

開發工具與專案結構介紹

目錄

| | |
|---------------------------------|----|
| Node 與其套件管理 npm | 2 |
| Visual Studio Code 環境操作簡介 | 5 |
| Git 版本控制及幾個常用的指令 | 6 |
| React 目錄結構 | 10 |
| 常見問題 FAQ | 12 |

Node 與其套件管理 npm



Node.js 簡介

官方的定義 ([官方文件](#))

Node.js® is a JavaScript **runtime** built on **Chrome's V8 JavaScript engine**.

- Node 是 JS 的 Runtime(執行環境)，JS 在瀏覽器可以運行，在 Node 下也可以運行
- 因此 Node 也可以來撰寫後端，也可以撰寫 API 程式

npm 簡介

官方在[這裡](#)

- npm 是 Node 的套件管理工具，任何透過 Node 撰寫的套件都可以透過 npm 來管理
- 在前端很常使用到的 npm CLI(指令)

- **npm init**

- 這個指令是讓下指令的那個資料夾被 npm 及 node 管理
- 下完這個指令，該資料夾就會出現生成 *package.json* (詳細的介紹在下一段) 的一些設定

```
C:\Users\jerryweng\Desktop\node-example>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

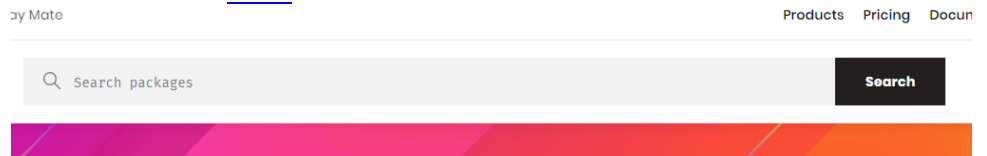
Press ^C at any time to quit.
package name: (example) 
```

- 這邊就是開始輸入 *package.json* 一些設定：像是名稱、版號等等，通常會一直按 Enter 跳過
- 或者也可以在一開始的指令後面加上 **-y**，也就是 **npm init -y**
 - 就會跳過設定 *package.json* 的程序
- **npm install**
 - 他是一個安裝的指令，也就是他會去讀 *package.json*(往下有說明) & *package-lock.json* 的上的設定，幫助你安裝這個專案需要的專案

- 如果要自己新增其他的專案

- `npm install 安裝包的名稱 --save-dev`

- 所有安裝包都可以在[這裡](#)找到，可以打關鍵字搜尋套件名稱



- `--save-dev` 表示，這個安裝包會在開發環境使用到

- 這些套件以及該套件的版本號碼會被放在 `package.json` 檔案的 `devDependencies` 這個 key 的值

```
"devDependencies": {
  "@babel/core": "^7.7.4",
  "@babel/preset-env": "^7.7.4",
  "@babel/preset-react": "^7.7.4",
  "autoprefixer": "^9.7.3",
  "babel-loader": "^8.0.6",
```

- 而如果 `--save-dev` 改成 `--save`

- 則會記錄在 `package.json` 檔案的 `dependencies` 這個 key 的值

```
"dependencies": {
  "@testing-library/jest-dom": "^4.2.4",
  "@testing-library/react": "^9.5.0",
  "@testing-library/user-event": "^7.2.1",
  "react": "^16.13.1",
  "react-dom": "^16.13.1",
  "react-router-dom": "^5.1.2",
  "react-scripts": "3.4.1"
```

- 而如果改成 `-g`

- 則代表全域安裝的意思

- `npm start`

- 其實完整的指令是 `npm run start`

- `run` 可以省略只有在 `npm start` 這個情形底下
- 其他 script 都還是要加上 `run`

- Npm 可以 run 自定義的 script(腳本)，這些腳本就記錄在 `package.json` 的 `script` 這個 key

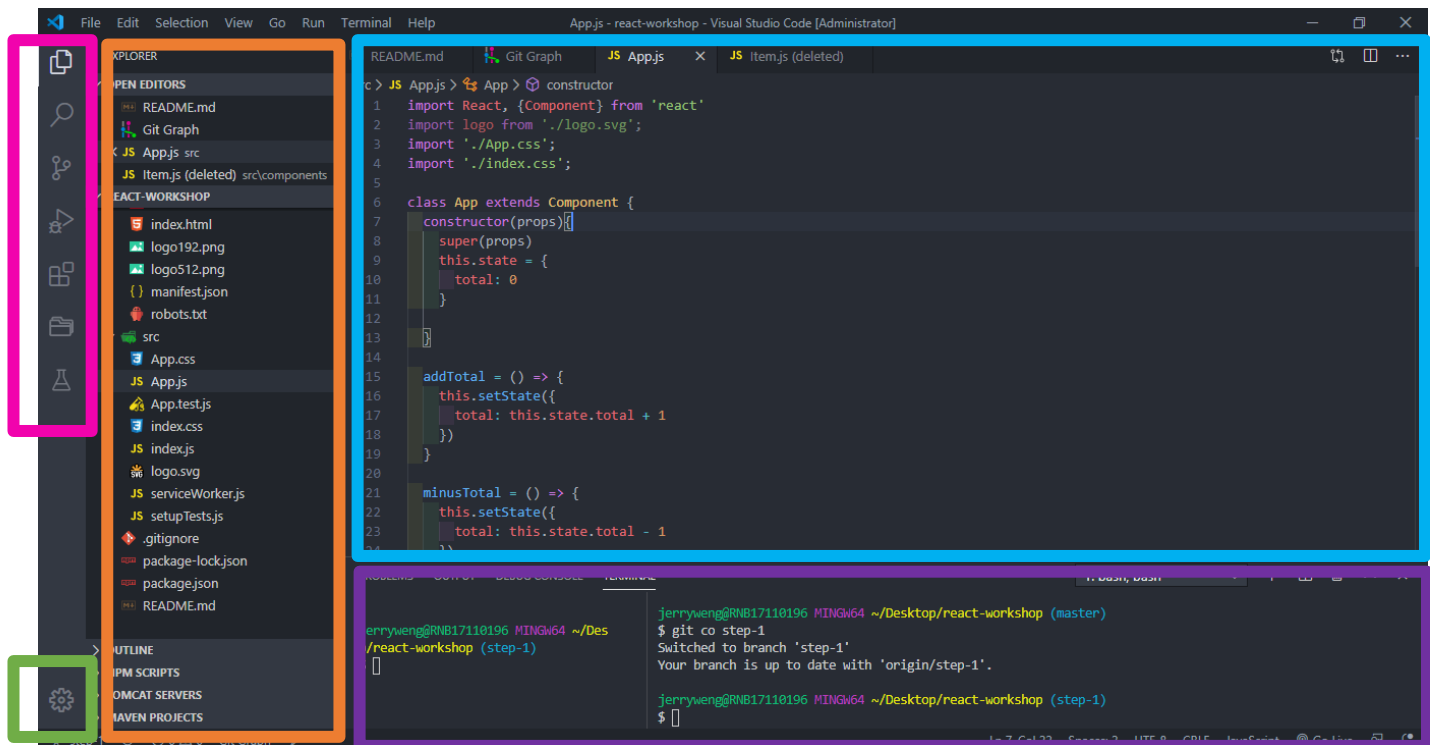
```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

- 像是這包專案的 `start` 是定義 react 幫我們 run local 端的環境
- 也可以自定義其他腳本，給定自定義的 key 值，再接到 `npm` 後面即可

package.json & package-lock.json

- *package.json*
 - *package.json* 是由 `npm init` 這個指令生成的檔案，會記錄這份專案的詳細資訊
 - *package.json* 上記錄的相依套件則是會安裝在 *node_modules* 這個資料夾
 - 這個 *node_modules* 資料夾通常我們會透過 *.gitignore* 來將它排除版本控制，原因是當我們把版本丟到遠端管理的時候，不會一併把這些套件也傳到遠端(太大)，而是透過 `npm install` 來讀取 *package.json* 上的套件，才將套件安裝
- *package-lock.json*
 - 更詳細的套件版本紀錄，可以視為 *package.json* 更詳細的紀錄

Visual Studio Code 環境操作簡介



- Visual Studio Code，簡稱 VSCode
- 粉紅框部分的按鈕，按下，橘色框會依照點選的按鈕不同呈現不同的內容
- 粉紅框部分，由上而下分別是
 - **檔案總覽:** 會列出所有開啟來的檔案、目前開啟來的專案目錄架構
 - 會依照你曾經安裝過的外掛有不同的變化
 - **搜尋:** 可以搜尋內容，找出內容出現在哪個檔案
 - **版本控制:** 如果有搭配 Git，它會顯示目前的版本控制狀況
 - **Debug:** 可以撰寫自己的 Debug 腳本並做 Debug
 - **外掛:** 可以安裝外掛，讓開發更有效率
 - 可以搜尋 VSCode 必裝外掛，可以自己選擇提升效率的外掛
 - 其他按鈕是會看你安裝甚麼外掛，有安裝才會顯示，上面五個是預設值
- 綠色框，為設定按鈕，所有 VSCode 相關的設定如字型大小、外觀都可以在這邊找到
- 藍色框部分，程式部分
 - 藍色框右上角有個類似窗戶的圖示，可以分割畫面
- 紫色框部分，終端機
 - 紫色框右上角有個類似窗戶的圖示，可以分割畫面

Git 版本控制及幾個常用的指令



推薦來讀這份[電子文件](#)，往下拉有目錄；[這份文件](#)也不錯

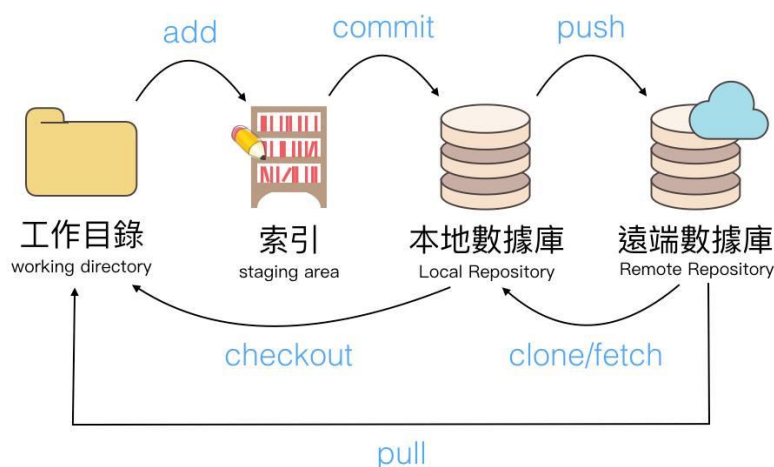
- 要使用 Git，就必須讓整個資料夾的檔案被 Git 託管(就如同我們使用 `node init` 來管理專案一樣)
 - 因此要讓整個資料夾的檔案被 Git 託管，必須使用 `git init` 指令
 - 下指令後，資料夾會生成一個隱藏資料夾 `.git` 表示此資料夾已經被 Git 託管(就是以版本控制的意思)
- .gitignore 檔案
 - 紀錄不要受 Git 控制的檔案
- 使用者相關設定指令

```
$ git config --global user.name "你的名字"
```

```
$ git config --global user.email "你的 Email"
```

- git 會紀錄是誰更動了檔案，這些資訊在之後上傳到遠端的時候也需要用到
- `--global` 是全域設定，如果只想要單獨設定某個資料夾，則可以在該資料夾將 `--global` 改成 `--local`

Git 版本控制的流程



圖片來源:<https://w3c.hexschool.com/git/7ca21e02>

- 加入索引

- 上面圖片的工作目錄，就是你受 git 控制的專案資料夾，今天你在這個資料夾裡面的檔案做更動後，可以下以下指令

- `git status`，會出現以下

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

- 代表這些檔案更動了，但 git 還沒有控制到它，因此要使用
- `git add 檔案名稱`，來將更動的檔案加入控管
  - 通常會一口氣使用 `git add .`，這個 `.` 就是將所有檔案再次加到控管

- 加入本地數據庫

- 當今天可能做完一個新功能後，也將這些更動加到控管，等於有了一個新的版本(比起還沒開始開發新功能時)，這時候要把這些更動記錄起來成為一個新的版本，就使用 `git commit` 指令，將這個版本存到本地的數據庫 (repository)
  - 詳細是: `git commit -m "自定義的內容"`
  - `-m` : 這個是可以打自定義內容的條件
  - 自定義的內容，通常會寫這次更動做了甚麼，像是"修正 bug" 等等

- 新增分支

- 專案通常有多人開發，每個人都要負責特定的功能，因此我們從主要分支(master)，各開各的支線，各自去開發自己的功能後，再合併(merge)起來
  - `git branch 新分支的名稱`

- 切換分支

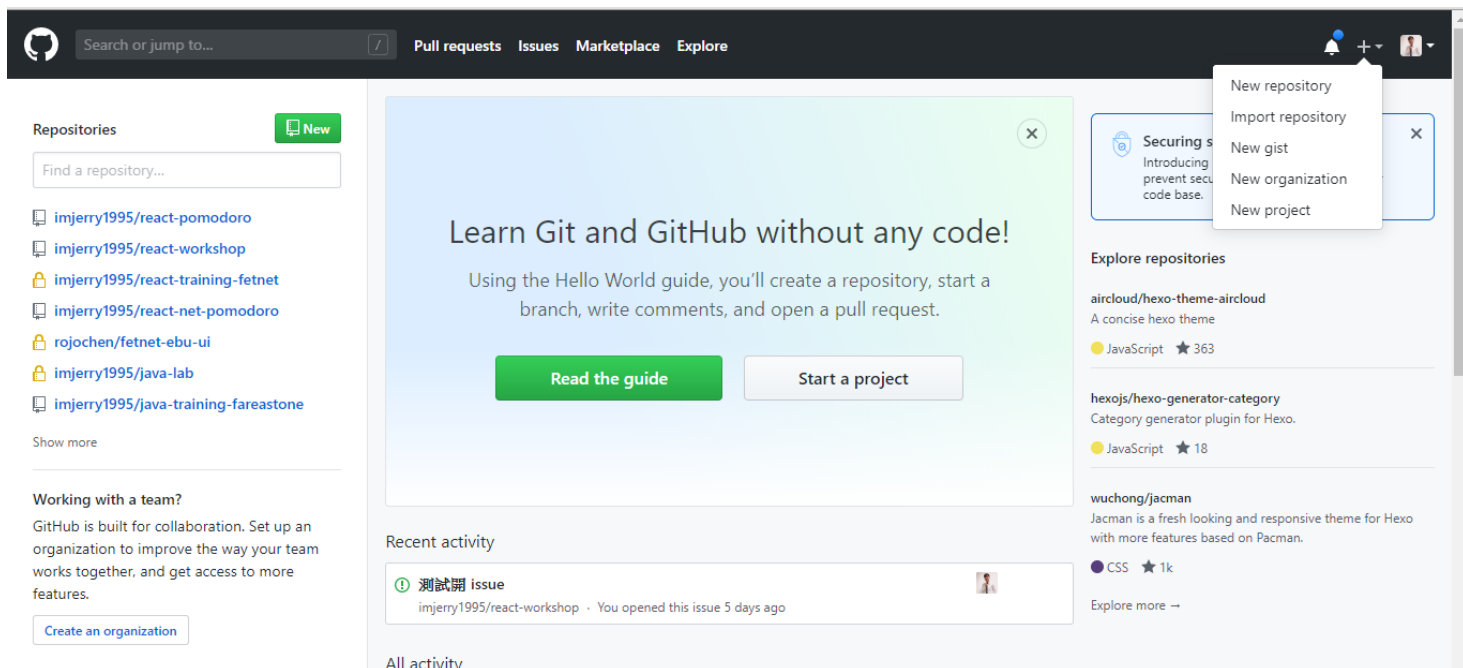
- `git checkout 分支的名稱`

- 合併分支

- 要先切換到主要的分支(不一定是 master，也有可能是小分支要合併掉小分支)
- `git merge 要被合併掉的分支名稱`

- 上傳(推送)到遠端

- 遠端數據庫就是類似 [GitHub](#)、[GitLab](#)、[BitBucket](#) 等等的平台
- 要推送到遠端的數據庫(repository，簡稱 repo)，就必須先在遠端新增一個數據庫，以下用 GitHub 舉例




- 在 GitHub 網頁畫面右上角有個 + 符號，按下 New repository

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)



Owner

Repository name \*

 imjerry1995 ▾ /

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

Description (optional)

- ☒  **Public**  
Anyone can see this repository. You choose who can commit.
- ☐  **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

- ☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾ ⓘ

Create repository



- 輸入數據庫名稱，選擇是否要私人，最後按下 Create repository

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

Set up in Desktop or HTTPS SSH `https://github.com/imjerry1995/exampleOpen.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 "# exampleOpen" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/imjerry1995/exampleOpen.git
git push -u origin master
```

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

```
git remote add origin https://github.com/imjerry1995/exampleOpen.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.

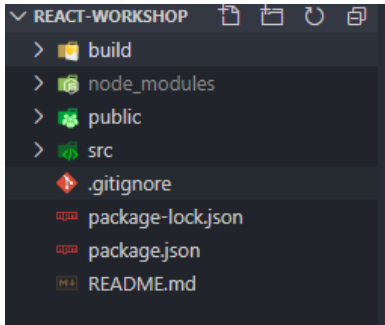
Import code

- 之後跳到這頁，會告訴你如何連結你的本地數據庫
  - 如果是還沒在本地建立資料夾，則在本地的任何一個資料夾下紅框(上圖)的指令
  - 如果是已經有本地專案資料夾，則在該專案資料夾下藍框(上圖)的指令
- 主要的上傳指令
  - `git remote add origin` 遠端數據庫位置
    - 這是連結本地數據庫跟遠端數據庫的指令
  - `git push -u origin master`
    - 將本地數據庫 master 分支的檔案上傳到遠端的 master 分支
    - 所以如果要上傳其他分支名稱，只要把 master 改成其他分支名稱即可

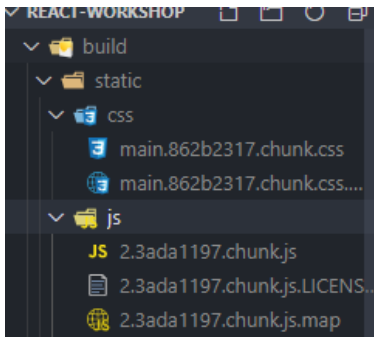
以上只是 git 的簡要介紹，詳細的學習建議還是看提供的電子書會比較能建立正確的概念！

# React 目錄結構

## 概觀



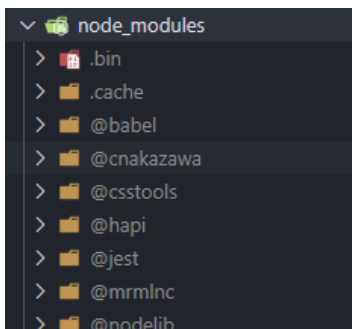
## build 資料夾



(圖片是部分擷取)

- 這個資料夾是透過 `npm run build` 指令生成的，是要放到伺服器或環境上的檔案，生成的檔案都是瀏覽器看得懂的
  - 也就是前處理如 scss 檔案，變成 css 檔案
  - ES6 的語法，變成以前的 JS 語法
  - React 的語法，全部打包成一般 JS 語法
  - 這個資料夾通常不受 git 控制

## node\_modules 資料夾

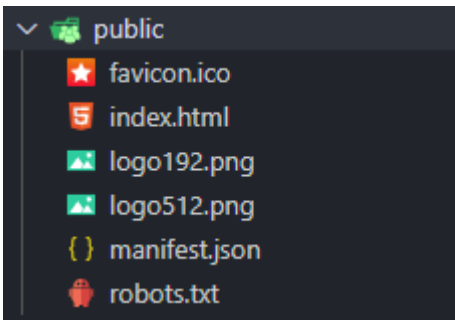


(圖片是部分擷取)

- 這個資料夾就是 `npm install` 安裝的套件及其相依套件的資料夾

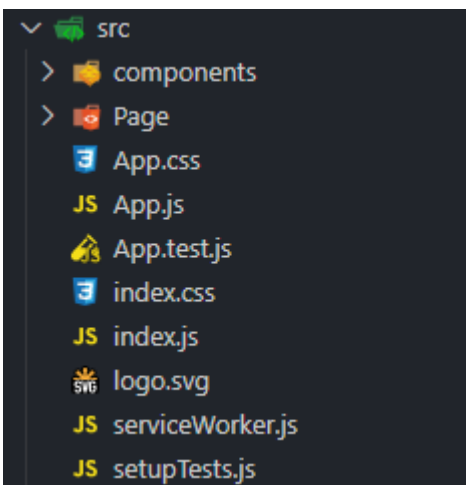
- 這個資料夾通常不受 git 控制

## public 資料夾



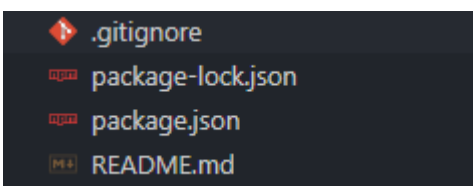
- 存放 html 、或其他圖片檔案

## src 資料夾



- React 原始碼、CSS、測試檔案等主要程式都放在這邊

## 根目錄



- *.gitignore* 檔案: 紀錄不被 git 控制的檔案
- *package-lock.json*: 記錄此專案的詳細資訊，由 `npm init` 指令生成
- *package.json*: 記錄此專案的資訊，由 `npm init` 指令生成
- *README.md*: markdown 檔案，可以自由輸入闡述此專案的內容

# 常見問題 FAQ

剛開始比較多人會遇到的問題

## Q1 為何我不能在專案資料夾使用 git 指令?

如果你下載 GitHub 上專案是透過 Download ZIP 檔案，此方式這個資料夾沒有被 git 管控，也就是沒有存在 *.git* 這個檔案，所以才不能使用 git 指令

這種方式若要使用不同分支，只能再去分支頁面下載不同分支的檔案

## Q2 為何我不能下 `npm install` 指令?

跟第一題是類似的問題，npm install 指令會去讀取 *package.json* 檔案，因此不能下指令可能是因為你並不在專案資料夾的路徑底下，所以 npm 找不到 *package.json* 檔案，切換到專案資料夾底下即可

## Q3 為何我不能使用 `npm start` 指令?

在我們的專案資料夾中，npm start 是透過 react 開啟一個本地端的 server，因此前提是你的專案資料夾要有安裝 react 套件，所以在 npm start 之前要先 npm install 讓 npm 去安裝 *package.json* 檔案上記錄的套件