

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 Git 版本控制

我們平時在編輯檔案時，常會為了確保資料修改後能恢復到編輯前的狀態，我們常會直接複製編輯前的檔案並且直接透過日期編號為該版本命名，如原始檔案為文檔.txt，過程中可能變為下圖 0-1：

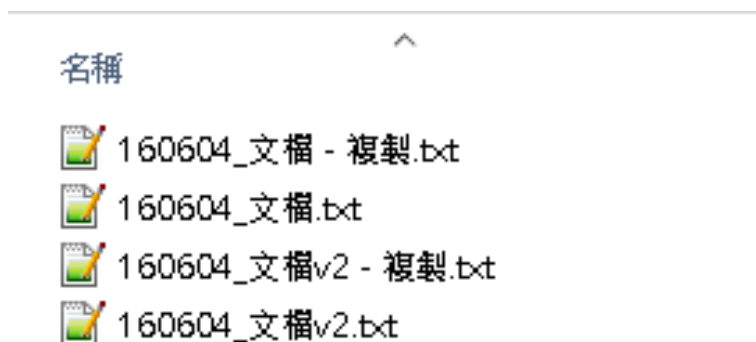


圖 0-1 雜亂的檔案管理

當需要還原時，才發現這些命名方式完全無助於找到想要的還原版本，不只相當麻煩，修改哪些細節內容完全不記得，更別說是多人開發團隊之中，與他人共同開發一個程式也是很常遇到的事，如果採用這種命名，相信大部分的人都會感到頭痛。

因此，這時候版本控制就非常重要，如圖 0-2 所示，版本控制是系統開發的標準做法，它能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開發流程。此外版本控制也能藉此在開發的過程中，確保不同人都能編輯同個程式，並能達到彼此同步。

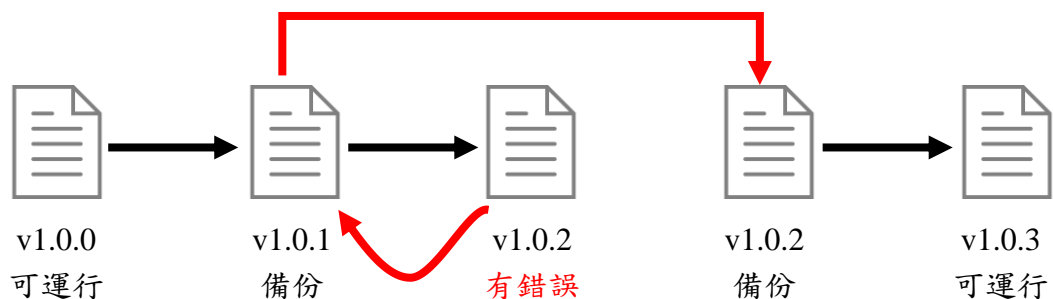


圖 0-2 透過版本控制保留檔案備份

0.1.1 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 0-3 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，但是如果需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。

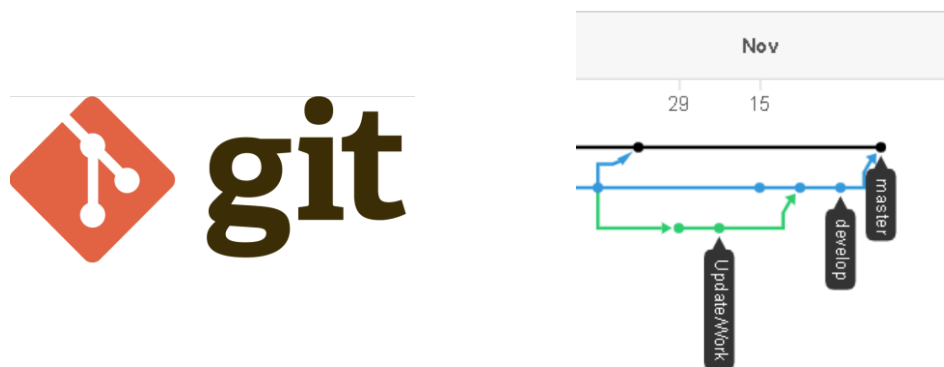


圖 0-3 Git（左）與版本控制（右）

0.1.2 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 0-4 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。

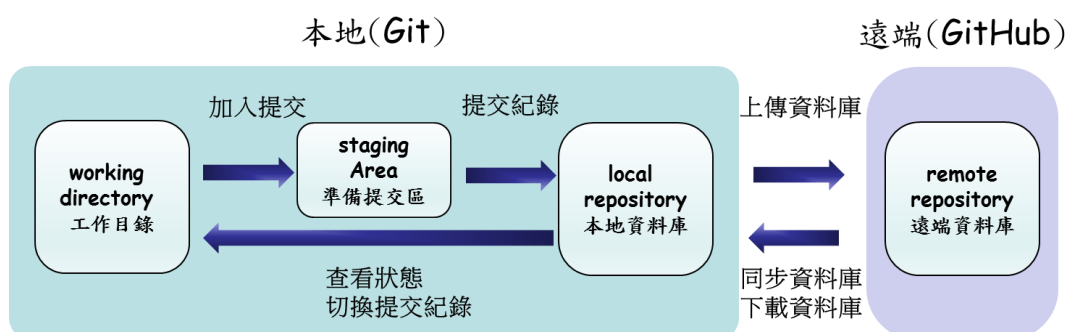


圖 0-4 使用 Git 備份資料到 GitHub

- **working directory (工作目錄)**：工作目錄主要是存放要被版本控制的檔案資料夾。我們可以選擇一個一般資料夾並在資料夾內建立 git 的資料庫，就能把該資料夾變成 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。當在工作目錄下的檔案有進行變更，且我們希望能提交這些變更，我們會將這些資料存入準備提交區之中，這狀態下只有標記那些資料要被提交，但還並未實際的做提交紀錄。
- **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 0-5 所示。若使用 git 的控制台查看此目錄可以看到圖 0-6 的路徑後增加了一個[master]的標籤，表示有偵測到資料庫，如圖 0-6 所示。

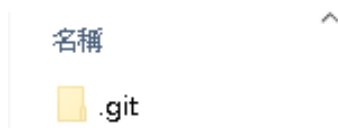


圖 0-5 目錄中的.git目錄

```
C:\Users\black\Documents\GitHub\MyProject> git init
Initialized empty Git repository in C:/Users/black/Documents/GitHub/MyProject/.git/
C:\Users\black\Documents\GitHub\MyProject [master]>
```

圖 0-6 [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 0-7 所示。

```
$ git status
```

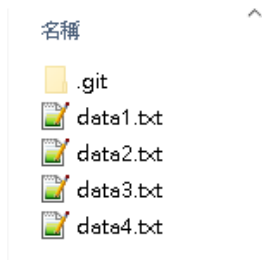


圖 0-7 工作目錄

```
C:\Users\black\Documents\GitHub\MyProject [master]> git status
On branch master

Initial commit

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

    data1.txt
    data2.txt
    data3.txt
    data4.txt

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\black\Documents\GitHub\MyProject [master +4 ~0 -0 !]>
```

圖 0-8 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，且 `[master]` 標籤多了一段紅色數字，表示偵測到有異動的檔案個數，如圖 0-8 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 .
```

```
C:\Users\black\Documents\GitHub\MyProject [master +4 ^B -0 ^I]> git add .
C:\Users\black\Documents\GitHub\MyProject [master +4 ^B -0 ^I]>
```

圖 0-9 加入檔案變更提交

執行指令之後，可以發現[master]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 0-9 所示。

4) 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 0-10 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。

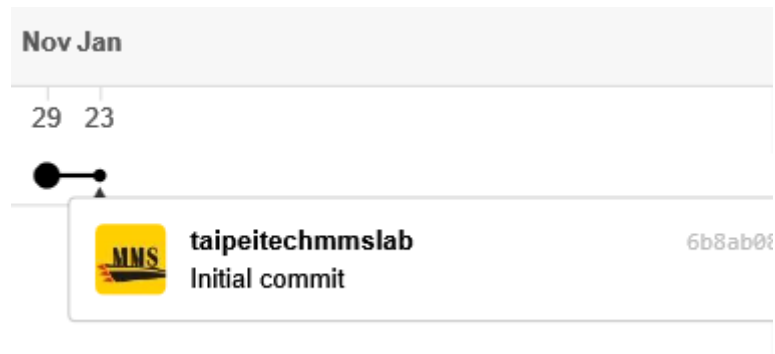


圖 0-10 GitHub 上的提交訊息

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"
```

```
nothing added to commit but untracked files present (use "git add" to track)
C:\Users\black\Documents\GitHub\MyProject [master +4 ^B -0 ^I]> git add .
C:\Users\black\Documents\GitHub\MyProject [master +4 ^B -0 ^I]> git commit -m "加入檔案"
[master (root-commit) a85b75b1] 加入檔案
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 data1.txt
 create mode 100644 data2.txt
 create mode 100644 data3.txt
 create mode 100644 data4.txt
Warning: Your console font probably doesn't support Unicode. If you experience strange characters in the output, consider switching to a TrueType font such as Consolas!
C:\Users\black\Documents\GitHub\MyProject [master]>
```

圖 0-11 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 0-11 所示。當標籤後的綠文字消失，代表工作目錄與本地資料庫已經同步。

5) 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 0-12、0-13 所示。

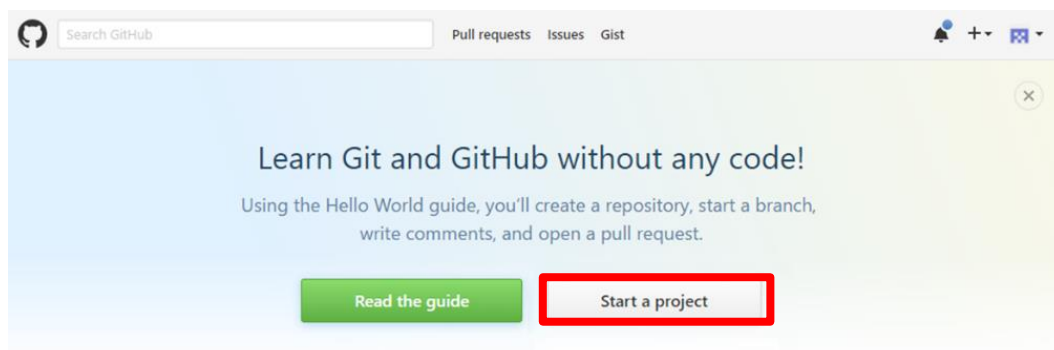



圖 0-12 點選 Start a project 建立新專案

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  / Repository name: ✓

Great repository names are short and memorable. Need inspiration? How about [upgraded-dollop](#).

Description (optional):

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: | Add a license: ⓘ

圖 0-13 輸入專案名稱並按下「Create repository」

建立資料庫後，如圖 0-14 所示，會產生一個連結，之後上傳資料庫時會需要。

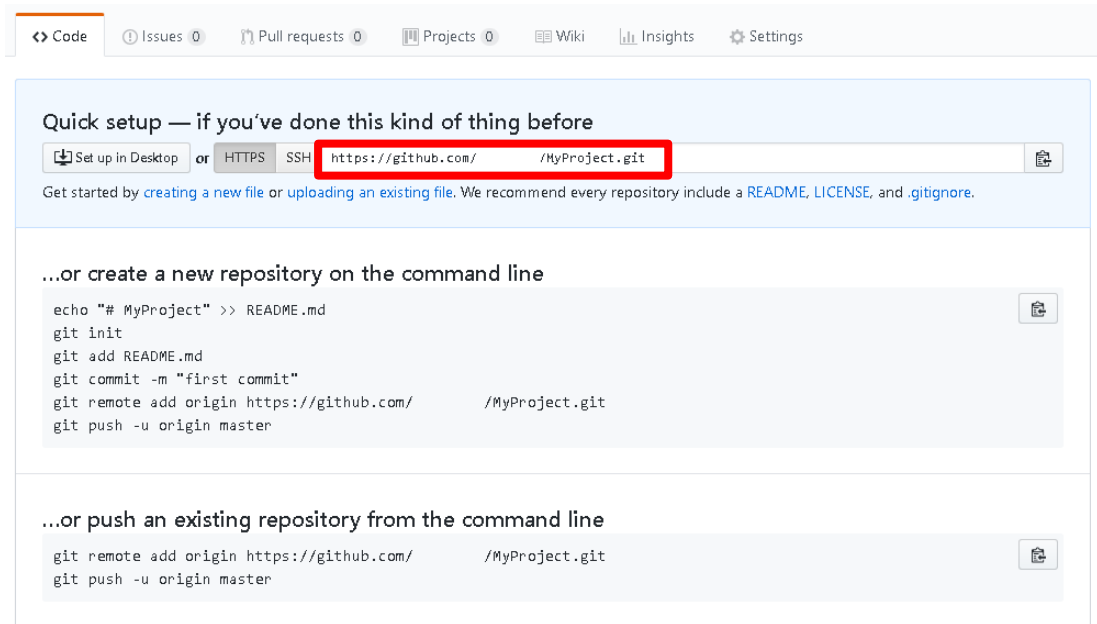


圖 0-14 GitHub 資料庫連結

6) 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 0-15 所示。

```
$ git remote add origin 資料庫連結
```

```
[master]> git remote add origin https://github.com/ /MyProject.git
```

圖 0-15 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 0-16 所示。

```
$ git push origin 標籤名稱
```

```
tHub\MyProject [master]> git push origin master
```

圖 0-16 同步資料到遠端資料庫

7) 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 0-17 所示。

```
$ git pull origin 標籤名稱
```

```
GitHub\MyProject [master]> git pull origin master
```

圖 0-17 從遠端資料庫同步資料到本地

8) 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 0-18 所示。

```
$ git clone 資料庫連結
```

```
$\GitHub> git clone https://github.com/ /MyProject.git
```

圖 0-18 複製遠端資料

9) 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 0-19 的圖形介面。

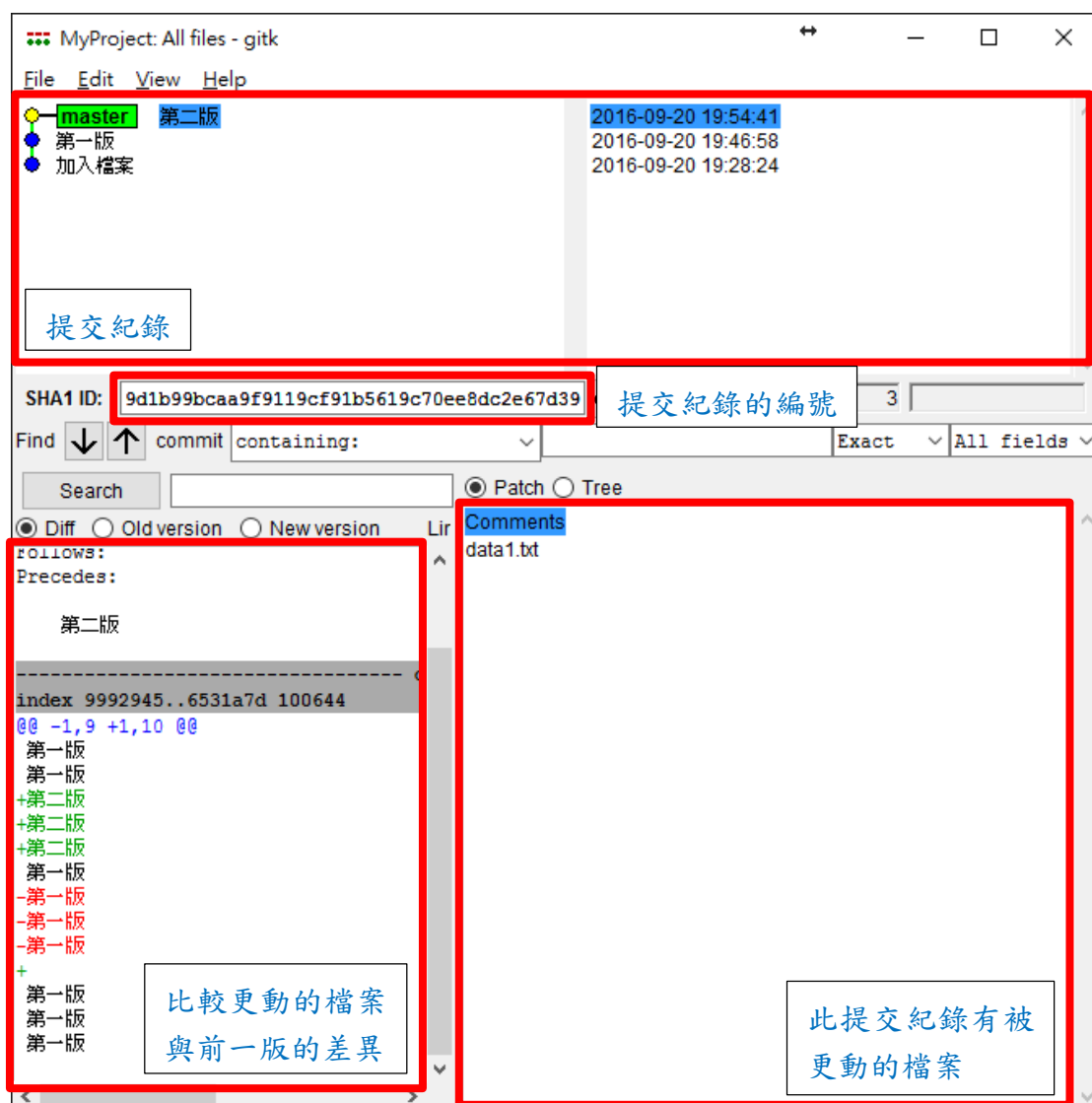


圖 0-19 Git 圖形化介面

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 0-20 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。

SHA1 ID: 9d1b99bcaa9f9119cf91b5619c70ee8dc2e67d39

圖 0-20 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。

10) 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 0-21 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
```

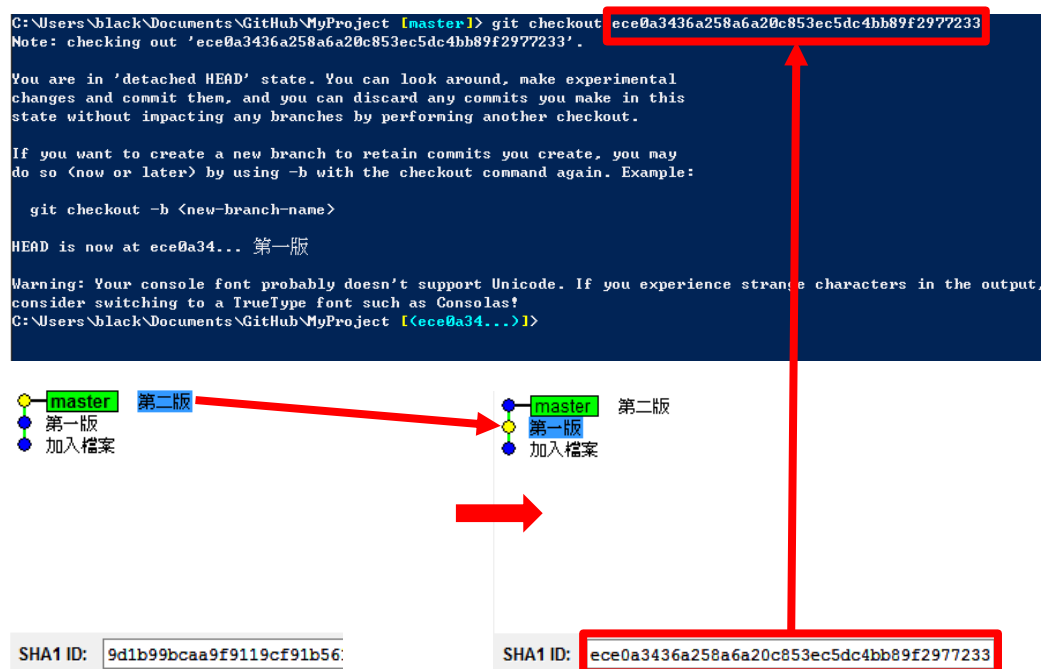


圖 0-21 切換提交紀錄

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。

0.2 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
- 實際練習 Git 與 GitHub 的基本使用情境
 - 情境 1：將撰寫完成的專案推送到 GitHub 上。
 - 情境 2：將某個專案複製到工作目錄下。

0.2.1 安裝 Git 使用環境 Git Bash

Step1 至 <https://git-for-windows.github.io/> 下載 Git 安裝檔，如圖 0-22 所示。

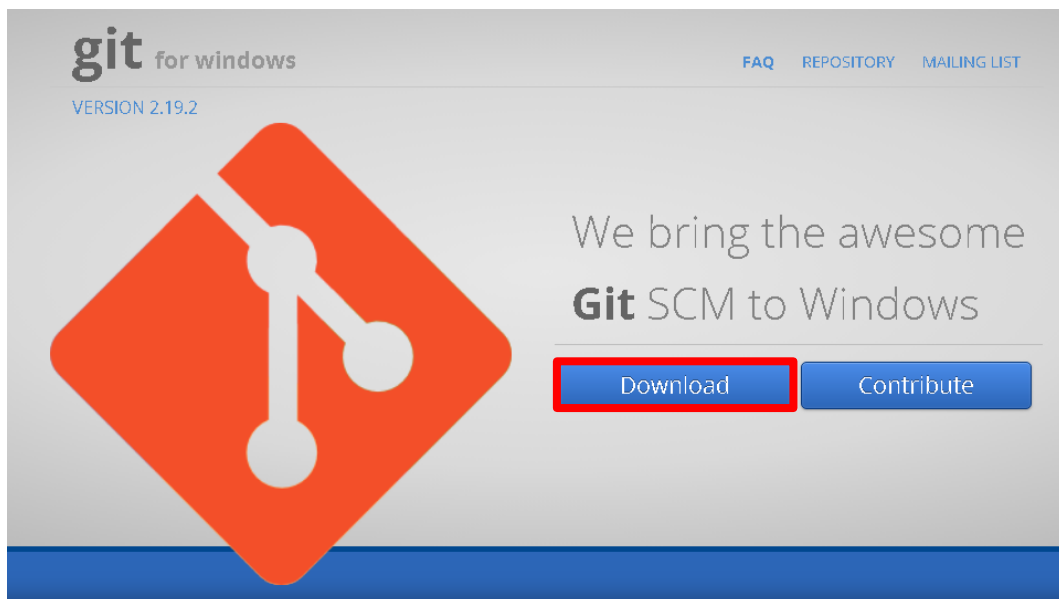


圖 0-22 下載 Git 安裝檔

Step2 開啟安裝檔，開始安裝 Git，如圖 0-23 所示。



圖 0-23 允許安裝 Git

Step3 閱讀並同意授權聲明，點擊「Next」，如圖 0-24 所示。



圖 0-24 Git 授權聲明

Step4 設定 Git 的安裝路徑，並點擊「Next」，如圖 0-25 所示。

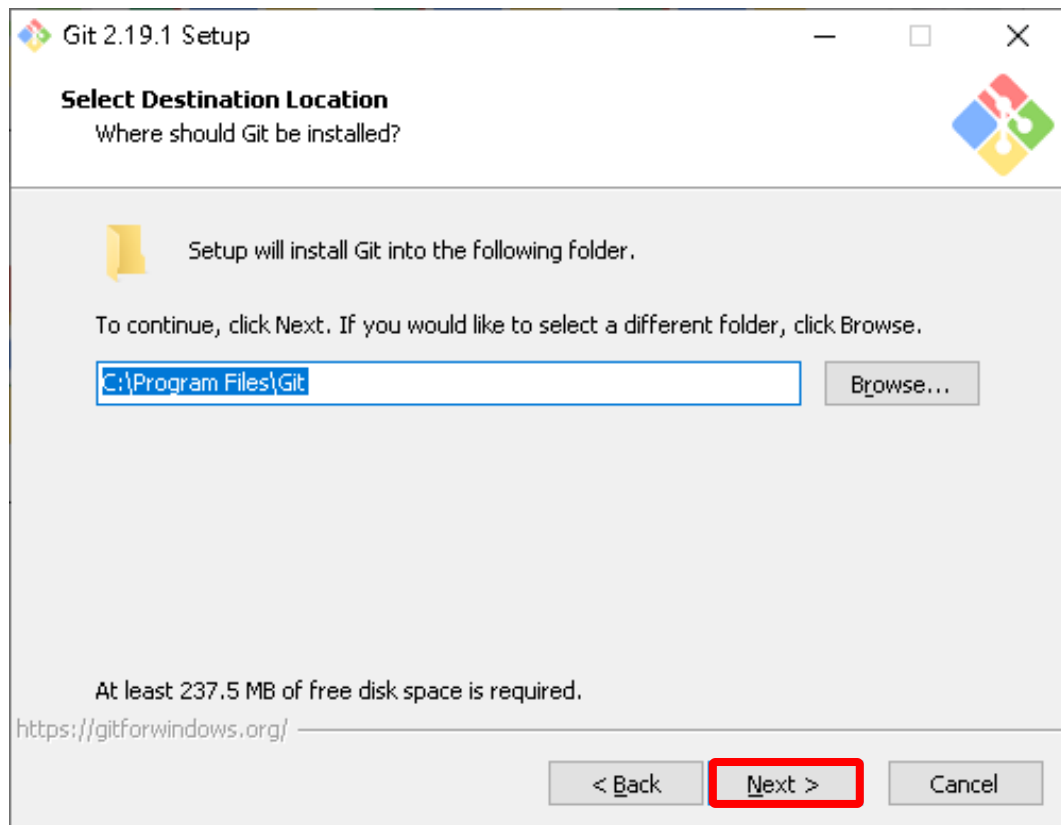


圖 0-25 選擇 Git 安裝路徑

Step5 勾選 On the Desktop，點擊「Next」，再點擊「Next」，如圖 0-26 所示。

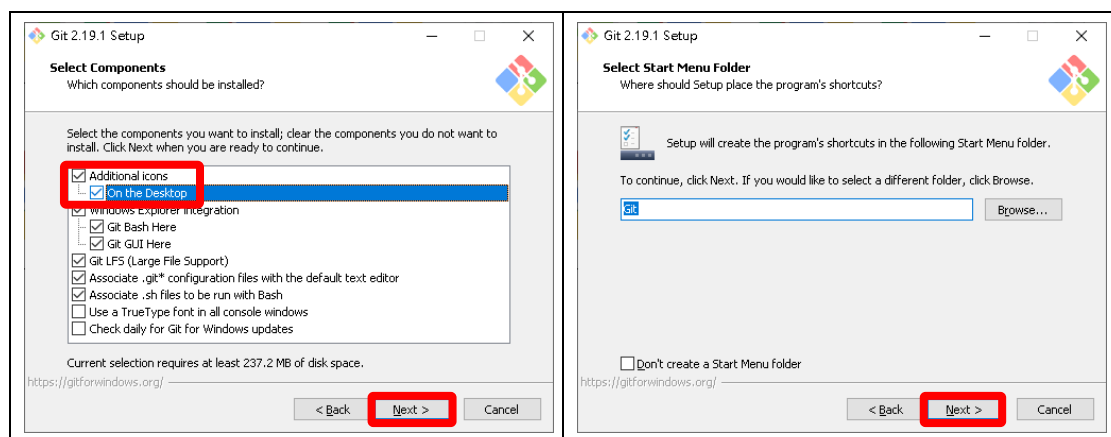


圖 0-26 勾選 On the Desktop（左）與設定開始目錄名稱（右）

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 0-27 所示。

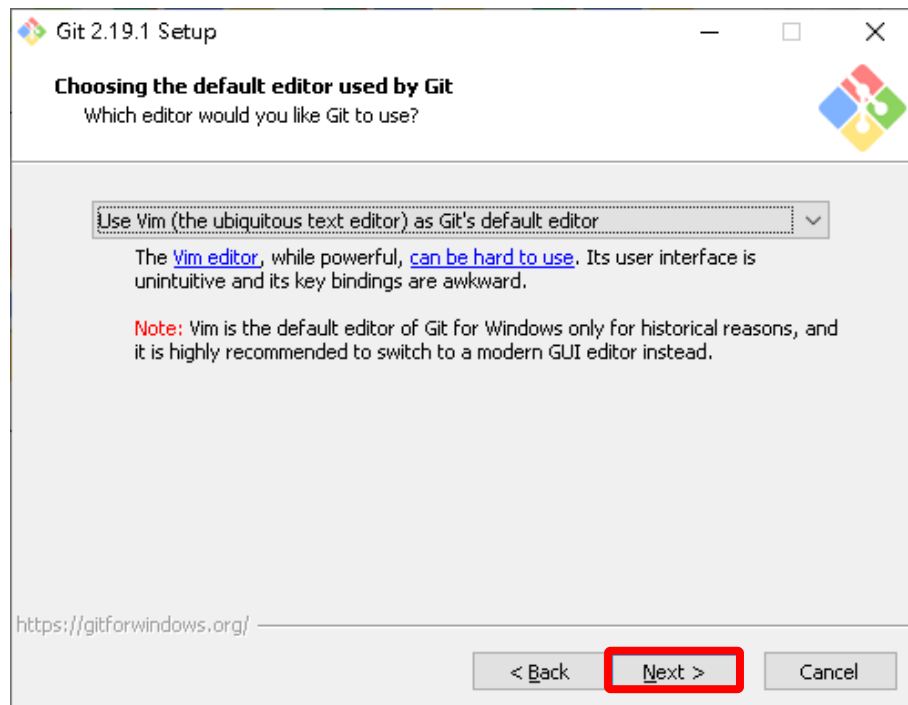


圖 0-27 選擇 Git 編輯器

Step7 設定 Git Bash，預設選擇 Use Git from Git Bash only，點擊「Next」，如圖 0-28 所示。

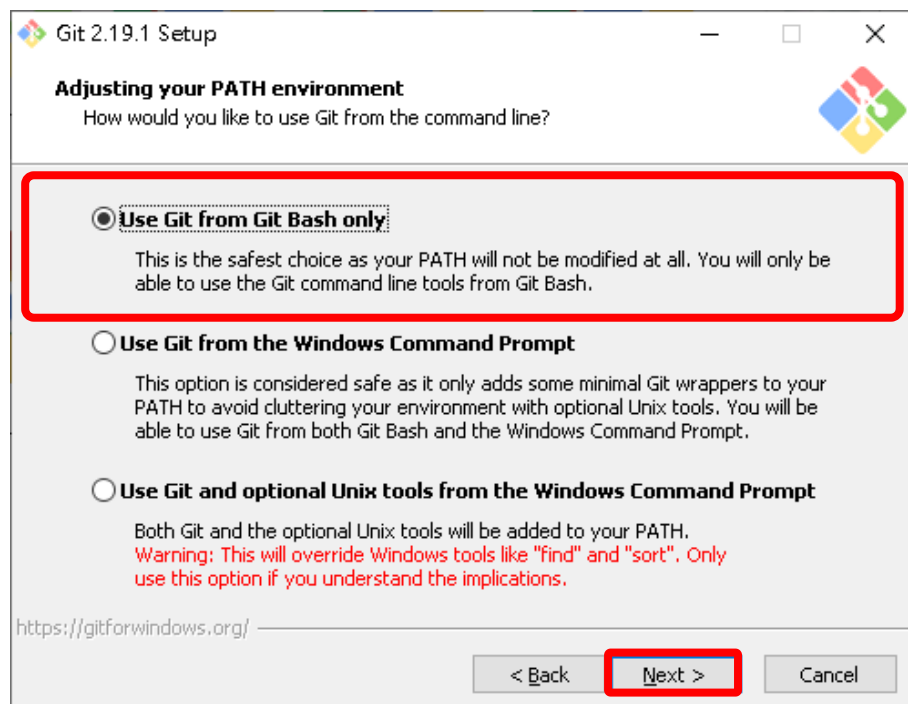


圖 0-28 設定 Git Bash

Step8 選擇 Use the OpenSSL library，點擊「Next」，如圖 0-29 所示。

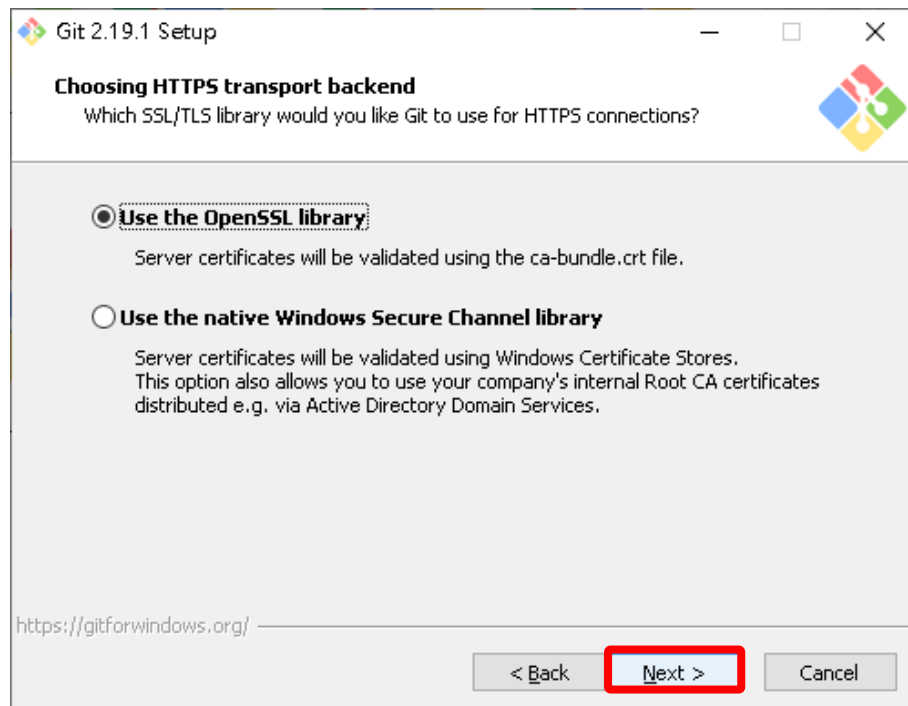


圖 0-29 選取 Use the OpenSSL library

Step9 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 0-30 所示。

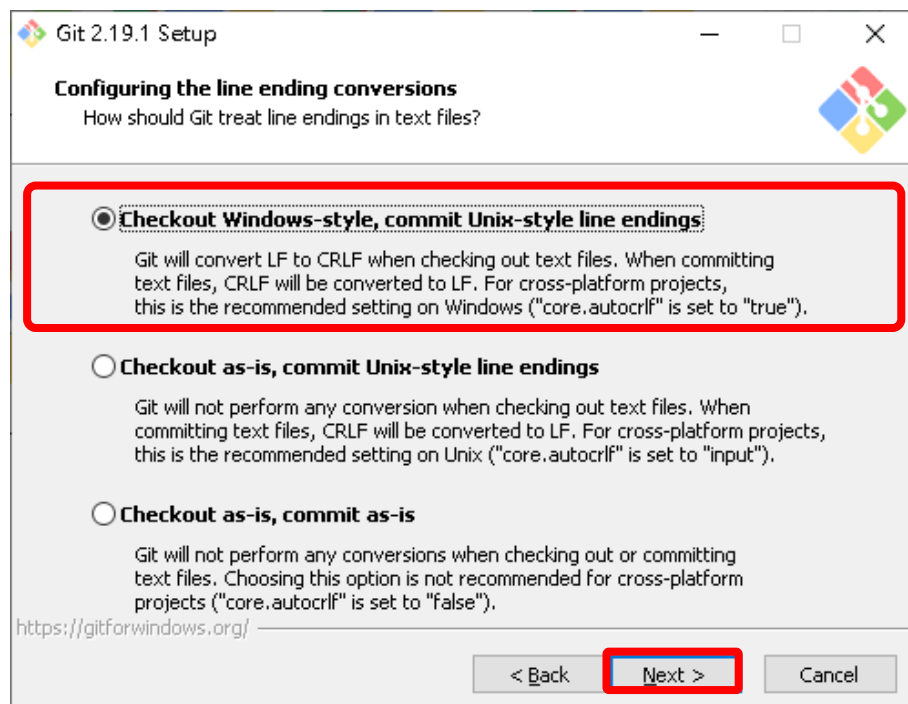


圖 0-30 設定結束符號

Step10 選擇 Use MinTTY，點擊「Next」，如圖 0-31 所示。

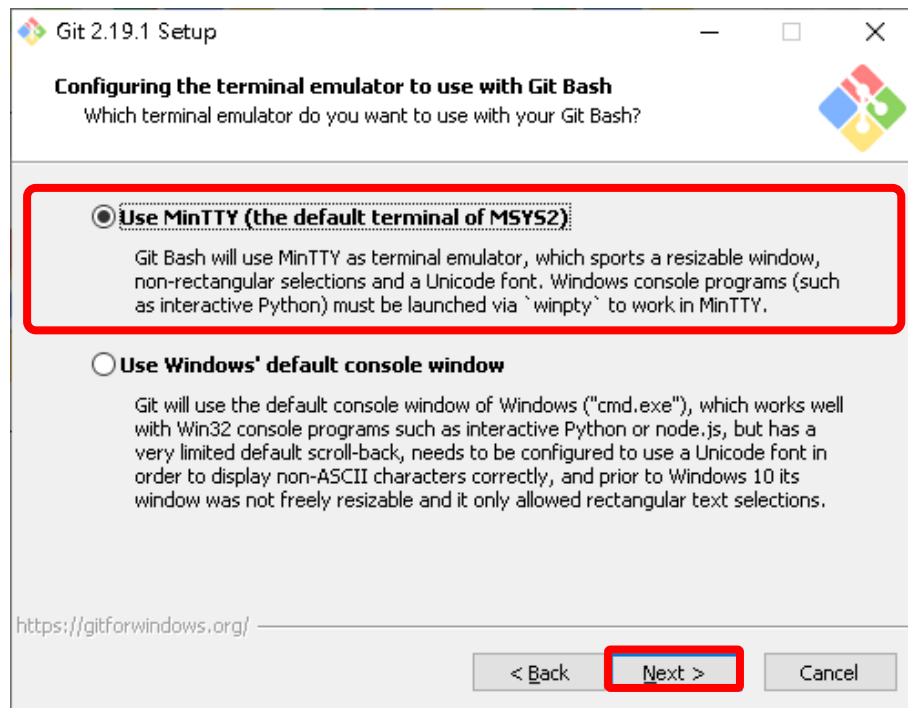


圖 0-31 選擇 Use MinTTY

Step11 選擇 Enable file system caching 與 Enable Git Credential Manager，點擊「Next」，如圖 0-32 所示。

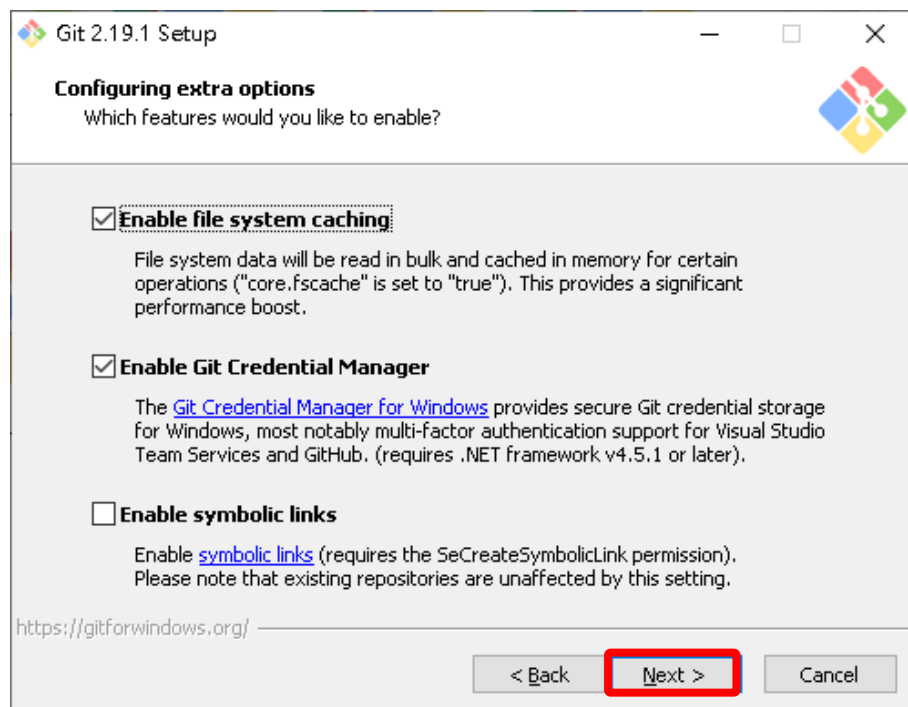


圖 0-32 選擇 Enable file system caching 與 Enable Git Credential Manager

Step12 直接點擊「Install」，如圖 0-33 所示。

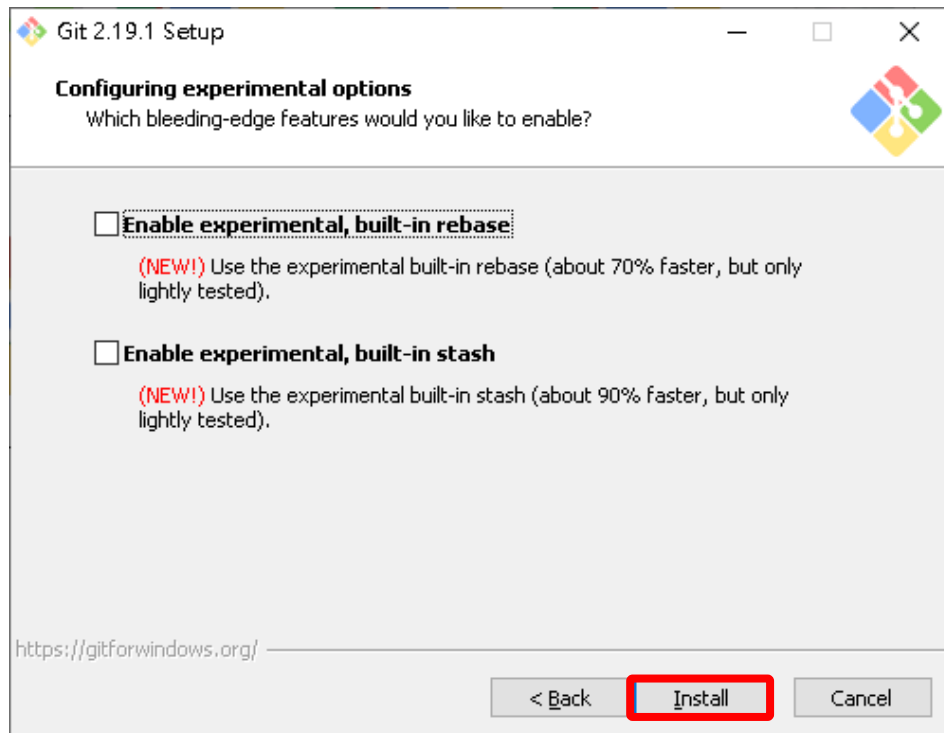


圖 0-33 開始安裝 Git

Step13 安裝完成後點擊「Finish」離開，如圖 0-34 所示。

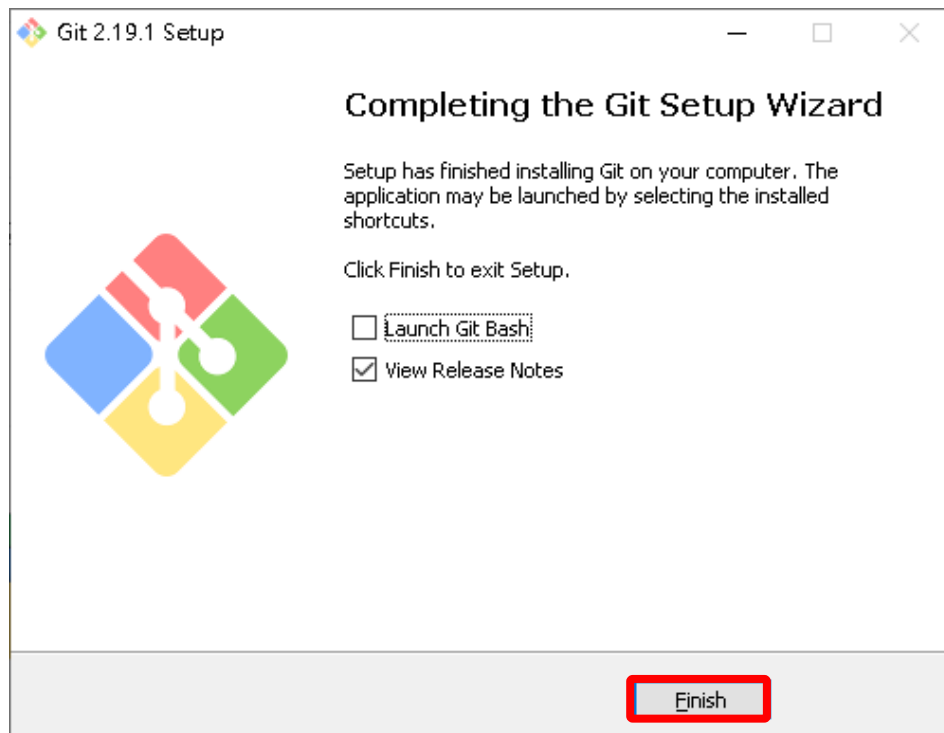


圖 0-34 安裝成功

Step14 到桌面點擊執行 Git Bash，畫面如圖 0-35 所示。

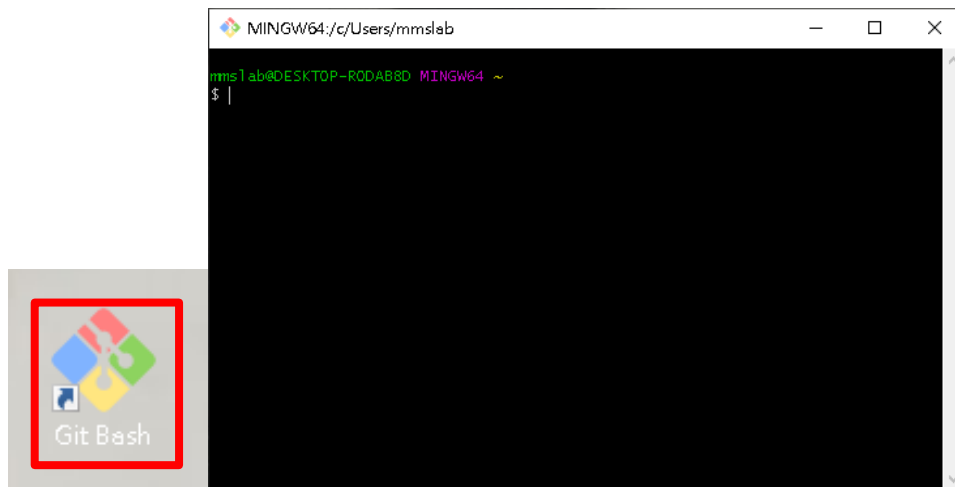


圖 0-35 開啟 Git Bash

Step15 在 git bash 內，設定自己的 user.email 和 user.name，如圖 0-36 所示。

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

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

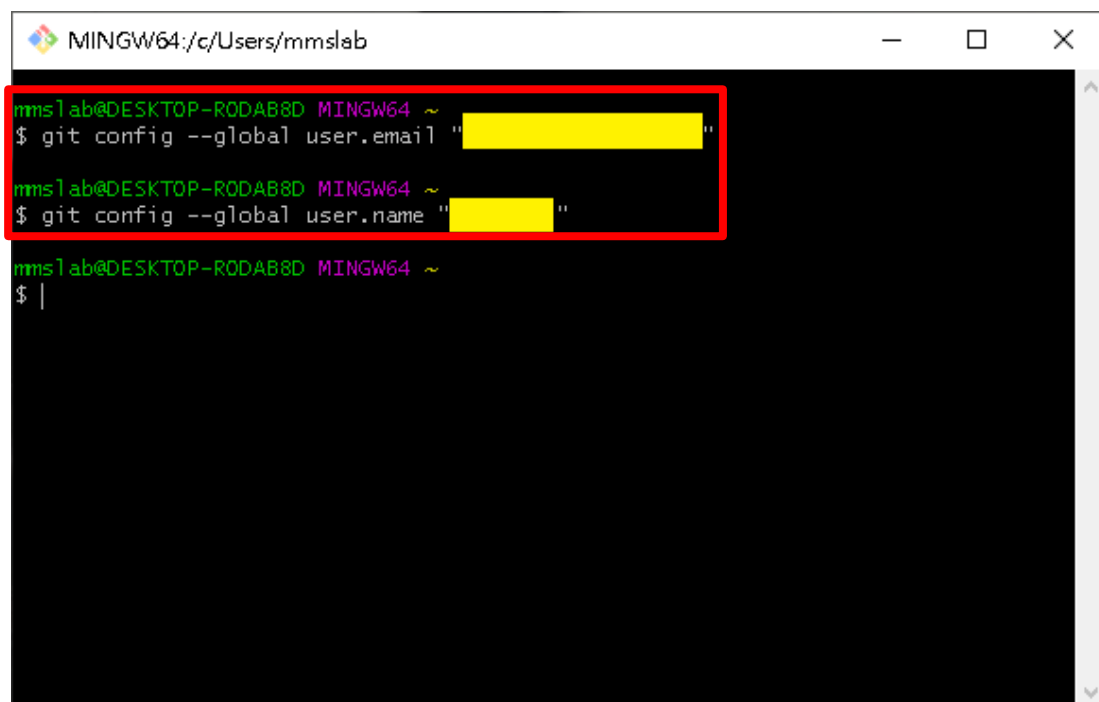
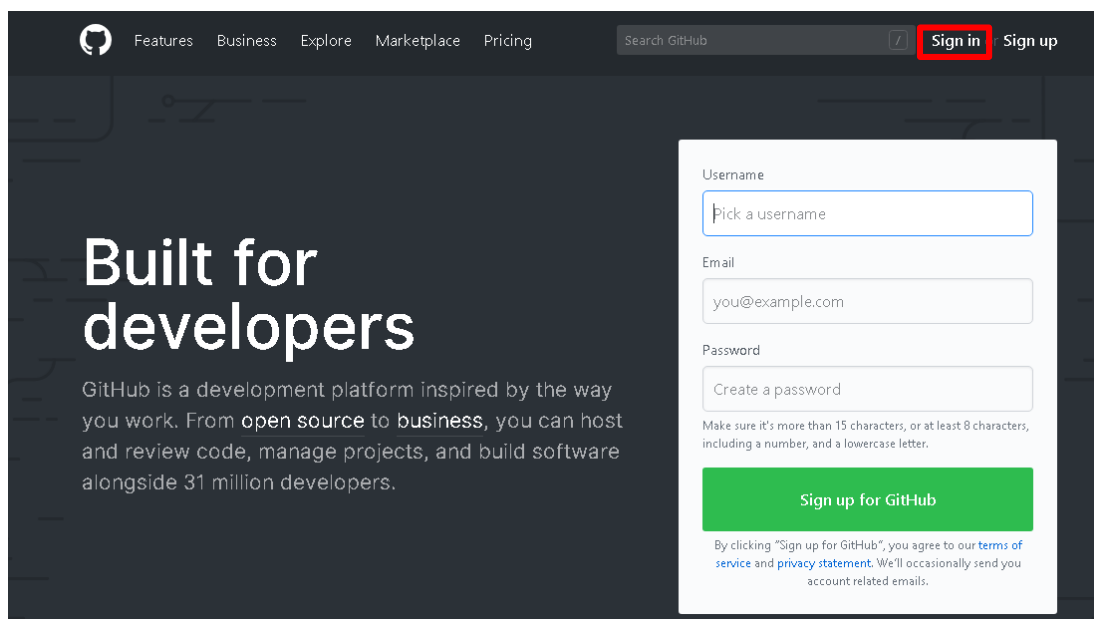


圖 0-36 設定使用者資料

0.2.2 註冊 GitHub 帳號與建立一個遠端資料庫

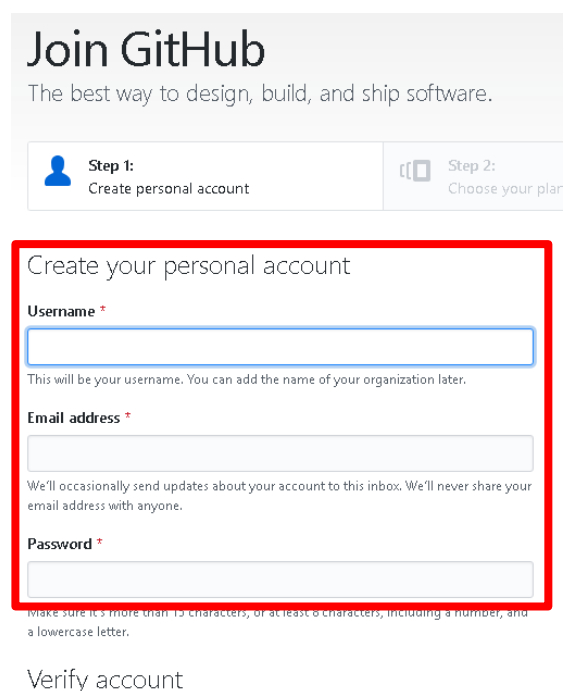
Step1 至 GitHub 官方網站 <https://github.com/> 註冊帳號，如圖 0-37 所示。



The screenshot shows the GitHub homepage with a dark background. On the left, the text "Built for developers" is prominent, followed by a description of GitHub as a development platform. On the right, there is a white sign-up form. The form has fields for "Username" (with a placeholder "Pick a username"), "Email" (with a placeholder "you@example.com"), and "Password" (with a placeholder "Create a password"). Below the password field, there is a note: "Make sure it's more than 15 characters, or at least 8 characters, including a number, and a lowercase letter." A green button labeled "Sign up for GitHub" is at the bottom of the form. Above the button, there is a line of text: "By clicking 'Sign up for GitHub', you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails."

圖 0-37 GitHub 官網

Step2 填寫註冊資料，並點擊「Create an account」，如圖 0-38 所示。



The screenshot shows the "Join GitHub" page. At the top, it says "Join GitHub" and "The best way to design, build, and ship software." Below this, there are two steps: "Step 1: Create personal account" and "Step 2: Choose your plan". The "Step 1" section is highlighted with a red box. It contains the title "Create your personal account" and three input fields: "Username" (with a red asterisk), "Email address" (with a red asterisk), and "Password" (with a red asterisk). Below the "Username" field, there is a note: "This will be your username. You can add the name of your organization later." Below the "Email address" field, there is a note: "We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone." Below the "Password" field, there is a note: "Make sure it's more than 15 characters, or at least 8 characters, including a number, and a lowercase letter." At the bottom of the form, there is a link "Verify account".

圖 0-38 欄位對應翻譯：

Username：使用者名稱。

Email Address：電子信箱。

Password：使用者密碼。

圖 0-38 註冊基本資料

Step3 選擇 GitHub 註冊帳號的方案，此處保持 Free Plan 的方案，然後直接點擊「Finish sign up」即可完成註冊，如圖 0-39 所示。

Welcome to GitHub

You've taken your first step into a larger world,

Completed
Set up a personal account

Step 2:
Choose your plan

Step 3:
Go to your dashboard

Choose your personal plan

Plan	Cost (view in TWD)	Private repositories	
Large	\$50/month	50	<button>Choose</button>
Medium	\$22/month	20	<button>Choose</button>
Small	\$12/month	10	<button>Choose</button>
Micro	\$7/month	5	<button>Choose</button>
Free	\$0/month	0	<button>Chosen</button>

Each plan includes:

- Unlimited** collaborators
- Unlimited** public repositories
- ✓ Free setup
- ✓ HTTPS Protection
- ✓ Email support
- ✓ Wikis, Issues, Pages, & more

Charges to your account will be made in **US Dollars**. Converted prices are provided as a convenience and are only an *estimate* based on *current* exchange rates. Local prices will change as the exchange rate fluctuates.
Don't worry, you can cancel or upgrade at any time.

- ☐ **Help me set up an organization next**
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations.](#)

Finish sign up

圖 0-39 選擇免費方案

Step4 創立一個遠端 Repository 在 GitHub 上，點選「Start a project」，如圖 0-40 所示。

Search GitHub

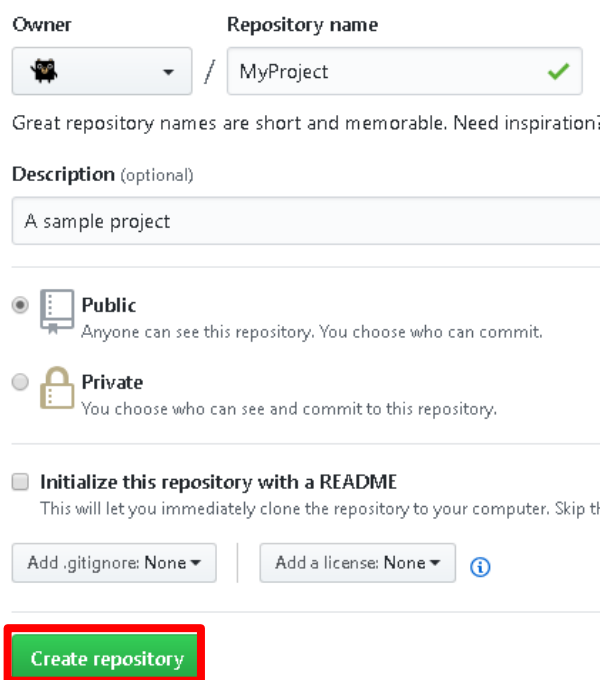
Pull requests Issues Gist

Learn Git and GitHub without any code!
Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide **Start a project**

圖 0-40 創建專案

Step5 輸入及設定 Repository 的資料及屬性，之後點擊「Create repository」，如圖 0-41 所示。



Owner

Repository name

Great repository names are short and memorable. Need inspiration:

Description (optional)

A sample project

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip tl

Add .gitignore: None

Add a license: None

Create repository

圖 0-41 欄位對應翻譯：

- 1) Repository name：專案名稱。
- 2) Description：專案描述。
- 3) Public/Private：專案隱私權。
- 4) Initialize this ... a README：是否加入初始 README 檔案。
- 5) Add .gitignore：忽略檔案設定。
- 6) Add a license：專案授權。

圖 0-41 設定專案屬性並按下 Create repository

Step6 完成遠端資料庫的建立，命名為「MyProject」，如圖 0-42 所示。



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

Set up in Desktop or HTTPS SSH https://github.com/ /MyProject.git

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

...or create a new repository on the command line

```
echo "# MyProject" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/ /MyProject.git
git push -u origin master
```

圖 0-42 專案建立成功

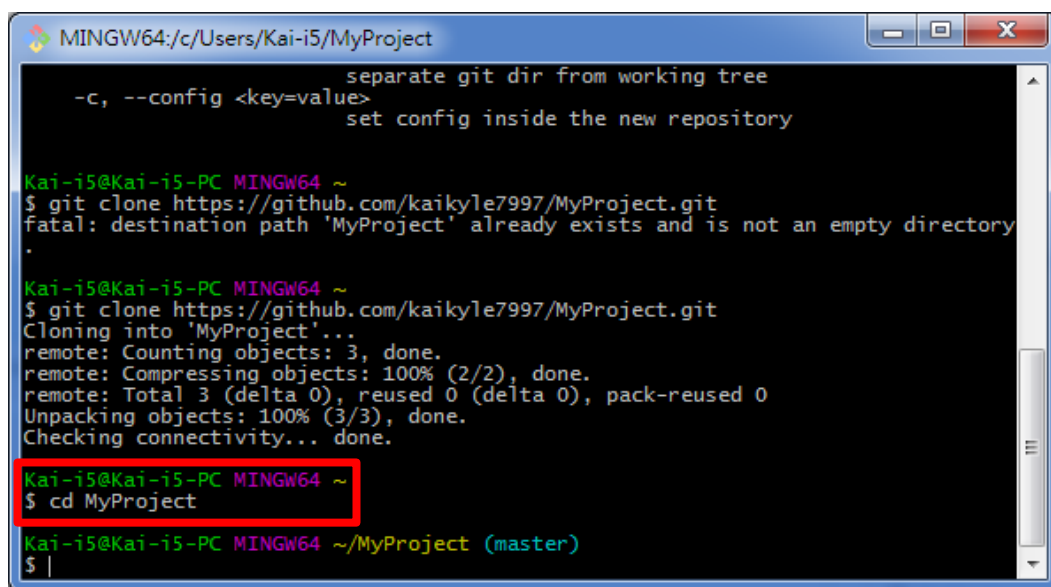
0.2.3 實際練習 Git 與 GitHub 的基本使用情境

情境 1、將撰寫完成的專案推送到 GitHub 上

Step1 切換到已經建立的資料夾作為工作目錄，如圖 0-43 所示。

執行下面指令，切換當前目錄到工作目錄路徑，如下：

```
$ cd MyProject
```



```
MINGW64:/c/Users/Kai-i5/MyProject
-c, --config <key=value>
                        separate git dir from working tree
                        set config inside the new repository

Kai-i5@Kai-i5-PC MINGW64 ~
$ git clone https://github.com/kaikyle7997/MyProject.git
fatal: destination path 'MyProject' already exists and is not an empty directory
.

Kai-i5@Kai-i5-PC MINGW64 ~
$ git clone https://github.com/kaikyle7997/MyProject.git
Cloning into 'MyProject'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
Kai-i5@Kai-i5-PC MINGW64 ~
$ cd MyProject
Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ |
```

圖 0-43 切換工作目錄

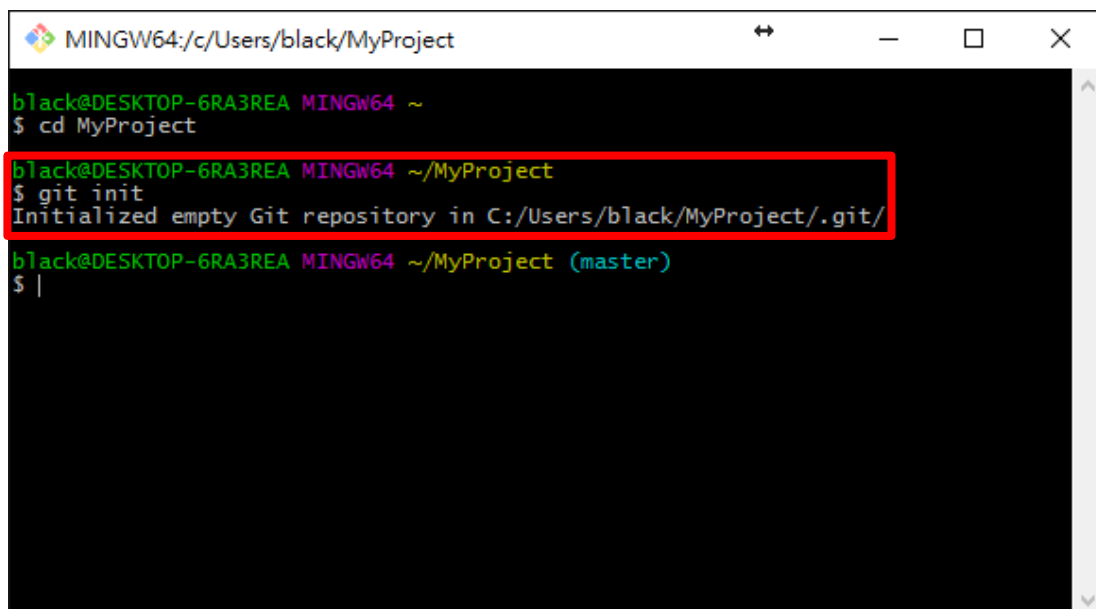
說明

「~」這符號代表使用者的目錄，為 C:\Users\[你的名稱]，所以 cd MyProject 後，當前目錄變為 C:\Users\[你的名稱]\MyProject\。

Step2 建立 Git 本地資料庫，如圖 0-44 所示。

在工作目錄輸入指令產生出本地資料庫：

```
$ git init
```



```
MINGW64:/c:/Users/black/MyProject
black@DESKTOP-6RA3REA MINGW64 ~
$ cd MyProject
black@DESKTOP-6RA3REA MINGW64 ~/MyProject
$ git init
Initialized empty Git repository in C:/Users/black/MyProject/.git/
black@DESKTOP-6RA3REA MINGW64 ~/MyProject (master)
$ |
```

圖 0-44 初始化 Git

Step3 將已經寫好的專案，移到切換的資料夾，為上傳專案做準備。

在檔案總管瀏覽路徑 C:\Users\[你的名稱]\MyProject\，並加入自己的程式專案及檔案到此工作目錄下，如圖 0-45 所示。

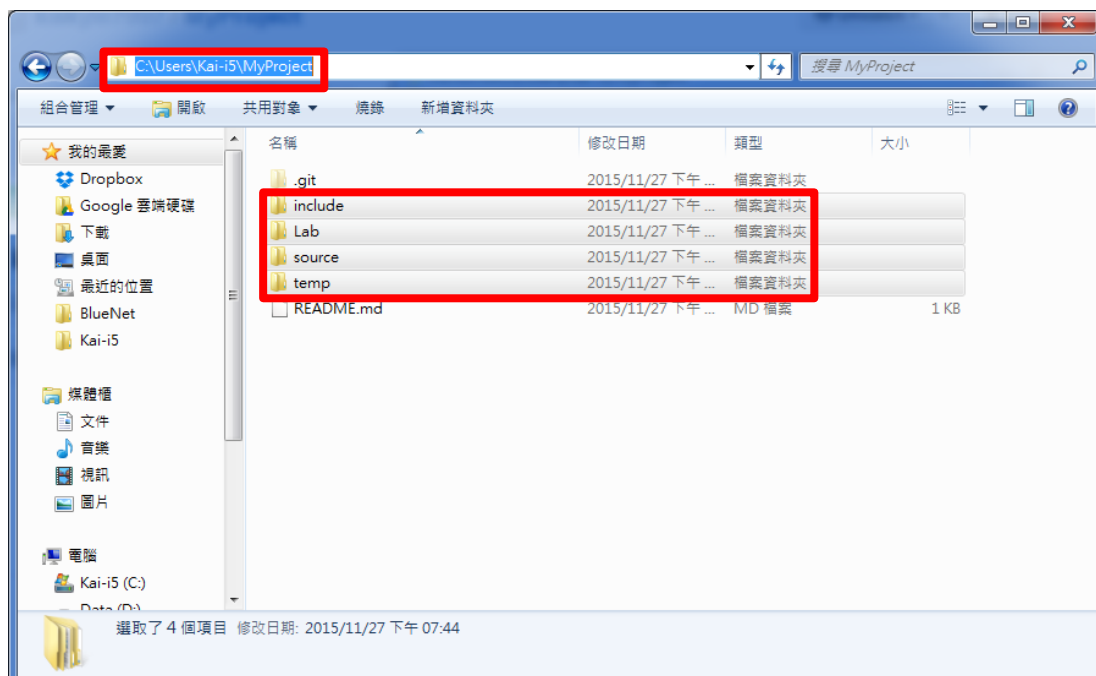
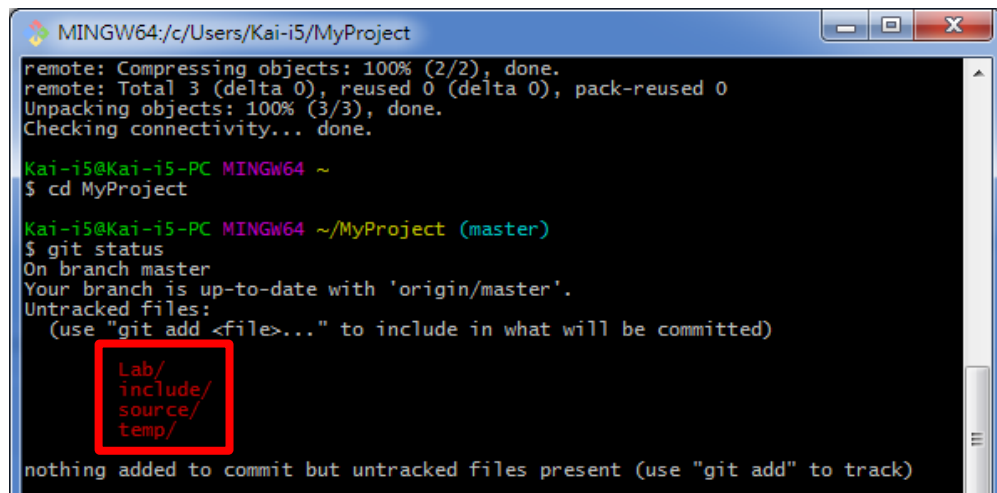


圖 0-45 加入你的專案

Step4 查看本地資料庫變更狀況，有紅色字體表示變更未被追蹤，如圖 0-46 所示。

```
$ git status
```



```
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.

Kai-i5@Kai-i5-PC MINGW64 ~
$ cd MyProject

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (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)

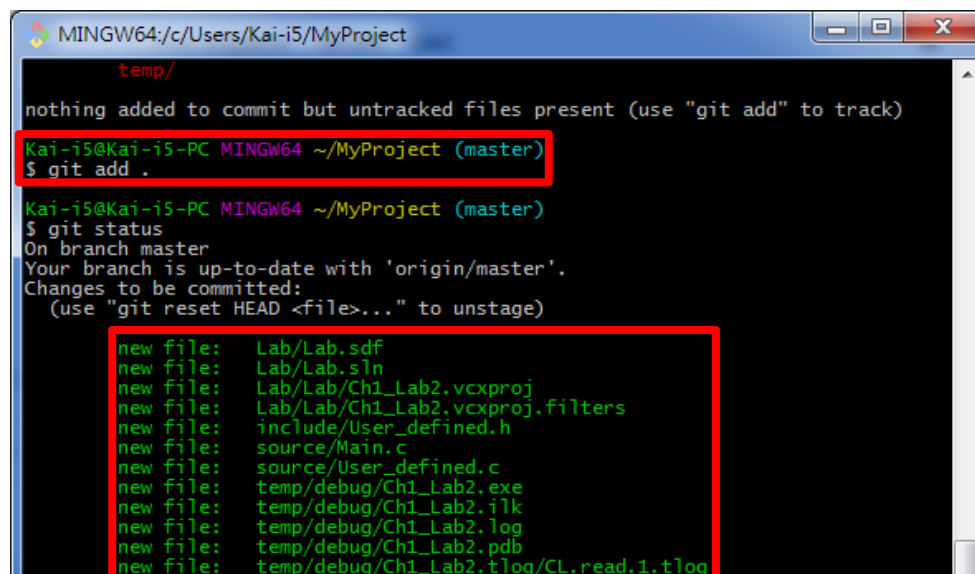
    Lab/
    include/
    source/
    temp/

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

圖 0-46 查看變更

Step5 將檔案的變更動作加入至準備提交區，如圖 0-47 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 .
```



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

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ git add .

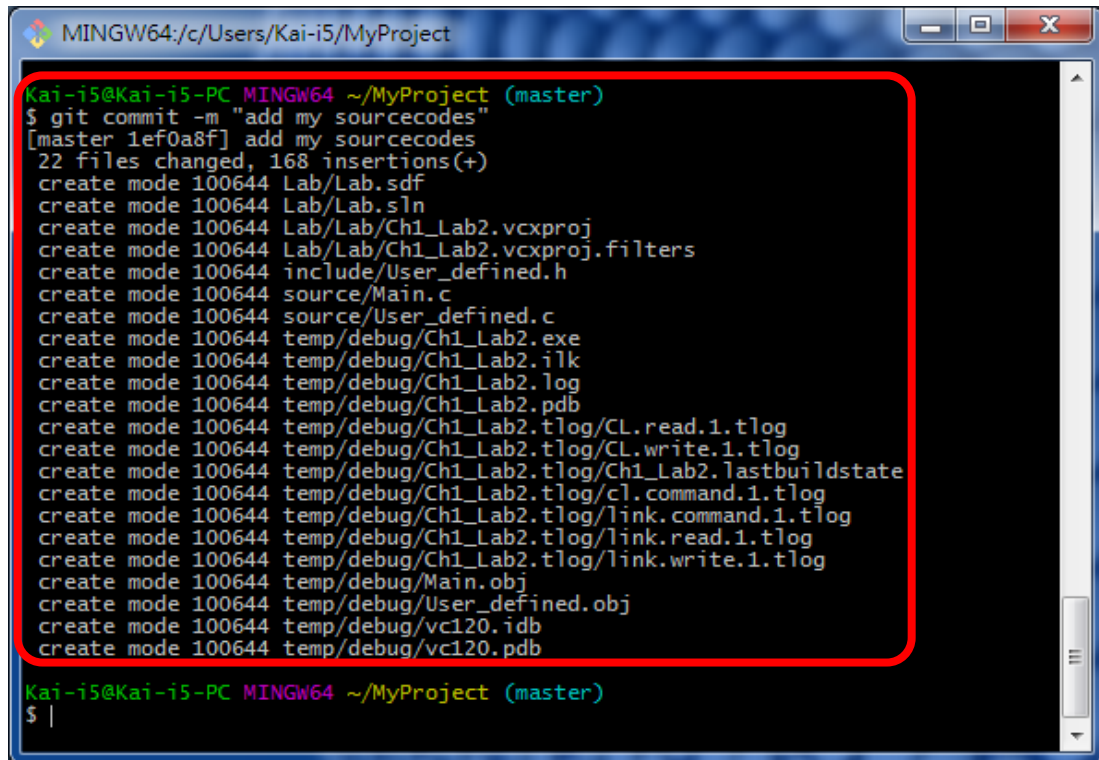
Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   Lab/Lab.sdf
    new file:   Lab/Lab.sln
    new file:   Lab/Lab/Ch1_Lab2.vcxproj
    new file:   Lab/Lab/Ch1_Lab2.vcxproj.filters
    new file:   include/User_defined.h
    new file:   source/Main.c
    new file:   source/User_defined.c
    new file:   temp/debug/Ch1_Lab2.exe
    new file:   temp/debug/Ch1_Lab2.ilc
    new file:   temp/debug/Ch1_Lab2.log
    new file:   temp/debug/Ch1_Lab2.pdb
    new file:   temp/debug/Ch1_Lab2.tlog/CL.read.1.tlog
```

圖 0-47 加入變更的檔案

Step6 將提交區的檔案至本地資料庫，如圖 0-48 所示。

```
$ git commit -m "add my sourcecode"
```




```
MINGW64: c:/Users/Kai-i5/MyProject

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ git commit -m "add my sourcecodes"
[master 1ef0a8f] add my sourcecodes
22 files changed, 168 insertions(+)
create mode 100644 Lab/Lab.sdf
create mode 100644 Lab/Lab.sln
create mode 100644 Lab/Lab/Ch1_Lab2.vcxproj
create mode 100644 Lab/Lab/Ch1_Lab2.vcxproj.filters
create mode 100644 include/User_defined.h
create mode 100644 source/Main.c
create mode 100644 source/User_defined.c
create mode 100644 temp/debug/Ch1_Lab2.exe
create mode 100644 temp/debug/Ch1_Lab2.ilc
create mode 100644 temp/debug/Ch1_Lab2.log
create mode 100644 temp/debug/Ch1_Lab2.pdb
create mode 100644 temp/debug/Ch1_Lab2.tlog/CL.read.1.tlog
create mode 100644 temp/debug/Ch1_Lab2.tlog/CL.write.1.tlog
create mode 100644 temp/debug/Ch1_Lab2.tlog/Ch1_Lab2.lastbuildstate
create mode 100644 temp/debug/Ch1_Lab2.tlog/cl.command.1.tlog
create mode 100644 temp/debug/Ch1_Lab2.tlog/link.command.1.tlog
create mode 100644 temp/debug/Ch1_Lab2.tlog/link.read.1.tlog
create mode 100644 temp/debug/Ch1_Lab2.tlog/link.write.1.tlog
create mode 100644 temp/debug/Main.obj
create mode 100644 temp/debug/User_defined.obj
create mode 100644 temp/debug/vc120.idb
create mode 100644 temp/debug/vc120.pdb

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ |
```

圖 0-48 提交檔案

Step7 將本地資料庫與遠端資料庫（GitHub）做連結，複製 GitHub 的資料庫連結，如圖 0-49 所示。



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

☐ Set up in Desktop or ☒ HTTPS ☒ SSH

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

...or create a new repository on the command line

```
echo "# MyProject" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/a1573595/MyProject.git
git push -u origin master
```

圖 0-49 GitHub 專案頁面

複製完後即可在工作目錄下使用指令將資料庫做連結：

```
$ git remote add origin 資料庫連結
```

Step8 將本地資料庫的紀錄提交到遠端資料庫（GitHub）上，如圖 0-50 所示。

第一次push時，遠端資料庫沒有對應的主幹，所以需要建立主幹，之後就可直接下push即可。

```
$ git push --set-upstream origin master
```

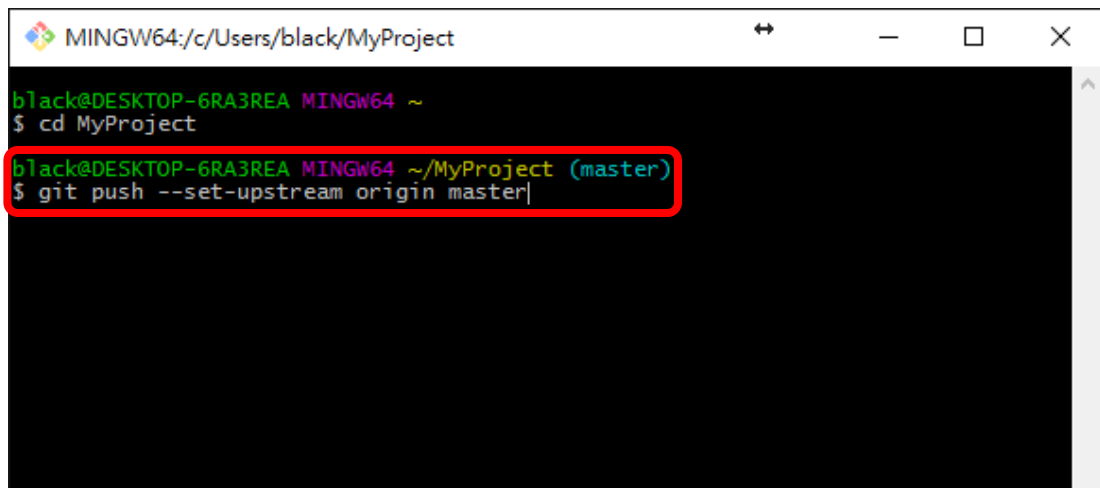


圖 0-50 提交檔案至遠端資料庫

Step9 輸入身分驗證，此時須輸入 GitHub 的使用者名稱及使用者密碼，完成 git push 提交時需要的身分驗證，點選「OK」，如圖 0-51 所示。

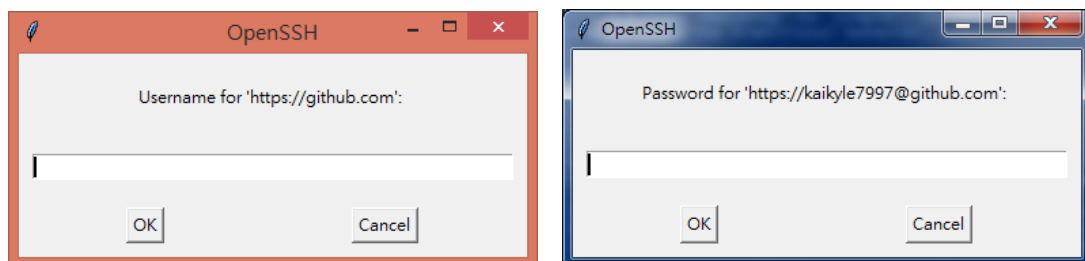
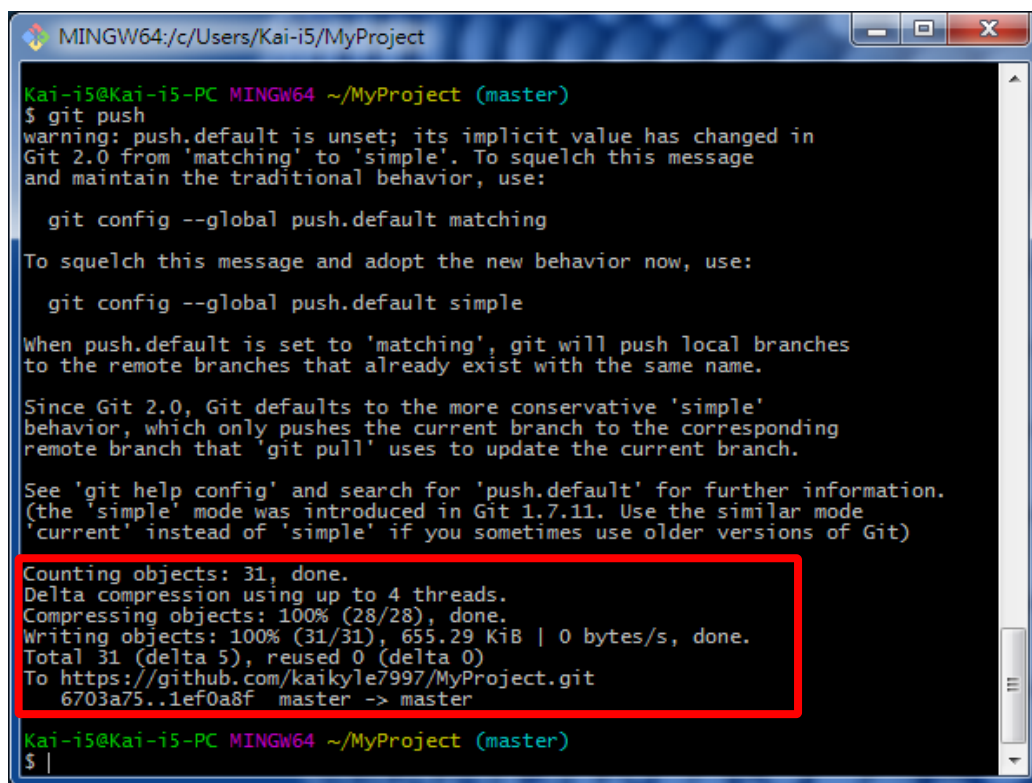


圖 0-51 輸入使用者名稱（左）與輸入密碼（右）

Step10 檔案成功推至遠端的 GitHub，如圖 0-52 所示。



```
MINGW64:/c/Users/Kai-i5/MyProject

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 31, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (28/28), done.
Writing objects: 100% (31/31), 655.29 KiB | 0 bytes/s, done.
Total 31 (delta 5), reused 0 (delta 0)
To https://github.com/kaikyle7997/MyProject.git
6703a75..1ef0a8f master -> master

Kai-i5@Kai-i5-PC MINGW64 ~/MyProject (master)
$
```

圖 0-52 檔案上傳成功

Step11 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 0-53 所示。

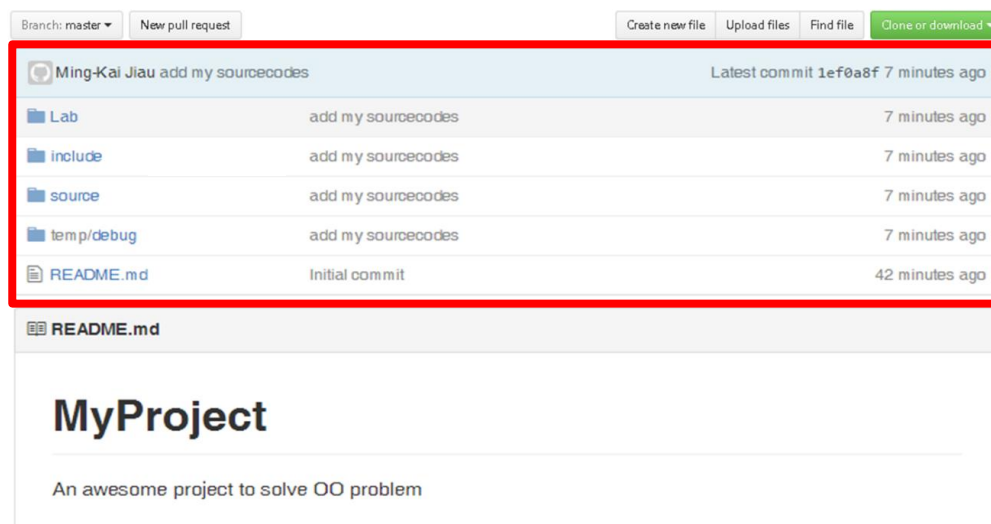


圖 0-53 查看 GitHub 上的專案

情境 2、將某個專案複製到工作目錄下。

Step1 從 GitHub 找到某人所寫的程式，如專案網址：<https://github.com/smartCarLab/smartCar>，首先點擊如下圖框框內按鈕，將 MyProject 的專屬 URL 複製，如圖 0-54 所示。

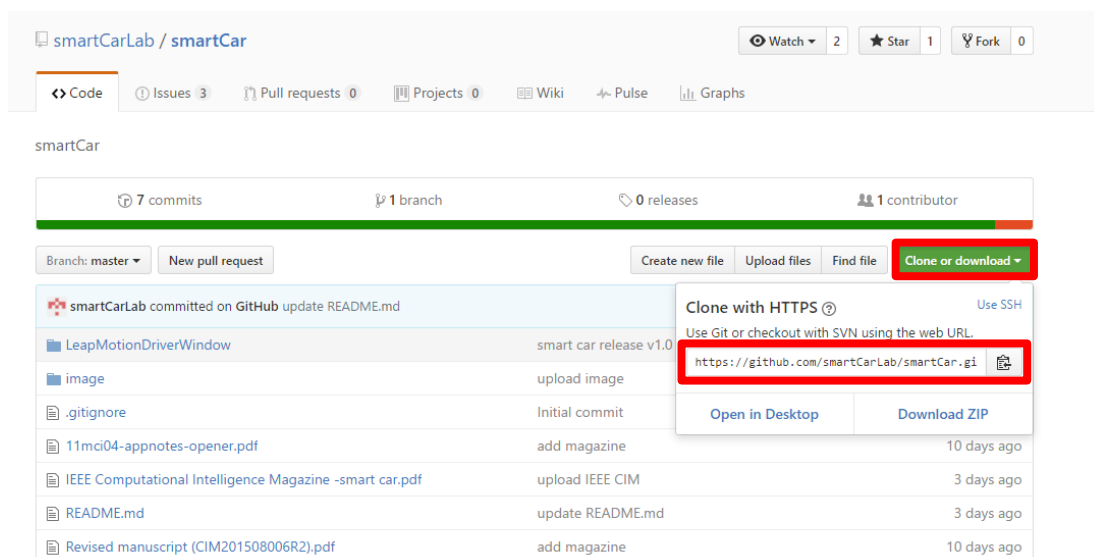


圖 0-54 複製專案 URL

Step2 圖 0-55 在 git bash 執行如下指令，進行遠端資料庫下載，如圖 0-55 所示。

```
$ git clone https://github.com/smartCarLab/smartCar.git
```

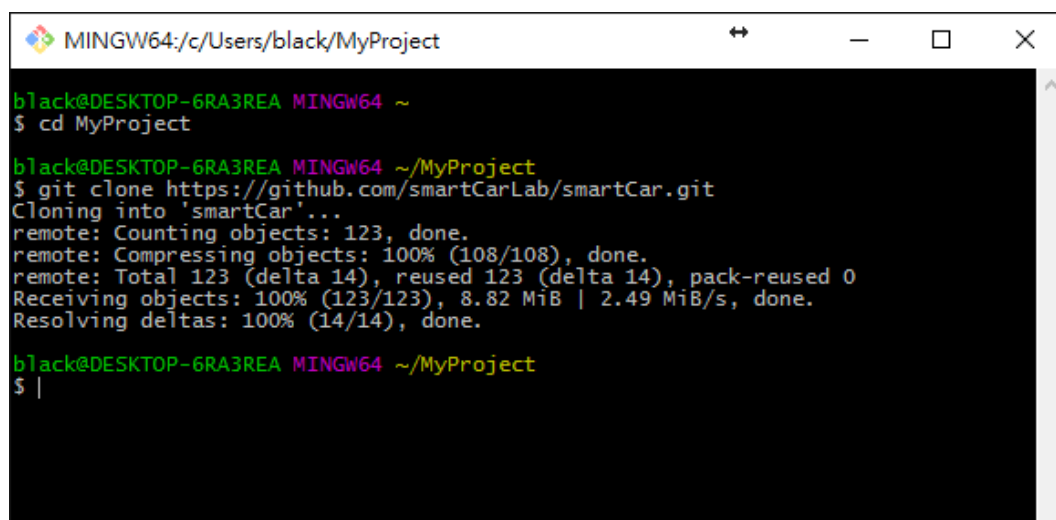


圖 0-55 複製專案到本地

Step3 下載完成之後可以移動至檔案目錄底下就可以使用該專案的資料庫，如圖 0-56 所示。

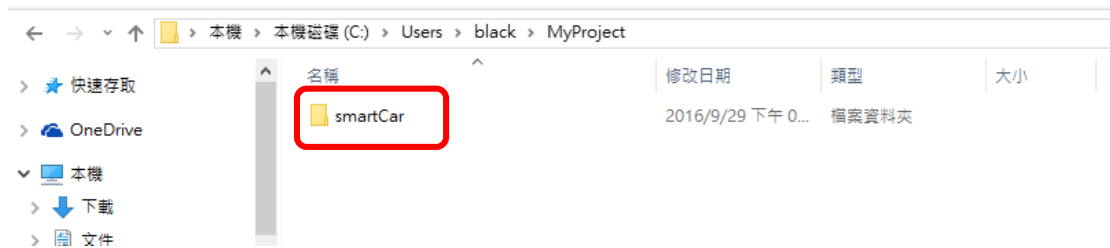


圖 0-56 檢查檔案目錄

Step4 用 Git 開啟目錄，如果有出現 master 標籤就表示成功，如圖 0-57 所示。

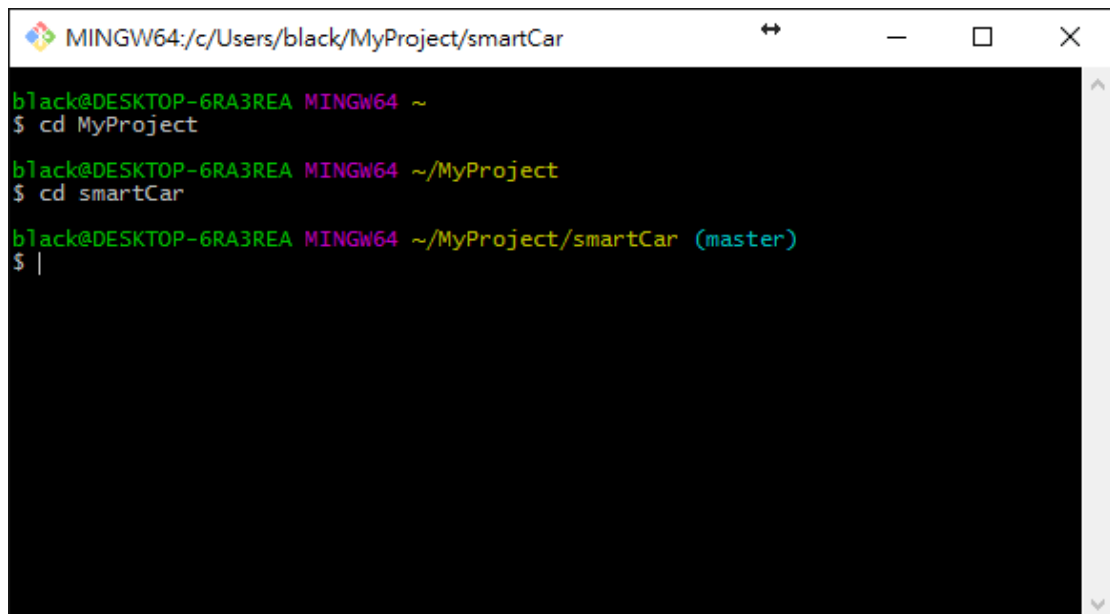


圖 0-57 出現 master 標籤

0.3 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 pull	將本地端的資料更新到遠端。
git pull --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。

<code>git stash apply</code>	取出暫存區中最新的一筆資料，但不會移除。
<code>git stash apply stash@{index}</code>	取出第 index 筆暫存資料。
<code>git stash clear</code>	把暫存區裡的資料都清掉。
<code>git reset --hard</code>	強制回復到上一版。
<code>git reset --hard xxxx</code>	強制回復到某個 commit 版本。
<code>git reset --hard origin/B</code>	強制回復到遠端 B 分支版本。
<code>git rebase</code>	衍合分支。
<code>git rebase -i HEAD~n</code>	衍合最後的 N 次提交。
<code>git rebase --interactive -i HEAD ~n</code>	開啟對話模式編輯（head~n表示要編輯到前n個）

說明

rebase 跟 reset 都是很危險的操作指令，因為它們都是改寫提交的歷史紀錄，弄不好的話資料可能就會不見了。所以若是在沒有把握的狀況下，最好先在操作前用 branch 備份，有問題的時候只要 reset 回備份的分支就行了。

1. **git status**：檢查目前分支狀態。

如圖 0-58 所示，在分支上修改過或是刪除，可以透過 git status 指令來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 0-58 git status

2. **git log**：查看分支近期 commit 過的點，如圖 0-58 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)
cfff4ee2 - 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 remnants of dynamic interpreter (1 year, 4 months ago)
6b29b21 - Remove unused IndirectLiterals (1 year, 4 months ago)
83db68a - Fixed Diaest requires in const.missina. (1 year, 4 months ago)
```

圖 0-59 Git log

3.**git add .**：將修改或變更檔案的部分暫存在 index 位址，如圖 0-60 所示。

※指令範例：git add .：

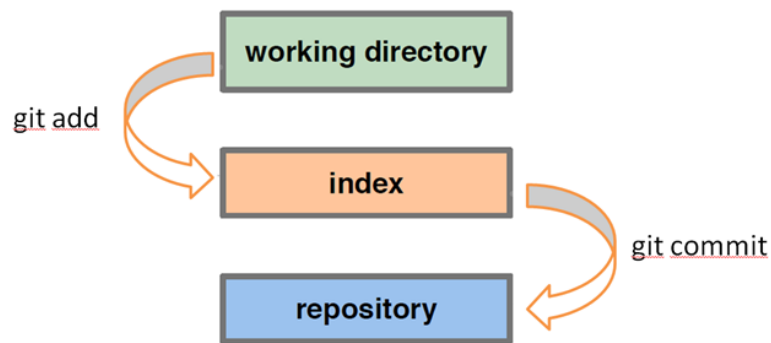


圖 0-60 Git add

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：

```
MINGW32/C/Users/Lin/Documents/GitHub/bn-ride-android
lin@LIN-PC /C/Users/Lin/Documents/GitHub/bn-ride-android (master)
$ git pull
Username for 'https://github.com': 102418005
Password for 'https://102418005@github.com':
remote: Counting objects: 1026, done.
remote: Compressing objects: 100% (977/977), done.
remote: Total 1026 (delta 398), reused 347 (delta 347), pack-reused 571
Receiving objects: 100% (1026/1026), 684.00 KiB | 391.00 KiB/s, done.
Resolving deltas: 100% (644/644), completed with 86 local objects.
From https://github.com/kaikyle7997/bn-ride-android
   1c338e9..11f83c3  master    -> origin/master
   5c26e6h..118caa27 develop  -> origin/develop
* [new branch]      feature/fleet -> origin/feature/fleet
* [new branch]      fix/chatroomAPI -> origin/fix/chatroomAPI
* [new branch]      hotfix/work -> origin/hotfix/work
Updating 1c338e9..11f83c3
error: Your local changes to the following files would be overwritten by merge:
       app/src/development/java/com/BlueNet/Constants.java
Please, commit your changes or stash them before you can merge.
Aborting
lin@LIN-PC /C/Users/Lin/Documents/GitHub/bn-ride-android (master)
$
```

圖 0-61 Git Pull 示範

6.git push：將自己在 local 端 commit 的點同步到 GitHub 上，更新目前所開發的專案到遠端伺服器上。

個人 local 端的 commit 點 ，最新 commit

點是 D，GitHub 上的 commit 點 ，最新 commit 點到 A 點，當自己 local 端有修改專案所 commit B、C、D 要同步到 GitHub 上時，執行指令 git push。

結果如下：

GitHub 上的 commit 點 。

※指令範例：git push，通常下完指令後，需要輸入帳號密碼確認才會開始同步。

7.gitk：開啟 Git GUI，點擊左上方區塊的 commit 點的詳細資料，如圖 0-62 所示。

※指令範例：gitk

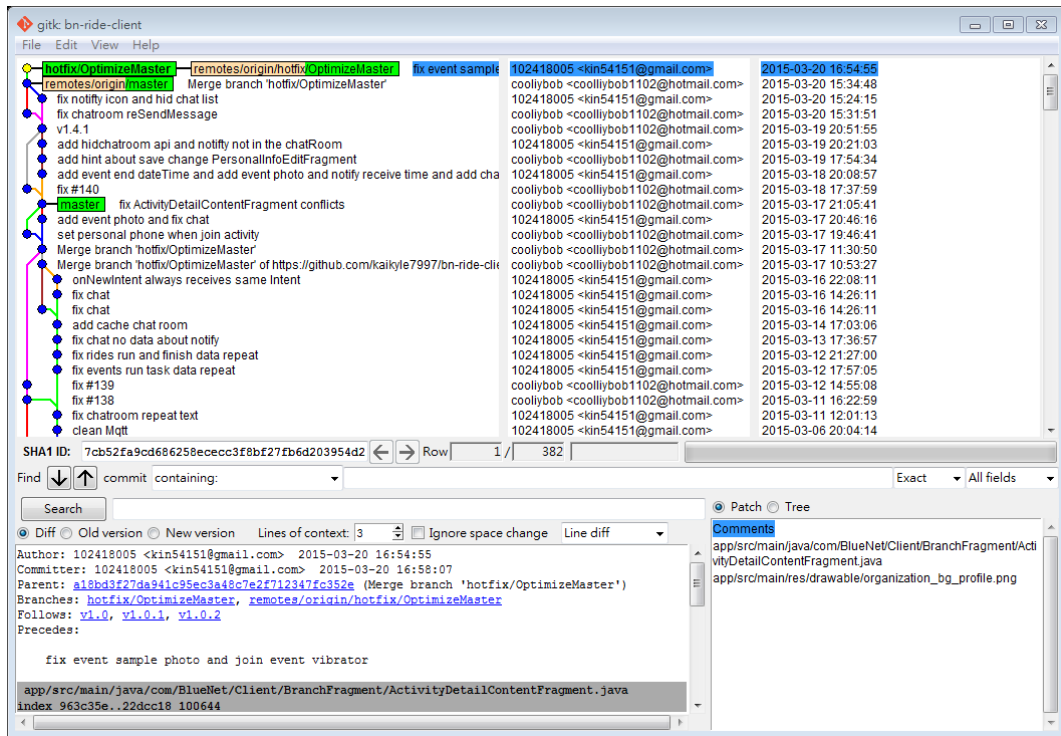




圖 0-62 Git GUI

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 。

0.4 書附範例專案

在介紹 GitHub 的專案管理後，我們將這本書的所有 Lab 專案程式碼放置在 GitHub 上，讀者可以自行將 Lab 專案複製下載到個人電腦的工作目錄下，Lab 專案程式碼放置的 GitHub 連結如下：

<https://github.com/taipeitechmmslab/MMSLAB-Android-Kotlin>

我們提供的專案程式碼主題分別如下：

項目	主題
Lab2	畫面設計與元件使用
Lab3	物件控制與監聽事件（承接 Lab2）
Lab4	Activity
Lab5	Fragment
Lab6	提示訊息元件
Lab7	清單元件
Lab8	進階清單元件
Lab9	Android 的非同步執行
Lab10	Service
Lab11	Broadcast & Receiver
Lab12	Google Map
Lab13	SQLite
Lab14	API
Lab15	Cloud Messaging