Git 版本控制：程式碼的時間機器

給超新手的 5 分鐘快速路徑

目標：讓 Git 幫你「存檔有版本」

Windows (PowerShell)

步驟 1-2：進入專案並初始化

```
# 1) 到你的專案資料夾
Set-Location ~/Desktop/my-project

# 2) 初始化 Git 倉庫
git init
```

Windows (PowerShell)

步驟 3-4：檢查狀態並建立檔案

```
# 3) 檢查狀態
git status

# 4) 建立測試檔案
New-Item readme.txt -Value "hello"
```

Windows (PowerShell)

步驟 5-7：暫存、提交並查看歷史

```
# 5) 加入到暫存區
git add readme.txt

# 6) 建立記錄點
git commit -m "第一次記錄：新增 readme"

# 7) 查看記錄歷史
git log --oneline
```

macOS (zsh)

步驟 1-2：進入專案並初始化

```
# 1) 到你的專案資料夾
cd ~/Desktop/my-project

# 2) 初始化 Git 倉庫
git init

# 3) 檢查狀態
git status

# 4) 建立測試檔案
echo "hello" > readme.txt
```

macOS (zsh)

步驟 5-7：暫存、提交並查看歷史

```
# 5) 加入到暫存區
git add readme.txt

# 6) 建立記錄點
git commit -m "第一次記錄：新增 readme"

# 7) 查看記錄歷史
git log --oneline
```

Git 小字典（晶晶體）

- `add` = 放進購物車（準備要存的東西）
- `commit` = 遊戲記錄點（可以讀檔回來）
- `status/log` = 現在/過去在幹嘛

常見卡關解決

問題：user.name/user.email 未設定

Windows PowerShell 設定

```
git config --global user.name "Your Name"
git config --global user.email "yourname@email.com"
```

macOS zsh 設定

```
git config --global user.name "Your Name"
git config --global user.email "yourname@email.com"
```

什麼是 Git？

Git 是一個超級厲害的「版本控制系統」！

Git 的比喻

想像你在寫小說，每寫一段就想保存一個版本：

- 第1版：寫了開頭
- 第2版：加了主角設定
- 第3版：修改了情節

傳統方式的問題

- 複製一堆資料夾：project_v1, project_v2, project_final...
- 檔案一多就亂了
- 不知道誰改了什麼
- 後悔改錯很難復原

Git 的解決方案

- 📚 **記錄所有變更**：誰什麼時候改了什麼
- ⏰ **時間旅行**：隨時回到過去的版本
- 👥 **團隊合作**：多人同時開發不會衝突
- 🌐 **分享程式碼**：推送到 GitHub 和大家分享

為什麼叫版本控制？

就像遊戲的「記錄點」系統：

- 每到關卡結束就存記錄點
- 打錯可以讀舊記錄點
- 可以比較不同記錄點的差異

Git 就是程式碼的「超級記錄點」！

認識 Git 的三個區域

Git 就像一個三層的倉庫：

三層倉庫結構

```
工作區 (Working Directory)
    ↓ git add
暫存區 (Staging Area)
    ↓ git commit
儲存庫 (Repository)
    ↓ git push
遠端倉庫 (Remote)
```

簡單解釋（像遊戲）

- **工作區**：你現在正在玩的關卡
- **暫存區**：準備要存的進度（像購物車）

區域解釋續

- **儲存庫**：已存好的記錄點
- **遠端倉庫**：雲端存檔（給別人看）

Git 基本指令大集合

建立 Git 倉庫

Windows PowerShell 初始化

```
# 初始化新專案（在專案資料夾裡執行）
git init

# 從別人的專案複製下來
git clone https://github.com/username/repo.git
```

macOS zsh 初始化

```
# 初始化新專案（在專案資料夾裡執行）
git init

# 從別人的專案複製下來
git clone https://github.com/username/repo.git
```

初始化是什麼意思？

就像「開店準備」：

- 建立 .git 隱藏資料夾
- 設定好 Git 的基本環境
- 準備開始記錄版本歷史

執行完 git init 後，你的資料夾就變成 Git 倉庫了！

日常開發的三個步驟

Git 工作流程就像遊戲存檔：

Windows PowerShell 工作流程

```
# 1. 準備要存的東西（放進購物車）
git add 文件名
git add .          # 全部檔案都放進去

# 2. 按下記錄點（存檔）
git commit -m "新增使用者登入功能"

# 3. 上傳到雲端（讓別人也能讀）
git push
```

macOS zsh 工作流程

```
# 1. 準備要存的東西（放進購物車）
git add 文件名
git add .          # 全部檔案都放進去

# 2. 按下記錄點（存檔）
git commit -m "新增使用者登入功能"

# 3. 上傳到雲端（讓別人也能讀）
git push
```

實際練習（一鍵複製）

Windows PowerShell:

```
# 在你專案資料夾裡執行
New-Item hello.py -Value "print('Hello Git')"
git add hello.py
git commit -m "第一次記錄：新增 hello.py"
```

Windows PowerShell 練習續

```
git status
git log --oneline
```

macOS zsh 練習

```
# 在你專案資料夾裡執行
echo "print('Hello Git')" > hello.py
git add hello.py
git commit -m "第一次記錄：新增 hello.py"
```

macOS zsh 練習續

```
git status
git log --oneline
```

分支管理：平行宇宙開發

分支就像遊戲的「不同存檔槽」！

分支的比喻

想像你在玩遊戲，同時要開發新功能和修 bug：

- 主分支 (main)：正式版本，就像遊戲主線劇情
- 功能分支 (feature/login)：開發新功能的分支故事
- 修復分支 (bugfix/header)：修 bug 的平行世界

查看分支指令

```
# 看現在有哪些分支
git branch -a
```

建立新分支

```
# 建立並切換到新分支
git checkout -b feature/new-login
# 或新版指令：
git switch -c feature/new-login
```

分支切換

```
# 在分支間切換
git checkout main
git switch feature/new-login
```

合併分支

```
# 合併分支 (從 feature 合併到 main)
git checkout main
git merge feature/new-login
```

實際練習步驟

```
# 在你專案資料夾執行
git switch -c feature/add-button
echo "<button>Click me</button>" >> index.html
git add index.html
git commit -m "新增按鈕功能"
git switch main
git merge feature/add-button
```

為什麼要分支？

就像寫論文大綱：

- 主分支：最終定案的版本
- 草稿分支：試寫不同的章節
- 修改分支：針對特定問題修改

分支讓你可以「安全地實驗」，不會弄壞主版本！

SSH 金鑰：安全的鑰匙配對

SSH 金鑰就像網站的「VIP 鑰匙」！

沒有鑰匙：每次 push 都要輸入帳號密碼
有鑰匙：一勞永逸，自動認證

💡 小提示：為什麼要用 SSH？

```
就像飯店房卡：
- 傳統：每次都要刷卡驗證身份
- SSH：給你一把專屬鑰匙，一刷就進

GitHub 就是那個飯店，SSH 鑰匙就是你的 VIP 卡！
```

設定 GIT 基本設定

告訴 Git 你是誰！

```
git config --global user.name "Your Name"
git config --global user.email "yourname@email.com"
```

GIT 心智模型（一眼看懂）

工作區 → 暫存區 → 本地儲存庫 → 遠端

- `git add` 放入暫存區
- `git commit` 寫入本地儲存庫
- `git push` 上傳到遠端
- `git pull` 更新本地
- 隨時 `git status` 、 `git log` 掌握狀態

最小實務流程（從修改到上傳）

檢查狀態

```
# 檢查現在狀態
git status
```

加入檔案

```
# 加入要存的檔案
git add path/to/file
```

建立記錄點

```
# 按下記錄點
git commit -m "feat: add intro section to README"
```

上傳到雲端

```
# 上傳到雲端
git push
```

實際練習（一鍵複製）

```
echo "print('Hello Git')" > hello.py
git add hello.py
git commit -m "第一次記錄：新增 hello.py"
```

練習檢查結果

```
git status
git log --oneline
```

分支基础（建/切/合併）

建立新分支

```
git switch -c feature/intro      # 建新分支並切過去
```

查看所有分支

```
git branch --all                    # 看看所有分支
```

合併分支

```
git merge feature/intro          # 把分支合併進來
```

分支實際練習

```
git switch -c feature/add-button
echo "<button>Click me</button>" >> index.html
git add index.html
git commit -m "新增按鈕功能"
git switch main
git merge feature/add-button
```

為什麼要分支？

就像寫論文大綱：

- 主分支：最終定案的版本
- 草稿分支：試寫不同的章節
- 修改分支：針對特定問題修改

分支讓你可以「安全地實驗」，不會弄壞主版本！

衝突處理提醒

⚠️ 若發生衝突，打開檔案處理 `<<<<<<<` 標記

常見還原與後悔藥（安全優先）

還原檔案

```
git restore --staged file.js     # 從購物車移除，保留檔案
git restore file.js              # 檔案回到最後記錄點
```

重設記錄點

```
git reset --soft HEAD^          # 回到前一個記錄點
git commit --amend --no-edit    # 修改最後的記錄訊息
```

什麼時候用這些？

- 加錯檔案到購物車：用 git restore --staged
- 程式碼改壞了：用 git restore
- 想修改最後記錄：用 git commit --amend
- 想回到上個記錄點：用 git reset

診斷好幫手（看看現在怎麼了）

狀態檢查

```
git status                              # 現在狀態怎麼樣
```

歷史查看

```
git log --oneline --graph --decorate -n 10  # 記錄點歷史
```

差異比較

```
git diff                    # 現在和最後記錄點差在哪
git diff --staged           # 購物車裡的東西差在哪
```

這些指令告訴你什麼？

- git status：現在遊戲進度
- git log：過去的記錄點
- git diff：現在和上個記錄點差多少

常見誤區（避免踩雷）

版本控制常見錯誤：

- 在主分支直接改：先建新分支 `git switch -c feature/x`
- 忘記忽略檔案：加 `.gitignore` 後 `git rm -r --cached node_modules`
- 名字/Email 沒設：用 `git config --global user.name "名字"`

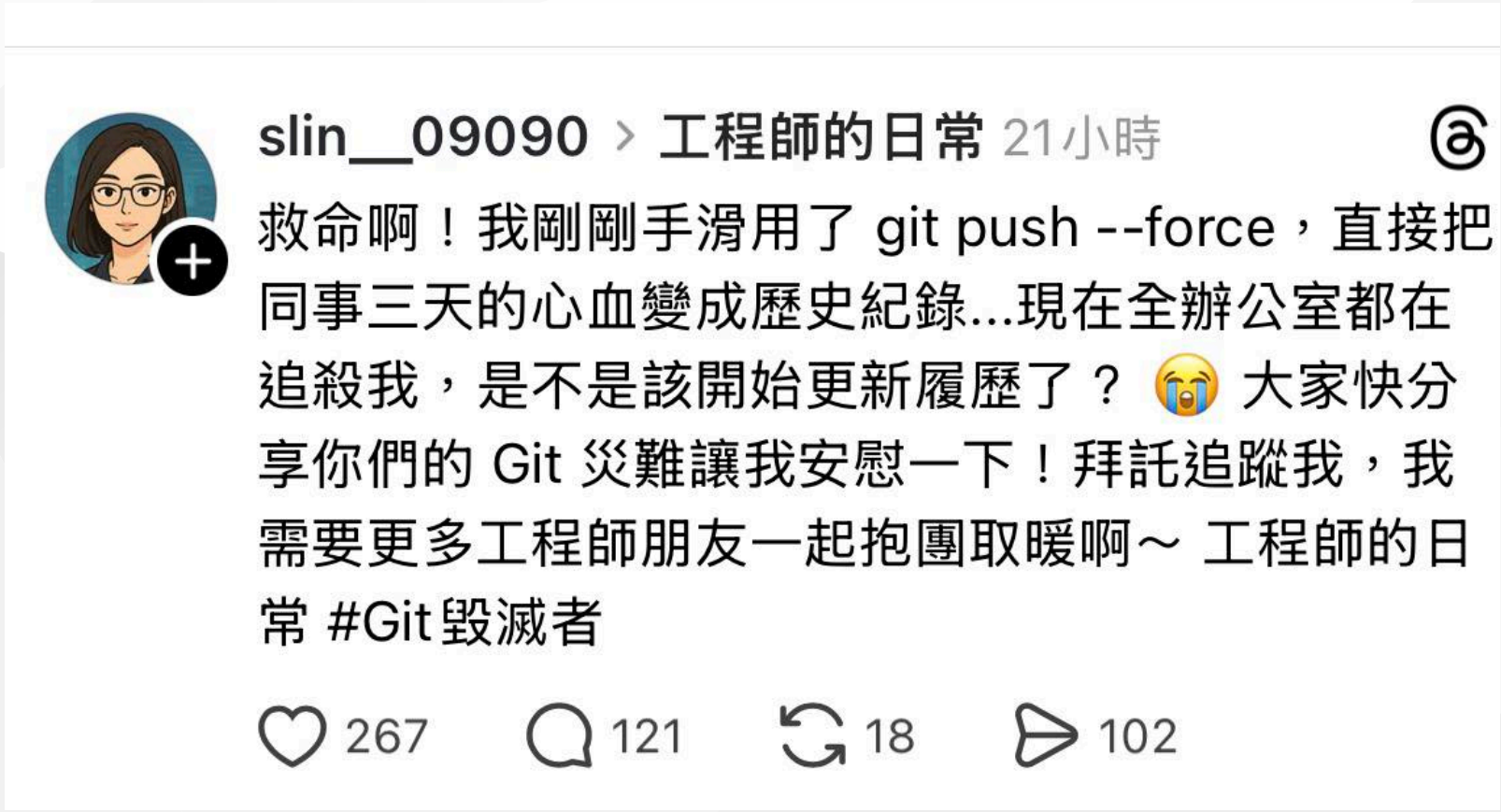
常見誤區續

- SSH 金鑰沒加到 GitHub：去 Settings 貼上 `~/.ssh/id_ed25519.pub`
- push 失敗：先 `git pull --rebase` 再推
- 強推別人分支：千萬別用 `--force`

補救方法

- 改歷史會影響別人：小心用 `rebase -i`
- 大檔案塞進 Git：用 Git LFS 或別的方法
- 迷路了：`git switch main` 回主分支
- 合併衝突：找 `<<<<<<<` 標記，手動修好

圖片：危險指令示意


slin__09090 › 工程師的日常 21小時
救命啊！我剛剛手滑用了 git push --force，直接把同事三天的心血變成歷史紀錄...現在全辦公室都在追殺我，是不是該開始更新履歷了？ 😭 大家快分享你們的 Git 災難讓我安慰一下！拜託追蹤我，我需要更多工程師朋友一起抱團取暖啊～ 工程師的日常 #Git毀滅者
267
121
18
102

圖片說明：Git 危險指令

這張圖片展示了 Git 中最危險的指令：

`git push --force` - 強制推送，會覆蓋遠端歷史！

為什麼危險？

- 會刪除遠端倉庫的歷史記錄
- 團隊成員的本地記錄會變成衝突
- 無法恢復被覆蓋的提交

安全替代方案

- `git push --force-with-lease` - 更安全的強制推送
- 先與團隊溝通再使用 `--force`
- 考慮使用 `git revert` 而非 `git reset`

系統級危險指令

- `rm -rf /` - 刪除整個系統
- `chmod -R 777 /` - 給所有人所有權限
- `curl ... | sh` - 亂執行網路程式
- `:(){ :|: & };:` - Fork炸彈

安全守則

- 永遠先在測試環境試
- 複製貼上前先閱讀
- 重要操作先備份

Git 快速參考（10 行）

基本指令

```
git status                          # 看現在狀態
git add <path>                      # 加到購物車
git commit -m "msg"                 # 按記錄點
git push                            # 上傳雲端
```

進階指令

```
git pull --rebase                    # 從雲端更新
git switch -c feature/x              # 建新分支
git merge feature/x                  # 合併分支
```

查看與還原

```
git log --oneline --graph -n 10     # 看記錄歷史
git diff / git diff --staged        # 看差異
git restore/restore --staged        # 還原檔案
```