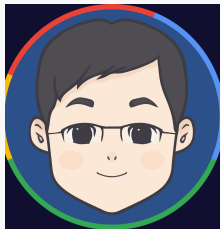


Git程式碼時光機

2023/10/04 19:00~22:00



小松
@XiaoSong

```
org = filterByOrg ? study.lead_organization === filterByOrg : true  
status = filterByStatus ? study.status === filterByStatus : true  
matchStatus) {  
  return true  
}  
  
function filterStudies(studies, filterByOrg, filterByStatus) {  
  return studies.filter(study => {  
    return filterByOrg === study.lead_organization &&  
    filterByStatus === study.status &&  
    !filterByStatus || study.status === filterByStatus  
  })  
}
```

About me

- 中央網學所碩一
- 年會志工愛好者
- 不務正業的資訊人
- 邁向全端工程師



My Telegram
@song0922

今日流程

- 開場
- Git簡介
- 環境安裝
- Git操作及練習

Slides



<https://reurl.cc/NyqXQm>

Handouts



<https://reurl.cc/Wvvbak>

今日提問Slido

#2272102

<https://reurl.cc/edX92K>



slido



你常用的編輯器

- ① Click **Present with Slido** or install our [Chrome extension](#) to activate this poll while presenting.

slido



你對Git的了解程度

① Click **Present with Slido** or install our [Chrome extension](#) to activate this poll while presenting.

有一天, 小明...

project.cpp

projectFinal.cpp

project.cpp

project2.cpp

FinalFinalFinal.cpp

projectFinalFinal.cpp



大家如何管理Code

EX:

test.cpp

test20230101.cpp

test20230101-2.cpp

test_final.cpp

test_final_ok.cpp

.....



試問

以上的例子中

前一個版本和後一個版本到底修改了哪些bug

我們無從得知

解決方式：

每次都另存新檔？可是牽涉讀檔的時候，檔名是寫死在code中的，無法動態修改成最新檔名，此法**不可行**。~~(把所有修改記錄寫在excel中)~~

情境

合作開發的專題

組員A:我先寫功能的部分, 界面你來處理

組員B:我的界面寫好了, 把功能的部分傳過來吧

悲劇發生:

A和B使用的檔名都相同, 結果B的檔案被A覆蓋了.....

B含淚重寫



怪我咯?

如何解決這樣的問題

- 萬能的Ctrl Z

缺點:檔案的覆蓋無法使用Ctrl Z

- 另存新檔

缺點:檔名一定要不同, 而且不知道每次修改了什麼

- 人腦記憶法

我都記得每次我改了什麼 呵呵呵呵



分散式版本控制系統

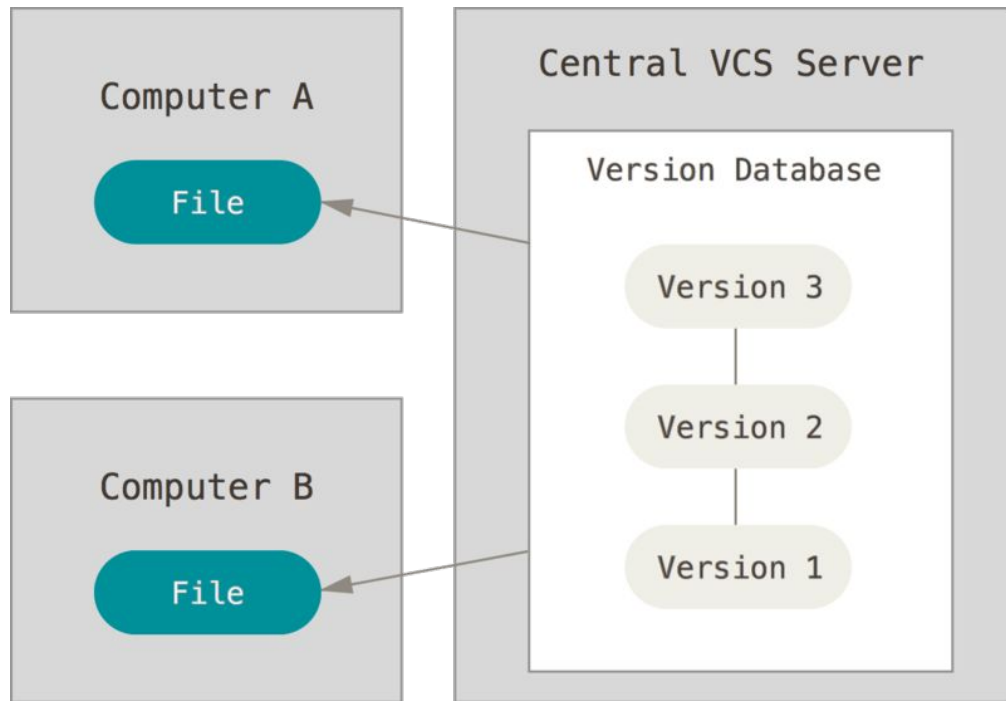


code的時光機

每完成一階段工作
都幫code拍張照片
想像成父母拍照記錄孩子成長的樣子

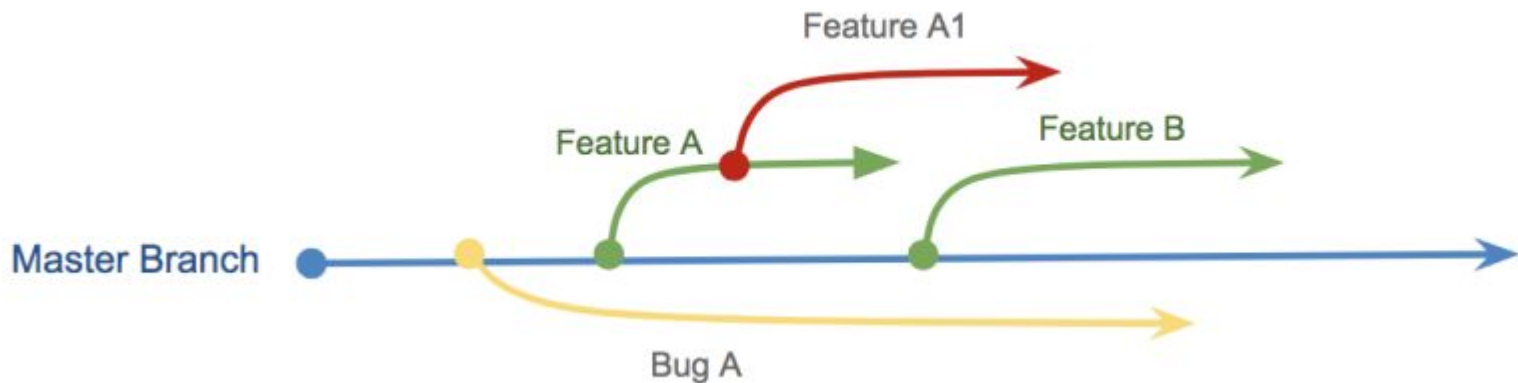
何為分散式？

- 程式碼保存在多地
- 優點：多地備份、離線存取
- 缺點：公司機密程式碼洩露



何為版本？

- 程式修改的過程
- 程式碼的歷史記錄



Bugfix-12: Large PRs with Many Diffs #16

Merged alexmdodge merged 4 commits into master from bugfix/12/many-diffs-loading on Apr 2

Conversation 0 Commits 4 Checks 0 Files changed 17

Changes from all commits ▾ Jump to... ▾ +684 -181



```
8 extension/manifest.json
@@ -21,9 +21,9 @@
21 21     "author": "Alex Dodge",
22 22     "content_scripts": [
23 23         {
24 -         "matches": ["https://github.com/*/pull/*/files"],
25 -         "css": ["lib/gdfe_styles.css"],
26 -         "js": ["lib/gdfe_script.js"],
24 +         "matches": ["https://github.com/*/pull/*"],
25 +         "css": ["lib/gde_styles.css"],
26 +         "js": ["lib/gde_script.js", "lib/gde_vendor.js"],
27 27         "run_at": "document_end"
28 28     }
29 29 ],
@@ -32,6 +32,6 @@
32 32     "tabs",
33 33     "http://*.google.com/"
34 34 ],
35 -     "short_name": "GDE",
35 +     "short_name": "GitHub Diff Explorer",
```

紅色刪除

綠色新增

VS Code安裝

<https://code.visualstudio.com/download>

- Visual Studio Code(簡稱 VS Code)是一款由微軟開發且跨平台的免費原始碼編輯器
- 套件支援

Visual Studio Code 預設支援非常多的程式語言

包括 JavaScript、TypeScript、CSS 和 HTML

支援 Python、C/C++、Java 和 Go 在內的其他語言。



Github 註冊

<https://github.com/>

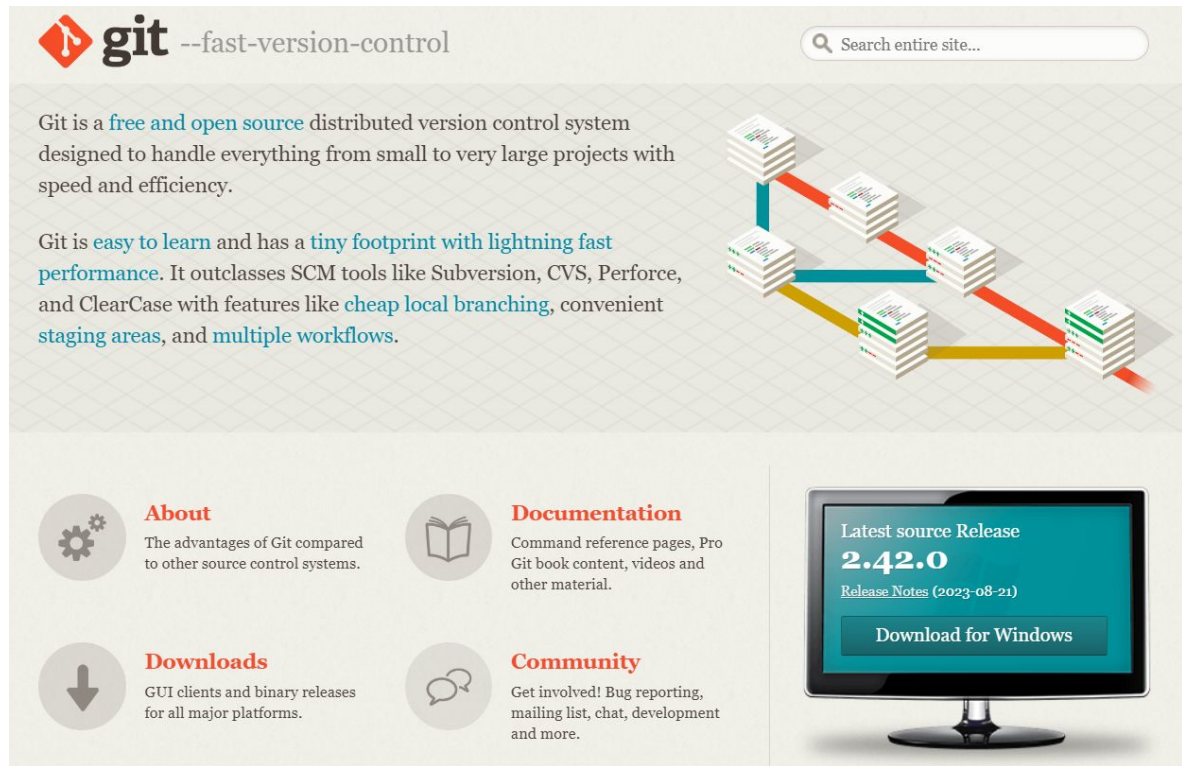
- 線上軟體原始碼代管服務平台，用Git作為版本控制軟體
- 截至2023年1月26日，已經有超過1億開發人員使用GitHub
- 協作、共同開發平台
- —工程師交友平台—



Git 下載

<https://git-scm.com/>

<http://git-scm.com/download/win>



The screenshot shows the Git website homepage. At the top left is the Git logo (a red diamond with a white 'G') followed by the text 'git --fast-version-control'. To the right is a search bar with the placeholder text 'Search entire site...'. Below the header, there are two paragraphs of text. The first paragraph describes Git as a 'free and open source' distributed version control system. The second paragraph describes Git as 'easy to learn' and having a 'tiny footprint with lightning fast performance'. To the right of the text is a diagram showing a network of Git repositories represented as stacks of books connected by colored lines (red, blue, yellow). Below the text and diagram, there are four circular icons with corresponding text: 'About' (gear icon), 'Documentation' (book icon), 'Downloads' (downward arrow icon), and 'Community' (speech bubble icon). On the right side, there is a monitor displaying the latest source release '2.42.0' and a button labeled 'Download for Windows'.

git --fast-version-control

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

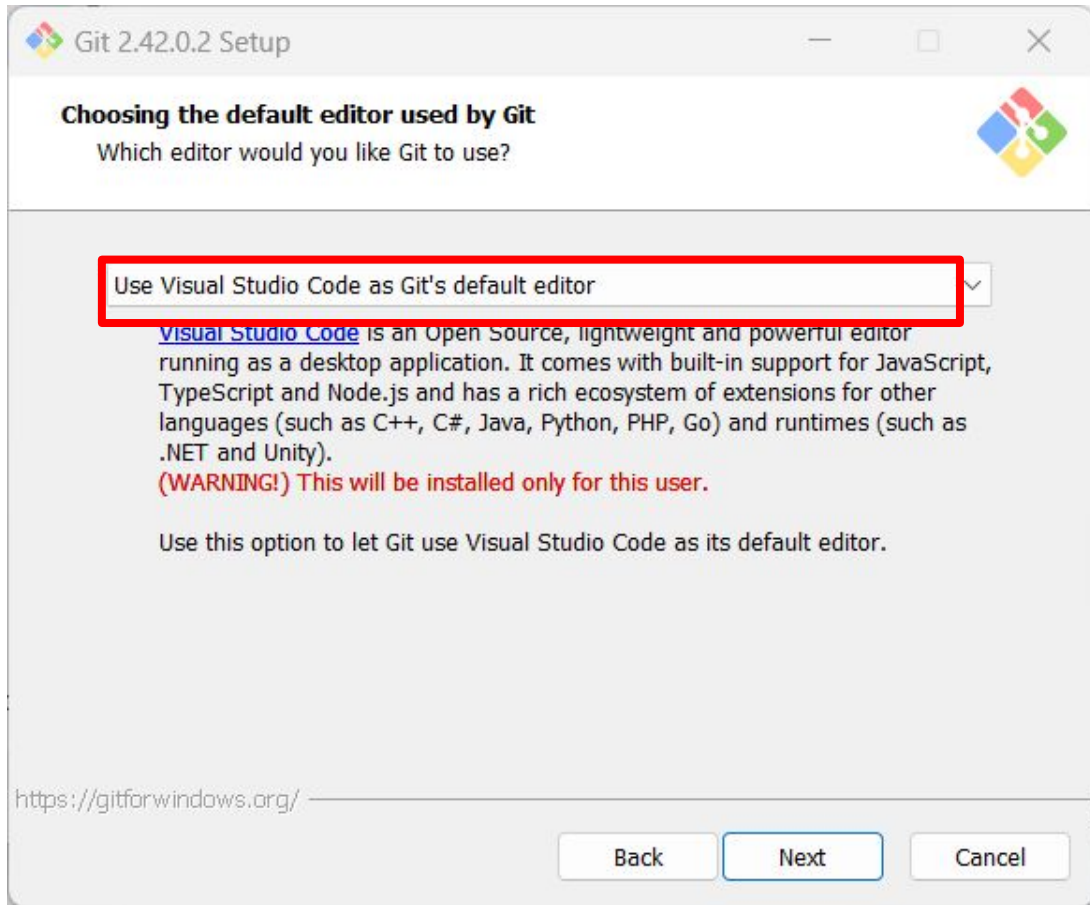
Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.42.0
Release Notes (2023-08-21)
[Download for Windows](#)

注意事項！

在安裝Git時

請選擇VS Code作為預設編輯器



如果不幸進入Vim

A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. In the top right corner, there are icons for a terminal, a plus sign, a minus sign, a window, a trash can, and a menu. The terminal content shows the output of a git commit command. It starts with a cursor on the first line. Then, it displays instructions: '# Please enter the commit message for your changes. Lines starting with '#' will be ignored, and an empty message aborts the commit.' followed by '#'. Then, it shows '# On branch master' and '#'. Next, it says '# Initial commit' and '#'. Then, it displays '# Changes to be committed:' followed by '# new file: index.html' and '#'. Below this, there are several lines of '~'. At the bottom, there is a light gray bar with the text '.git/COMMIT_EDITMSG[+] [unix] (23:08 02/10/2023)' on the left and '1,1 All' on the right. Below this bar, the text '-- INSERT --' is visible.

如果不幸進入Vim

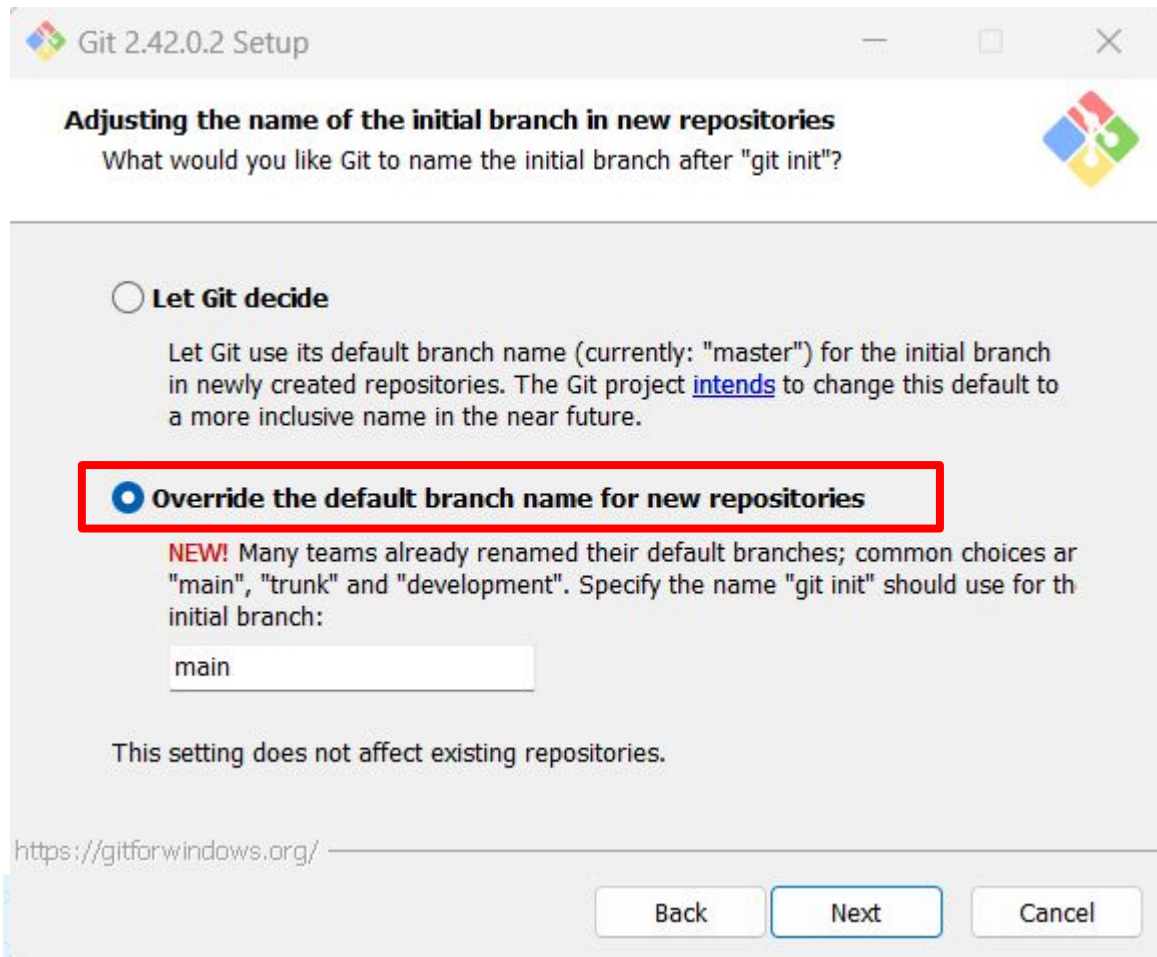
```
$ git config --global core.editor "code --wait"
```

修改預設編輯器為VS Code

注意事項！

在安裝Git時

main作為預設主分支



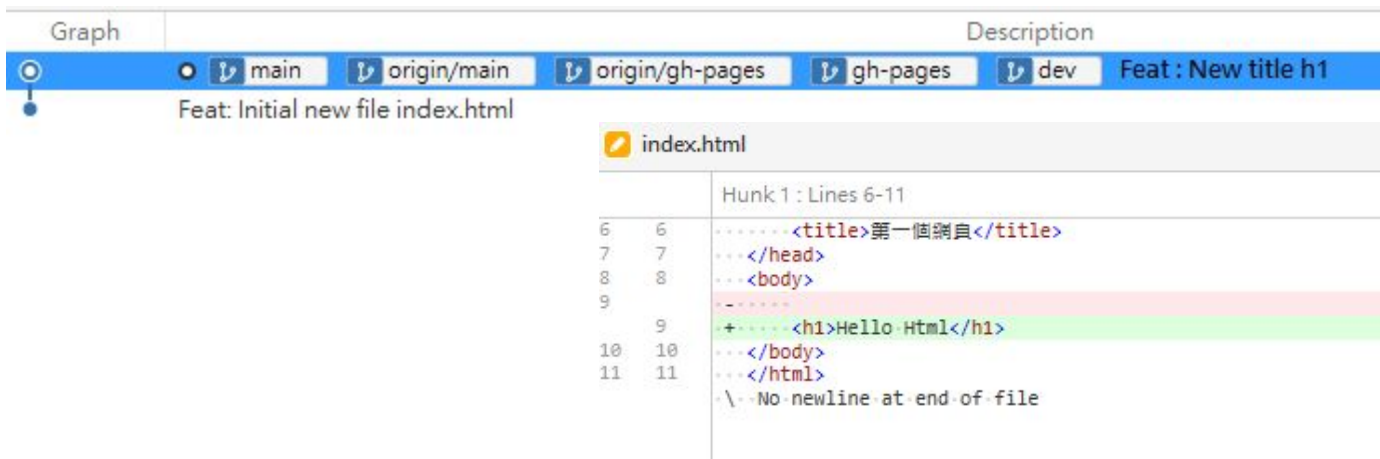
將 Git 預設分支從 Master 改成 Main

```
$ git config --global init.defaultBranch main
```

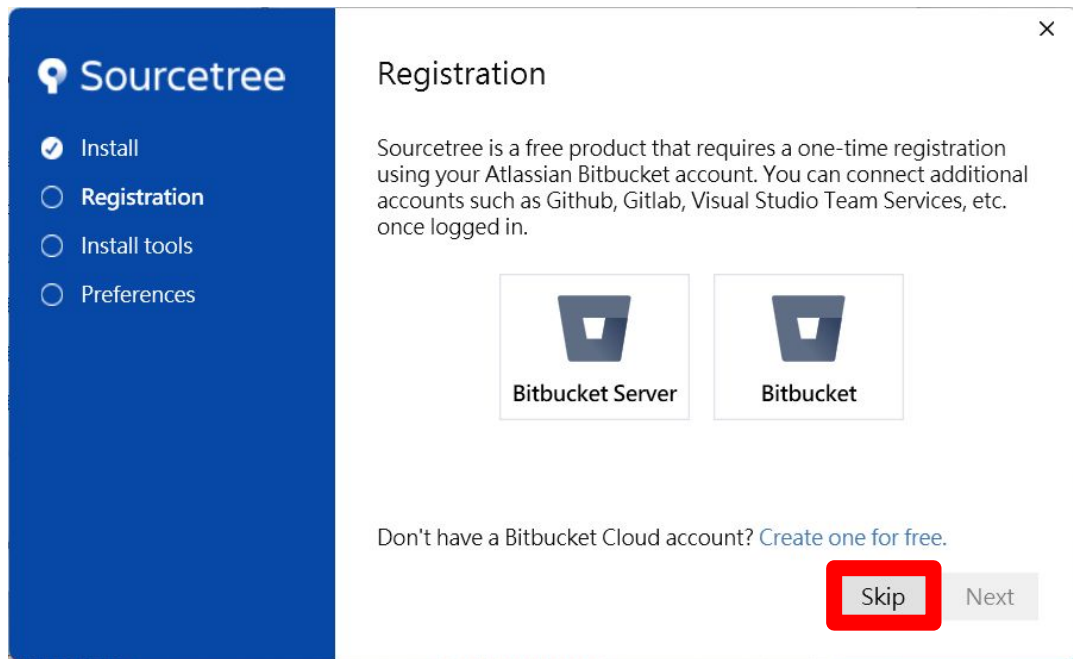
Sourcetree 下載(非必須)

<https://www.sourcetreeapp.com/>

- 用 GUI 介面來管理版本控制內容的軟體



Sourcetree 下載



免費帳號可使用的私有版本庫
不限數量
但最多可支援5名使用者

安裝實作時間

20:00分鐘

1. vscode安裝
2. Github註冊
3. Git安裝
4. sourcetree安裝

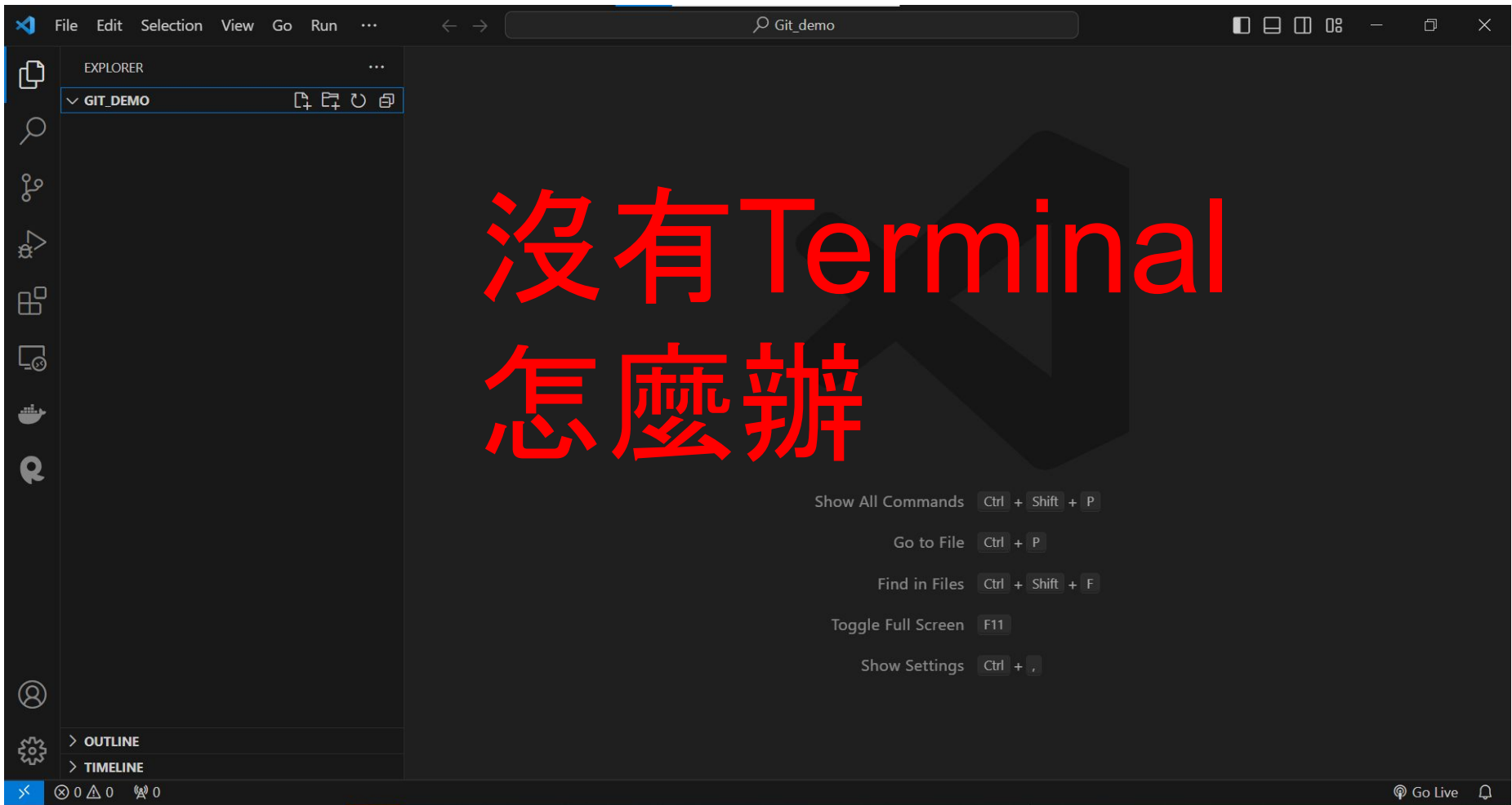
VS Code 打開專案新資料夾

先建立一個資料夾

File > Open Folder > 選擇你的專案所在的資料夾

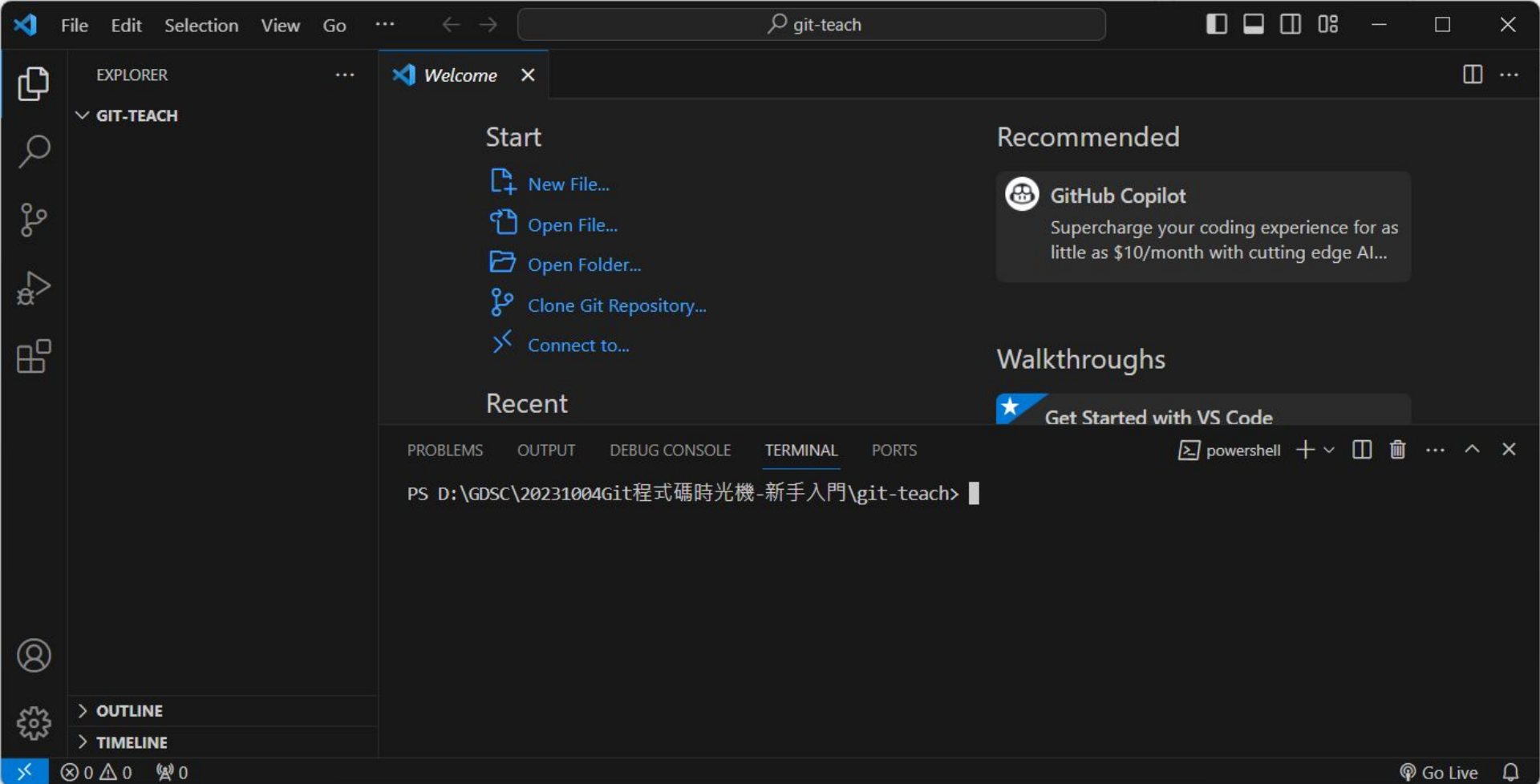
Tips:

請使用【資料夾】而不是單獨打開單一文件



VS Code 打開terminal

ctrl + ~



Git 本地端操作(#0)-所有指令教學

```
$ git help
```

查詢git各種功能及說明

git help

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an
existing one

.

.

.

Git 本地端操作(#1)-版本號查詢

```
$ git --version
```

查詢目前git版本

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git --version  
git version 2.42.0.windows.2
```

```
PS D:\git-teach> git --version
```

```
git version 2.42.0.windows.2
```

Git 本地端操作(#2)-參數說明

```
$ git config
```

查詢目前git系統參數說明

PS D:\git-teach> **git config**

usage: git config [<options>]

Action

--get

get value: name

[value-pattern]

Config file location

--get-all

get all values:

--global

use global

key [value-pattern]

config file

--get-regexp

get values for

--system

use system

regexp: new-name

config file

--remove-section

remove a section:

--local

use repository

name

config

-l, --list

list all

Git 本地端操作(#2-1)-本機參數說明

```
$ git config --list
```

查詢目前git本機參數

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.fsmonitor=true
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
core.editor="C:\Users\ccssl\AppData\Local\Programs\Microsoft VS Code\bin\code" --wait
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=120061203
user.email=53130716+120061203@users.noreply.github.com
```

Git 本地端操作(#3)-設定賬號

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

讓 Git 知道這台電腦做的修改要連結到哪一個使用者

XXX請修改成自己的代號(數字英文皆可)

EX: `git config --global user.name "xiaosong"`


```
PS D:\git-teach>
```

```
git config --global user.name "xiaosong"
```

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach>
```

```
git config --list
```

```
user.name=xiaosong
```

Git 本地端操作(#4)-設定郵箱

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

讓 Git 知道使用者github郵箱

YYY@gmail.com請修改成自己的郵箱

EX: git config --global user.email "xiaosong@gmail.com"

```
PS D:\git-teach>
```

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

```
PS D:\git-teach>
```

```
git config --list
```

```
user.email=xiaosong@gmail.com
```

恭喜邁入Git新手村



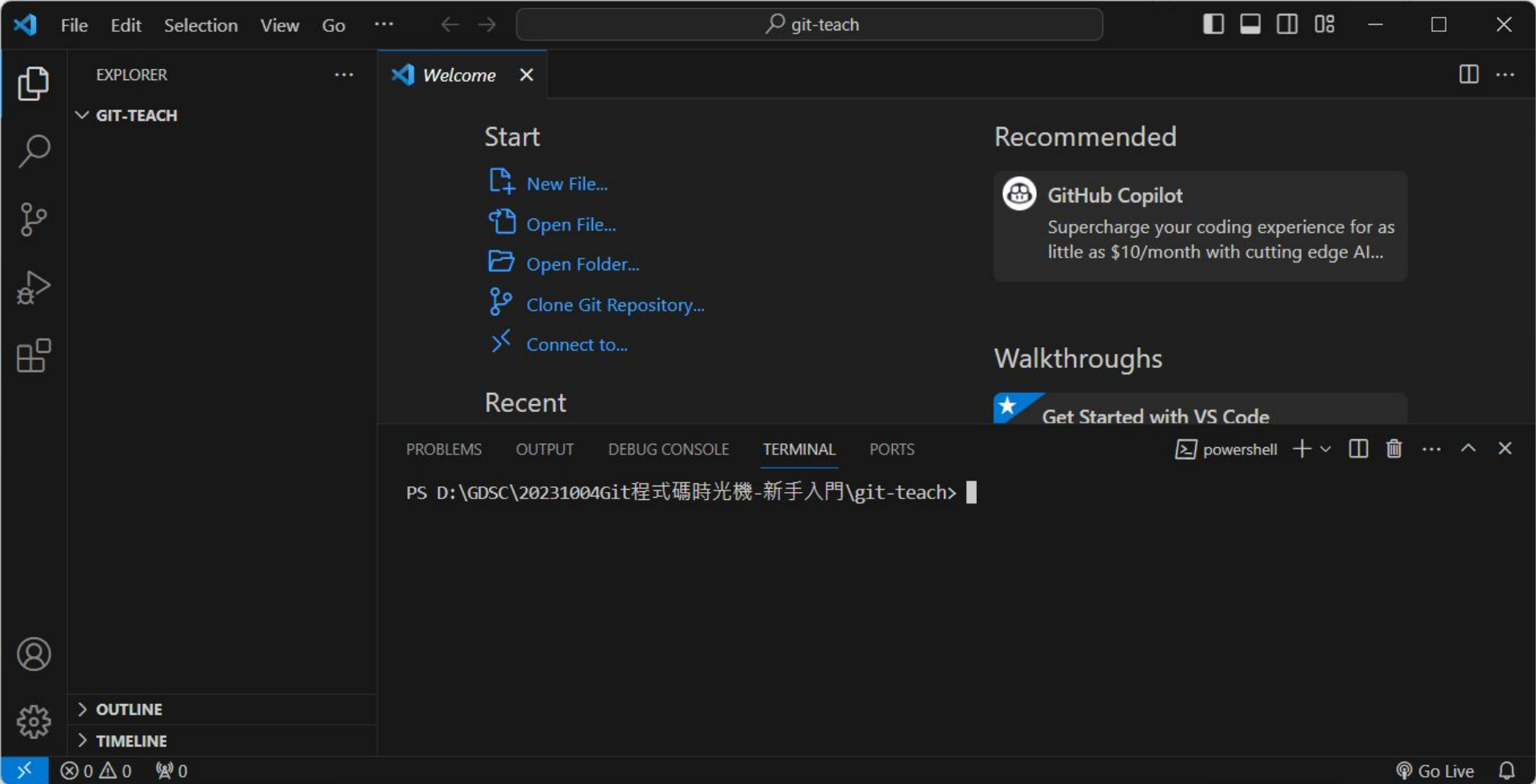
實作時間

10:00分鐘

0. `git help`
1. `git --version`
2. `git config --list`
3. `git config --global user.name
"XXX"`
4. `git config --global user.email
"YYY@gmail.com"`

新增及初始化 Repository





Git 本地端操作(#5)-初始化

```
$ git init
```

讓 Git 知道目前這個資料夾需要用Git版本控制

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git init  
Initialized empty Git repository in D:/GDSC/20231004Git程式碼時光機-新手入門/git-teach/.git/
```



```
PS D:\git-teach> git init
```

```
Initialized empty Git repository in  
D:/git-teach/.git/
```

我們做了什麼

使用 `git init` 指令初始化這個目錄

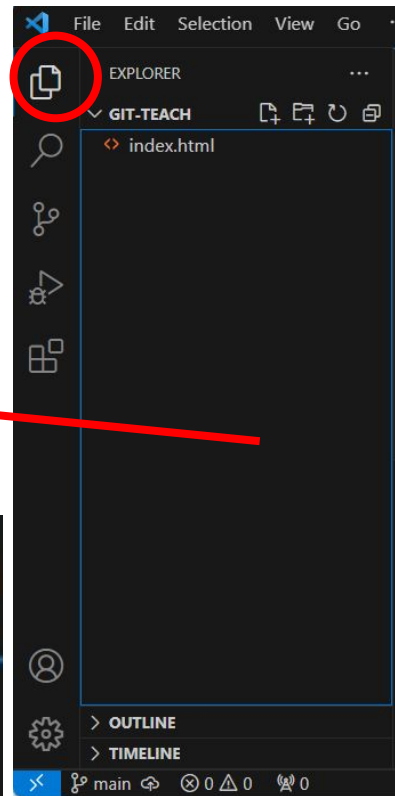
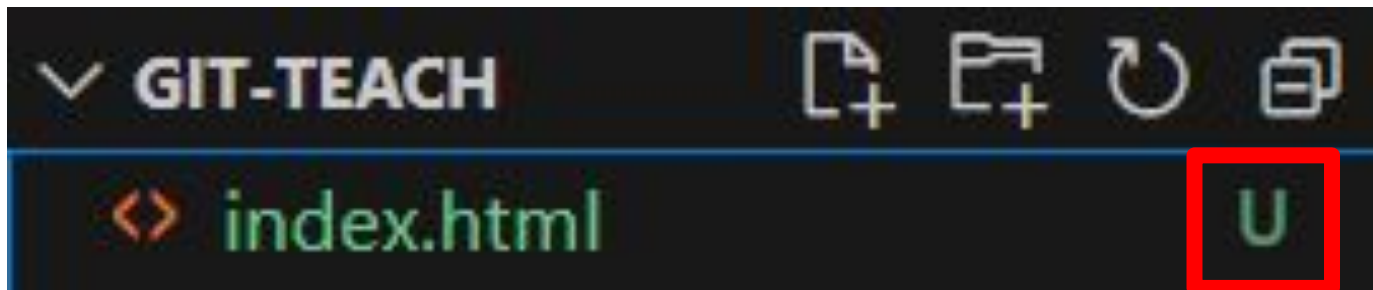
主要目的是要讓 Git 開始對這個目錄進行版本控制。

tips:

小數點開頭的目錄或檔案名稱(例如 `.git`)，在一些作業系統中預設是隱藏的，可能會需要開啟檢視隱藏檔之類的設定才看得得到。

建立index.html

右鍵 > New File... > index.html



Git 本地端操作(#6)-查詢檔案狀態

\$ git status

Untracked files是什麼？

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git status
On branch main

No commits yet

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

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

實作時間

10:00分鐘

5. `git init`

建立index.html

6. `git status`

Why Untracked

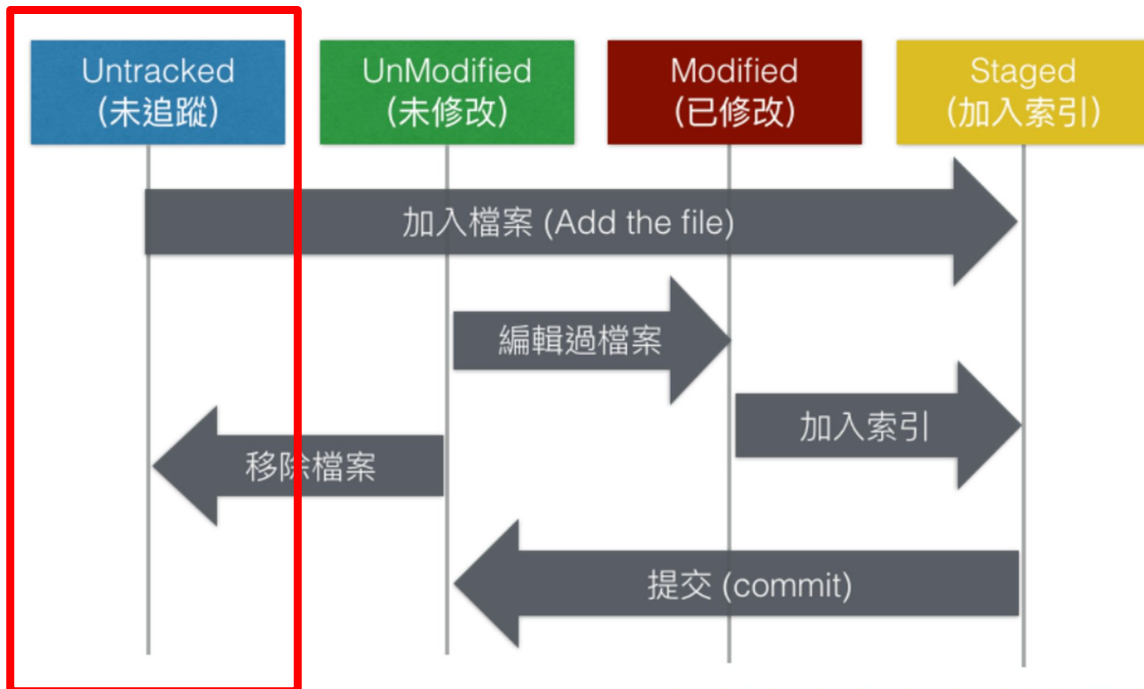
檔案目前的狀態是 `Untracked files`

意思是這個檔案尚未被加到 Git 版控系統裡

還沒開始正式被 Git 「追蹤」

它只是剛剛才加入這個目錄而已

Git 檔案追蹤狀態



建立專案目錄後
新檔案
將會是未追蹤狀態

Git 本地端操作(#7)-追蹤檔案

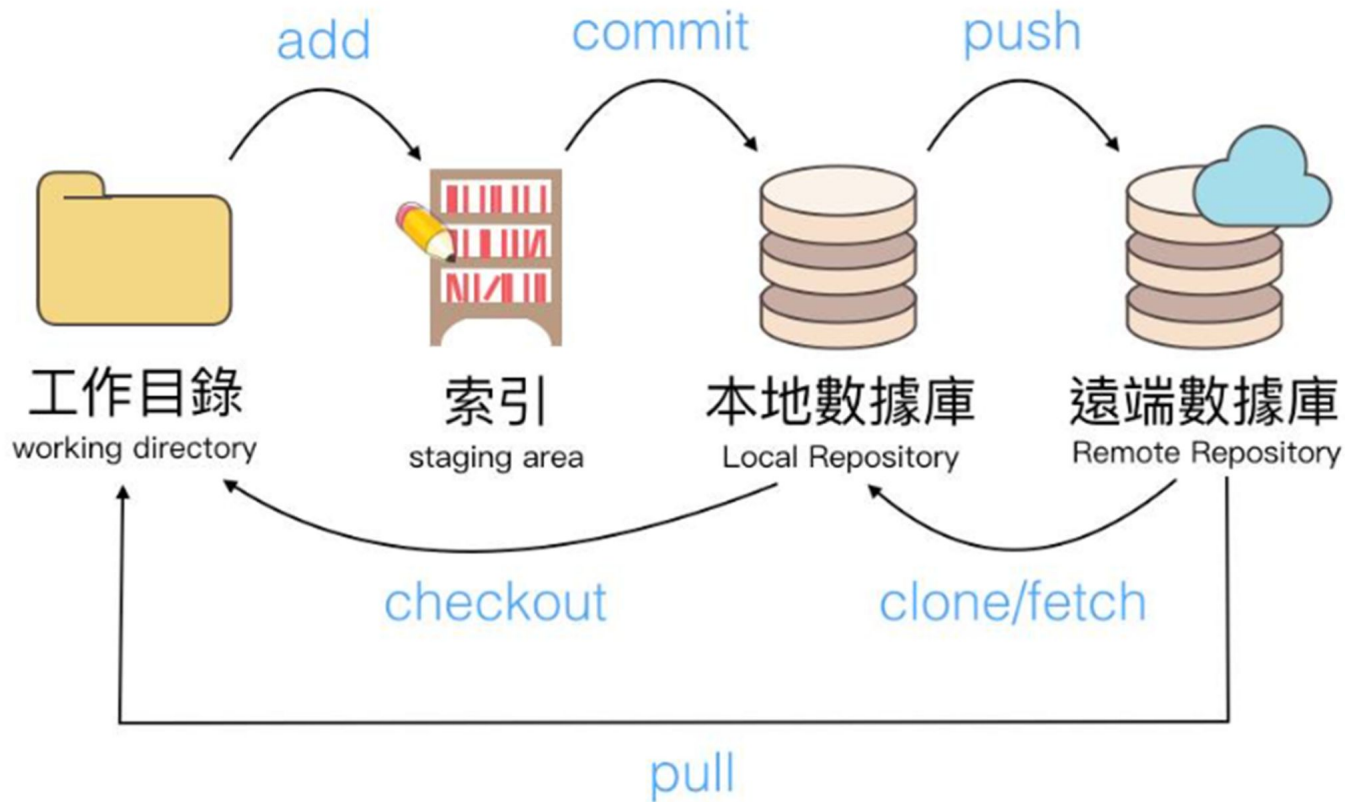
```
$ git add index.html
```

把 `welcome.html` 這個檔案交給 Git, 讓 Git 開始「追蹤」它

```
$ git add .
```

把當下這個目錄, 以及它的子目錄、子子目錄..裡的異動
全部加到**索引**

Git 檔案追蹤目錄



不想追蹤特定檔案

不要追蹤(手動)

不要把該檔案add到索引中

加入 .gitignore

把不想追蹤的檔案名稱寫入，git會自動忽略該檔案

在把資料同步到github時，有時候會同步到一些不必要的檔案。

例如：環境設定(Eclipse的.metadata檔)、編譯檔(java的.class檔)...等等

```
/node_modules  
/storage/*.key  
/vendor  
.env
```



Git 本地端操作(#6)-再次查詢狀態

\$ git status

✓ GIT-TEACH

<> index.html

A

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git status
```

```
On branch main
```

```
No commits yet
```

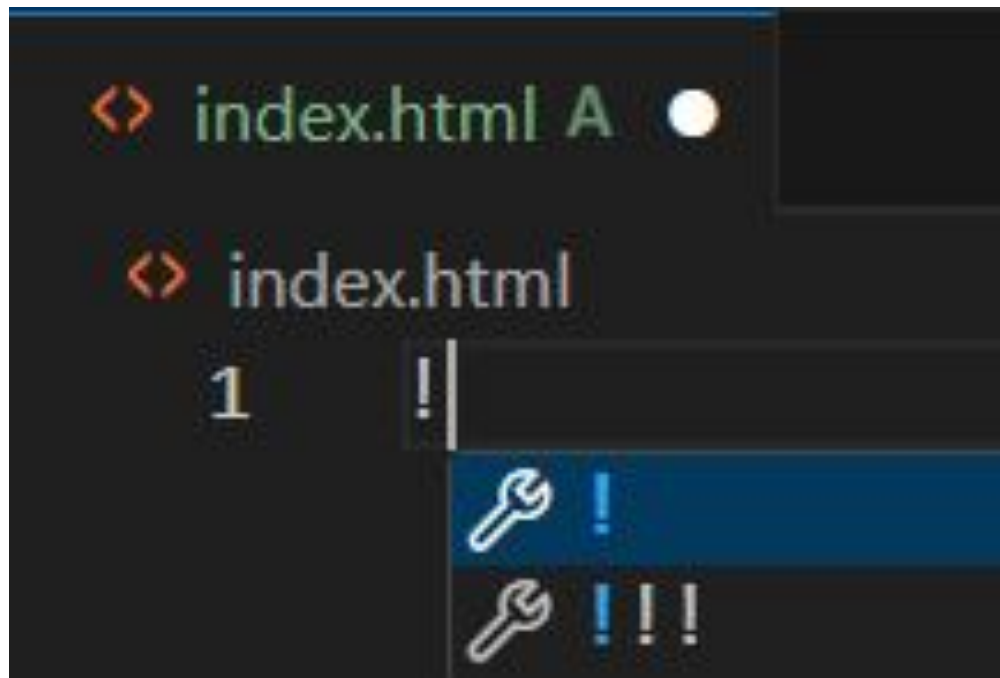
```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   index.html
```

建立基本HTML架構

\$! + TAB鍵



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>我的第一個網頁</title>
</head>
<body>
</body>
</html>
```

Git 本地端操作(#6)-再次查詢狀態

記得Ctrl + S存檔

\$ git status

▼ GIT-TEACH

<> index.html M

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git status
On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   index.html
```

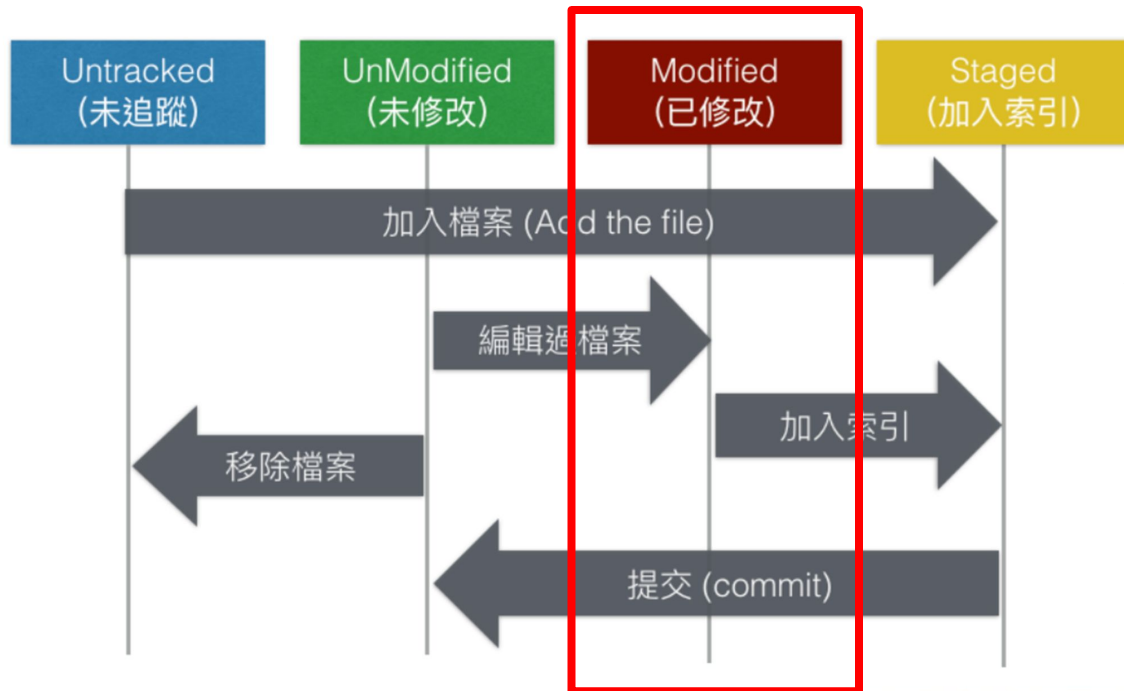
```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git restore <file>..." to discard changes in working directory)
```

```
modified:   index.html
```

Git 檔案追蹤狀態

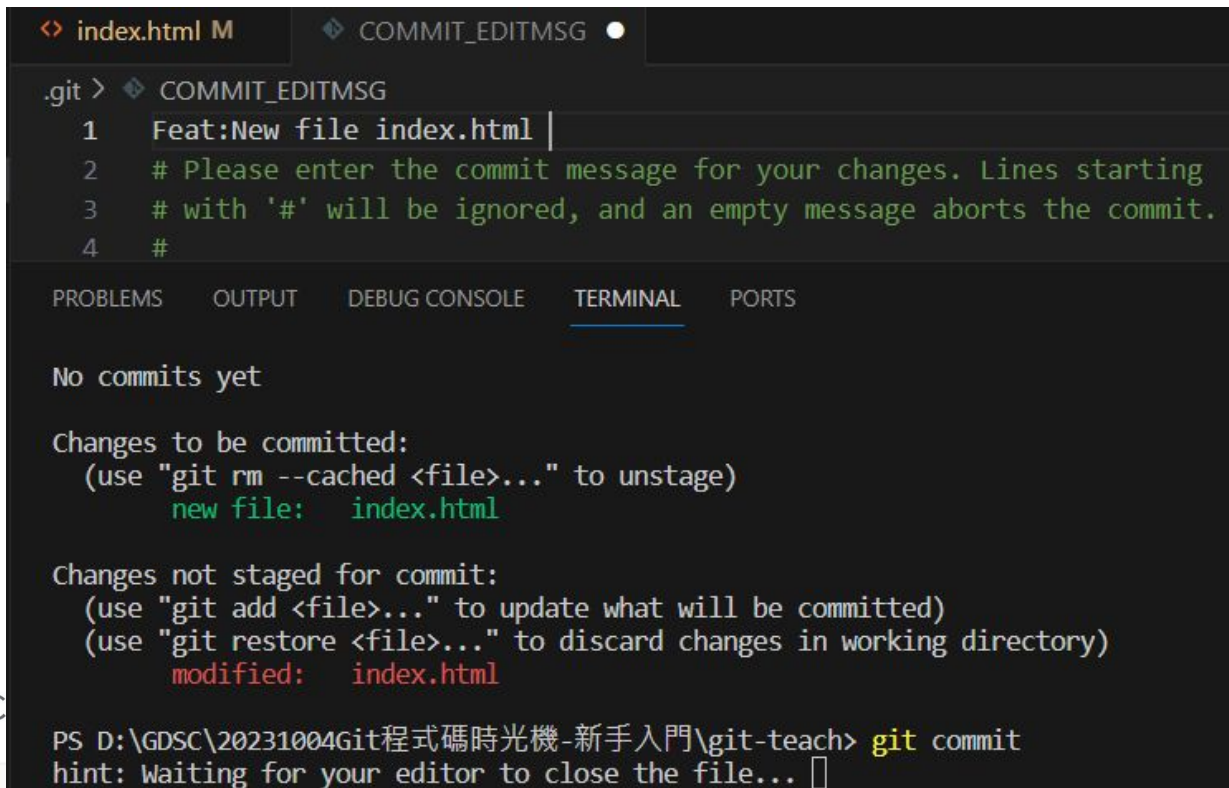


加入目錄後
將會是已修改狀態

Git 本地端操作(#8)- 版本提交

```
$ git add .
```

```
$ git commit
```



```
<> index.html M  COMMIT_EDITMSG  ●

.git >  COMMIT_EDITMSG
1  Feat:New file index.html |
2  # Please enter the commit message for your changes. Lines starting
3  # with '#' will be ignored, and an empty message aborts the commit.
4  #

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git commit
hint: Waiting for your editor to close the file...  □
```


如果不幸進入Vim

```
$ git config --global core.editor "code --wait"
```

修改預設編輯器為VS Code

如何退出Vim by ChatGPT

退出 Vim 取決於您當前的模式和正在執行的操作。以下是一些退出 Vim 的常見方法：

普通模式(按下 `esc` 鍵)下退出：

- 輸入 `:q` 然後按下回車鍵，以退出 Vim。如果您已經對文件進行了更改，但沒有保存，Vim 會提示您保存或放棄更改。
- 如果您強制退出而不保存更改，可以使用 `:q!`。

保存並退出：

- 如果您希望在退出 Vim 之前保存當前文件，可以使用 `:wq`。

退出而不保存更改：

- 如果您不想保存更改並退出 Vim，可以使用 `:x` 或 `ZZ`。

強制退出：

- 如果您想強制退出 Vim，不論是否有未保存的更改，可以使用 `:q!` 或在普通模式下輸入 `:qa!` 來關閉所有打開的文件。

使用快捷鍵退出：

- 在普通模式下，按下 Shift 鍵並同時按下 Z 兩次，即 `Shift + Z + Z`，可以保存並退出當前文件。

如何寫Commit

Feat:New file index.html

<> index.html M

COMMIT_EDITMSG X

.git > COMMIT_EDITMSG

1 Feat:New file index.html |

2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.

Commit 基本規範

它最主要的目的就是告訴你自己以及其它人
「這次的修改做了什麼」。

1. 將訊息用空白斷行 區分標題與內容
2. 標題限制在 **50** 字元以內
3. 標題開頭使用**大寫**
4. 標題不以句點結尾
5. 使用**祈使句**設計標題 (#第一個單字是動詞--例:修正密碼明碼問題)
6. 內容每行最多 72 字, 過多文字則需要斷行

```
Fix: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
#  
# On branch main  
# Changes to be committed:  
#   modified:   index.html  
#
```

Commit 基本規範

Header: `<type>(<scope>): <subject>`

- type: 代表 commit 的類別: feat, fix, docs, style, refactor, test, chore, 必要欄位。
- scope 代表 commit 影響的範圍, 例如資料庫、控制層、模板層等等, 視專案不同而不同, 為可選欄位。
- subject 代表此 commit 的簡短描述, 不要超過 50 個字元, 結尾不要加句號, 為必要欄位。

Commit 基本規範

Body: 72-character wrapped. This should answer:

- * Body 部份是對本次 Commit 的詳細描述，可以分成多行，每一行不要超過 72 個字元。
- * 說明程式碼變動的項目與原因，還有與先前行為的對比。

Footer:

- 填寫任務編號（如果有的話）。

Commit 範例

224

Header

wade · 下午6:19 · 6 weeks ago

type

subject

fix: 修正捐款單資料管理>匯入資料時間格式錯誤問題

Body

PHPExcel_Shared_Date::ExcelToPHP(\$field_value) 轉譯出來的 timestamp 還需要扣除 timezone 的時區差異，才是正確的 timestamp

issue 11161

Footer

Commit 範例2

Commits on Jun 12, 2019

Fix #17



denny0223 committed on 12 Jun ●

Force #rooms width to match the content width for background



othree committed on 12 Jun ✓

Adjust arrangement of sponsors



bingluen committed on 12 Jun ✓

Merge pull request #15 from toomore/individual_sponsor ...



denny0223 committed on 12 Jun ✓

Add individual_sponsor btn



toomore committed on 12 Jun ✓

Update version of node on building



bingluen committed on 12 Jun ✓

常見Type規範

- feat : 新增/修改功能 (feature)。
- fix : 修補 bug (bug fix)。
- docs : 文件 (documentation)。
- style : 格式 (不影響程式碼運行的變動 white-space, formatting, missing semi colons, etc)。
- refactor: 重構 (既不是新增功能, 也不是修補 bug 的程式碼變動)。
- perf : 改善效能 (A code change that improves performance)。
- test : 增加測試 (when adding missing tests)。
- chore : 建構程序或輔助工具的變動 (maintain)。
- revert: 撤銷回覆先前的 commit 例如 : revert: type(scope): subject (回覆版本 : xxxx)。

什麼時候Commit

這個問題沒有標準答案，你可以很多檔案修改好再一口氣全部一起Commit，也可只改一個字就Commit。

常見的Commit的時間點有：

1. 完成一個「**任務**」的時候：不管是大到完成一整個電子商務的金流系統，還是小至只加了一個頁面甚至只是改幾個字，都算是「任務」。
2. **下班**的時候：雖然可能還沒完全搞定任務，但至少先Commit今天的進度，除了備份之外，也讓公司知道你今天有在努力工作。（然後帶回家繼續苦命的做？）
3. 你**想要**Commit的時候就可以Commit。

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>我的第一個網頁</title>
</head>
<body>
  <h1>Hi</h1>
  <h2>I'm xiaosong</h2>
</body>
</html>
```

實作時間

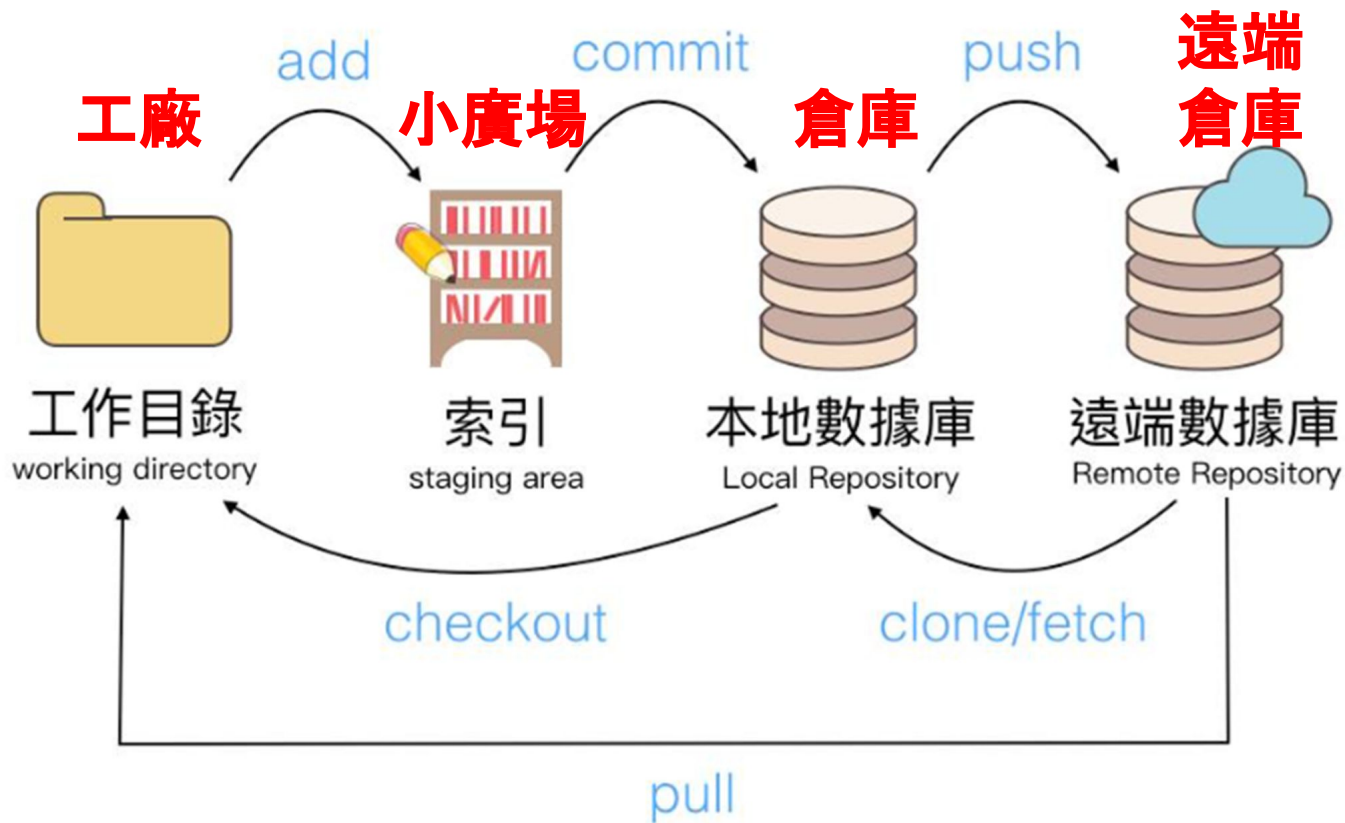
10:00分鐘

! + Tab (建立HTML)

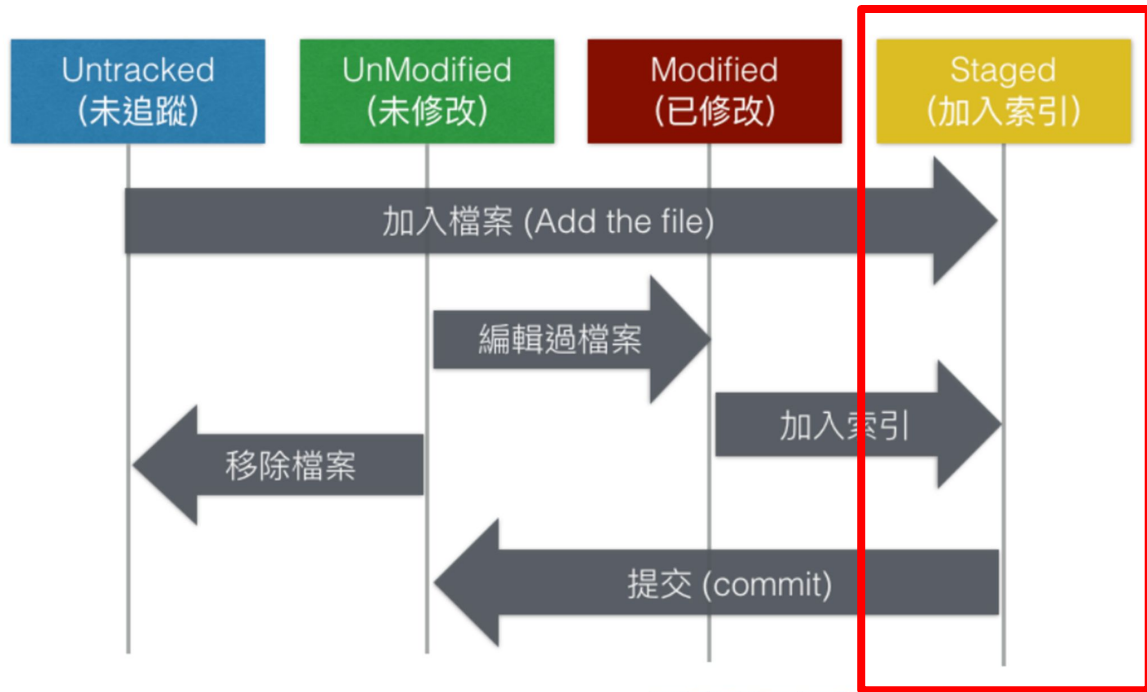
7. `git add .`

8. `git commit` (可以多下幾個試試看)

我們剛才做了什麼



Git 檔案追蹤狀態




提交commit後
將會是working tree
clean狀態

Git 本地端操作(#9)- 查看commit歷史

```
$ git log --oneline
```

查詢commit歷史 --oneline表示每個commit以一行文字呈現

新commit



```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git log --oneline
eaee9d9 (HEAD -> main) Feat:New h2 title
b64ea9d Feat: New h1 title
5c33404 Feat:New file index.html
```

舊commit

建立分支

Branch



Git 本地端操作(#10)- 查看目前分支

```
$ git branch
```

目前處於main分支

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git branch
* main
```

```
PS D:\git-teach> git branch
```

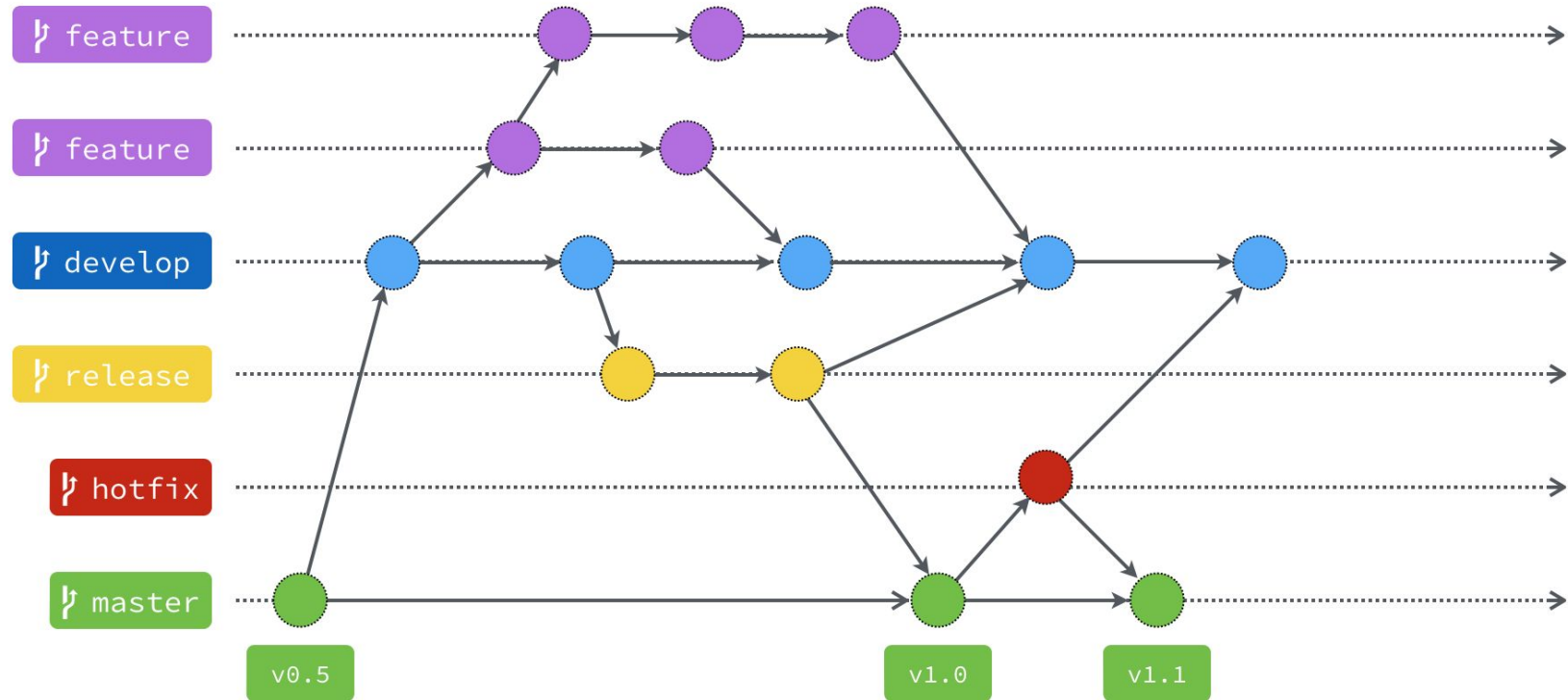
```
* main
```

為何使用分支

分支的概念就有點像「影分身術」，當你做出一隻新的分身（分支），這個分身會去執行任務或是打倒敵人，如果執行失敗了，最多就是那個分身消失，就再做一隻新的分身就行了，本體不會受到影響。

例如想要增加新功能，或是修正 Bug，或是想實驗看看某些新的做法，都可以另外做一個分支來進行，待做完確認沒問題之後再合併回來，不會影響正在運行的產品線。

Git flow



Git 本地端操作(#10-1)- 建立其他分支

```
$ git branch [branchName]
```

Ex : git branch dev

#建立一個名為dev的分支

Git 本地端操作(#10)- 再次查看目前分支

```
$ git branch
```

目前處於main分支(看*的位置判斷)

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git branch dev
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git branch
dev
* main
```

```
PS D:\git-teach> git branch
```

```
dev
```

```
* main
```

Git 本地端操作(#10-2)- 修改分支名字

```
$ git branch -m dev gh-pages
```

原本的名字 修改後的名字

原本dev分支名字修改成gh-pages分支

```
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git branch -m dev gh-pages
PS D:\GDSC\20231004Git程式碼時光機-新手入門\git-teach> git branch
  gh-pages
* main
```



```
PS D:\git-teach> git branch -m dev
```

```
gh-pages
```

```
PS D:\git-teach> git branch
```

```
gh-pages
```

```
* main
```

實作時間

15:00分鐘

請嘗試建立一個分支
分支名字: gh-pages

10. `git branch gh-pages`

```
PS D:\git-teach> git branch
```

```
gh-pages
```

```
* main
```



Git 本地端操作(#11)- 切換分支

```
$ git checkout [branchName]
```

EX: git checkout gh-pages

切換至gh-pages這個分支

```
PS C:\Users\user\Downloads\Git_demo> git checkout gh-pages  
Switched to branch 'gh-pages'
```

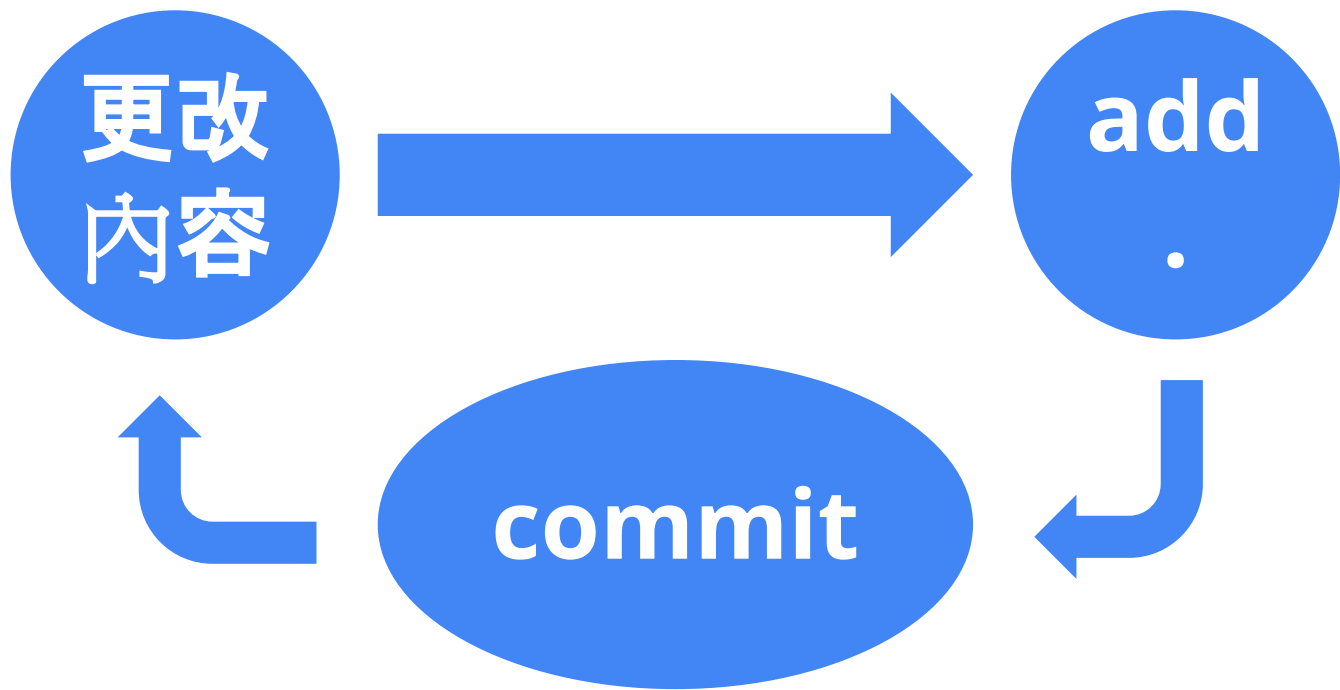
```
$ git checkout gh-pages
```

```
D:\git-teach> git branch
```

```
* gh-pages
```

```
main
```





Branch 的新功能完成了

然後呢？



Git 本地端操作(#12)- 合併分支

\$ `git checkout main` // 切換至main分支

\$ `git merge gh-pages` // 將gh-pages合併至main

```
PS C:\Users\user\Desktop\git-teach> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\user\Desktop\git-teach> git merge dev
Updating b8eeb0f..865dfe4
Fast-forward
 index.html | 1 +
 1 file changed, 1 insertion(+)
```

Fast-forward是什麼

實作時間

15:00分鐘

切換到gh-pages分支
修改內容進行commit
將gh-pages合併至main

解答

11. git checkout gh-page

7. git add .

8.git commit

11. git checkout main

12.git merge gh-page

Fast-forwarding 快轉 (#12-1)

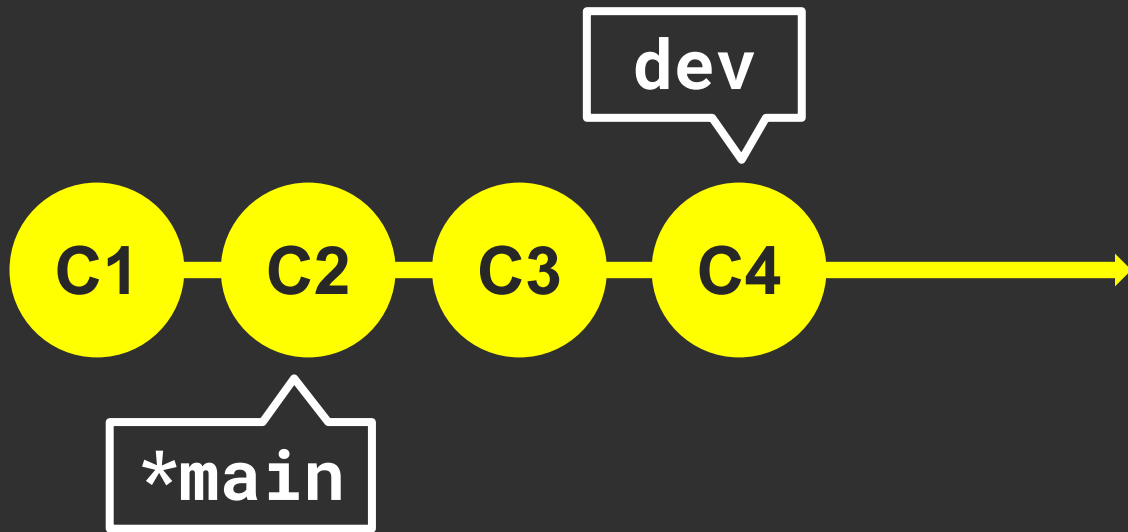
為了簡化commit graph

因此有了快轉機製的

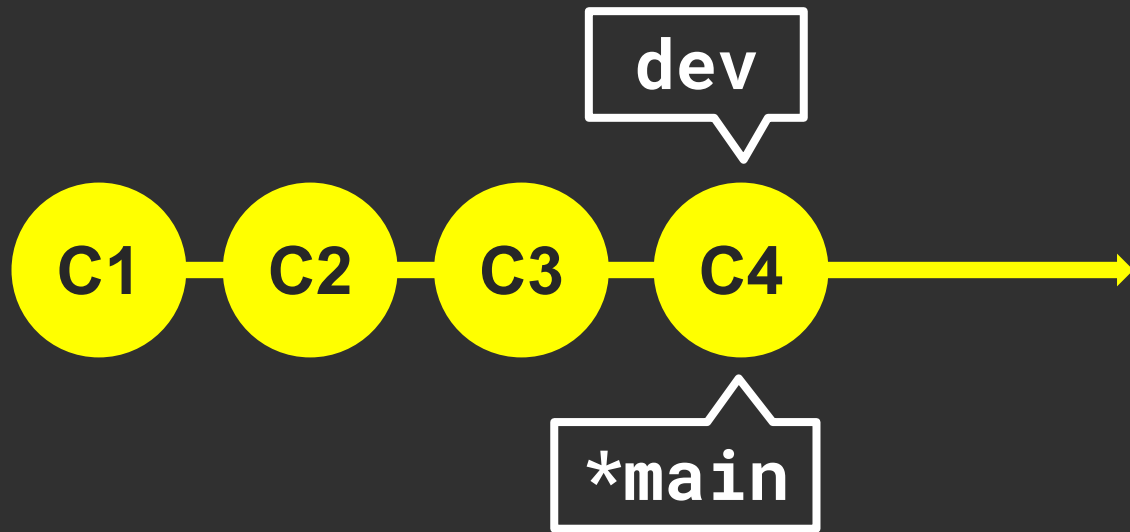
但不是所有時候我們都需要快轉

例如需要告知後續維護人員此時已經合併分支時

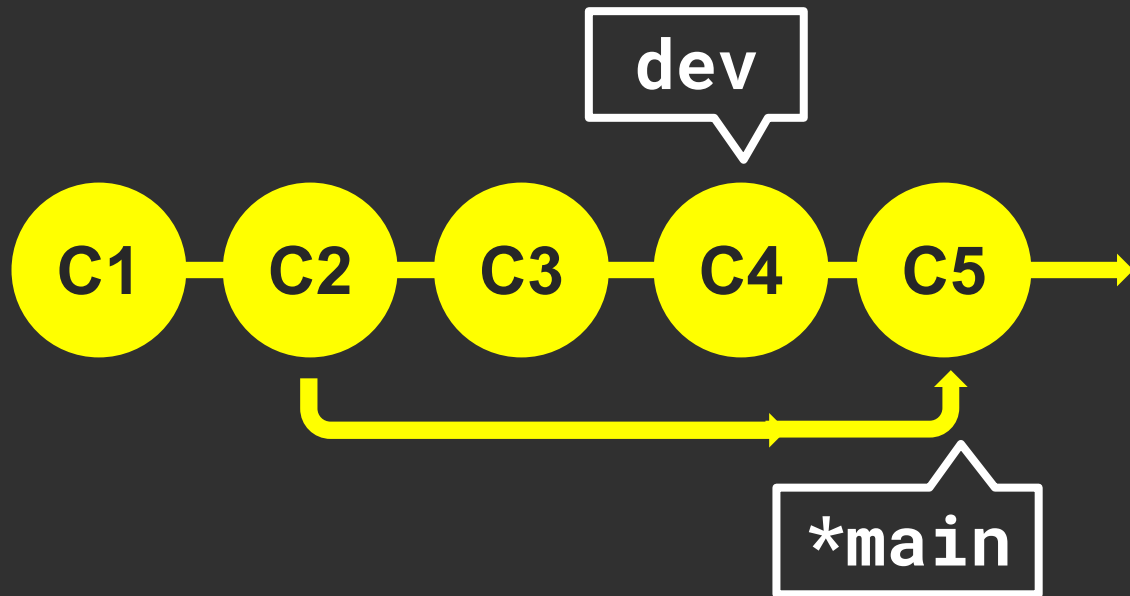
初始狀態



有快轉 `git merge dev`

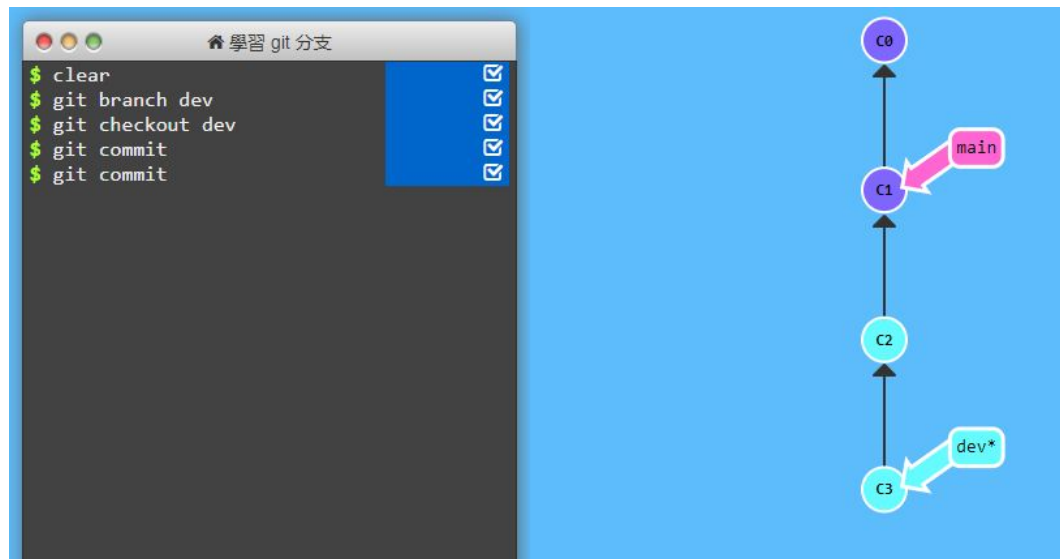


無快轉 `git merge dev --no-ff`



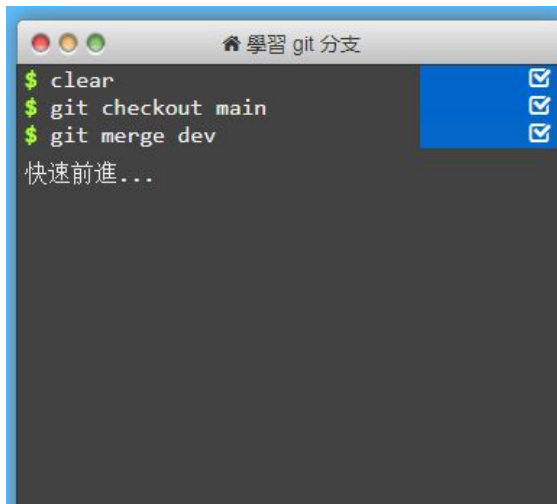
Fast-forwarding 快轉 -情境解釋-step1

dev分支領先main分支
當專案主管認為dev分支
可以合併時
會**切換回main分支**
並執行 **git merge dev**

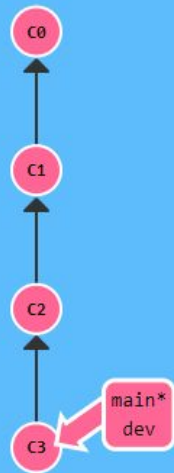


Fast-forwarding 快轉 -情境解釋-step2

如右圖可見
main分支前進至dev分支
但如果在他人看來
並**無法察覺**進行了合併



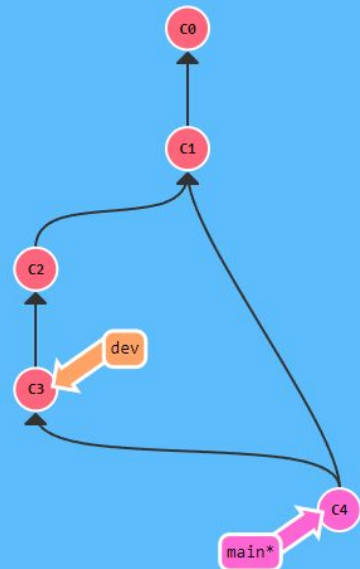
```
$ clear
$ git checkout main
$ git merge dev
快速前進...
```



Fast-forwarding 快轉 -情境解釋-step3

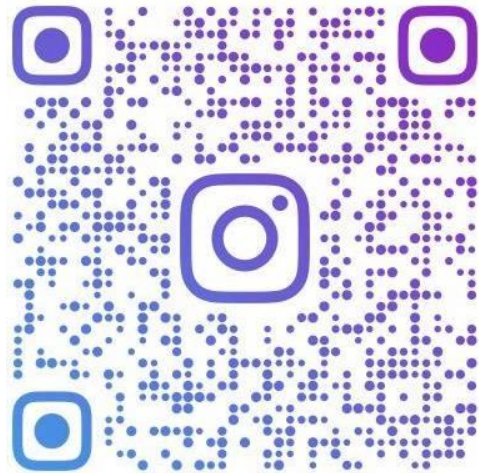
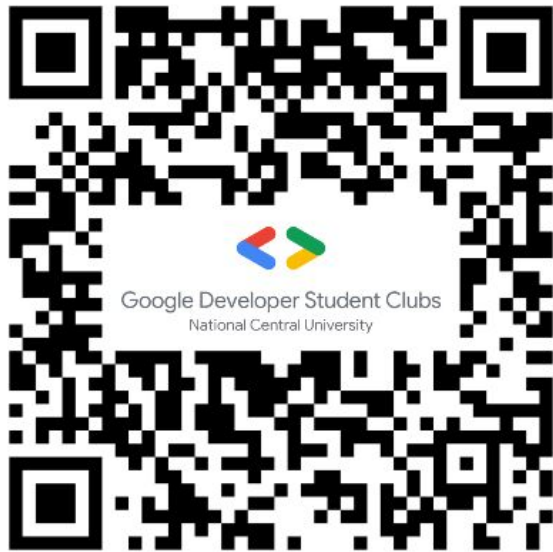
如右圖可見
main分支使用--no-ff參數
前進至dev分支
但是可以**清晰看出**
此時進行了合併

```
學習 git 分支
$ clear
$ git checkout main
$ git merge dev
快速前進...
$ undo
$ git merge dev --no-ff
```



Contact Us

按讚、訂閱、開啟小鈴鐺



DSCNCU

參與證明領取

<https://reurl.cc/WvGo37>



Git 指令查詢表

<https://code.yidas.com/git-commands/>

下回預告

- 時間倒轉之術 - checkout
- 合併衝突 - merge conflict
- 遠端協作 - github相關指令與應用
- github student package



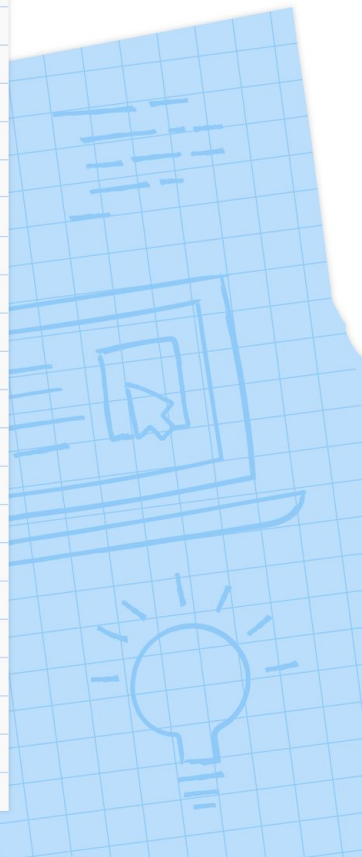
Google Developer Student Clubs

National Central University



時間倒轉之術

Checkout



合併衝突

Merge Conflict



