Unleash Innovation

以 GitHub 來學習團隊的分工機制

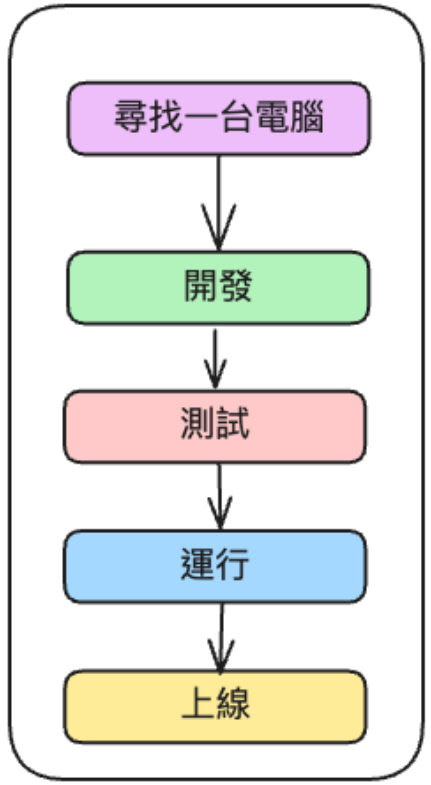
PLED/CID-1 邱宏瑋

課程目標對齊

- 你會學到
  - 多人團隊協同開發的流程
  - 以 GitHub 為範例的合作流程
  - 以 GitHub Action 為基礎的合作流程
- 你不會學到
  - 各式各樣的 Git 指令
  - 進階的各種指令操作與語法介紹
  - 各種雲端環境整合 (GCP, AWS, Azure)

開發困境

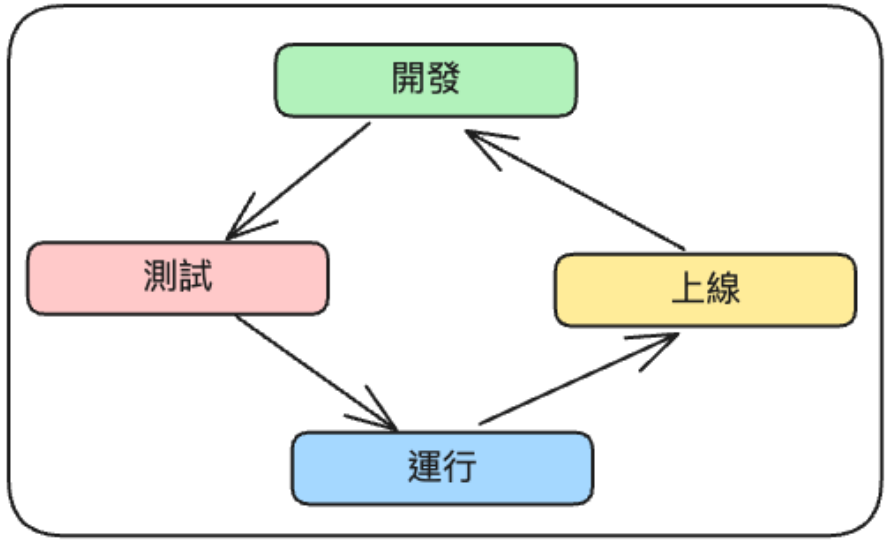
- **以學校課程為範例**
  - 參與人數: 1 人
  - 目標: 滿足課程作業的需求
  - 驗證方式: 現場 Demo
- 開發流程
  - 準備一台機器
  - 準備開發環境
  - 根據作業需求撰寫程式
  - 根據作業需求測試並且構思各種可能的 Input
  - 運作並且 Demo 給助教看



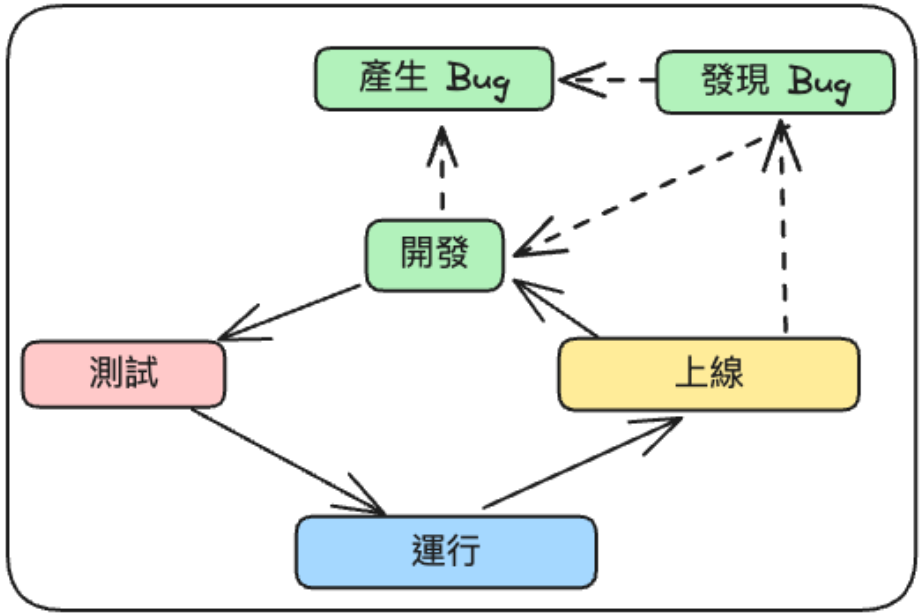
開發困境

- **實務上的開發工作會更加複雜**
  - 參與人數: 多人
  - 目標: 滿足商業需求
    - 營收提高
    - 提高使用者滿意度
    - …等
  - 驗證方式: 數字反映/使用者回饋
- **常見挑戰**
  - 目標需求不固定，根據結果來調整需求
  - 加了新功能，壞了舊功能
    - 隨者程式碼愈來愈複雜，這件事情可能會消耗愈多時間去處理

理想流程



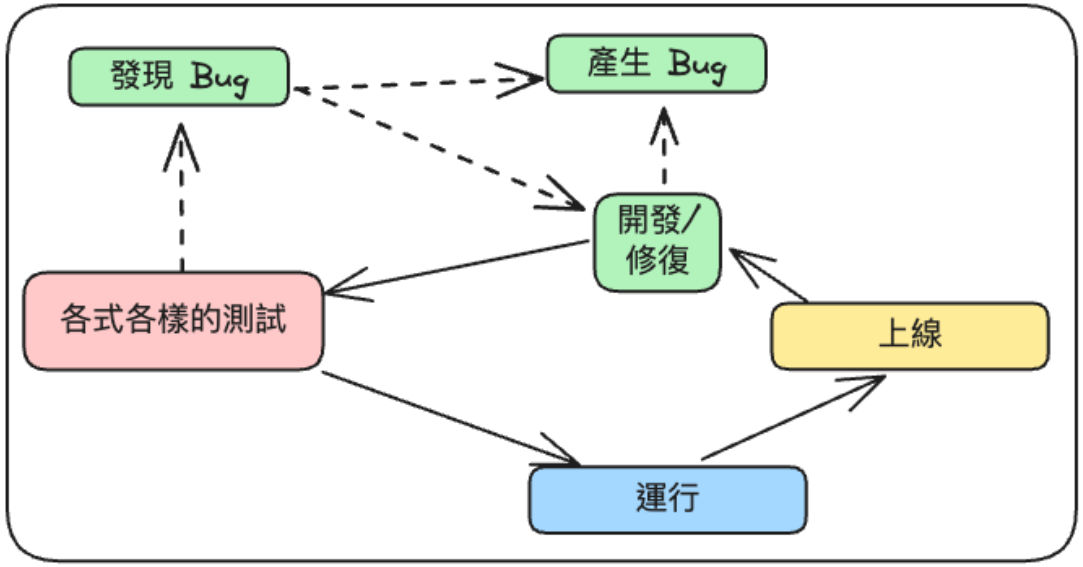
實務流程



開發困境

- 為了避免夜長夢多，改一個壞一個的情況日加嚴重
  - 軟體開發普遍都要求撰寫測試程式，及早發現問題
- 有很多相關測試流程
  - Unit Test
  - Integration Test
  - E2E Test
  - Smoke Test
- 這些測試的目的都是希望能夠於程式上線前找出問題，提早發現
  - 內部找出問題總比被使用者找出問題好

理想流程



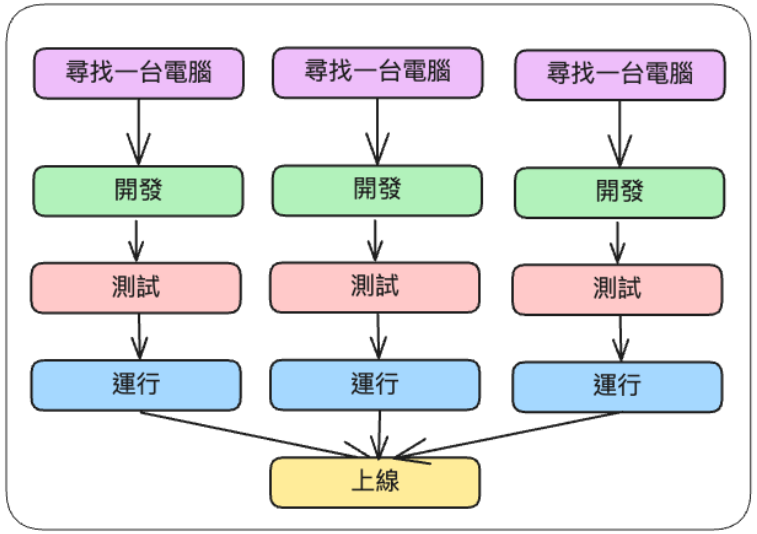
開發困境 - Demo

- 一個以 Python 撰寫的應用程式
- 實作基本非常簡單的功能
  - 加法
  - 階層
  - 奇偶數判斷
- 可以撰寫一些測試程式，反覆運行確認你每次修改都沒有改壞既有功能

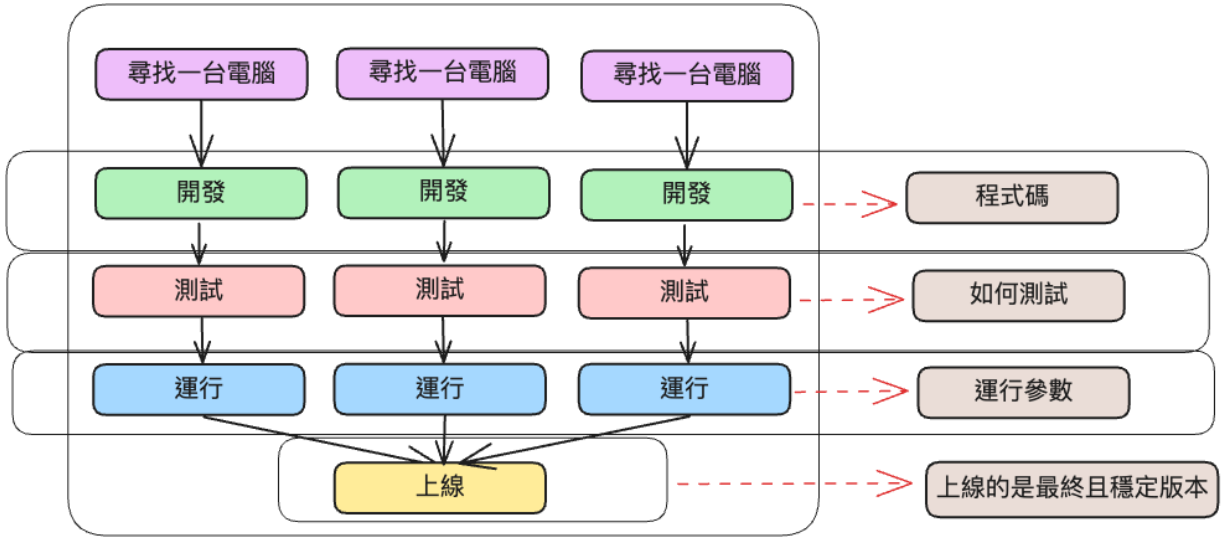
開發困境

- 測試工具與範疇百百種，從程式語言本身的語法撰寫到功能的驗證都是一種測試
- 譬如你撰寫一個 大數加減法的作業，可能的測試範圍如下
  - 程式碼語法一致性 (Coding Style)
  - 程式碼寫法調整 (找出不好的寫法，譬如 if (A=3))
  - 功能測試
    - 簡單的 +,-,*,/ 結果是否符合預期
    - 各種奇怪的輸入 (非數字，全形數字 隱藏字元)
    - 不支援的運算符號(%^&@*)

理想流程



實務困境



開發困境

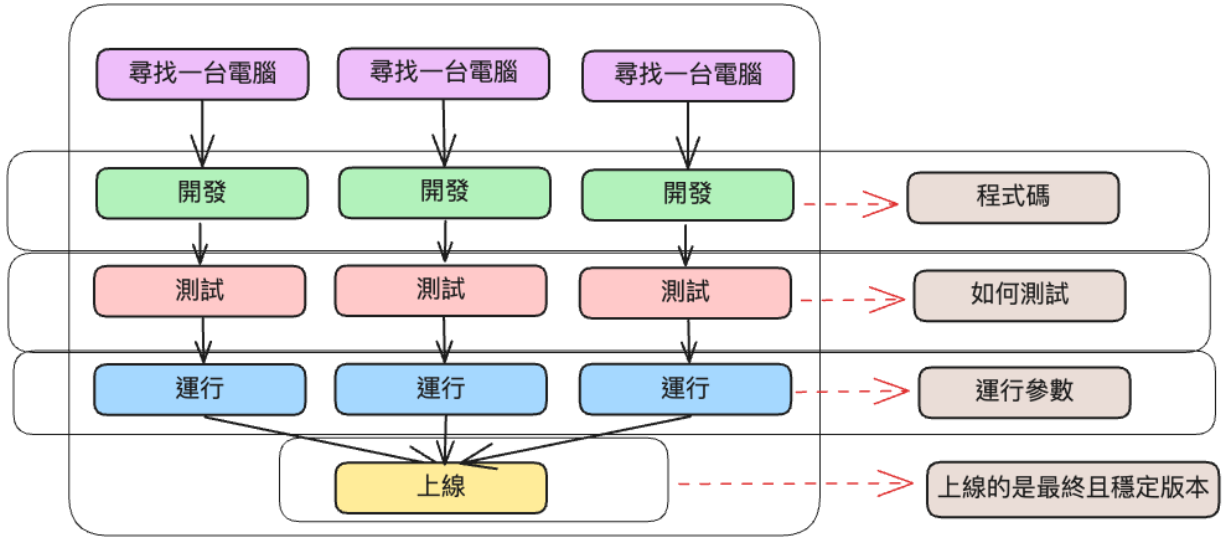
- 為了簡化這些測試的操作，這類型的測試都希望可以透過程式去執行，避免每次都要人為重複輸入與操作
  - 撰寫程式去執行測試，來測試你的應用程式
- 流程比較:
  - 修改完成，手動執行測試
    - 初期容易，隨者程式碼愈來愈多，手動測試要跑的指令愈多
    - 可能需要的測試資料或是相關套件也愈來複雜
  - 修改完成，自動執行測試
    - 先期需要投資去開發與建立
- 哪個流程更適合多人開發?

開發困境

- 上述的問題不單是同一個組織/公司內的團隊會有，實務上現在很多開源軟體都是來自全球的開源貢獻者一起協同合作
- 特點
  - 跨時區
  - 跨組織
- 非同步的合作方式變得非常重要，不可能時時刻刻都可以有口頭討論的機會
  - 沒有一個標準的流程與工具可以套用到全世界的開發開發者
  - 但是概念與精神都是雷同的，只是工具與流程會因應軟體特性而有所微調

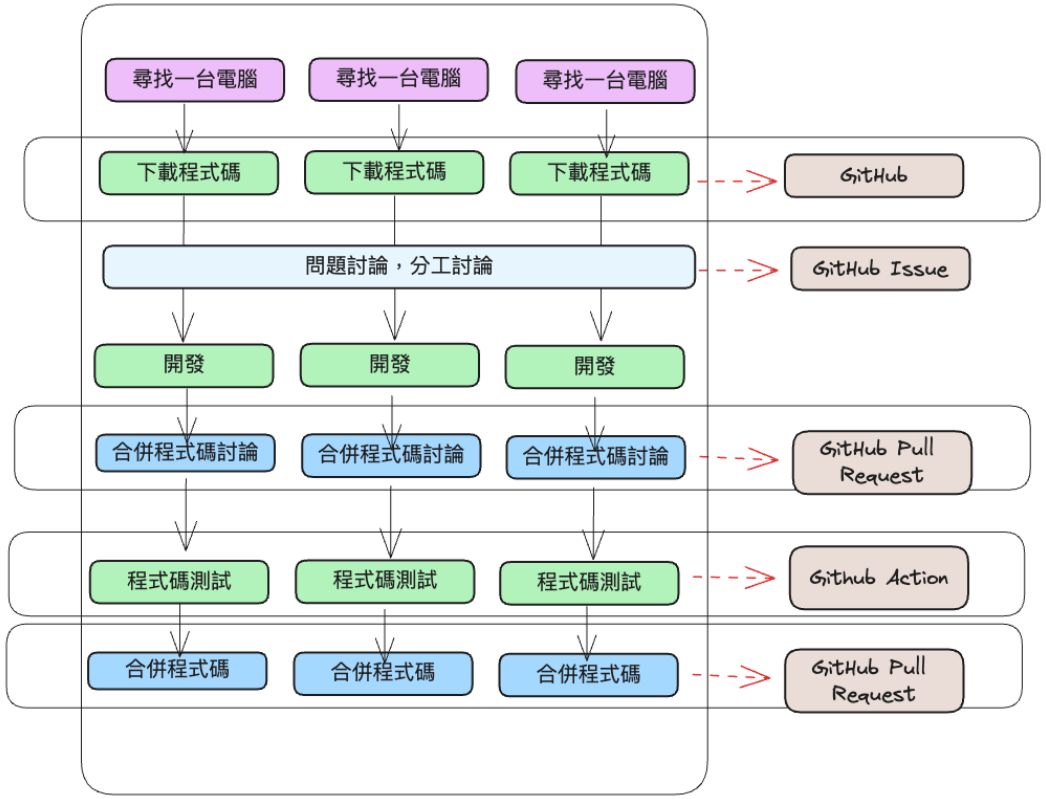
開發困境 – 示範解法

- 以下是一種解決方式，但不是 唯一且標準，**不過概念大部分都雷同**
- 常見問題
  - 程式碼要如何共享與管理 → 程式碼版本管理系統
    - **Git**, SVN
  - 如何本地運行與測試 → 撰寫相關文件/套件管理工具/Container
    - **Markdown** 語法撰寫文件
    - 套件管理工具 (**apk, pip, npm, gomod, pkg-config**)
    - **Docker Container**
  - 分工複雜 → 有系統的分工方式，透過流程與系統來加強團隊內的合作
    - 精神: Kanban, Scrum
    - 工具: Jira, **GitHub**, Azure DevOps, GitLab …等
  - 程式品質 → Code Review / 自動化框架去測試程式
    - 框架: **GitHub Action**, Gitlab pipeline, Jenkins …等
    - Code Review: **GitHub Pull Request**, Gitlab Merge Request



作業目標

- 能夠使用 GitHub 來存放你的程式碼
  - 能夠有多版本的管理與開發
  - 能夠有系統的知道程式碼的修改紀錄與細節
- 能夠使用 GitHub Issue 來發文討論問題
  - 能夠開創新的 Issue 提出疑問
- 能夠使用 GitHub Pull Request
  - 透過網頁的方式去檢視每次的改動，並且討論程式碼的問題
  - 將該 Pull Request 取 Issue 連動，更清楚的知道每次程式碼的修正是為了什麼問題而修正
- 能夠使用 GitHub Action 來設計程式的自動化流程
  - 程式碼有更動的時候可以自動執行測試流程，包含
    - 準備一個環境
    - 安裝需要的相關套件
    - 執行測試結果
    - 將結果回饋到網頁上，即時知道這次的改動有沒有弄壞東西



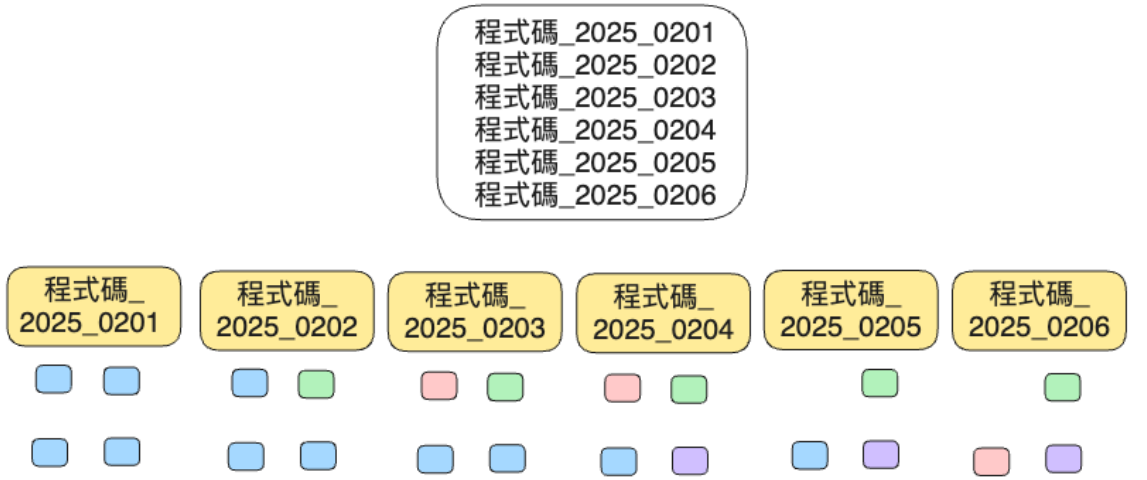
Git

- Git 是一個免費且開源的分散式版本系統
  - 追蹤程式碼的變更
  - 適合多人協作
  - 目前作為主流的程式碼管理工具
- 使用 Git 的好處
  - 版本控制: 保留每次變更的紀錄 (但是還是要仰賴開發者的使用習慣)
    - 每天撰寫一次更新紀錄
    - 每個禮拜撰寫一次更新紀錄
  - 分支管理: 同一套程式碼可以因應開發需求同時維護多個版本，又不會互相覆蓋
  - 回朔管理: 有問題可以快速回復到任何時間點的版本

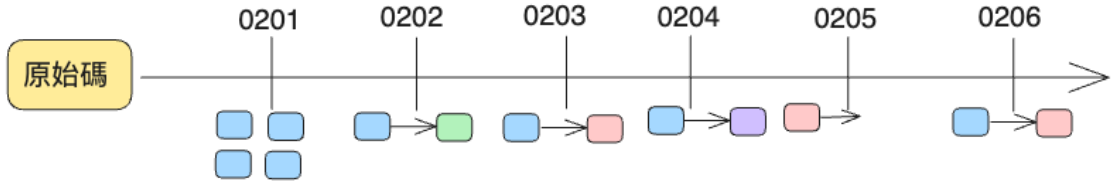
https://git-scm.com/

Git

- 當程式碼有很多的迭代與修正，很久以前會有開發者透過資料夾的方式去紀錄每個時間點的版本
  - 缺乏變動紀錄，每份副本都是完整內容
  - 不容易知道每個版本的差異，還常常需要寫份文件去記錄差異

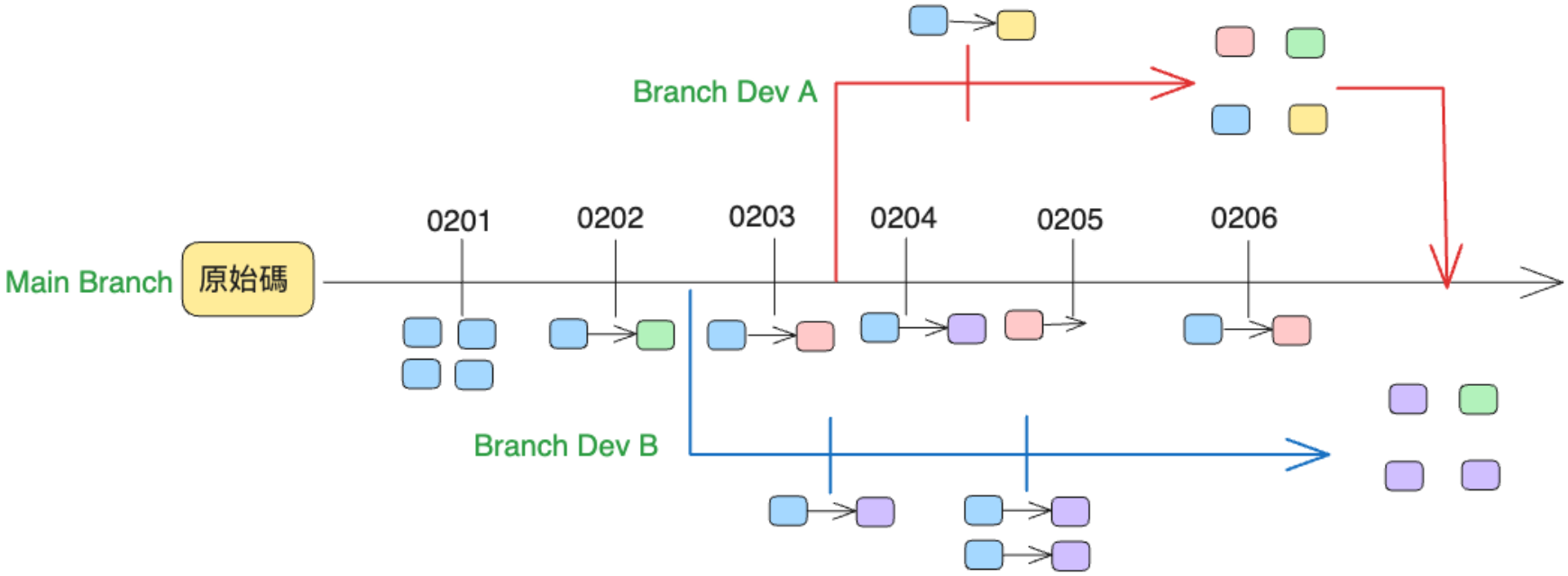


- 版本管理系統會被透過背後的演算法去紀錄更動的差異，可以更快且更清楚的知道不同版本之間的變動
  - 有需求也可以回復或是取消任何一個時間點的更動



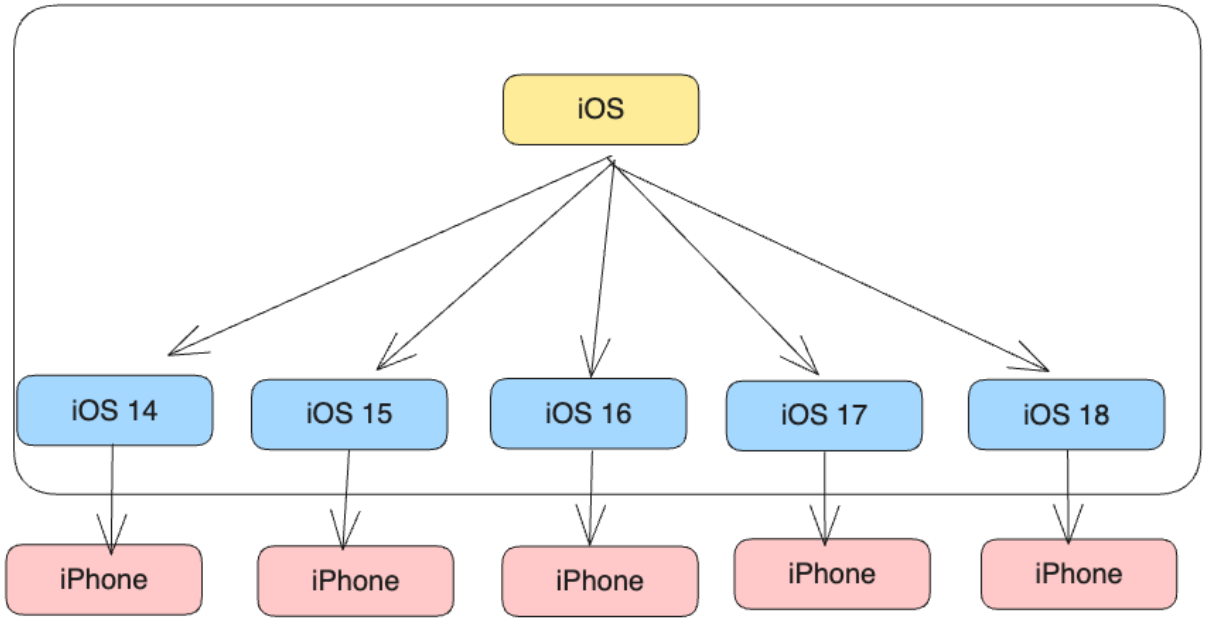
Git

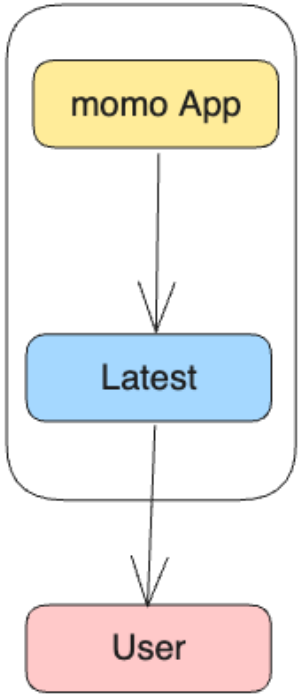
- 多人合作時很常透過分支(Branch)來互相合作
- 從程式碼的一個時間點開出很多的副本
  - 大家獨立開發自己的部分
  - 最後再將程式碼合併回去
- 通常會稱預設的為 Main (以前叫做 Master)



Git

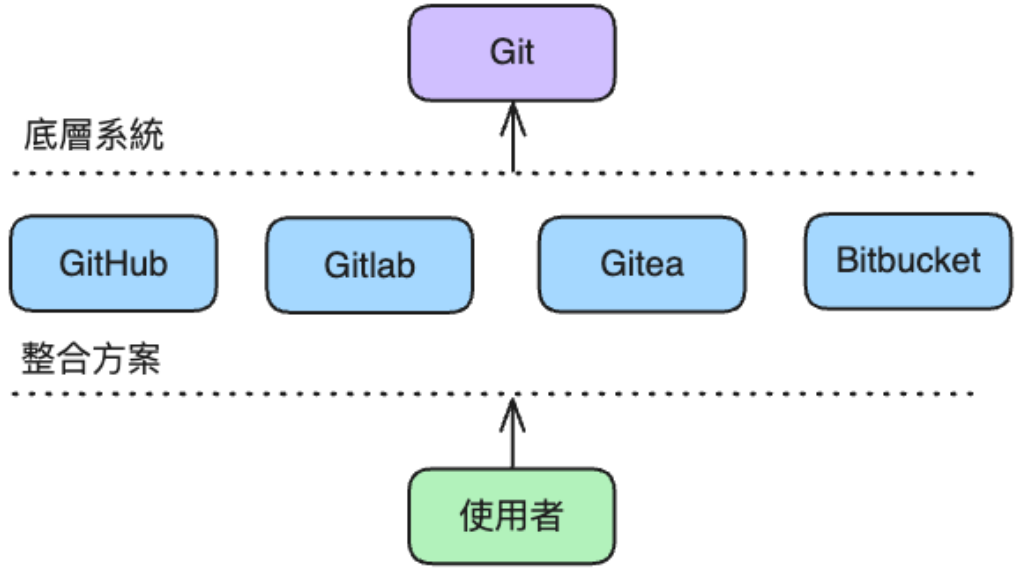
- 不同的軟體特性，會有不同的分支管理策略
  - 軟體要不要支援眾多舊版本?
    - iOS, Linux
  - 軟體對使用者來說是不是只有最新版本?
    - 許多網頁服務
- 分支的管理沒有絕對標準
  都需要根據團隊的特性
  去修正
- 有很多知名的流程範例
  - GitHub Flow
  - Gitlab Flow
  - Git Flow





GitHub

- 實務上大部分人不會直接使用 **純 Git** 服務
  - 沒有 GUI，只有 CLI 可以使用
  - 需要自己下載與安裝，並且找一台機器運行
    - 需要準備儲存空間，需要處理加密網路連線
- 因此現在有很多整合服務，讓大家可以更輕鬆的去使用 Git 服務
  - 有簡潔且明確的 GUI 操作
    - 兼容 Git 的操作，學習一套 Git 的操作到處都可以用
  - 更多服務整合
    - 帳號管理，權限管理
  - 不需要自己維護與安裝，直接使用雲端服務



GitHub

- GitHub 使用流程
  - GitHub 網頁操作
    - 創建帳號
    - 創建 Repo
      - 我們都將放程式碼的地方稱為 Repository (Repo)
  - 自己電腦操作
    - 將 Repo 的內容下載
    - 程式碼修改與開發
    - 透過 Git 紀錄修改內容
    - 將內容推回到 GitHub

GitHub - Demo

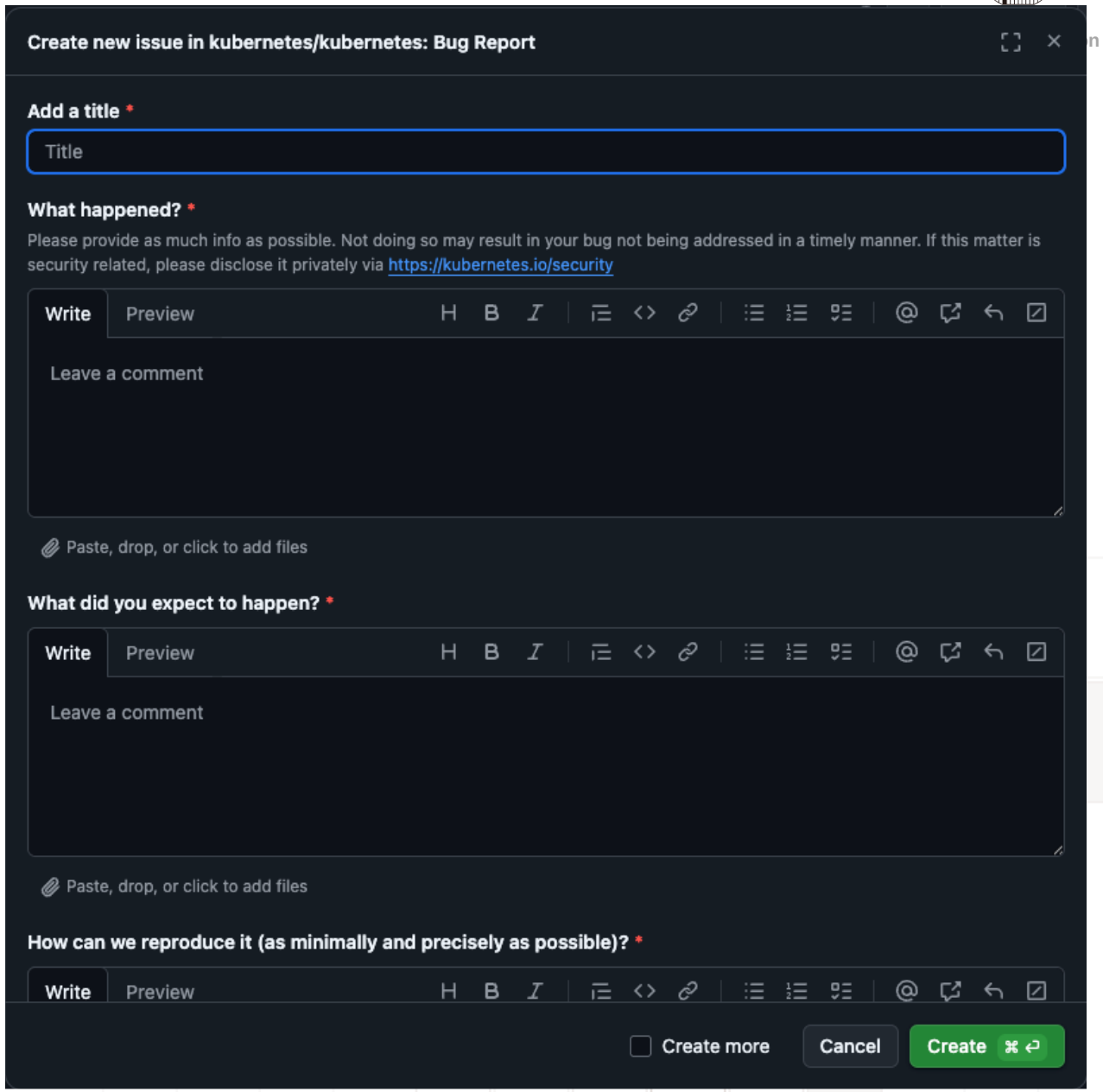
- 操作示範
  - 網頁操作
    - 創建 Repo
    - 瀏覽 程式碼 的變動與修正
  - 環境操作
    - 下載 Repo
    - 修改程式碼
    - 以 Git 紀錄修正內容
    - 推送到 GitHub

GitHub - Demo

- 操作示範
  - Branch 操作與管理
  - 透過網頁檢視 Git 的內容
  - 為 Repo 撰寫一個首頁內容
    - README.md
      - Md => **M**ark**d**own 的縮寫

GitHub Issue

- GitHub 整合的 Issue 系統
- 團隊可以於此內討論各種問題
  - 發現 Bug
  - 功能許願
  - 使用詢問
  - 閒聊
- 格式
  - 預設是一片空白，自由發揮
  - 可以透過 Template 的方式撰寫一個模板，迫使大家只能按照格式去發文與留言

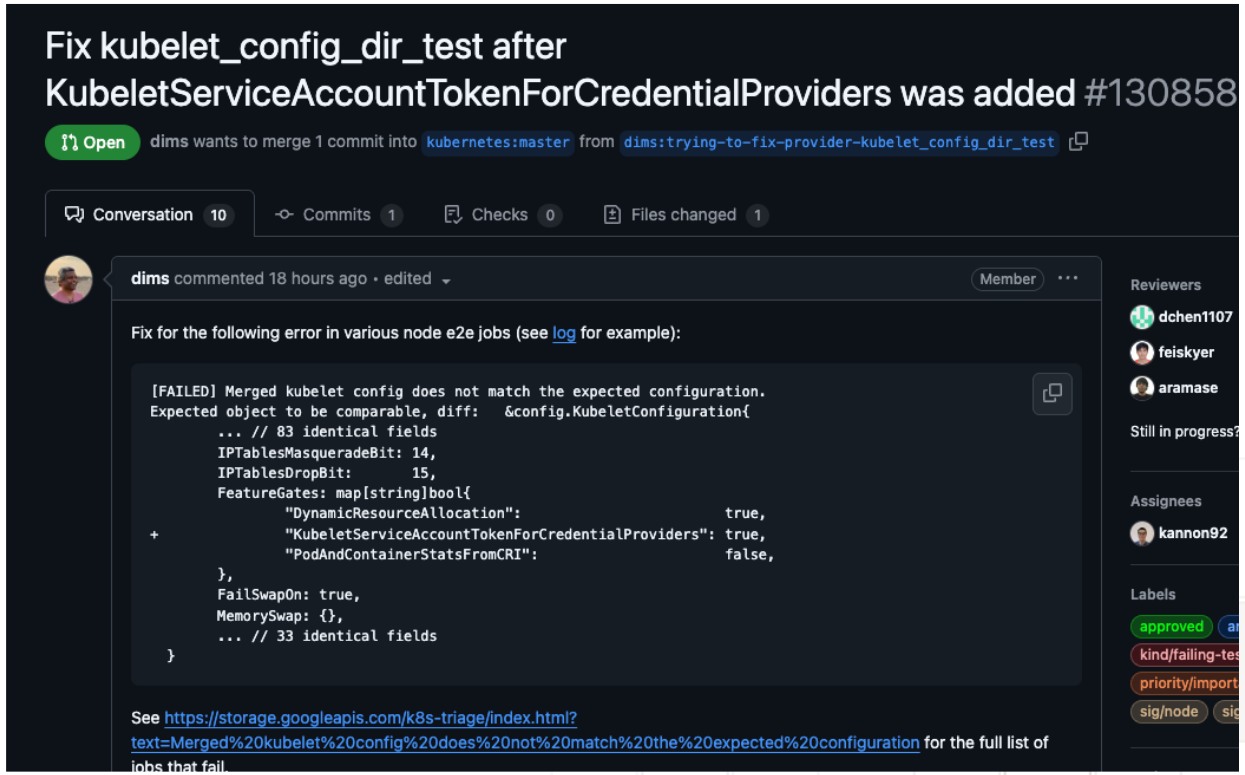


GitHub Issue - Demo

- 操作示範
  - 如何開 Issue
  - 如何撰寫一個樣板去調整發文格式

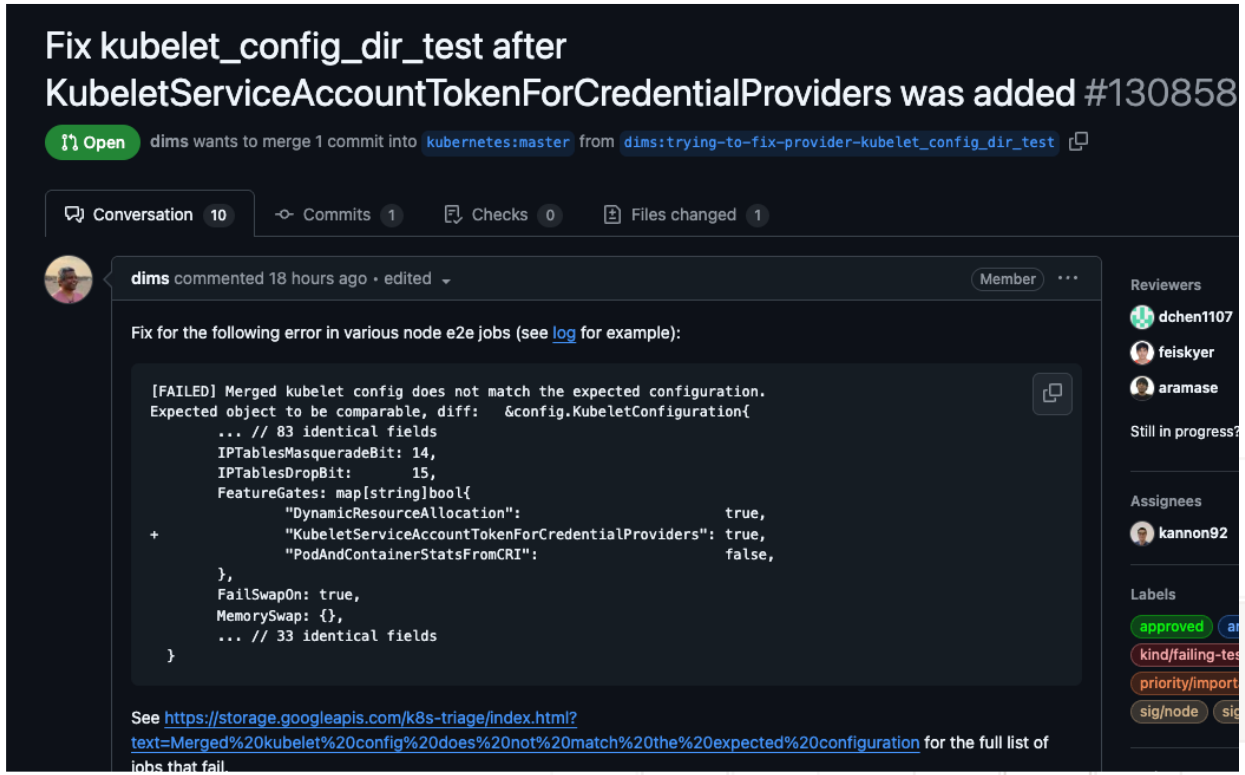
GitHub Pull Request

- GitHub 提供的程式碼合併工具
  - 提供 GUI 幫你快速檢視程式碼的差異
  - 提供 GUI 讓團隊可以互動討論內容
- 通常都會描述
  - 為了什麼目的
  - 怎麼修正
  - 搭配 Git 也可以看到每個小修正的差異



GitHub Pull Request - Demo

- 示範
  - 如何開創一個 Pull Request
    - 習慣簡稱 PR
  - 如何透過 PR 內去留言討論問題
  - 如何與 Issue 關聯



GitHub Action

- 前述的介紹都是基於 GitHub 的使用，主要圍繞於目前 GitHub 上的多人協作流程
  - 跨時區/跨地區 的非同步工作型態
- 除了基本的 Git 操作與互動功能外，還支援 GitHub Action 來達成自動化指令
  - 透過事件觸發 (Event-Driven) 的方式去執行指令
- 事件觸發範例
  - 有人開了一個 Issue
  - 有人開了一個 PR
    - 針對修改的程式碼去進行驗證，確保修改的程式碼沒有搞破東西
  - 有程式碼合併到特定的 Branch
    - 譬如有人合併程式碼到 Main Branch，就自動執行指令產生安裝檔

GitHub Action

- GitHub Action 的概念相對單純，由幾個元件組成
- 觸發條件
  - 譬如當 PR 發生，有程式碼想要合併時
- 執行環境
  - 譬如 Ubuntu 22.04
- 執行步驟
  - 譬如
    - 下載程式碼
    - 安裝需要的套件
    - 嘗試編譯
    - 執行測試

GitHub Action

- GitHub Action 的概念相對單純，由幾個元件組成
- 觸發條件
  - Main branch 有任何程式碼發生變動
- 執行環境
  - Ubuntu 最新版本
- 執行步驟
  - 譬如
    - 下載程式碼
    - 建立環境 (nodejs 18)
    - 安裝需要的套件 (npm install)
    - 執行測試 (npm test)
- 撰寫上述的檔案，就可以達成每次程式碼更動都會自動跑測試

```yaml
name: CI Test

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: 取得程式碼
        uses: actions/checkout@v3

      - name: 設定 Node.js 環境
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: 安裝依賴
        run: npm install

      - name: 執行測試
        run: npm test
```

GitHub Action

- 要完成一個完整的 GitHub Action 來改善前述提到的開發流程
- 我們需要清楚且明確的知道
  - 如何搭建軟體開發環境
    - 安裝那些套件
  - 如何進行測試
    - 程式碼測試
    - 功能性測試
- 上述的流程都要可以用指令或是程式去準備，才有辦法整合到 GitHub Action 去自動操作
  - 減少人為操作

```yaml
name: CI Test

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: 取得程式碼
        uses: actions/checkout@v3

      - name: 設定 Node.js 環境
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: 安裝依賴
        run: npm install

      - name: 執行測試
        run: npm test
```

GitHub Action - Demo

- 描述一個 GitHub Action 的範例檔案
- 將該檔案透過 Git 的方式整合到 Repo 中
- 觸發條件
  - 當 Pull Request 發生的時候就執行
- 執行環境
  - Ubuntu 22.04 的版本
- 執行步驟
  - 下載程式碼
  - 安裝需要的 Python 套件
  - 進行測試
    - 測試失敗要回傳失敗
- Pull Request 內可以看到測試結果

回家作業

Category	Topic	Score	Remark
Repo 操作 (20)	創建一個 GitHub Repo	5	必須要是 Public，助教才可以點選
	有準備 README.md	5	要有修改過，不能用預設的
	從網頁上可以看到除了 Main 外有額外兩個 Branch	10 (各五分)	hw1-p hw1-f
Issue (20)	創建任意 Issue	5	創一個 issue，open 狀態
	有 Issue Template	15	創 issue 時有 template 可以選，內容不拘
Pull Request(20)	以 hw-p 此 branch 對 main 創立 Pull Request	5	需針對任何檔案修改
	以 hw-f 此 branch 對 main 創建 Pull Request	5	需針對任何檔案修改
	上述任何一個 PR 內有留言互動，針對程式碼變動留言	10	針對修改必須要留言
GitHub Action(40)	點選 GitHub Action 內有任何 Action 運行過痕跡	10	成功或失敗的都可以
	Action 內扣除預設，至少要有額外兩個步驟(Steps)	10	Setup, Post 這些不算
	hw-p 的 PR 必須要有 GitHub Action 運行，要成功	10	
	hw-f 的 PR 必須要有 GitHub Action 運行，要失敗	10	hw-f 的程式碼修改必須要讓你 GitHub Action 會跑失敗