**關於Git**

Git 是一種**版本控制系統(**分散式版本控制系統)，主要用途是追蹤檔案隨著時間的變更，每個開發者的電腦上都有完整的版本歷史，而不是只依賴某一台伺服器，讓多人協作開發時可以更有條理地管理專案、還原修改、處理衝突等。

Git[易於學習](https://git-scm.com/doc)，佔用 [空間小，效能卻非常快。它憑藉著](https://git-scm.com/about/small-and-fast)[低成本的本地分支](https://git-scm.com/about/branching-and-merging)、便利的[暫存區域](https://git-scm.com/about/staging-area)和 [多種工作流程](https://git-scm.com/about/distributed)等功能，超越了 Subversion、CVS、Perforce 和 ClearCase 等 SCM 工具。

**為什麼要用 Git？**

* **追蹤所有修改** ：它會記錄你檔案的每一個變動，你可以隨時查看、比較，甚至回到過去任何一個版本。
* **不怕搞砸** ：不小心刪錯或改壞了？沒關係，Git 能讓你輕鬆還原，就像有了「復原」的無限次數按鈕。
* **團隊合作超有效率** ：多個人可以同時獨立工作，互不干擾。完成後，Git 還能幫你把大家的成果完美整合在一起。
* **離線也能工作** ：因為每個人都有完整的專案副本，所以即使沒網路，你也能繼續工作和儲存進度。

**Git 跟 GitHub (或 GitLab、Bitbucket) 差在哪？**

| **Git** | **GitHub / GitLab / Bitbucket** |
| --- | --- |
| 版本控制工具，可在本機管理檔案版本 | 平台服務，用於遠端儲存、協作與分享 Git 專案 |
| 在電腦上使用，不需要網路 | 需要網路連線，可與他人同步協作 |
| 適合個人或本機專案版本追蹤 | 適合多人專案、開源社群或公司內部協作 |

Git 的應用場景

1️ 軟體/網站開發

* 跟蹤程式碼修改
* 團隊多人協作（例如你改 A 模組，我改 B 模組）
* 管理不同版本（v1.0、v1.1、v2.0）
* 開發新功能時開分支，測試沒問題後再合併

2️文件編輯與協作

* 技術文件、學術報告、筆記等
* 記錄每次誰改了哪裡
* 回復到之前版本（防止改壞或誤刪）

3️學術研究

* 記錄實驗程式碼與結果分析腳本的歷史
* 可重現研究成果（版本可控）

4️資料分析與 AI 專案

* 專案中含程式碼、模型訓練紀錄、資料集紀錄
* Git 搭配 DVC 或 Git LFS 可管理大型資料

5️製作部落格或個人網站（如 GitHub Pages）

* 把寫好的 HTML/CSS 放上 GitHub，就能免費架站

6️多人報告/小組專題

* 用 Git 管理報告內容和排版格式，避免檔案版本混亂

**Git 的起源與歷史**

Git 是由 Linux 之父 Linus Torvalds 在 2005 年所開發，目的是用來管理 Linux 核心的程式碼。當時他們捨棄了舊的版本控制系統 BitKeeper，因此 Linus 決定自己開發一套免費、快速且分散式的版本控制工具，這就是今天的 Git。

**Git 與其他版本控制系統的比較**

|  |  |  |
| --- | --- | --- |
| 特性 | Git（分散式） | SVN（集中式） |
| 資料儲存 | 每台電腦都有完整歷史 | 只存在中央伺服器 |
| 離線操作 | 完整功能 | 需連線 |
| 分支管理 | 輕鬆快速 | 分支操作較複雜 |
| 效能 | 高效能（C 語言撰寫） | 中等 |
| 安全性 | 每個版本有雜湊  （SHA-1）驗證 | 較弱 |

**Git 的核心概念與架構**

Repository（倉庫）：用來儲存你的版本控制歷史的資料夾。

Working Directory（工作區）：你正在開發或編輯的實體檔案區域。

Staging Area（暫存區）：暫時儲存修改變更，用來組合一次 commit 的內容。

Commit（提交）：將修改記錄永久寫入版本歷史。

Branch（分支）：平行開發的副本，不會影響主分支。

Merge（合併）：把分支內容整合回其他分支。

Remote（遠端）：如 GitHub 等託管平台的版本倉庫。

|  |  |  |
| --- | --- | --- |
| Git 名詞 | 用途 | 比喻說明 |
| Repository（倉庫） | 儲存專案及歷史紀錄 | 像是整個專案的資料夾歷史庫 |
| Commit（提交） | 記錄每次變更 | 像是幫專案拍張照片 |
| Branch（分支） | 開發不同功能用的分線 | 像是開另一條路做實驗 |
| Merge（合併） | 把分支成果合併到主線 | 像是把兩條路的成果合成一條 |

**Git 常用指令整理**

|  |  |
| --- | --- |
| 指令 | 功能 |
| git init | 初始化 Git 倉庫 |
| git clone <URL> | 複製遠端倉庫 |
| git status | 查看當前狀態 |
| git add <檔名> | 將變更加入暫存區 |
| git commit -m "訊息" | 提交版本 |
| git push | 推送到遠端 |
| git pull | 下載遠端更新 |
| git branch | 查看/建立分支 |
| git merge | 合併分支 |

**GitHub / GitLab 的簡介**

GitHub 是最常見的 Git 倉庫託管平台，提供免費與付費服務。GitLab 是另一個支援私有倉庫與 CI/CD 的平台，常用於企業內部專案。這些平台提供專案協作、Issue 管理、Pull Request、Wiki、部署整合等功能。

**Git 進階概念補充**

1. Git 的三個狀態（States）：

- Modified（已修改）：檔案已變更但尚未加入暫存區

- Staged（已暫存）：變更已加入暫存區

- Committed（已提交）：變更已記錄到版本庫

2. HEAD 的意義：HEAD 指向目前所處分支的最新提交，可以理解為「目前所在的快照狀態」。

3. Rebase vs Merge：

- git merge：保留歷史，產生合併紀錄

- git rebase：線性歷史，適合整理提交紀錄（但要小心用在公開分支）

**Git 工作流程（Workflow）範例**

1. Git Flow（推薦給中大型專案）

- main：穩定、可部署版本

- develop：開發版

- feature/：新功能開發

- release/：發布準備

- hotfix/：緊急修復

2. GitHub Flow（推薦給小團隊、快速迭代）

- 所有新功能或修正以 Pull Request（PR）形式進行

- 僅 main 分支可部署

- 每次合併都需經過 Review

**Git 安全性與版本不可竄改性**

Git 的每次提交都透過 SHA-1 雜湊碼計算生成，包含：

- 前一次提交的雜湊值

- 提交內容

- 提交者資訊與時間

這讓版本歷史具有「鏈結」與「防篡改」特性，有助於審計與追溯。

**Git 延伸工具與整合**

1. Git LFS（Large File Storage）：管理大型二進位檔案（圖片、模型等），減少 Git 資料庫膨脹

2. DVC（Data Version Control）：適用資料科學與 AI 專案，能追蹤資料集與模型訓練過程

3. CI/CD 整合工具：

- GitHub Actions：自動測試、部署

- GitLab CI：流程自訂度高，企業常用

**使用 Git 的好習慣**

1. 每個 commit 都應簡潔且有意義的描述（例如：「fix: 修復登入錯誤」）

2. commit 原則：一次提交只處理一件事

3. 避免將敏感資訊（如密碼、API 金鑰）加入版本庫

4. 定期 pull 避免衝突，並妥善處理 merge 衝突

**Git 常見錯誤與處理方式**

**1. fatal: refusing to merge unrelated histories**

* **錯誤說明**：發生在你的本機 Git 專案與 GitHub 的初始化方式不同步
* **解法**：git pull origin main --allow-unrelated-histories
* **來源紀錄**：【07/22 共享 Flask 專案畫面】

**2. 合併衝突 Merge conflict**

* **情境說明**：你和組員同時修改某檔案相同區段。
* **解決方式**：
  + 手動解衝突（使用 VSCode）
  + 解決後：

git add .

git commit -m "fix conflict"

* **來源紀錄**：【07/22】與【07/17】多人開發同步問題

**3. Flask 搭配 Git/XAMPP 開發路徑問題**

* **情境說明**：你提過 Flask 結合 XAMPP 的目錄結構造成組員環境不一致。
* **解法建議**：
  + 在 app.py 中設定正確路徑：

app = Flask(\_\_name\_\_,

template\_folder="../../frontend/templates",

static\_folder="../../frontend/static")

* + 使用 url\_for() 處理 HTML 靜態資源：

<img src="{{ url\_for('static', filename='img/logo.jpg') }}">

* **來源紀錄**：【07/17】Flask 結構與模板路徑問題
* **補充紀錄**：【07/15】前台後台區分與 Flask 網頁呈現路徑

**參考文獻**

* **fatal: refusing to merge unrelated histories**

- Git 官方說明：<https://reurl.cc/gY2KmQ>

* **Merge Conflict 合併衝突**

- Atlassian Git Tutorials：<https://reurl.cc/z5ZxW6>

* **XAMPP + Flask 路徑衝突**

- Flask 官方文件：<https://reurl.cc/pY1O1e>