

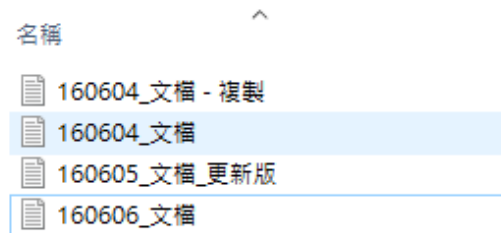
Lab0 Github 版本控制

一、本節目的：

- 了解什麼是版本控制行為、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

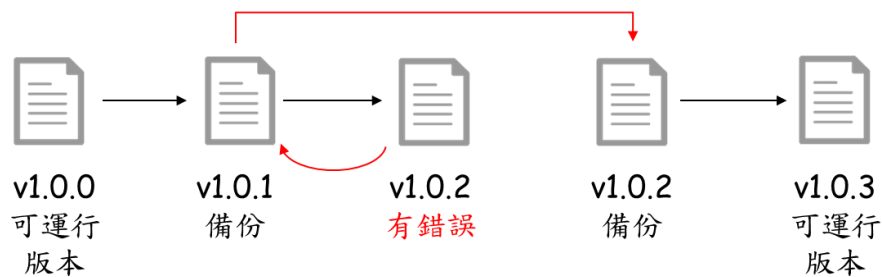
二、觀念說明：

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，過程中可能變為如下：



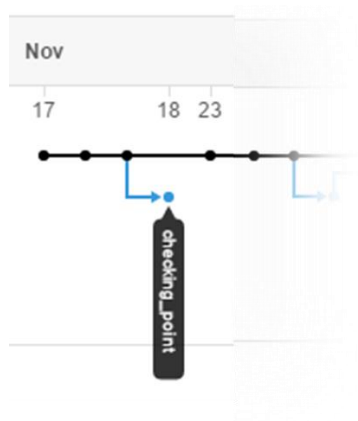
當需要還原時，才發現這些命名方式完全無助於找到想要的還原版本，不只相當麻煩，修改哪些細節內容完全不記得，更別說是多人開發團隊之中，與他人共同開發一個程式也是很常遇到的事，如果採用這種命名，相信大部分的人都會感到頭痛。

因此這時候版本控制就非常重要，版本控制是系統開發的標準作法，它能系統化的管理備份資料，讓開發者能從開始一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。



1 Git 版本控制系統

Git 為分散式版本控制系統，可以把檔案每一次的狀態變更保存為歷史記錄。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，我們就會需要借助到遠端資料庫來做管理。

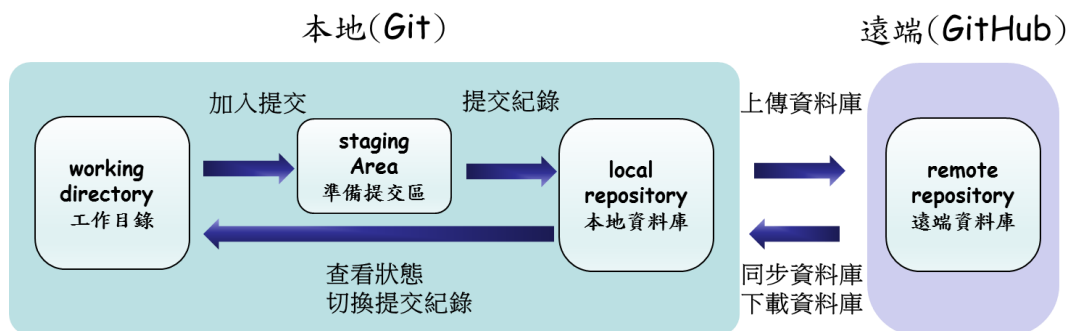


2 GitHub

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。GitHub 最主要的功能是能將位於在電腦端經由 Git 操作後的歷史記錄上傳至網路上，進行備份或是分享，除了允許個人或是組織建立、存取資料庫以外，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。



程式專案在 Git 控管下與 GitHub 上的不同階段變化，如下：



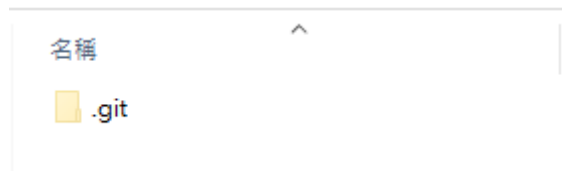
- **working directory (工作目錄)：**工作目錄主要是存放要被版本控制的檔案資料夾。我們可以選擇一個一般資料夾並在資料夾內建立 git 的資料庫，就能把該資料夾變成 git 的工作目錄
- **staging area (準備提交區)：**準備提交區用於記錄即將要被提交的資料。當在工作目錄下檔案的有進行變更，且我們希望能提交這些變更，我們會將這些資料存入準備提交區之中，這狀態下只有標記那些資料要被提交，但還並未實際的做提交紀錄。
- **local repository (本地資料庫)：**即為自己電腦端上的資料庫。當確定好所有要提交的資料都加入到準備提交區之後，可以將準備提交區的資料做提交紀錄，提交後的資料會被記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。

● 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，在目錄下會產生一個「.git」的目錄，即為本地資料庫，若使用使用 `git` 的控制台查看此目錄可以看到路徑後增加了一個[master]的標籤，表示有偵測到資料庫。

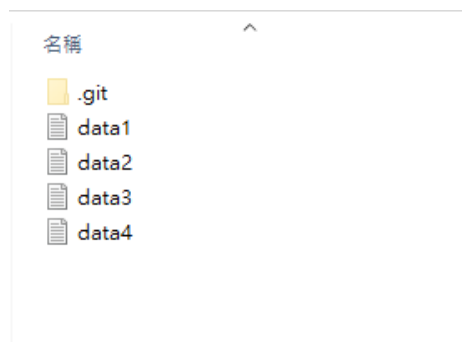


```
C:\Users\black\Documents\GitHub\MyProject> git init
Initialized empty Git repository in C:/Users/black/Documents/GitHub/MyProject/.git/
C:\Users\black\Documents\GitHub\MyProject [master]>
```

● 查看狀態

建立好資料庫後，當該工作目錄有更動，例如增加檔案，或是既有的檔案有更動，可以使用 `git status` 查看更動過的資料

```
$ git status
```



```
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 !]>
```

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。

● 加入提交

在記錄檔案版本之前，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 將檔案作加入到準備提交區，也可以用 git add .加入所有檔案。

```
$ git add 檔案名稱 或 git add .
```

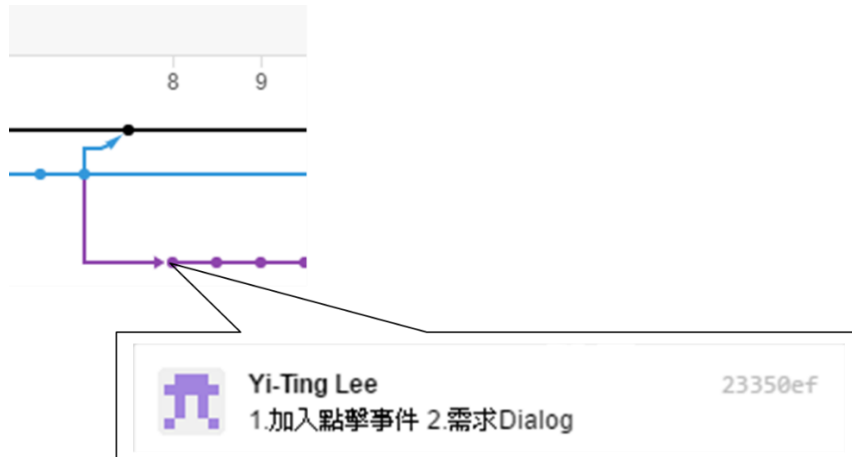
```
C:\Users\black\Documents\GitHub\MyProject [master +4 ~0 -0 !]> git add .
C:\Users\black\Documents\GitHub\MyProject [master +4 ~0 -0 ~]>
```

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。

● 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。

執行提交時，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執行提交是會失敗的。



在 git commit 後面加上 -m 的語法可以輸入說明文字。

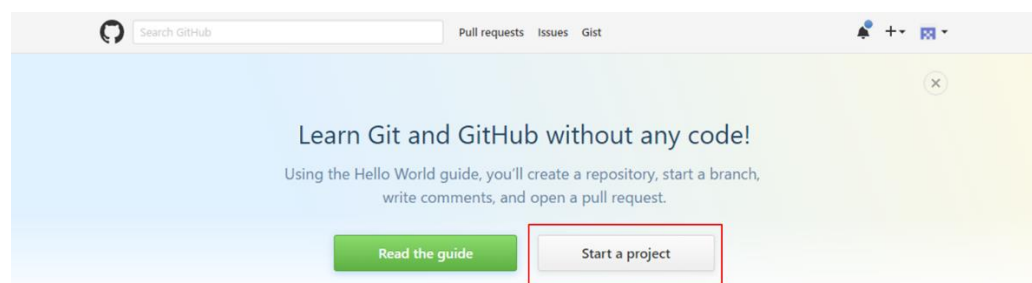
```
$ git commit -m "說明文字"
```

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

這步驟會將我們修改的內容做保存紀錄，當標籤後的綠字消失，代表工作目錄與本地資料庫已經同步。

● 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。



Create a new repository

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

Owner



Repository name

MyProject ✓

Great repository names are short and memorable. Need inspiration? How about **special-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: None ▾

Add a license: None ▾



Create repository

建立資料庫後，會產生一個連結，之後上傳資料庫時會需要。

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Pulse](#) [Graphs](#) [Settings](#)

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

[Set up in Desktop](#) or [HTTPS](#) [SSH](#)

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
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/MyProject.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

資料庫連結

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。下 git remote 指令連結遠端資料庫，這邊需要加上資料庫的連結。

```
$ git remote add origin 資料庫連結
```

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

然後繼續下 push 指令，就可以將資料同步至遠端。

```
$ git push origin 標籤名稱
```

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

● 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，如果本地端已經有對應的資料庫，可以用 pull 同步資料庫到工作目錄下。

```
$ git pull origin 標籤名稱
```

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

● 下載遠端資料庫

有些時候，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。這時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。

```
$ git clone 資料庫連結
```

```
s\GitHub> git clone https://github.com/[redacted]/MyProject.git
```

● 查看本地資料庫

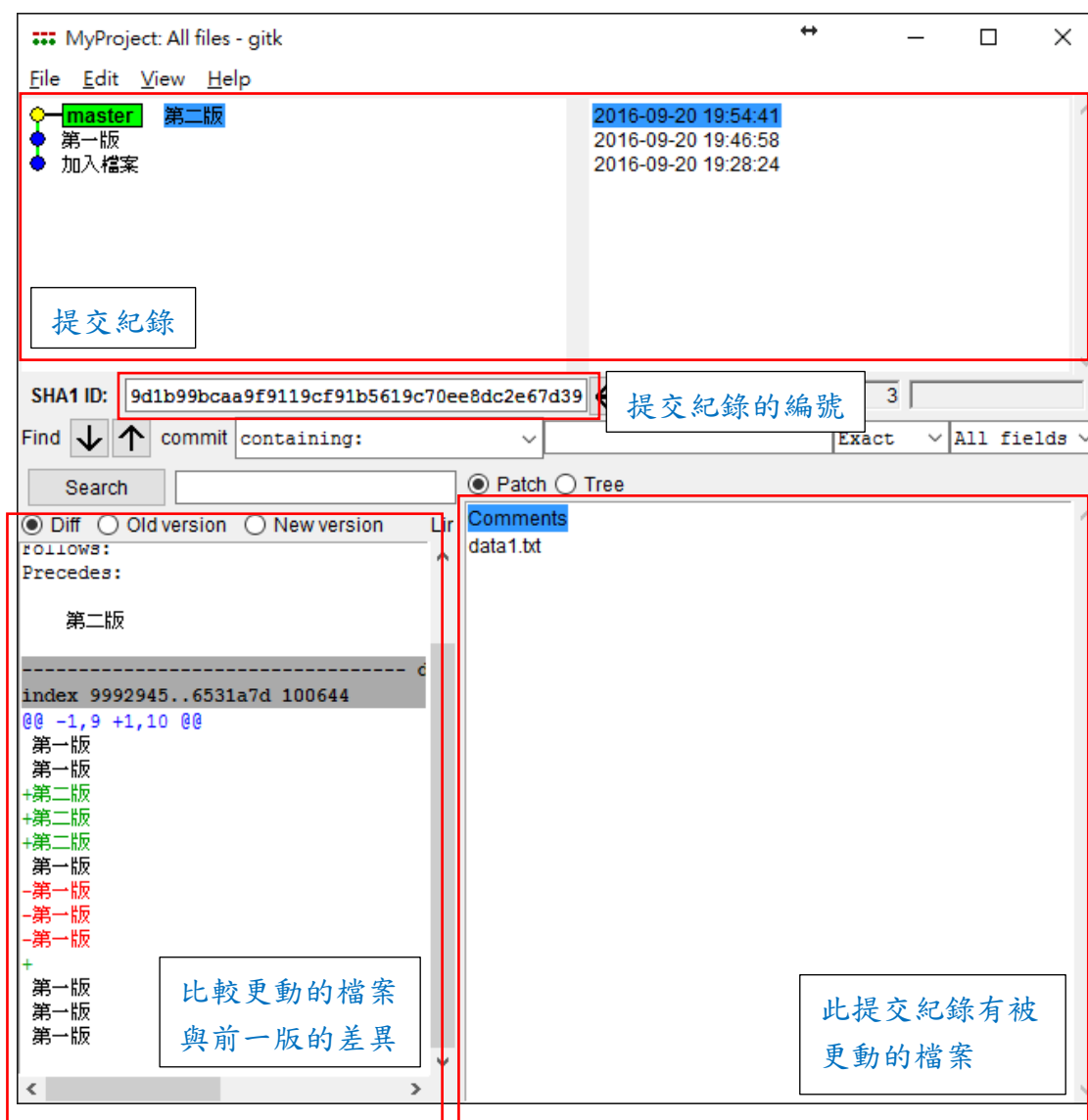
任何時間點，我們都可以查看之前的本地資料庫中紀錄的所有提交紀錄，這

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形介面



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深長的樹狀圖，每次提交都會產生出新的節點，每個節點都代表一次的版本紀錄。

為了區分每個提交紀錄，系統會產生一組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。指定識別碼就可以在資料庫中找到對應的提交紀錄。

SHA1 ID: 9d1b99bcaa9f9119cf91b5619c70ee8dc2e67d39

執行提交後，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。

● 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，在前面我們知道每個提交紀錄都有唯一識別碼，我們可以透過識別碼移動到對應的提交紀錄。

```
$ git checkout 識別碼
```

The image shows a terminal window and a diagram illustrating a Git checkout operation. The terminal window displays the command `git checkout ece0a3436a258a6a20c853ec5dc4bb89f2977233` and its output, including the message "HEAD is now at ece0a34... 第一版". The diagram below the terminal shows a commit history with two versions of the `master` branch. The first version is labeled "第一版" (First Version) and the second version is labeled "第二版" (Second Version). A red arrow points from the "第二版" label to the "第一版" label, indicating the checkout operation. Below the diagram, the SHA1 IDs are shown: `9d1b99bcaa9f9119cf91b5619c70ee8dc2e67d39` for the first version and `ece0a3436a258a6a20c853ec5dc4bb89f2977233` for the second version.

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。

三、設計重點：

- 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
 - ◆ 情境一、將撰寫完成的專案推送到 GitHub 上。
 - ◆ 情境二、將某個專案複製到工作目錄下。

四、設計步驟：

- 安裝 Git 使用環境 Git Bash

Step1 請先至 <https://git-for-windows.github.io/> 下載 Git

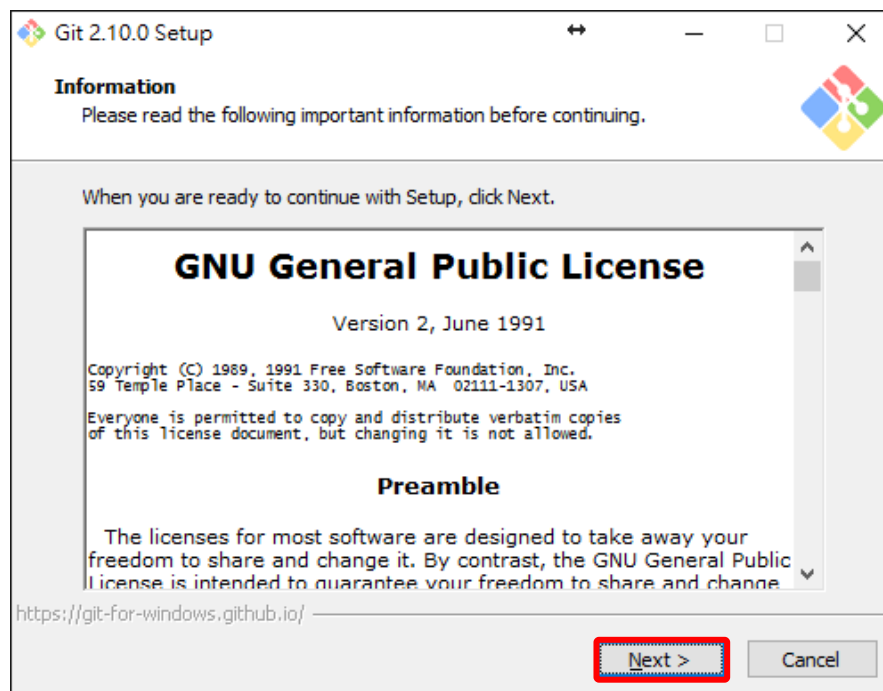


Step2 點擊 Git-2.6.3-64-bit.exe 開始安裝 Git Bash

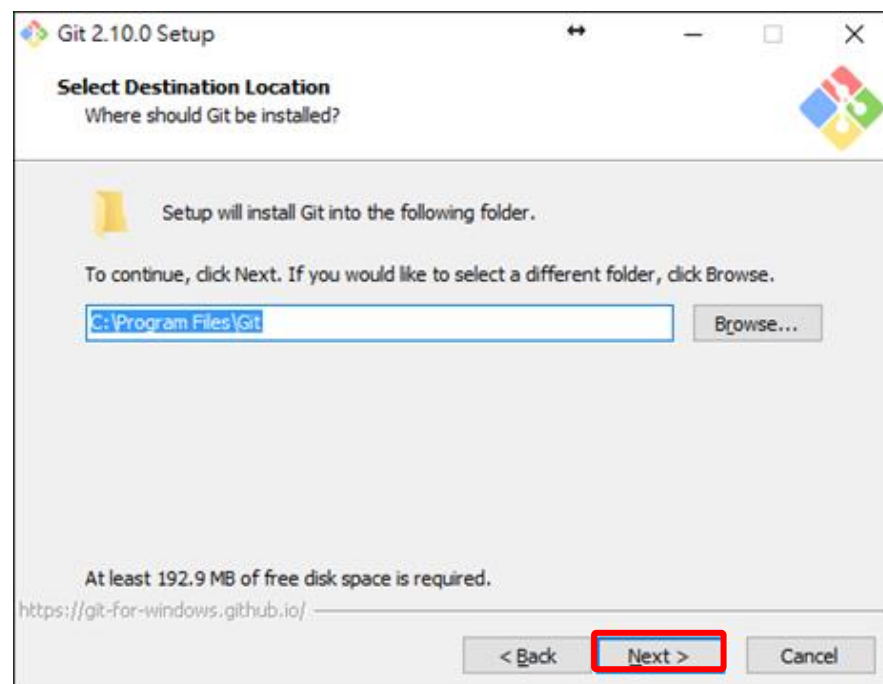




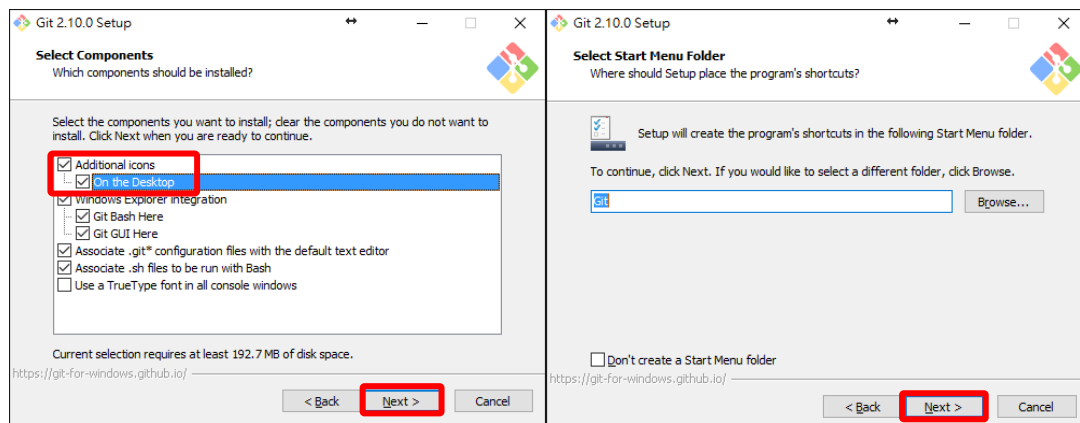
Step3 點擊 Next，再點擊 Next



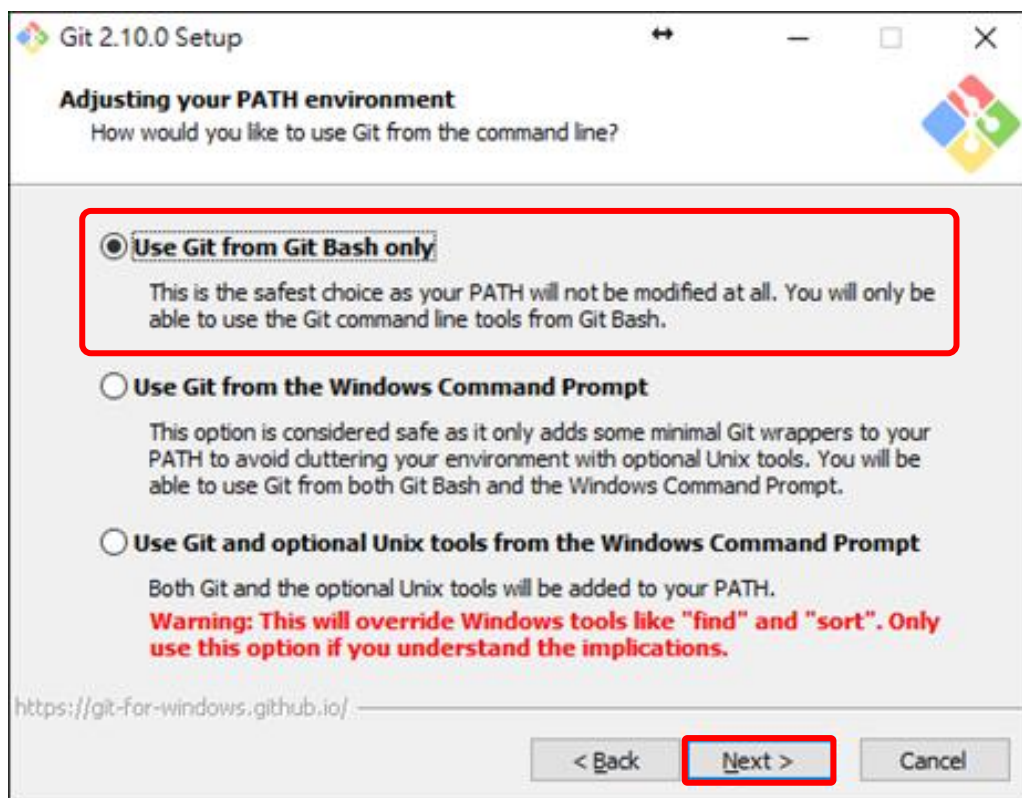
Step4 點擊 Next



Step5 勾選 On the Desktop，點擊 Next，再點擊 Next

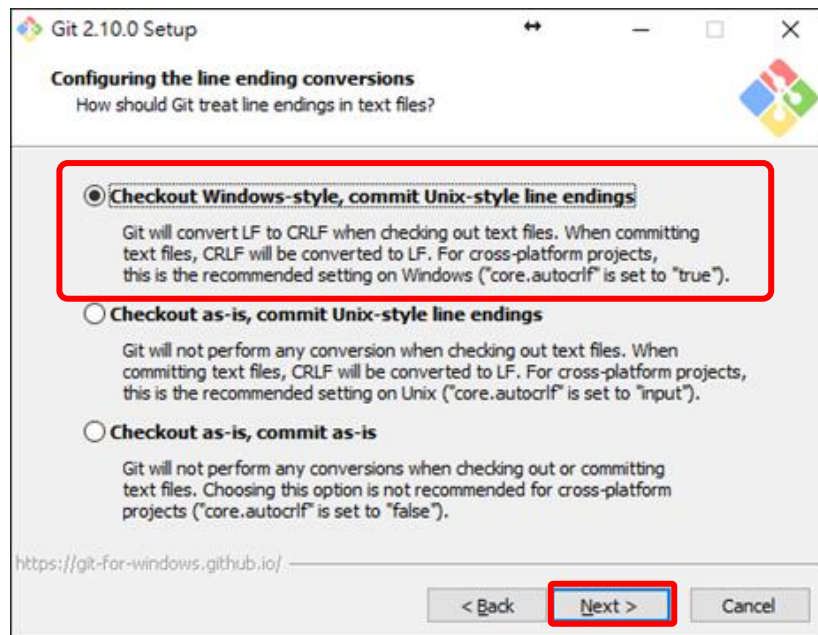


Step6 預設選擇 Use Git from Git Bash only，點擊 Next

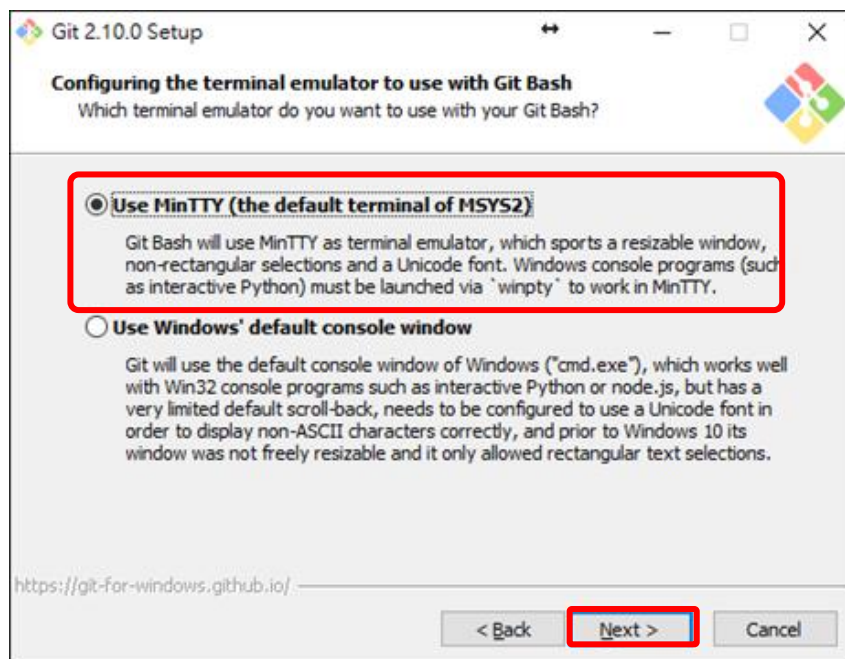


Step7 預設選擇 Checkout Windows-style, commit Unix-style line endings，點擊 Next

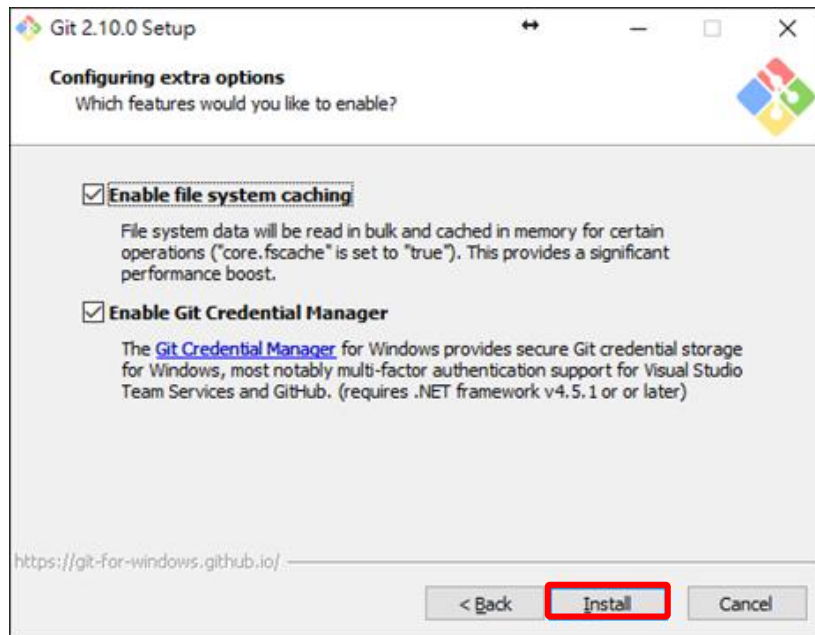
註：設定結束符號類型



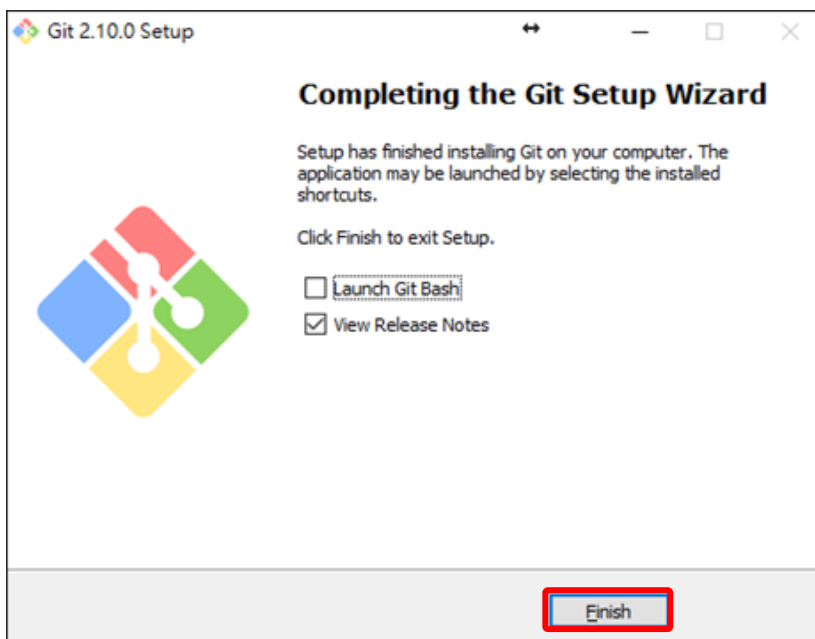
Step8 預設選擇 Use MinTTY，點擊 Next



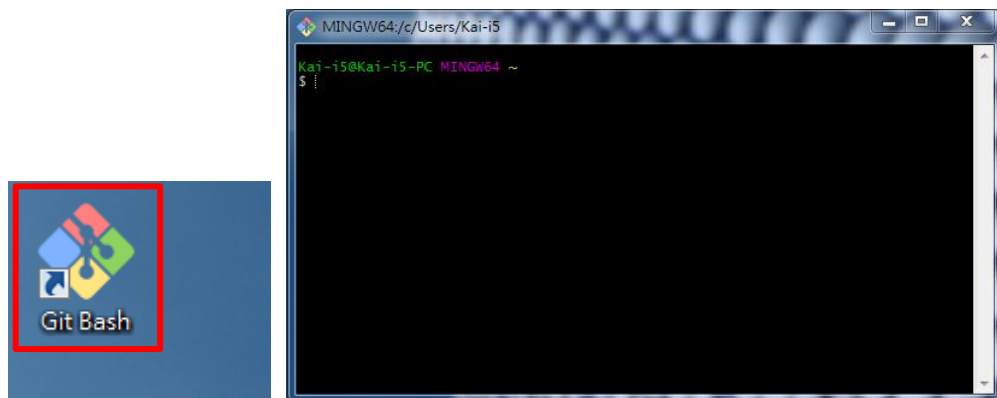
Step9 直接點擊 Install



Step10 完成安裝點擊 Finish



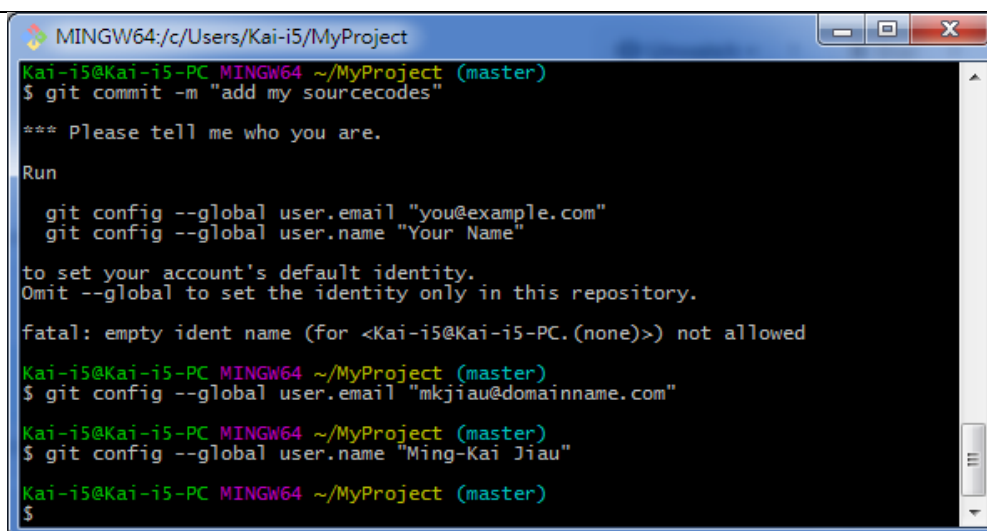
Step11 到桌面點擊如下圖示，執行 Git Bash，如下黑色畫面



Step12 在 git bash 內，設定自己的 user.email 和 user.name，使用如下指令：

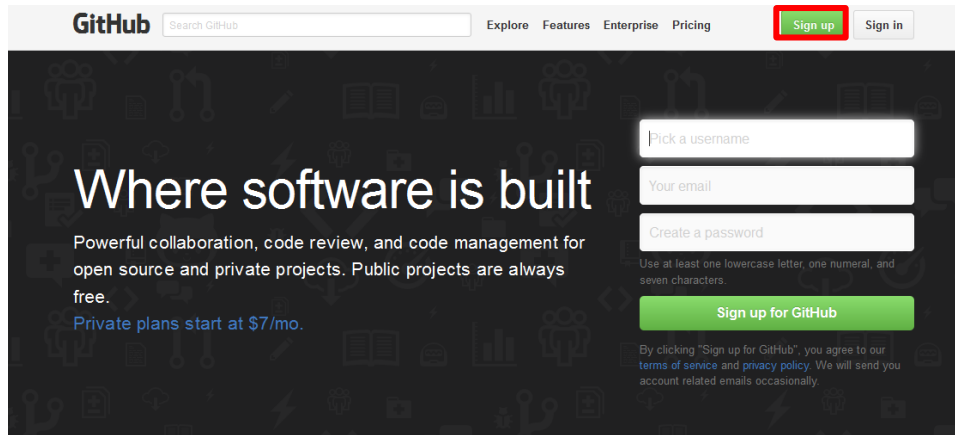
```
$ git config --global user.email "xxx@gmail.com"
```

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



- 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



Step2 填寫註冊資料，並點擊 Create an account

A screenshot of the GitHub "Join GitHub" page. The header includes the GitHub logo, navigation links for Explore, Features, Enterprise, Pricing, Sign up, and Sign in. The main content area has a light background with the text "Join GitHub" and "The best way to design, build, and ship software." Below this, there is a three-step process: Step 1: Set up a personal account, Step 2: Choose your plan, and Step 3: Go to your dashboard. The "Create your personal account" section is highlighted with a red box. It contains three input fields: "Username", "Email Address", and "Password". Below the "Password" field, there is a note: "Use at least one lowercase letter, one numeral, and seven characters." To the right of the form, there is a section titled "You'll love GitHub" with the text "Step1: 填寫基本註冊資料", "Username : 使用者名稱、", "Email Address : 電子信箱、", and "Password : 使用者密碼". At the bottom of the form, there is a green "Create an account" button.

Step2: 送出註冊



Step3 選擇 GitHub 註冊帳號的方案，此處保持 Free Plan 的方案，直接點擊 Finish sign up 完成註冊

Search GitHub Pull requests Issues Gist

Welcome to GitHub

You've taken your first step into a larger world, @chienhua7243.

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	Choose
Medium	\$22/month	20	Choose
Small	\$12/month	10	Choose
Micro	\$7/month	5	Choose
Free	\$0/month	0	Chosen

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

Step4 創立一個遠端 Repository 在 GitHub 上，點選 Start a project

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](#)

Step5 輸入及設定 Repository 的資料及屬性，之後點擊 Create repository

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 **propitious-rutabaga**.

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: **None** | Add a license: **None** ⓘ

Step 1

Repository name : [程式庫名稱 (=專案名稱)]

Description : [程式庫描述]

Public/Private : [程式庫是否別人看得到]

Initialize this ... a README : [是否加入初始的 README.md 檔案]

Add .gitignore : [加入要忽略的檔案設定]

Add a lincense : [加入 sourcecode 想使用的 license 方式，如：MIT 最為自由]

Create repository

Step 2

點擊進行建立

Step6 完成遠端資料庫的建立，命名為「MyProject」

Navigation: <> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

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

Set up in Desktop or **HTTPS** **SSH**

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/TingWeiZhang/MyProject.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/TingWeiZhang/MyProject.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

- 實際練習 Git 與 GitHub 的基本使用情境

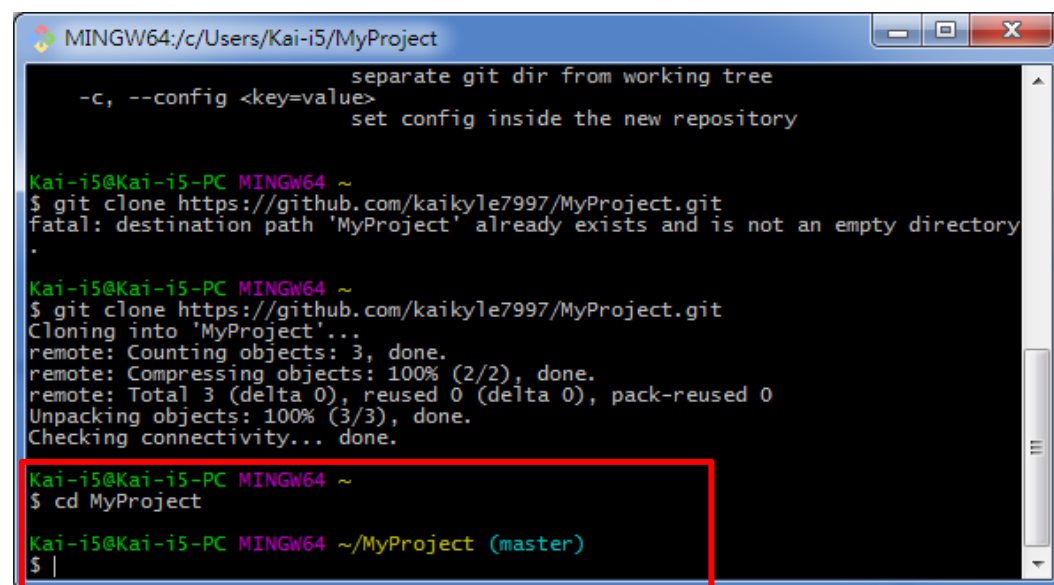
使用情境一、將撰寫完成的專案推送到 GitHub 上

Step1 切換到已經建立的資料夾作為工作目錄

執行下面指令，切換當前目錄到工作目錄位置，如下：

```
$ cd MyProject
```

註：~ 這符號代表使用者的主目錄，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\

A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/Kai-i5/MyProject". The window shows the following commands and output:

```
-c, --config <key=value>
    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)
$ |
```

The last three lines of the command prompt are highlighted with a red rectangle.

Step2 建立 Git 本地資料庫

在工作目錄輸入指令產生出本地資料庫

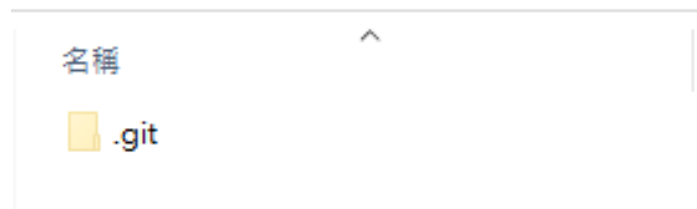
```
$ git init
```

```
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/

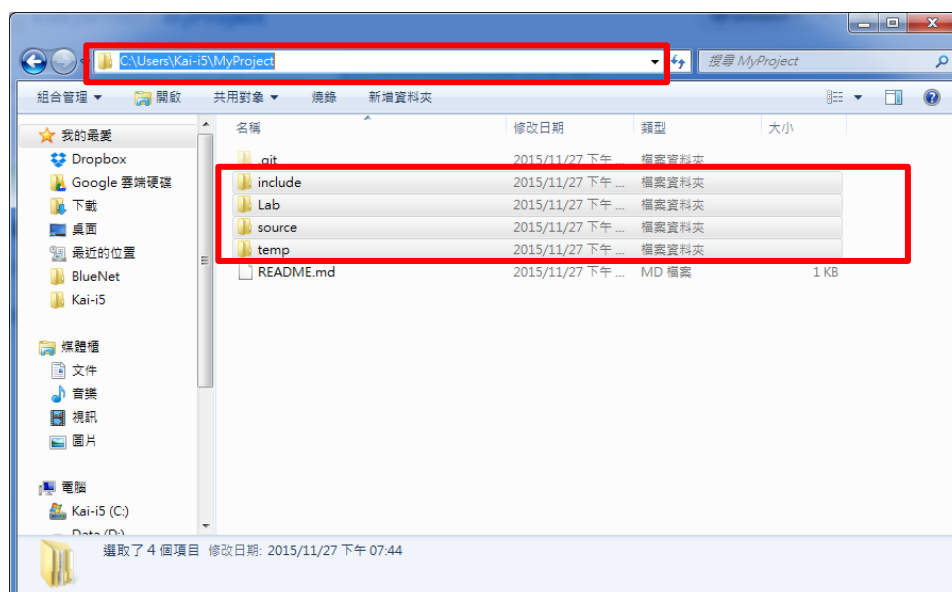
black@DESKTOP-6RA3REA MINGW64 ~/MyProject (master)
$ |
```



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，如下圖

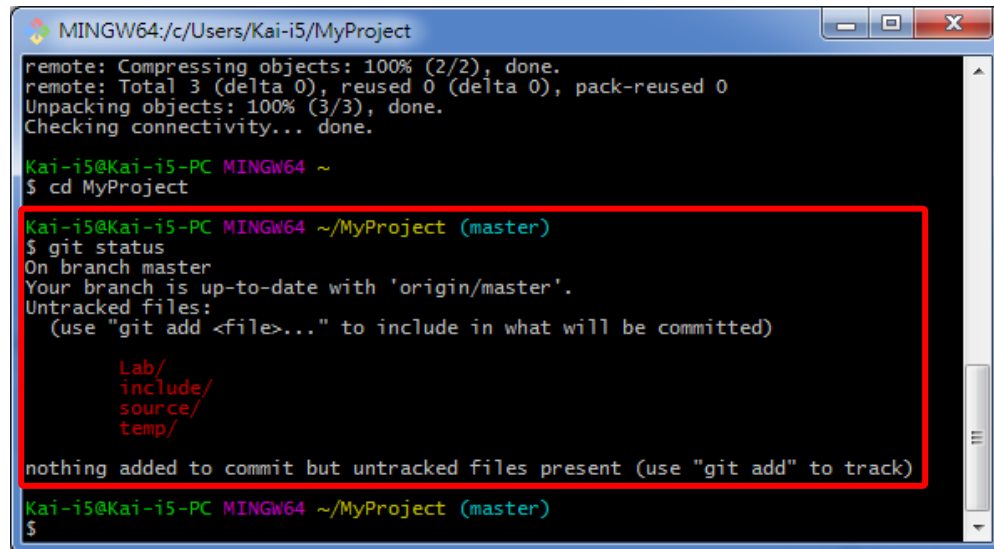
註：這邊請加入 LAB1 的專案，並命名為 LAB1



Step4 查看本地資料庫變更狀況

```
$ git status
```

註：有紅色字體表示未被追蹤



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

Step5 將此次的檔案變更動作加入 (加入到準備提交區)，指令如下：

```
$ git add .
```

註：執行 git add 後可以，執行 git status 看一下詳細資訊，有哪些檔案被追蹤/新增了

```
MINGW64/c/Users/Kai-i5/MyProject
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
    new file:   temp/debug/Ch1_Lab2.tlog/CL.write.1.tlog
    new file:   temp/debug/Ch1_Lab2.tlog/Ch1_Lab2.lastbuildstate
    new file:   temp/debug/Ch1_Lab2.tlog/cl.command.1.tlog
    new file:   temp/debug/Ch1_Lab2.tlog/link.command.1.tlog
    new file:   temp/debug/Ch1_Lab2.tlog/link.read.1.tlog
    new file:   temp/debug/Ch1_Lab2.tlog/link.write.1.tlog
    new file:   temp/debug/Main.obj
    new file:   temp/debug/User_defined.obj
    new file:   temp/debug/vc120.idb
    new file:   temp/debug/vc120.pdb

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

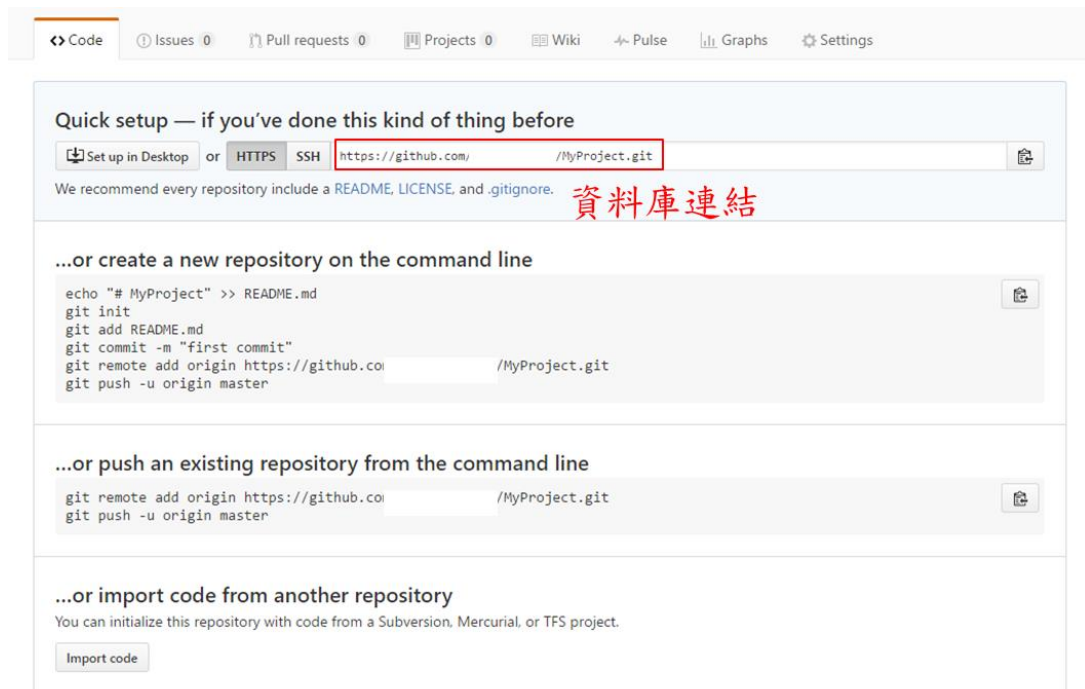
Step6 提交準備提交區的檔案至本地資料庫

```
$ 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)
$ |
```


Step7 將本地資料庫與遠端資料庫(GitHub)做連結
資料庫連結要先到 Github 頁面上複製



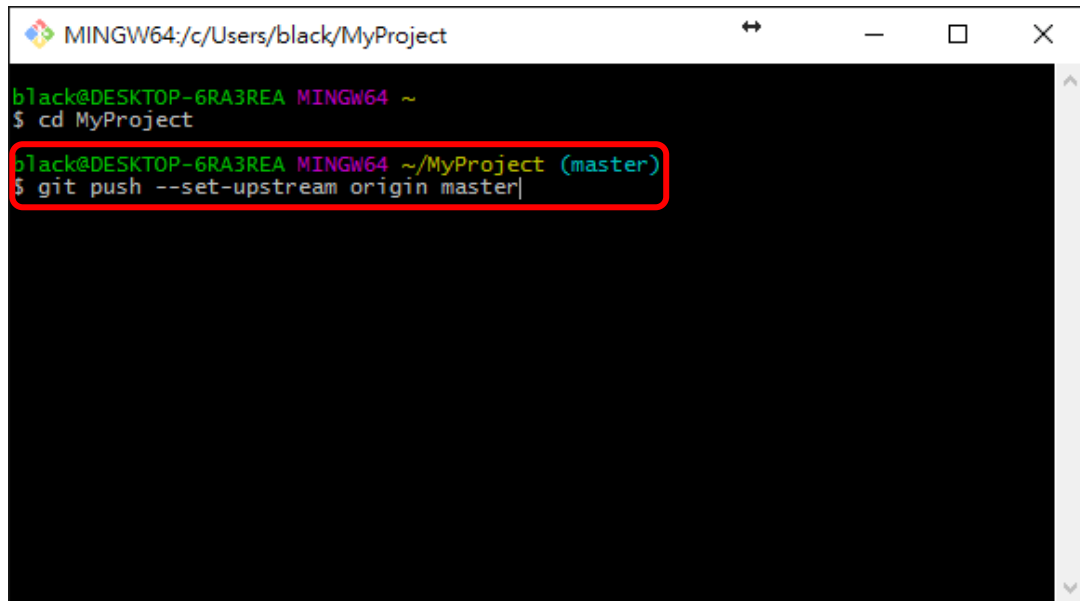
複製完後即可在工作目錄下使用指令將資料庫做連結

```
$ 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

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

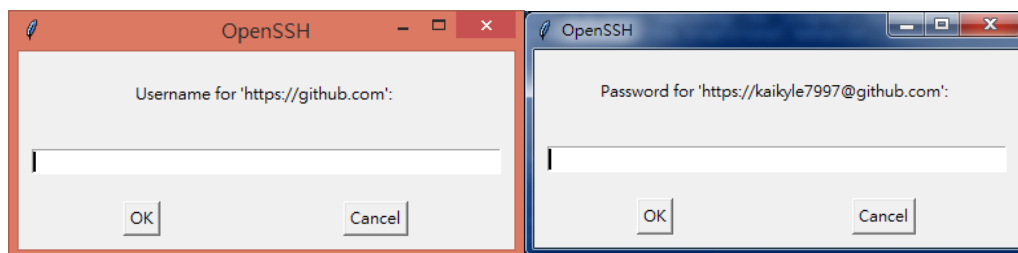
```
$ git push --set-upstream origin master
```

A terminal window titled 'MINGW64:/c/Users/black/MyProject'. The prompt is 'black@DESKTOP-6RA3REA MINGW64 ~'. The command 'cd MyProject' has been executed. The next prompt is 'black@DESKTOP-6RA3REA MINGW64 ~/MyProject (master)'. The command 'git push --set-upstream origin master' is being entered and is highlighted with a red rectangle.

```
MINGW64:/c/Users/black/MyProject
black@DESKTOP-6RA3REA MINGW64 ~
$ cd MyProject
black@DESKTOP-6RA3REA MINGW64 ~/MyProject (master)
$ git push --set-upstream origin master|
```

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



Step10 成功推至遠端的 GitHub

```
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)
$ |
```

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

Branch: master ▾ New pull request

Create new file Upload files Find file Clone or download ▾

Ming-Kai Jiau add my sourcecodes Latest commit 1ef0a8f 7 minutes ago

Lab	add my sourcecodes	7 minutes ago
include	add my sourcecodes	7 minutes ago
source	add my sourcecodes	7 minutes ago
temp/debug	add my sourcecodes	7 minutes ago
README.md	Initial commit	42 minutes ago

README.md

MyProject

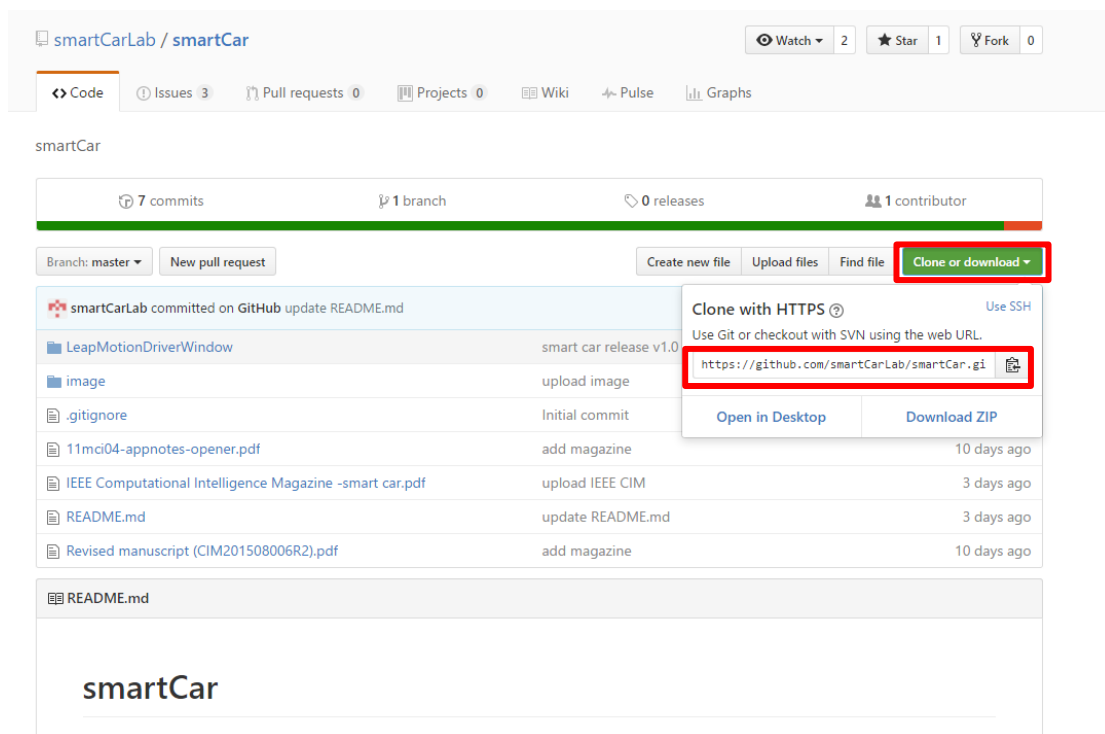
An awesome project to solve OO problem

使用情境二、將某個專案複製到工作目錄下。

Step1 找到要複製的遠端專案

從 GitHub 找到某人所寫的程式，首先點擊如下圖紅色框框內按鈕，將 MyProject 的專屬 URL 複製

註：測試的專案 <https://github.com/smartCarLab/smartCar>



Step2 複製遠端資料庫到本地資料庫

回到剛剛已經執行的 git bash 視窗，執行如下指令，進行遠端資料庫下載：

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

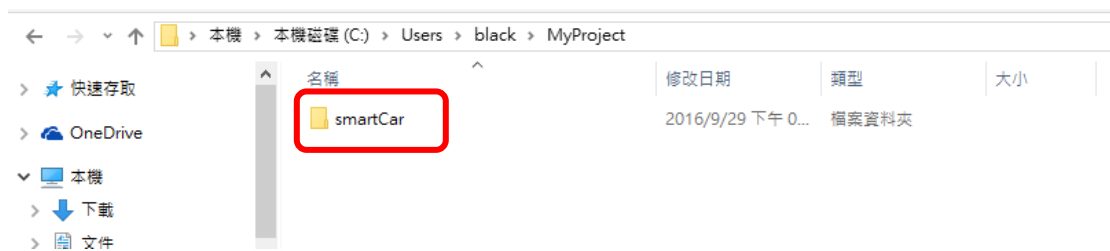
```
MINGW64:/c/Users/black/MyProject

black@DESKTOP-6RA3REA MINGW64 ~
$ cd MyProject

black@DESKTOP-6RA3REA MINGW64 ~/MyProject
$ git clone https://github.com/smartCarLab/smartCar.git
Cloning into 'smartCar'...
remote: Counting objects: 123, done.
remote: Compressing objects: 100% (108/108), done.
remote: Total 123 (delta 14), reused 123 (delta 14), pack-reused 0
Receiving objects: 100% (123/123), 8.82 MiB | 2.49 MiB/s, done.
Resolving deltas: 100% (14/14), done.

black@DESKTOP-6RA3REA MINGW64 ~/MyProject
$ |
```

下載完成之後可以移動至該檔案目錄底下就可以使用該專案的資料庫



Step3 檢查是否成功連結資料庫

試著用 Git 開啟目錄，如果有出現 master 標籤就表示成功

```
MINGW64:/c/Users/black/MyProject/smartCar

black@DESKTOP-6RA3REA MINGW64 ~
$ cd MyProject

black@DESKTOP-6RA3REA MINGW64 ~/MyProject
$ cd smartCar

black@DESKTOP-6RA3REA MINGW64 ~/MyProject/smartCar (master)
$ |
```

←

→

⌵

⬆

📁

>

本機

>

本機磁碟 (C:)

>

Users

>

black

>

MyProject

>

smartCar

>

📌 快速存取

>

OneDrive

>

本機

>

📂 下載

>

📄 文件

>

🎵 音樂

>

🖥️ 桌面

>

🖼️ 圖片

>

🎬 影片

>

本機磁碟 (C:)

>

本機磁碟 (E:)

>

本機磁碟 (F:)

名稱

修改日期

類型

大小

📁 .git

2016/9/29 下午 0...

檔案資料夾

📁 image

2016/9/29 下午 0...

檔案資料夾

📁 LeapMotionDriverWindow

2016/9/29 下午 0...

檔案資料夾

📄

2016/9/29 下午 0...

文字文件

3 KB

📄 11mci04-appnotes-opener

2016/9/29 下午 0...

Adobe Acrobat ...

316 KB

📄 IEEE Computational Intelligence Maga...

2016/9/29 下午 0...

Adobe Acrobat ...

1,602 KB

📄 README.md

2016/9/29 下午 0...

MD 檔案

2 KB

📄 Revised manuscript (CIM201508006R2)

2016/9/29 下午 0...

Adobe Acrobat ...

4,334 KB

五、 參考資料—Git 常用指令

1. `gitk` :顯示樹狀圖形化介面
2. `git add` :加入新增的檔案
3. `git pull` :將遠端的資料更新到本地端
4. `git branch` :創建一個分支
5. `git merge` :合併分支
6. `git log` :檢視提交的歷史紀錄
7. `git reset` :回到特定節點(跟 `revert` 不同，`reset` 是直接砍掉 `commits` 歷史記錄)

※`git push` 將本地端的資料更新到遠端

`--force -f` 強制更新就算可能有危險

※`git commit` 提交一個新的版本

`--amend` 更改目前分支最新版的 `commit`

`--message -m` 直接在後面輸入 `commit message`

※`git checkout` 查看分支或節點

`-b` 先建立分支再切換

※`git stash` 暫存資料

`git stash` 丟進暫存區

`git stash list` 列出所有暫存區的資料

`git stash pop` 取出最新的一筆，並移除。

`git stash apply` 取出最新的一筆 `stash` 暫存資料。但是 `stash` 資料不移除

git stash apply stash@{index} 取出第 index 筆 stash 暫存資料

git stash clear 把 stash 都清掉

※git rebase 重新 commit 一遍

--interactive -i HEAD~n 開啟對話模式編輯(head~n 表示要編輯到前 n 個)

※從 master 版 merge 到主線的指令

git checkout master

git merge develop --no-ff -m "V1.0.x"

git push

※更改過去的 commit 訊息流程

git rebase -i HEAD~n

要修改的 commit 訊息改成 reword

git checkout branch 名稱

git reset --hard

※不小心將要 Commit 到 A 的版本 Commit 到 B

git checkout B

git rebase A

git reset --hard origin/B

git checkout A

git reset --hard xxxx

rebase 跟 reset 都是危險的操作，因為它們是改寫歷史歷史，一不小心資料就不見囉。沒把握的話，操作前最好先用 branch 保存下來。

1.git status：檢查目前分支狀態

如下圖所示，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/itemap.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/
```

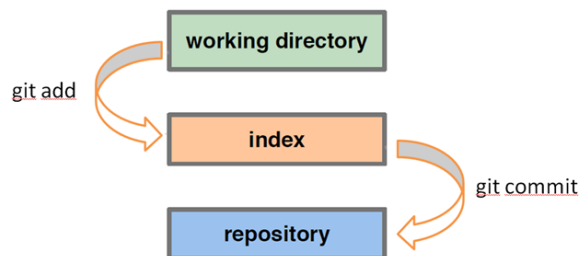
2.git log：查看分支近期 commit 過的點，如下圖所示，查看 commit 的名稱、幾天前、版本號。

※指令範例: git log

```
10f632c - (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)
b7d93fd - 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)
addf636 - yet another way of removing the first elements from an array (1 year, 4 months ago)
6e4ed58 - new bench for Array#slice (1 year, 4 months ago)
9d9ace - 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)
c713e6b - 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)
a2be2d5 - Remove unused rubinius::guards (1 year, 4 months ago)
32ed7a5 - Remove used flag and file it was defined in (1 year, 4 months ago)
5ff4ee2 - Remove unused CallFrameList and some maps (1 year, 4 months ago)
add8f2b1 - Removed unused async message and mailbox code (1 year, 4 months ago)
c4b54ba - Remove unused code (1 year, 4 months ago)
744e9f8 - Fix tiny typo's (1 year, 4 months ago)
912d538 - Cleanup last remnants of dynamic interpreter (1 year, 4 months ago)
6b29b21 - Remove unused IndirectLiterals (1 year, 4 months ago)
83db63a - Fixed Diaest requires in const.missing. (1 year, 4 months ago)
```

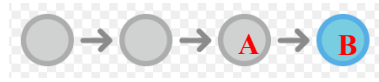
3.git add .：將修改或變更檔案的部分暫存在 index 位址，如下圖所示

※指令範例: 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，原先你在 A 點經由修改專案後 git commit 產生新點 B



※指令範例: `git commit -m"敘述這次 commit 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% (97/97), done.
remote: Total 1026 (delta 398), reused 347 (delta 347), pack-reused 571 receiving objects: 97% (996/1026), 684.00 KiB | 391.00 KiB/s
Receiving objects: 100% (1026/1026), 690.19 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
  5c26e6b..18caa27  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)
$
  
```

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，指令 git push

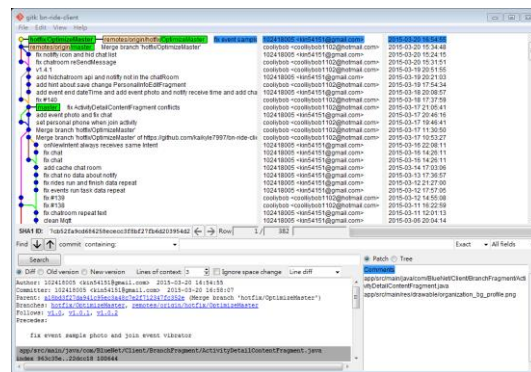
結果如下

GitHub 上的 commit 點 

※指令範例: git push，通常下完指令後需要打帳密確認才開始同步


7.gitk：開啟 git GUI，點擊左上方區塊的 commit 點的詳細資料

※指令範例: gitk



8.git checkout：切換到不同分支上 **develop**

現在在 develop 分支上 ，當下 git checkout master 指令後

master
就會切到 master 分支上 

※指令範例: git checkout master