成功

Step7 回瀏覽器，F5 重新整理 GitHub 的頁面，可以看到剛剛上傳的檔案，如圖 3-40 所示。

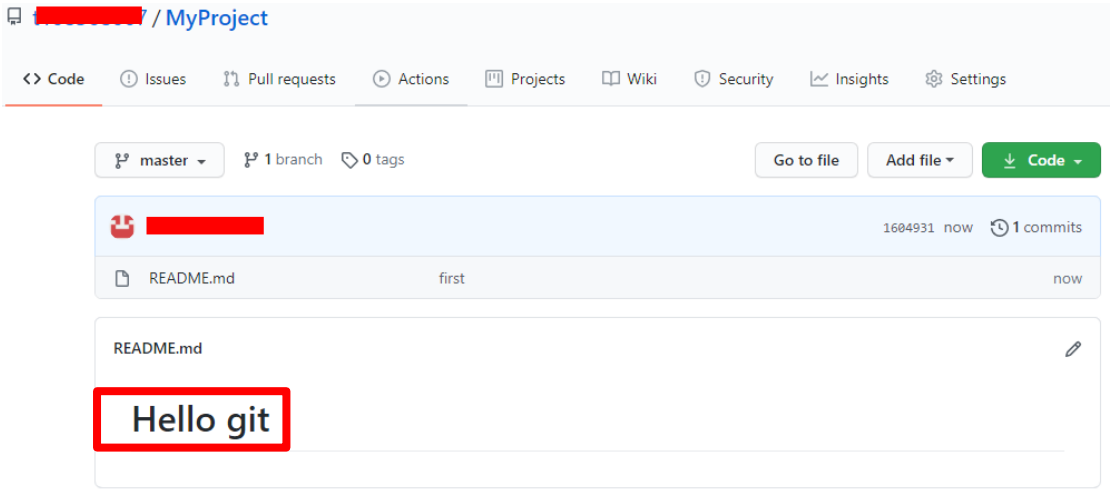


圖 3-40 查看 GitHub 上的專案

0.4 參考資料－Git 常用指令

指令	說明
gitk	顯示歷史紀錄的樹狀圖形化介面。
git add	加入要新增的檔案。
git pull	將遠端的資料更新到本地端。
git branch	創立一個新的分支。
git merge	合併分支。
git mergetool	呼叫一個適當的視覺化合併工具並引導你解決衝突。
git log	檢視所有提交的歷史紀錄。
git reset	回到某個特定的 commit（reset 會直接砍掉提交的歷史紀錄）
git push	將本地端的資料更新到遠端。
git push --force -f	強制更新並覆蓋遠端的分支。
git commit	提交一個新的版本。
git commit --amend	更改目前分支最新版的 commit。
git commit –message -m	直接在後面輸入提交訊息。
git checkout	查看分支或節點。
git checkout -b	先建立分支再切換。
git stash	把資料放到暫存區。
git stash list	列出暫存區裡全部的資料。
git stash pop	取出暫存區中最新的一筆資料，並將其移除。
git stash apply	取出暫存區中最新的一筆資料，但不會移除。
git stash apply stash@{index}	取出第 index 筆暫存資料。

git stash clear	把暫存區裡的資料都清掉。
git reset --hard	強制回復到上一版。
git reset --hard xxxx	強制回復到某個 commit 版本。
git reset --hard origin/B	強制回復到遠端 B 分支版本。
git rebase	衍合分支。
git rebase -i HEAD~n	衍合最後的 N 次提交。
git rebase --interactive -i HEAD ~n	開啟對話模式編輯（head~n表示要編輯到前n個）

說明

rebase 跟 reset 是很危險的指令，因為它們都是改寫提交的歷史紀錄。所以最好在操作前先用 branch 備份，有問題時只要 reset 回備份的分支就行了。

1.git status：檢查目前分支狀態。

如圖 4-1 所示，透過 git status 指令可以確認自己修改的檔案。

※指令範例：git status：

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   public/index.html
#       deleted:    public/sitemap.xml
#       new file:   public/stylesheets/mobile.css
#
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       deleted:    app.rb
#       deleted:    test/add_test_crash_report.sh
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       public/javascripts/
```

圖 4-1 git status

2.git log：查看分支近期 commit 過的點，如圖 4-2 所示，查看 commit 的名稱、幾天前、版本號。

※指令範例：git log：

```
107632c - (HEAD, release-1.2.4, b1.2.4) Update website for 1.2.4 (1 year, 4 months ago)
88d9f68 - Bump version number (1 year, 4 months ago)
b7df3fd - regenned site for new blog post about status board (1 year, 4 months ago)
f76e532 - new blog post: rubinius status board (1 year, 4 months ago)
42f7c72 - added capitalize to String case benchmarks (1 year, 4 months ago)
bddf636 - yet another way of removing the first elements from an array (1 year, 4 months ago)
6e4ed98 - new bench for Array#slice (1 year, 4 months ago)
049bace - Remove tags for now passing specs (1 year, 4 months ago)
44c3886 - Socket needs it's own shutdown (1 year, 4 months ago)
8374734 - regenned site for new blog post (map pins) (1 year, 4 months ago)
f90da99 - new blog post: rubinius around the world map and pins of shirts/tshirts (1 year, 4 months ago)
cf13e6b - Add a few more errno's based on OS X and Linux (1 year, 4 months ago)
0b8b477 - Add a bunch of errno's from FreeBSD (1 year, 4 months ago)
4b34345 - Load correct digest file, fixes broken Rubygems (1 year, 4 months ago)
e2be2d5 - Remove unused rubinius::guards (1 year, 4 months ago)
23e97d5 - Remove used flag and file it was defined in (1 year, 4 months ago)
cff4ee2 - Remove unused CallFrameList and some maps (1 year, 4 months ago)
dd8f2b1 - Removed unused async message and mailbox code (1 year, 4 months ago)
c4b54ba - Remove unused code (1 year, 4 months ago)
744e9f0 - Fix tiny typo's (1 year, 4 months ago)
912d530 - Cleanup last remnands of dynamic interpreter (1 year, 4 months ago)
6b29b21 - Remove unused IndirectLiterals (1 year, 4 months ago)
83db68a - Fixed Digest requires in const_missing. (1 year, 4 months ago)
```

圖 4-2 Git log

3.git add .：將修改或變更檔案的部分暫存在 index 位址，如圖 4-3 所示。

※指令範例：git add .：

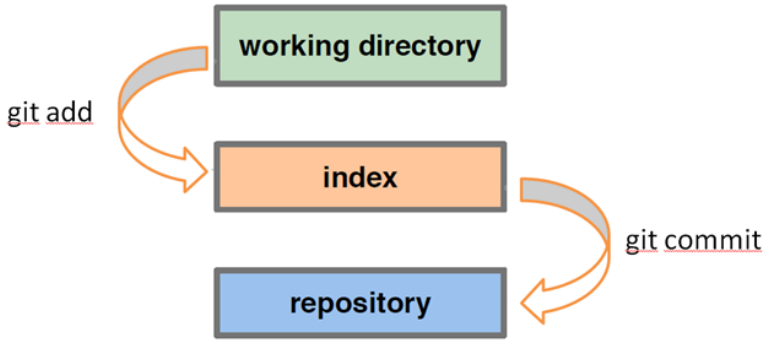


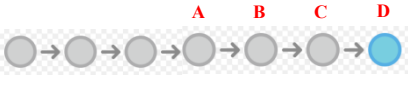
圖 4-3 Git add

4.git commit：將暫存在 index 位址內容存到 commit 點的容器裡。

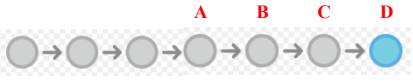
如下圖所示，原先在 A 點，經由修改專案後，git commit 產生出新的節點 B。

※指令範例 git commit –m "敘述這次提交修改的內容"

5.git pull：將目前 GitHub 上最新的 commit 點同步下來，下載目前最新的專案。

個人 local 端的 commit 點  ，最新 commit 點只到 A 點，GitHub 上的 commit 點 

，最新 commit 點是 D，當下 git pull 的指令時會將 GitHub 上 commit 點同步到 local 端，結果如下：

個人 local 端的 commit 點 

。

※指令範例：git pull：

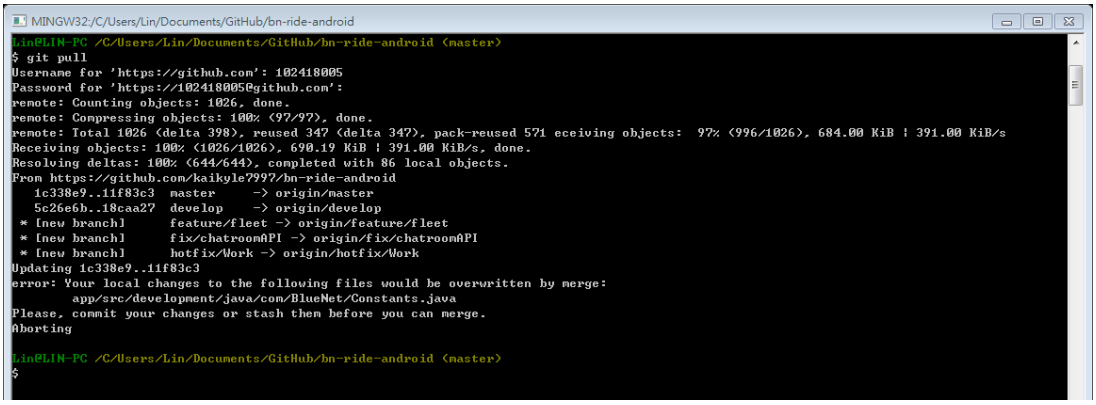


圖 4-4 Git Pull 示範

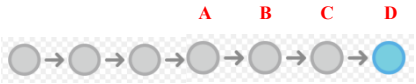
6.git push：將 local 端 commit 的點同步到 GitHub 上，更新程式碼。

local 端的 commit 點 

，最新 commit 點是 D，GitHub 上的 commit 點 

，最新 commit 點到 A 點，因為 local 端有修改，所以同步到 GitHub 時要執行指令 git push。

結果如下：

GitHub 上的 commit 點 

。

※指令範例：git push，通常下指令後，需要輸入帳號密碼才會同步。

7.gitk：開啟 Git GUI，點擊左上方區塊的 commit 點的詳細資料，如圖 4-5 所示。

※指令範例：gitk

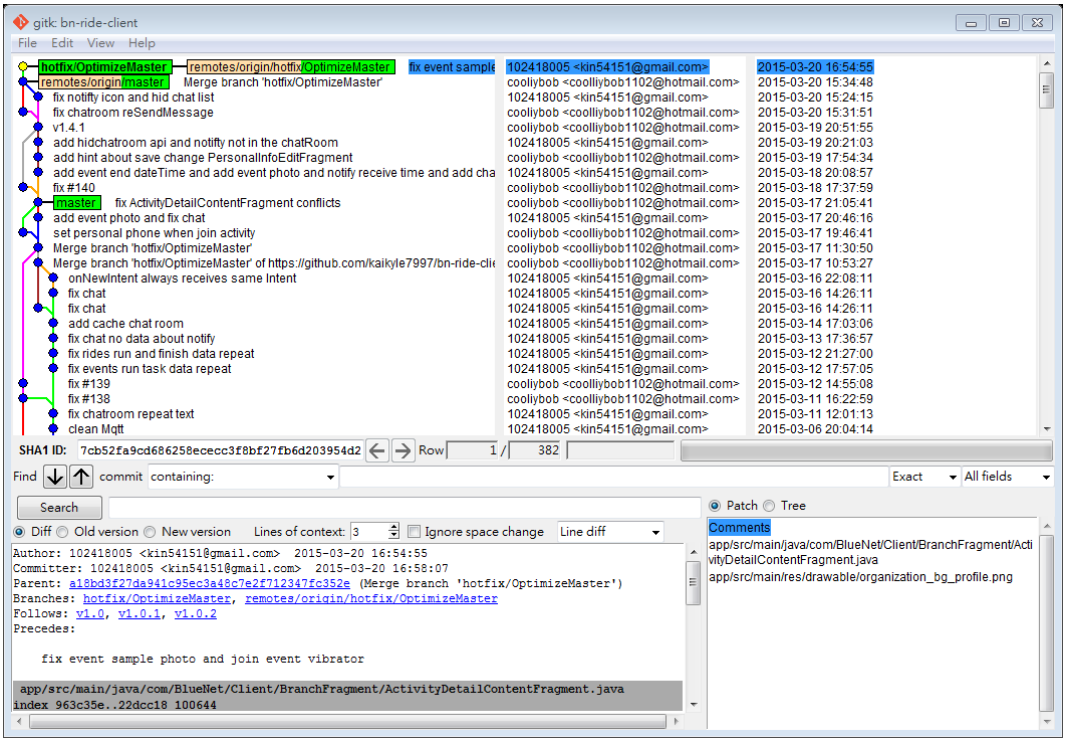


圖 4-5 Git GUI

8.git checkout：切換到不同分支上。

現在在 develop 分支上 

，當下 git checkout master 指令後，就會切到 master 分支上 

。