UNIT 2

原來 Git 可以這樣用

在單元 1 我們是從專案備份的觀點切入來介紹 Git，但是其實 Git 的應用不僅於此。在開始學習 Git 以前，我們先畫一下大餅，讓讀者對 Git 的用法有一個概括性的了解。

2-1 Git 有哪些功能

圖 2-1 是把 Git 的應用做一個通盤的展示，我們把它分成三個階段來介紹。

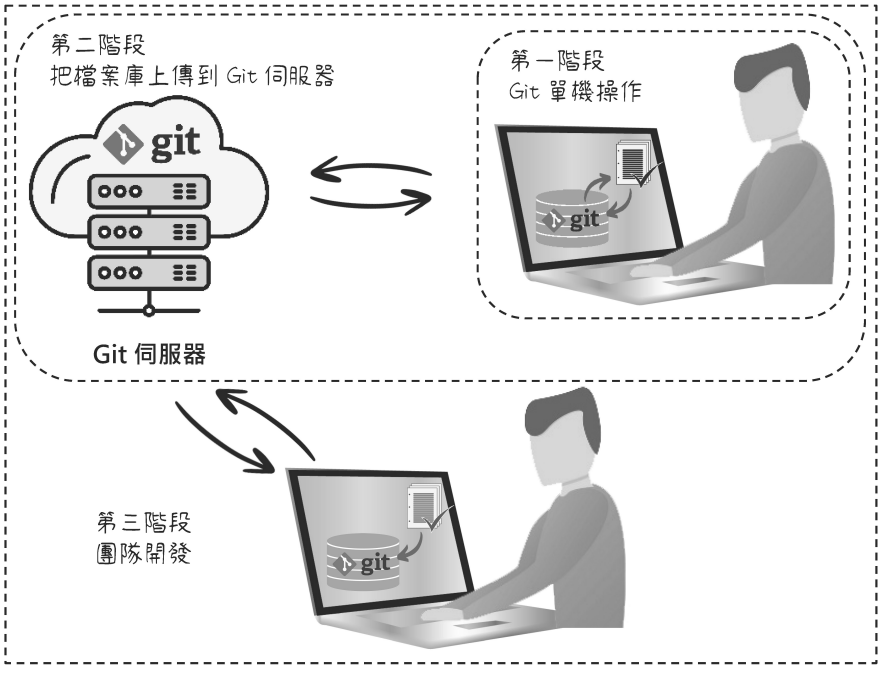

圖 2-1　Git 用法概觀示意圖

在說明之前，我們先解釋一個 Git 專用術語 Repository。這個字是儲藏室的意思，在 Git 裡頭它是儲存檔案內容的地方，本書稱它為檔案庫，這個名稱簡單易懂！檔案庫說穿了，就是一個隱藏的子資料夾，Git 會把檔案變更的內容，全部存到這個資料夾裡頭。

現在回到圖 2-1，我們用虛線框把它分成三個部分，這三個部分也是我們學習 Git 的順序。首先是第一階段，這部分是 Git 單機版操作，它是學習 Git 的基礎，這裡大概佔六到七成的學習份量。把它弄懂、熟練之後，後續二個階段就能水到渠成！

Git 單機版操作包含下列幾個學習重點：

1. Git GUI 程式的用法和 Git 設定檔。
2. 如何執行 Commit，和從檔案庫取回檔案。
3. 檔案比對。
4. 幫專案建立分支，執行合併和解決衝突。

學會以上技巧之後，就能夠隨心所欲地管理自己電腦上的程式專案。

圖 2-1 的第二個階段是把檔案庫上傳到 Git 伺服器。這樣做有二個目的，第一是備份，也就是把檔案庫備份到雲端，這樣就不怕電腦資料遺失或毀損，因為我們隨時都可以從伺服器下載檔案庫。第二個好處是開發團隊可以利用 Git 伺服器上的檔案庫，分享每一個人的成果，這是第三個階段的基礎。

圖 2-1 中的第三個階段是學習團隊開發的運作模式。團隊開發會遇到的問題是，當你在修改程式的時候，其他人可能也在修改。這種情況會導致專案出現二個不同的版本。當你把修改後的結果上傳到 Git 伺服器時，如果其他人已經比你早一步做了修改，Git 伺服器就會要求你先下載修改後的版本，然後把你的修改合併到這個新版本，最後再上傳合併後的結果。

除此之外，我們還會介紹幾個比較知名的 Git 伺服器網站，像是 GitHub、Bitbucket 和 GitLab。它們不僅可以讓我們上傳檔案庫，還提供專案開發模式的建議，像是 GitHub Flow 和 GitLab Flow，

幫助我們提升軟體開發的品質。這些開發模式也會在本書中做介紹。

除了圖 2-1 的用法之外，其實 Git 還有其他用途，其中有一種應用叫做 GitBook。從名稱上看，很容易就可以猜到它應該和書有關。是的，它就是用來寫書的工具。

Git 為什麼會和寫書扯上關係？我們可以想像一下寫書的過程，其實和開發程式專案有很多雷同的地方，它們都會不斷地加入文字、圖片或是其他型態的內容，或是不斷地修改它們。在整個過程中，我們希望可以留下修改紀錄，以便將來可以回頭查詢，這不就是前面討論過的 Git 使用情境。

當然，書籍內容的呈現還需要搭配版面控制的功能，所以 GitBook 需要安裝其他套件，但是它的核心就是 Git。另外如果再加上圖 2-1 的團隊開發功能，就可以達成多人一起寫書的目標，而且每一個人所做的修改都可以完整地保存下來。

前面花了一些篇幅介紹 Git 的應用，目的是要讓讀者先對 Git 的用途有一個通盤的了解，這樣有助於後續學習某一項功能的時候，可以很清楚的對應到它的應用層面。接下來我們要開始進入 Git 的世界，不過在開始之前要先說明一下 Git 的操作方式。基本上 Git 有二種操作方法，第一種是利用圖形介面，第二種是使用指令。圖形介面的優點是使用起來比較方便、快速，但是無法涵蓋所有的功能。相對而言，指令比較麻煩，但是它支援 Git 全部的功能。本書教學是以 Git GUI 圖形操作為主，它同時適用 Windows、Mac 和 Linux 平台。另外還會補充相關指令的用法，如此一來，讀者就可以同時具備圖形介面和指令的運用能力。

2-2 Git 操作初體驗

Git 的出現是為了管理專案開發的過程，但是其實我們可以換用一個比較務實而且容易理解的說法：Git 的功能就是追蹤一個資料夾內容的變動，而且無論資料夾裡頭儲存哪一種類型的檔案，或是有多少層子資料夾，都可以用 Git 控管，並不僅限於程式專案資料夾。

要讓 Git 控管一個資料夾，必須先在該資料夾建立 Git 檔案庫，這個步驟稱為初始化。為了讓讀者了解 Git 如何追蹤資料夾內容的變動，我們用一個內容很簡單的資料夾作示範。這個資料夾中只有三個檔案（請參考圖 2-2）：

1. program.py
2. 本書幫助你成為 Git 專家.docx
3. git-log.png

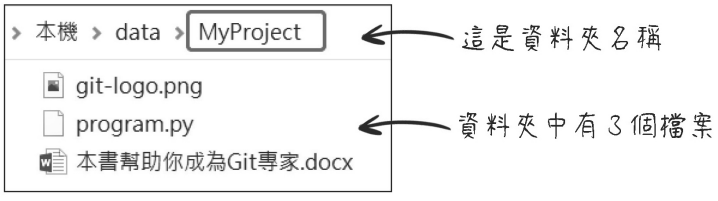

圖 2-2　用 Git 控管這個資料夾

這三個檔案分別代表三種不同檔案類型。「program.py」是純文字檔。不管是哪一種程式語言，它的程式檔都是純文字檔。純文字檔的格式最簡單，所以它很容易解讀。這裡我們是以目前最常見的 Python 程式檔為例，其他程式語言的程式檔也會是一樣的結果。

「本書幫助你成為 Git 專家.docx」是 Word 文件檔。雖然把 Word 檔打開會看到文字，但是其實它的內容不是只有文字，還有一些特殊的控制碼，它們用來標示字型、字體大小、行距、縮排等資訊，所以 Word 檔不是純文字檔。接下來的「git-log.png」是影像檔。影像檔也不是文字，而是所謂的二進位檔。二進位檔是用最原始的電腦資料格式來儲存，它的內容無法用文字的方式解讀，必須用特定的軟體來處理，才能夠看到正確的結果。

現在我們要讓 Git 管控這個資料夾，請依照下列步驟操作：

1 STEP 依照上一個單元的說明啟動 Git GUI 程式，選擇第一項 Create New Repository，參考圖 2-3。

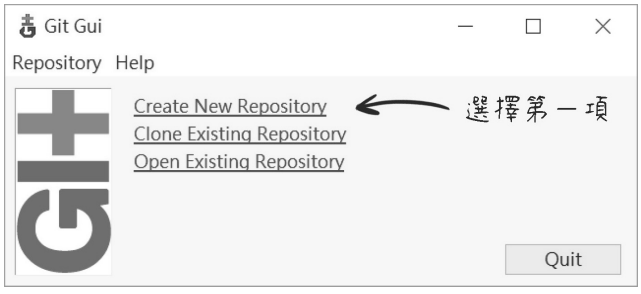

圖 2-3　選擇 Git GUI 程式的 Create New Repository

2 STEP 畫面會顯示圖 2-4 的對話盒。先按下 Browse 按鈕，選擇要讓 Git 管控的資料夾。按鈕左邊的欄位會顯示該資料夾路徑，完成後按下 Create 按鈕。

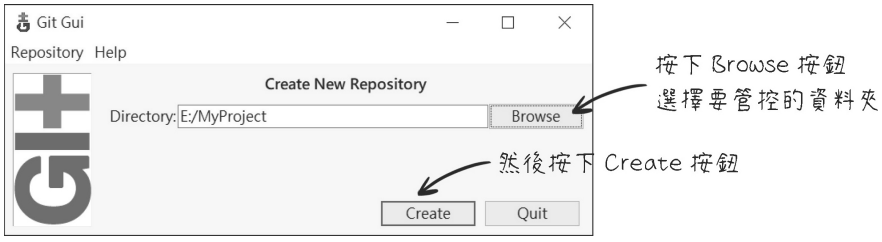

圖 2-4　選擇要管控的資料夾

3 STEP 接下來會出現圖 2-5 的操作畫面。這個畫面是後續學習的重點，它能夠讓我們執行很多 Git 的功能，但是第一次使用的時候要先設定操作者姓名和 Email 等資料。請依照圖 2-5 的說明操作，就會顯示圖 2-6 的畫面。

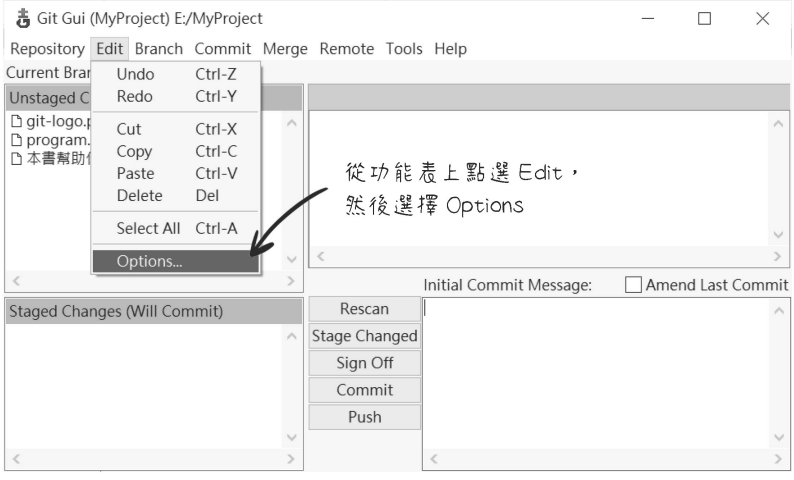

圖 2-5　開啓設定操作者姓名和 Email 的畫面

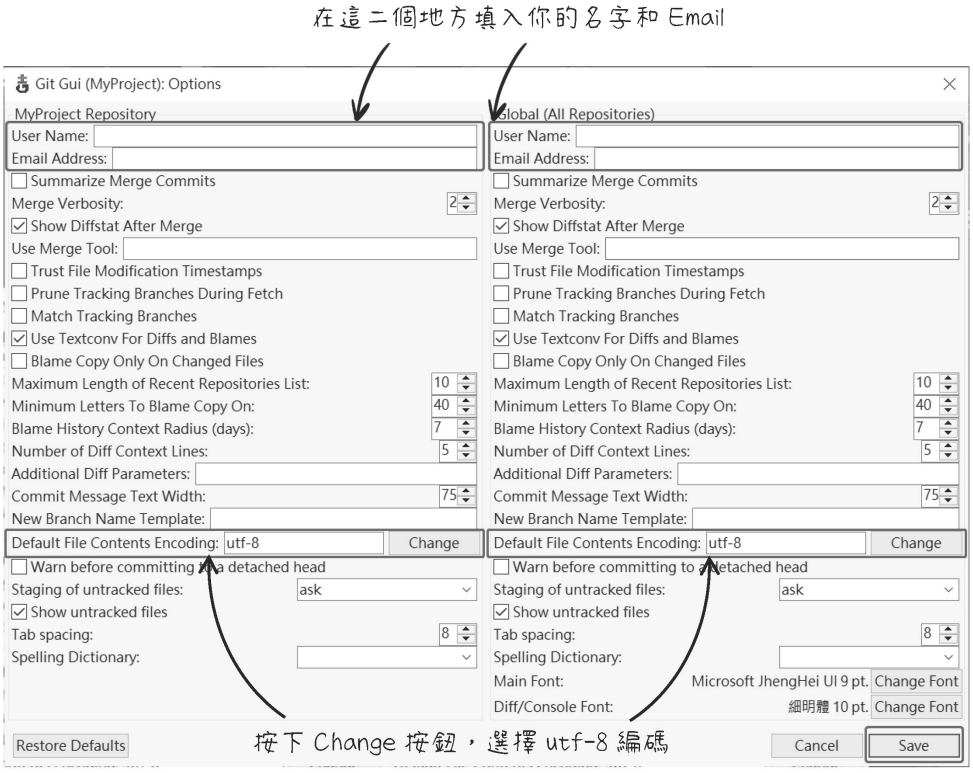

圖 2-6　填入操作者姓名和 Email

4 STEP 在圖 2-6 的對話盒填入你的名字和 Email。對話盒有左右二個部分，左邊的設定只針對目前管控的這個資料夾有效，右邊的設定會套用到未來所有被 Git 管控的資料夾。另外還要把檔案的編碼方式設為 utf-8。完成後按下右下角的 Save 按鈕。

5 STEP 現在我們要把資料夾的內容送進 Git 檔案庫儲存，這個過程需要三個步驟，請參考圖 2-7 的說明。首先要找出有更動的檔案，包括：新檔案、內容有修改的檔案和被刪除的檔案。檔案是否有更動是透過比對資料夾的內容和 Git 檔案庫內儲存的內容來決定。如果該檔案不存在 Git 檔案庫裡頭，表示它是一個新檔案。如果該檔案的內容，和 Git 檔案庫裡頭儲存的不一致，表示它被修改過。如果 Git 檔案庫裡頭的檔案已經不存在資料夾中，表示該檔案已經被刪除。按下圖 2-8 的 Rescan 按鈕會執行比對，比對結果會顯示在畫面左上角的窗格。

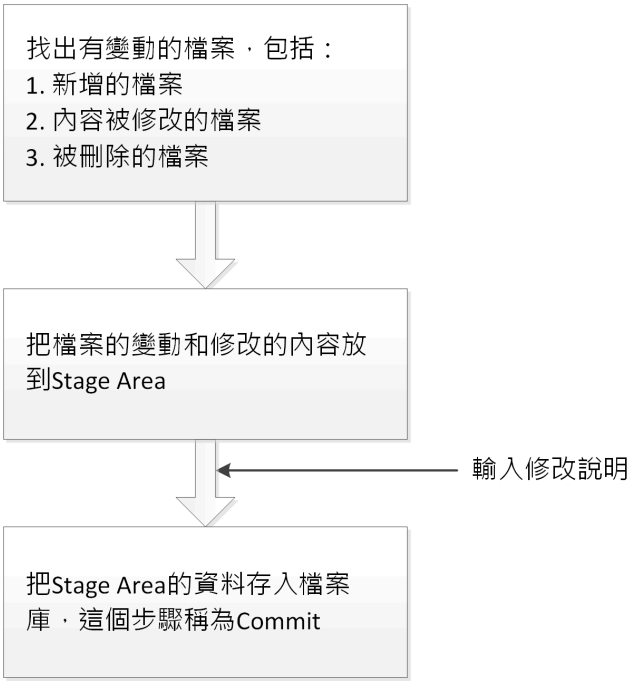

圖 2-7　Git 操作的三個基本步驟

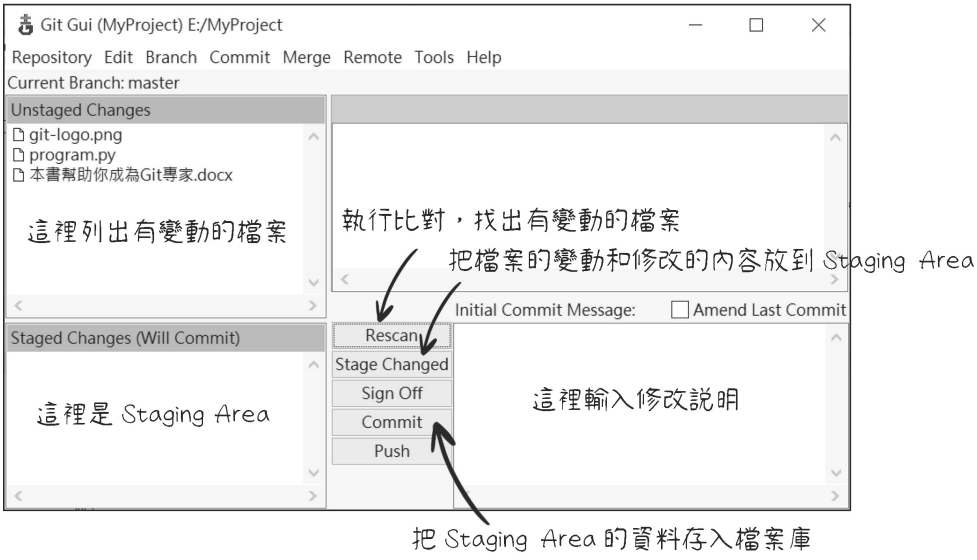

圖 2-8　Git GUI 程式操作說明

6 STEP 接著按下 Stage Changed 按鈕，螢幕會出現一個對話盒，說明有 3 個 Untracked files（還沒被追蹤的檔案，也就是新檔案的意思）。按下是，就會把修改的內容放到 Staging Area，也就是左下角的窗格。

7 STEP 在把 Staging Area 的內容存入檔案庫以前，必須先在右下角窗格輸入修改說明，例如「建立 Git 教學範例」。最後按下 Commit 按鈕，就會把 Staging Area 的內容存入 Git 檔案庫。所有窗格中的資料都會清空，表示所有更動的檔案都已經完成儲存。

8 STEP 接下來讓我們檢視儲存的結果。請參考圖 2-9 的操作畫面，選擇功能表的 Repository > Visualize All Branch History，就會出現另一個視窗，顯示 Git 檔案庫裡頭的 Commit 資訊，如圖 2-10。這個程式叫做 gitk，它也是我們要介紹的另一個主角。

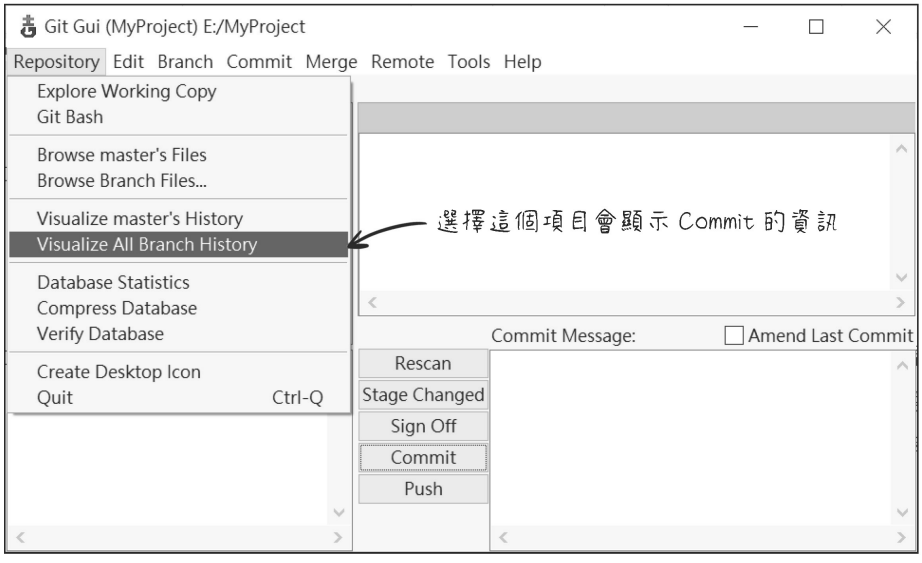

圖 2-9　啓動 Commit 檢視畫面

圖 2-10　用 gitk 程式檢視 Commit 資訊

現在讓我們設想一個狀況。如果在步驟 6 之後（也就是把檔案的更動放到 Staging Area），我們又去修改了某一個或是某幾個檔案，然後執行步驟 7，也就是把資料送進檔案庫儲存。這時候 Git 檔案庫裡頭儲存的是執行步驟 6 當時的內容，還是最後修改的內容？如果依照直覺來想，可能會覺得是最後修改的內容，但是正確答案是前者，也就是執行步驟 6 當時的內容。這是因為當我們按下 Stage Changed 按鈕時，是把當下檔案的變更放到 Staging Area。如果之後又去修改檔案，必須重新執行 Rescan 和 Stage Changed，才會包含最後修改的結果。這點很容易被初學者忽略，請特別留意。

補充

Git Repository 的廬山真面目

Git 檔案庫其實是一個名稱叫做「.git」的子資料夾。Windows 檔案總管預設會把它隱藏起來，不會顯示給我們看。如果要看到這些隱藏的檔案和資料夾，必須改變 Windows 檔案總管的檢視選項（參考圖 2-11），然後就會看到如圖 2-12 的結果。一旦打開 Windows 檔案總管的隱藏檔案和資料夾的顯示功能之後，要特別注意，不可以把「.git」資料夾刪除。因為所有的檔案修改紀錄都在裡頭。一旦刪除它，所有的歷史資料都會消失，除非你有把 Git 檔案庫上傳到 Git 伺服器。

圖 2-11　讓 Windows 檔案總管顯示隱藏的資料夾和檔案

圖 2-12　Git 檔案庫其實是一個名稱叫做「.git」的子資料夾

我們已經完成第一次 Git 操作體驗，過程中不需要輸入任何指令，全部用圖形介面的方式完成。這種作法比較簡單、方便。但是如同前面曾經提到過，指令才能夠發揮 Git 完整的功能，尤其是遇到某些特殊狀況的時候，就非得用 Git 指令才能處理。因此花點時間學習 Git 指令的用法是值得的！接下來我們要介紹如何用 Git 指令來達到和前面操作流程一樣的結果。

2-3 用指令操作 Git

Git 指令必須在 Git Bash 程式中執行。要啟動 Git Bash 最簡單的方法是開啟檔案總管，找到要讓 Git 管控的資料夾，然後用滑鼠右鍵點它，再從選單中選擇 Git Bash Here（參考圖 2-13），就會啟動 Git Bash，而且自動切換到該資料夾。圖 2-14 是 Git Bash 程式的執行畫面。只要在畫面最後一行的提示字元後面輸入指令，按下 Enter 鍵，就會執行該指令。

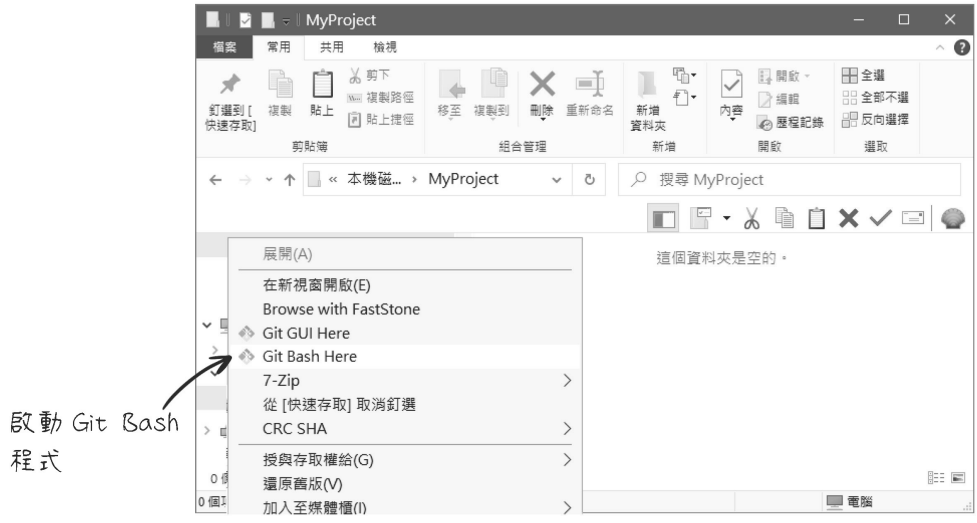

圖 2-13　啟動 Git Bash 程式

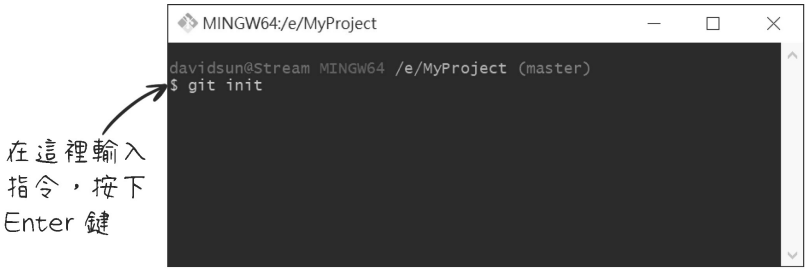

圖 2-14　Git Bash 程式執行畫面

另一種啟動 Git Bash 程式的方法，是從 Windows 程式集啟動它，然後在 Git Bash 視窗中，用 cd 指令切換到要管控的資料夾。假設我們的專案資料夾是 d 磁碟中的 MyProject，可以執行以下指令進行切換：

```
cd /d/MyProject
```

如果資料夾名稱中有空格，必須用單引號或是雙引號把資料夾路徑括起來，例如：

```
cd '/d/My Project'
```

接下來我們要用 Git 指令做出和前一小節的操作流程相同的結果。首先，我們要讓資料夾回復到還沒有被 Git 管控的狀態。這個不難，我們可以把前一個小節的資料夾中的檔案複製到一個新的資料夾，然後依照前面介紹的方法，在這個新資料夾中啟動 Git Bash 程式，再依照下列步驟操作。

1 STEP 執行以下指令完成 Git 檔案庫初始化：

```
git init
```

執行之後會顯示：

```
Initialized empty Git repository in 資料夾路徑/.git/
```

以上訊息最後是「.git」子資料夾，這就是我們在上一節解釋的情況。

2 STEP 執行以下指令檢視資料夾的狀態：

```
git status
```

以我們前一節的資料夾範例來說，會顯示：

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        git-logo.png
        program.py
"\346\234\254\346\233\270\345\271\253\345\212\251\344\275\24
0\346\210\220\347\202\272Git\345\260\210\345\256\266.docx"

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

請留意粗體字的部分，它列出三個 Untracked files，和前一小節的結果一樣。其中有一個檔案是中文檔名，但是 Git Bash 程式無法正確顯示中文。不過這不會造成任何問題，因為 Git 還是可以正常運作。上面訊息的最後一行還提示我們可以用 git add 指令來處理！

3 STEP 接下來要把這三個檔案的內容放到 Staging Area，這個動作的指令是：

```
git add -A
```

指令最後的「-A」是指令選項。它會把新檔案、有修改的檔案和刪除的檔案全部放到 Staging Area。

4 STEP 再一次執行步驟 2 的「git status」指令，會顯示以下訊息：

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   git-logo.png
        new file:   program.py
        new file:   "\346\234\254\346\233\270\345\271\253\345\
212\251\344\275\240\346\210\220\347\202\272Git\345\260\210\3
45\256\266.docx"
```

粗體字部分同樣列出前面提到的三個檔案，但是這三個檔案已經被標示為 new file，並且已經準備好要送進 Git 檔案庫儲存。這是上一個步驟的「git add」指令做的事。

5 STEP 執行下列指令，把 Staging Area 的資料送進檔案庫儲存：

```
git commit -m '修改說明'
```

6 STEP 再一次執行「git status」指令，會顯示下列訊息：

```
On branch master
nothing to commit, working tree clean
```

它的意思是說：我們目前是在 master 分支，不需要執行 commit，資料夾內容沒有新的變更。

完成以上步驟之後，就會得到和上一個小節相同的結果。但是現在讀者心中可能有一個疑問：用指令要怎麼看 Commit 資訊？可不可以做到類似前一小節 gitk 程式的效果？答案是肯定的，下列指令會列出 Git 檔案庫裡頭的 Commit：

```
git log
```

以前面建立的 Git 檔案庫來說，執行上述指令會顯示以下訊息：

```
commit 6798aff14eea9bd58276917e9af63eac4b39ca81 (HEAD -> master)
Author: peter <peter@gmail.com>
Date:   Sun Apr 30 15:52:43 2023 +0800

    建立 Git 教學專案
```

這段訊息的第一行有一串很長的十六進位數字，它是這個 Commit 的識別碼，我們會在單元 5 說明它的用途。從第二行開始依序是操作者、日期和 Commit 說明。

前面的「git log」指令其實沒有畫出類似 gitk 程式的結果，它只是把 Commit 依照時間先後順序列出來。我們可以在指令後面加上選項來控制它的輸出方式，以下是二個最常用的組合：

```
git log --graph
git log --graph --oneline
```

「--graph」選項會在每一個 Commit 前面加一個星號，做出類似 gitk 程式的效果。「--oneline」選項會限制每一個 Commit 資訊只佔一行。例如下列訊息是執行上面第一行指令的結果：

```
* commit 6798aff14eea9bd58276917e9af63eac4b39ca81 (HEAD -> master)
  Author: peter <peter@gmail.com>
  Date:   Sun Apr 30 15:52:43 2023 +0800

      建立 Git 教學專案
```

如果執行第二行指令，會得到下列結果：

```
* 6798aff (HEAD -> master) 建立 Git 教學專案
```

本書附錄提供常用的 Git 指令使用說明供讀者參考。

Git 指令和圖形介面是相輔相成的操作方式。我們也可以用指令啟動 Git GUI 和 gitk 程式：

```
git gui&
gitk&
```

第一行是啟動 Git GUI，指令最後的「&」是避免指令視窗被鎖住，第二行指令是啟動 gitk 程式。反過來，也可以從 Git GUI 功能表的 Repository > Git Bash 啟動指令視窗，如圖 2-15。

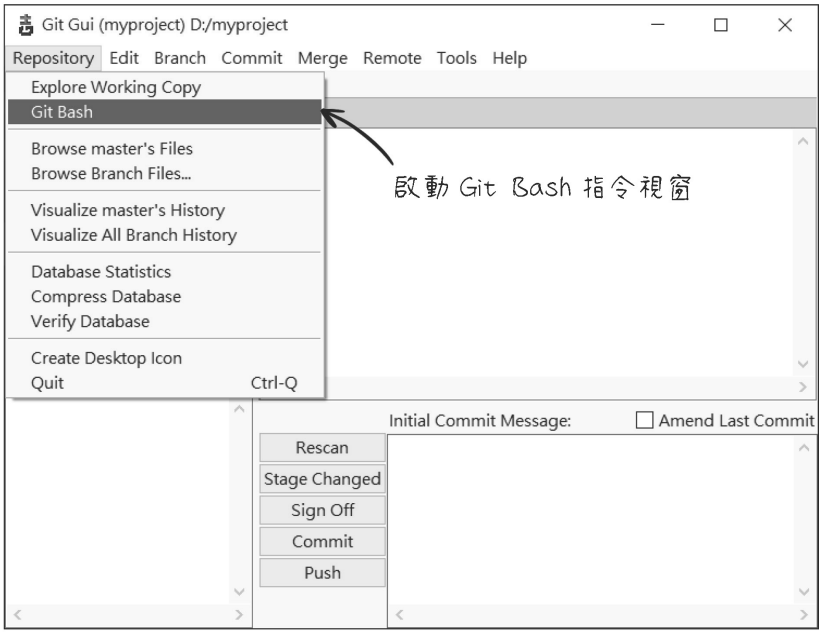

圖 2-15　從 Git GUI 啓動 Git Bash 指令視窗

Git 指令可以和前一小節介紹的圖形操作交互運用。例如執行「git init」之後，可以啟動 Git GUI，選擇 Open Existing Repository，打開 Git 管控的資料夾，再進行操作。或是在 Git GUI 按下 Stage Changed 按鈕之後，再用「git commit」指令把 Staging Area 的資料存入檔案庫。

基本上，比較常用的操作都可以用 Git GUI 完成。如果遇到特殊情況，無法用 Git GUI 處理，這時候再開啟 Git Bash 視窗來執行 Git 指令。

補充

顯示 Git 指令說明

單獨執行「git」指令會顯示輔助說明。執行「git help -a」則顯示完整的指令清單。執行「git 指令 --help」（例如「git init --help」）會顯示該指令的網頁說明檔。

如果指令太長，想要換到下一行繼續輸入，可以用反斜線字元'\'結尾，然後按下 Enter 鍵，繼續輸入。

補充

Git 的術語

執行 Git 指令時，常常會顯示一些訊息，訊息中可能會出現一些專用術語，以下是一些常見術語的說明：

1. Working Tree

   它是指目前這個資料夾。因為資料夾是一個樹狀結構，所以 Git 將它稱為 Working Tree。

2. Index

   它就是 Staging Area，也就是用來儲存檔案修改的內容。把檔案修改的內容放到 Index 稱為 Stage，從 Index 移除稱為 Unstage。