XMY Mac 硬碟安全清理事件報告

從「差點無法開機」到建立可重複的安全清理制度

報告對象	謝銘元（艾克斯）／Meow Universe
清理事件	2026 年 8 月 3 日晚間
前因事件	2026 年 8 月 2 日因空間與資源壓力，Mac 曾無法正常開機
文件性質	事件復盤、專有名詞說明與預防制度
核心結論	安全清除可重建資料，釋放約 40 GiB；原始碼與 Git 資產未列入刪除範圍

一句話結論

2026 年 8 月 3 日做的不是「大掃除式亂刪」，而是一套有盤點、有清單、有確認字、有紀錄、可回復的工程清理。真正被刪除的是可以重新安裝或重新產生的建置物；真正重要的原始碼、Git Commit、分支、資料庫與 .env 都被排除。

報告版本：2026-08-05｜依現有紀錄重建，非完整磁碟鑑識報告

執行摘要

> **董事長版結論**
>
> 這次清理成功把 Data Volume 的可用空間從 83 GiB 提高到 123 GiB，約增加 40 GiB；Developer 工作區則從 77 GB 降到 31 GB，縮減約 46 GB。清理程序先找出 2,613 個可重建資料夾，預估 45.3 GiB，要求輸入大寫 CLEAN 才執行，並留下清單與 Log。之後又完成 Worktree 稽核及 GitHub 雲端封存，最後顯示 BACKUP_COMPLETE。

指標	清理前	清理後	變化	判讀
Data Volume 可用空間	83 GiB	123 GiB	+40 GiB	約 +48%
Data Volume 使用率	82%	73%	-9 個百分點	風險顯著下降
~/Developer 容量	77 GB	31 GB	-46 GB	約 -60%
可重建資料夾	2,613 個	已依清單清理	預估 45.3 GiB	不含原始碼
GitHub 安全封存	尚未完成	BACKUP_COMPLETE	指定分支與未提交內容已封存	未合併 main

清理後「實際增加約 40 GiB」低於掃描預估的 45.3 GiB，這是正常現象。原因包括檔案系統計算方式、清理過程新增的 Log、Git metadata、系統快取與背景服務重新寫入資料。真正應看的是 df 的清理前後差額，而不是只看預估值。

本次事件的範圍

- 主要清理範圍：/Users/xmy/Developer 之下的開發專案建置物與測試輸出。
- 配套處理：清除已不存在 Worktree 的 Git 登記、稽核仍有效 Worktree、檢查未提交內容、把指定內容封存到 GitHub。
- 未處理範圍：iCloud CloudDocs、Google Drive、FileProvider、照片資料庫等雲端同步資料，沒有直接用 rm -rf 清除。
- 前因：8 月 2 日 Mac 曾因資源與磁碟緩衝不足而無法正常開機；8 月 3 日清理是為了避免重演。

一、事件背景與時間線

1.1 前因：8 月 2 日曾發生無法正常開機

前一天，Mac 在大量 AI、開發、瀏覽器與雲端同步工作同時進行後，出現正常模式與安全模式都無法完成啟動的情況。Recovery 中曾看到 Data Volume 尚未掛載且 FileVault 顯示 Locked，後來透過解鎖、掛載與磁碟工具程式修復，才重新回到桌面。

依現有證據，最合理的事件鏈不是「硬碟硬體突然壞掉」，而是 RAM、Swap、磁碟剩餘空間、強制關機與檔案系統掛載狀態互相放大。這也是為什麼回到桌面後，不能只慶祝開機成功，還必須立刻治理容量。

1.2 8 月 3 日晚間的清理時間線

時間	動作
20:47 後	盤點磁碟、Developer 容量、Git Worktree 與可重建目錄。
21:34 左右	建立安全清理清單：2,613 個資料夾，預估 45.3 GiB；清單存到桌面。
21:43:15	輸入 CLEAN 後正式執行；記錄清理前 Data Volume 83 GiB 可用、Developer 77 GB。
21:43-21:47	清除不存在的 Worktree 登記，依清單刪除可重建目錄並逐項寫入 Log。
21:47:05	清理後 Data Volume 123 GiB 可用、Developer 31 GB；保留有效 Worktree。
21:57 後	執行 Worktree 安全稽核，辨識遠端追蹤、ahead/behind、未提交項目與 detached HEAD。
22:49 後	啟動 GitHub 雲端安全封存；不合併 main、不刪 Worktree、不改未提交內容。
封存完成	終端機顯示 BACKUP_COMPLETE；當時 Data Volume 約 122 GiB 可用。

二、2026 年 8 月 3 日實際做了哪些事情

2.1 先做容量盤點，不直接刪除

第一步是確認「整顆磁碟還剩多少」以及「哪個資料夾最肥」。這一步相當於先做財務盤點，再決定砍哪一筆成本；沒有盤點就刪檔案，等於董事會沒看報表就關部門。

```
df -h /System/Volumes/Data
du -sh "$HOME/Developer"
```

清理前記錄為：Data Volume 總容量約 460 GiB、已使用 356 GiB、可用 83 GiB、使用率 82%；Developer 資料夾約 77 GB。

2.2 掃描「可重新產生」的資料夾

接著使用 find 掃描 Developer 下的建置物與測試輸出，而不是掃描所有檔案。主要類型如下：

目錄	為什麼可刪
node_modules	JavaScript／Node.js 套件安裝結果，可由 package.json 與 lockfile 重新安裝。
node_modules.partial-*	套件安裝中斷後留下的半成品。
.next	Next.js 編譯、快取與建置輸出，可重新 build。
.turbo	Turborepo 的任務快取，可重新計算。
dist / build / out	編譯後的成品或匯出結果，通常可由原始碼重建。
.cache	工具為加速而暫存的資料。
coverage	測試覆蓋率報表。
playwright-report / test-results	瀏覽器自動化測試的報告、截圖與結果。

2.3 先建立清單，再要求人工確認

系統共找到 2,613 個可重建資料夾，預估占用 45.3 GiB，並把完整路徑存成：

```
/Users/xmy/Desktop/meow-rebuildable-20260803-213401.txt
```

正式刪除前，腳本明確宣告不會刪除原始碼、Git Commit、分支、資料庫或 .env，並要求輸入大寫 CLEAN 才繼續。這個設計叫做「人工核准閘門」：系統可以先自動做功課，但真正不可逆的動作必須由人確認。

2.4 清除已不存在的 Worktree 登記

Git 仍記得一些以前建立過、但實體資料夾已經不存在的 Worktree。這些項目顯示 gitdir file points to non-existent location，屬於過期登記。清理程序移除的是 Git 內部的失效索引，不是再刪一次程式碼。

> **重要區分**
>
> Worktree prune 清掉的是「地址簿裡已失效的地址」，不是把仍存在的房子拆掉。只有被確認為 prunable、且實際路徑早已不存在的登記才被移除。

2.5 依清單刪除可重建目錄並留下 Log

腳本逐一讀取清單、先記錄該資料夾容量，再執行 rm -rf。所有結果寫進：

```
/Users/xmy/Desktop/meow-cleanup-20260803-213401.log
```

rm -rf 是不可逆指令，因此本次能安全使用，關鍵不在於它本身溫柔，而在於它只讀取事先產生的白名單、路徑有加引號、刪除類型受限制，且執行前要求輸入 CLEAN。電鋸不是問題，問題是有沒有在客廳蒙眼睛揮。

2.6 清理後立即驗證

清理完成後，腳本再次執行 df、du 與 git worktree list，確認容量真的下降，而且有效 Worktree 還在。結果：Data Volume 可用空間 123 GiB，Developer 31 GB。

```
清理前：356 GiB used / 83 GiB available / Developer 77 GB
清理後：316 GiB used / 123 GiB available / Developer 31 GB
```

2.7 再做 Worktree 安全稽核

清理後不是立刻宣布勝利，而是逐一檢查有效 Worktree 的分支、HEAD、遠端追蹤、領先／落後、未提交項目與最後 Commit。稽核發現有些工作區乾淨、有些分支尚未設定遠端，也有工作區存在未提交檔案；這些都被列為後續封存對象，而不是直接刪除。

2.8 完成 GitHub 雲端安全封存

之後執行 GitHub 封存程序。腳本明確承諾：不合併 main、不刪除 Worktree、不刪原始碼、不改變現有未提交內容，只建立或更新安全分支。最終結果顯示 BACKUP_COMPLETE，代表指定分支與未提交內容均完成雲端封存。

> **封存結果**
>
> GitHub 封存完成後，磁碟約剩 122 GiB。比清理後的 123 GiB 少約 1 GiB，屬於 Git 操作、Log、物件資料與背景寫入造成的正常變化。

三、哪些東西被刪除，哪些東西被保留

分類	內容	判讀
被刪除	node_modules、.next、.turbo、dist、build、out、.cache、coverage、playwright-report、test-results 等可重建資料。	重新安裝套件或重新 build 即可產生。
被清理	已不存在路徑的 Worktree 登記。	只移除失效 Git metadata，不刪有效工作區。
被保留	原始碼、package.json、lockfile、Git Commit、分支、資料庫、.env。	屬於不可隨意重建或含重要版本／設定的資產。

分類	內容	判讀
被保留	仍有效的 Worktree，包括主工作區、整合分支與 Claude 工作區。	後續先稽核、再雲端封存。
未直接處理	CloudDocs、Google Drive、FileProvider、照片與其他 App 資料。	雲端同步資料不可直接從 Application Support 用 rm -rf 刪。

資料安全判斷

依現有清理清單、Log、Worktree 稽核與 GitHub 封存紀錄，沒有發現誤刪原始碼、資料庫、.env 或有效 Git Worktree 的證據。不過本報告是依操作紀錄重建，不等同於逐 sector 的專業磁碟鑑識。

四、清理成果與風險改善

4.1 空間改善

Data Volume 由 82% 使用降到 73%，等於重新取得約 40 GiB 的緩衝。對一般辦公使用，83 GiB 其實不算「完全沒空間」；但對你同時跑多代理人、Next.js build、套件安裝、AI 工具與大量分支的工作方式，83 GiB 仍可能在短時間內被 Swap、node_modules 與 build cache 吃掉。

Developer 從 77 GB 降到 31 GB，代表原本接近六成空間其實是可重新產生的建置物，而不是獨一無二的程式資產。這是最重要的營運洞察：未來不是靠更勤勞地人工刪，而是要把「可重建資料生命週期」制度化。

4.2 風險改善

風險	清理前	清理後	依據
磁碟逼近滿載	高	中低	可用空間增加至約 122-123 GiB。
誤刪原始碼	若人工亂刪則高	低	白名單、確認字、路徑加引號、Log。
Git 分支遺失	中高	低	Worktree 稽核＋GitHub BACKUP_COMPLETE。
失效 Worktree 干擾	中	低	prune 移除不存在路徑的登記。
再次爆滿	仍存在	需持續治理	若大量建立 worktree/build，空間會再長回來。

4.3 仍未完全解決的事情

- Homebrew 路徑錯誤仍存在：每次 Terminal 開啟會顯示 /opt/homebrew/bin/brew 不存在。這不是本次爆滿主因，但會影響開發工具環境。
- HermesWorkspace、playground、Library/Application Support 等其他大型區域仍需治理；這次主力清理的是 Developer。
- iCloud／Google Drive 的本機快取仍可能占用數十 GB，必須透過官方 App 或 Finder 管理，不應直接刪底層資料夾。
- 部分 Worktree 曾有未提交項目或沒有追蹤遠端；雖已做指定封存，未來仍要建立每日自動檢查。

五、專有名詞白話解釋

名詞	白話解釋
Terminal／終端機	用文字指令操作 Mac 的工具。像直接對系統下達精確命令，速度快，但空格、大小寫、引號都不能亂。
Shell／zsh	真正解讀你輸入指令的程式。macOS 預設常用 zsh；command not found 是它找不到你輸入的命令。
路徑 Path	檔案在電腦裡的完整地址，例如 /Users/xmy/Developer。路徑有空格時必須加引號。
df -h	看整個磁碟或 Volume 還剩多少空間；df 可理解成 disk free。
du -sh	計算某個資料夾實際占用多少；du 可理解成 disk usage。
GiB／GB	容量單位。GiB 以 1024 為進位，GB 以 1000 為進位，所以數字不會完全一樣。
Data Volume	macOS 真正存放使用者資料、App 資料與工作檔案的主要磁碟區。
APFS	Apple 的檔案系統，負責管理 Volume、快照、加密與空間共享。
FileVault	Mac 的磁碟加密。Recovery 中顯示 Locked 常只是尚未輸入密碼，不等於資料壞掉。
Recovery	macOS 的救援環境，可執行磁碟工具、重裝系統與 Terminal。
Mount／掛載	把磁碟區接到系統的目錄樹上，讓檔案可以被讀取。未掛載不代表資料消失。
Swap	RAM 不夠時，macOS 暫時借用硬碟當記憶體。硬碟越滿，Swap 越容易把系統逼到極限。
Cache／快取	為了下次更快而保存的暫存資料，通常可重新生成，但會逐漸膨脹。
node_modules	Node.js 專案下載的第三方套件。很大，但可依 package.json 與 lockfile 重裝。
package.json	專案的套件與執行腳本說明書，相當於「需要哪些零件」的清單。

名詞	白話解釋
lockfile	記錄每個套件精確版本，確保重裝後仍接近原來環境，例如 package-lock.json、pnpm-lock.yaml。
.next	Next.js 的編譯輸出與快取。刪除後第一次啟動會比較慢，但可重新 build。
.turbo	Turborepo 儲存的任務快取，用來加速重複 build／test。
dist／build／out	由原始碼產生的成品資料夾，通常不是唯一資料。
coverage	測試覆蓋率報表，用來看程式碼有多少被測試到。
Playwright	自動操作瀏覽器進行測試的工具；playwright-report 與 test-results 是測試產物。
Git Repository／Repo	一個被 Git 管理版本歷史的專案資料夾。
Commit	一次有編號、可追溯的版本存檔。它是 Git 的核心資產。
Branch／分支	從某個版本分出去的一條工作線，讓不同功能可平行開發。
Worktree	同一個 Git Repo 同時在不同資料夾打開不同分支，適合多代理人平行工作，但很容易吃硬碟。
HEAD	目前工作區指向的版本位置。
Detached HEAD	工作區直接停在某個 Commit，而不是正常分支上；若有新修改，較容易忘記保存。
Remote／origin	遠端 Git 伺服器的別名，通常 origin 指 GitHub 上的主要 Repo。
Ahead／Behind	本地分支比遠端多幾個 Commit／少幾個 Commit，用來判斷是否需要 push 或同步。
Uncommitted changes	已修改但尚未 Commit 的內容，是最容易因刪資料夾而遺失的部分。
Prunable／prune	Git 判斷某個 Worktree 登記已失效，可安全清除 metadata。
Manifest／目標清單	正式動作前先列出的完整路徑清單，讓刪除範圍可審查、可追蹤。
Log／紀錄檔	逐步記錄腳本做過什麼、何時做、結果如何，方便稽核與復盤。
rm -rf	永久遞迴刪除資料夾的指令，不會進垃圾桶。只能在路徑已核對、範圍被限制、且有備份時使用。
.env	專案的環境變數，常含 API Key、資料庫網址與秘密設定；通常不可隨便刪或上傳公開 Repo。
Homebrew	Mac 常用的開發工具安裝管理器。
.zprofile	每次開啟 zsh 登入 Shell 時讀取的設定檔；錯誤路徑會讓 Terminal 每次開啟都報錯。

六、這次最重要的教訓

教訓	意義
空間不足不是單純「檔案太多」	你的工作型態會同時產生套件、建置物、Worktree、AI 快取、雲端快取與 Swap。它們會互相放大，不能只靠偶爾清垃圾桶。
可重建資料也需要治理	node_modules 與 .next 本身沒有錯；問題是 100 個專案各留一份，並且每個 Worktree 再複製一次。
GitHub 不等於整機備份	GitHub 很適合保護原始碼與版本，但不會自動保護所有未提交檔案、資料庫、桌面文件與 App 設定。
雲端同步不等於不占本機	iCloud 與 Google Drive 為了離線與加速，仍可能保留大量本機快取。
多代理人需要退出機制	建立 Worktree 很容易；若沒有在 PR 完成後自動封存與移除，它會像會議室只訂不退，最後整層樓都被占滿。
危險指令必須制度化	不是禁止 rm -rf，而是所有不可逆動作都必須具備掃描、清單、確認字、Log、驗證與回復策略。

七、未來如何預防

7.1 建立硬碟交通燈

狀態	門檻（內部管理規則）	行動
綠燈	≥ 120 GiB 可用	可正常進行開發，但仍維持每週掃描。
黃燈	80-119 GiB 可用	停止新增大型 Worktree／模型；先清理 build、cache、node_modules。
紅燈	50-79 GiB 可用	暫停多代理人與大量 build；先 Git 封存，再清理。避免 macOS 更新。
危急	< 50 GiB 可用	停止高負載工作、同步與重開機；先備份與釋放空間。

以上不是 Apple 官方硬性門檻，而是依你的高負載 AI／多站開發工作制定的內部風險規則。一般人可能 50 GiB 還能用；你在同一時間跑十幾個代理人，50 GiB 比較像油箱已亮燈。

7.2 每週執行一次「只掃描、不先刪」

1. 記錄 df -h，確認 Data Volume 可用空間。
2. 列出 Developer、HermesWorkspace、playground、Downloads 與 Application Support 的容量排行。

3. 掃描 node_modules、.next、.turbo、dist、build、out、coverage 與測試報告。
4. 先產生 manifest 與預估容量，送出摘要。
5. 只有在人工輸入 CLEAN 後才刪除，完成後再次 df、du 與 Git 稽核。

7.3 每月底治理 Git Worktree

- 只保留目前真的在做的 Worktree；建議常態上限 5-10 個，而不是數十個永久堆著。
- 每個 Worktree 都要有：用途、負責代理人、建立日、分支、遠端、最後活動日、到期日。
- PR 合併或任務取消後，自動執行：確認已 push → 產生封存紀錄 → 移除 Worktree → prune metadata。
- detached HEAD 若有重要 Commit，先建立正式分支並 push；不要讓重要成果只掛在一串 SHA 上。

7.4 重新設計本機、GitHub、外接碟與雲端的分工

位置	應放什麼	管理原則
內建 SSD	只放正在開發的專案、近期資料與必要 App。	維持至少 120 GiB 可用。
GitHub	原始碼、Commit、分支、PR 與可追溯版本。	每個正式 Repo 都要有 origin；每日收工前 push。
外接 NVMe SSD	封存舊專案、資料集、大型模型、下載包與可保留的建置產物。	建議至少 2 TB；不要把所有 1,000 站都常駐在 512 GB 內建碟。
Time Machine 磁碟	整機備份與系統災難復原。	與工作／Archive 磁碟分開。
Google Drive／iCloud	商務文件、報告、照片、跨裝置同步。	採串流或按需下載；不要拿來同步 node_modules。

7.5 讓建置移到雲端，不要每站都在本機囤一套

喵宇宙要擴到約 1,000 個網站，長期解法不是買更大的垃圾桶，而是減少本機產生垃圾。正式 build 應盡量交給 GitHub Actions／Vercel；本機只保留必要依賴與少數活躍站點。共用模板、共用 package 與中央 Registry 可減少 100 個 Repo 各複製一份相同依賴。

- 預覽站優先用遠端 Preview Deployment，不為每個版本保留一個本機 recovered／candidate／preview 副本。
- 多代理人同時工作時，設定最大並行數；當可用空間低於黃燈門檻，自動拒絕再開 Worktree。
- 大型資料庫匯出、模型、ZIP 與報表集中到 Archive，不散落在 Developer、Downloads、Documents 與 playground。

7.6 雲端同步資料只用官方介面清理

先前盤點顯示 Application Support 中 CloudDocs、Google、Claude、FileProvider 與 MobileSync 占用明顯。這些資料夾與同步狀態、資料庫索引、離線檔案相關，不能照 node_modules 的方法直接 rm -rf。

- iCloud：在 Finder 對大型檔案使用「移除下載」，保留雲端版本。
- Google Drive：使用串流檔案或取消不必要資料夾的離線使用。
- iPhone／iPad 備份：在 macOS 儲存空間或 Finder 裡刪除舊備份。
- Claude／其他 AI App：先確認 App 是否提供清除快取或歷史資料功能，再處理底層資料夾。

7.7 修正 Homebrew 啟動錯誤

目前 .zprofile 第 6 行引用不存在的 /opt/homebrew/bin/brew。建議把它改成「只有 brew 存在才載入」，避免 Terminal 每次開啟都報錯：

```
if [ -x /opt/homebrew/bin/brew ]; then
  eval "$(/opt/homebrew/bin/brew shellenv)"
fi
```

這個修正不會直接增加硬碟空間，但能讓開發環境更穩定，也避免真正錯誤被每天固定出現的假警報淹沒。

八、未來標準清理 SOP

> **原則**
>
> 永遠遵守「掃描 → 清單 → 備份／Git 稽核 → 人工確認 → 清理 → 驗證 → 留 Log」。不要從「看到一個很大的資料夾」直接跳到 rm -rf。

步驟	動作
1. 停止寫入	先停大量 build、套件安裝、AI 多代理人與雲端大量同步。
2. 看總空間	df -h /System/Volumes/Data，記錄可用 GiB 與使用率。
3. 看最大來源	du 與 find 盤點 Developer、HermesWorkspace、playground、Downloads、Library。
4. Git 安全檢查	找出未提交內容、未 push 分支、detached HEAD、無 origin Repo。
5. 建立白名單	只列可重建目錄，輸出 manifest 與預估容量。
6. 人工核准	顯示「不會刪什麼」，要求輸入大寫 CLEAN。
7. 執行與紀錄	逐項記錄容量、路徑與結果；不要隱藏錯誤。
8. 清理後驗證	再次 df、du、git worktree list、git status。
9. 雲端封存	對尚未上 GitHub 的重要分支與未提交內容建立安全分支。
10. 寫報告	留下清理前後數字、Log 路徑、異常與下一步。

Mac 再次無法開機時的簡化救援順序

1. 不要反覆強制關機，也不要立刻重裝 macOS。
2. 進入 Recovery，先用「磁碟工具程式 → 急救」檢查 Volume。
3. 若 Data Volume 顯示 Locked／Not Mounted，先用 diskutil apfs list 確認正確磁碟代號，再解鎖與掛載。
4. 確認 /Volumes/Data/Users/xmy 可讀後，優先備份重要資料。
5. 空間不足時只刪明確可重建的 cache/build，不碰 CloudDocs、FileProvider、照片圖庫與不明系統資料夾。
6. 成功回桌面後先備份、清理、修復，再考慮更新或重開機。

九、建議的 30 天落地計畫

優先級	工作	完成標準
P0	建立 meow-mac-health 每日只讀檢查	輸出可用空間、前 30 大資料夾、Worktree、未 push／未 Commit 狀態。
P0	建立每週安全清理腳本	預設 dry-run；產生 manifest；輸入 CLEAN 才執行。
P0	完成所有正式 Repo 的 origin 與雲端備份	無 origin、未追蹤分支、detached HEAD 全部列入治理。
P1	把 Archive 搬到 2 TB 外接 NVMe	舊專案、資料集、ZIP、模型與大型快照移出內建 SSD。
P1	收斂工作根目錄	正式開發以 ~/Developer 為主；HermesWorkspace／playground 僅保留明確用途。
P1	建立 Worktree 到期政策	任務完成 7 天內封存並移除；活躍 Worktree 設上限。
P2	調整 iCloud／Google Drive 離線策略	CloudDocs 與 Google 快取採按需下載，避免重複常駐。
P2	修正 .zprofile／Homebrew	移除失效啟動命令或重新安裝並驗證。

最優先的三件事

第一，讓磁碟低於 120 GiB 時自動警告；第二，把每週清理做成「先掃描後核准」；第三，讓每個 Worktree 在任務結束後自動封存與退場。這三件做完，未來不需要靠董事長親自判斷 2,613 個資料夾誰該走。

附錄 A｜本次留下的主要證據檔

證據	位置／檔名	用途
清理目標清單	/Users/xmy/Desktop/meow-rebuildable-20260803-213401.txt	2,613 個可重建目錄。
清理執行紀錄	/Users/xmy/Desktop/meow-cleanup-20260803-213401.log	逐項容量、刪除與清理前後驗證。
Worktree 稽核	meow-worktree-audit-20260803-215713.txt	分支、HEAD、遠端、ahead/behind、未提交項目。
GitHub 封存紀錄	/Users/xmy/Desktop/meow-cloud-safety-backup-20260803-224928.log	最終結果 BACKUP_COMPLETE。
前因復盤	Mac 無法開機事件復盤與預防報告_XMY_2026-08-02	Recovery、FileVault、APFS、掛載與原因分析。

附錄 B｜本次關鍵指令的意思

指令	作用
df -h /System/Volumes/Data	查看 Data Volume 的總容量、已用與可用空間。
du -sh ~/Developer	計算 Developer 總容量。
find ... -name node_modules ...	只找指定名稱的可重建資料夾。
git worktree list --porcelain	用容易被程式解析的格式列出所有 Worktree。
git worktree prune --verbose	移除路徑已不存在的 Worktree 登記。
rm -rf "$DIR"	永久刪除清單內指定資料夾；引號可保護含空格路徑。
tee -a "$LOG"	同時把畫面輸出顯示在 Terminal 並追加到 Log。
2>/dev/null	把不重要的錯誤訊息丟棄，讓掃描結果較乾淨；正式清理仍應保留關鍵錯誤。

結論

這次清理是成功的。它不只釋放了約 40 GiB，更重要的是把「什麼可以刪、什麼絕對不能碰」分清楚，並驗證了 Git 資產仍可追蹤、重要內容已完成雲端封存。

但這不是一次性勝利。以喵宇宙目前的開發速度與未來 1,000 站規模，硬碟會再次長滿是必然，不是意外。真正的解法是把容量門檻、Worktree 到期、雲端 build、外接 Archive、Git 封存與安全清理腳本一起變成作業系統。這樣下次硬碟不會等到罷工才發言。

— 報告完 —