

Claude Code 最佳實務指南

系統優化與操作手冊 ■

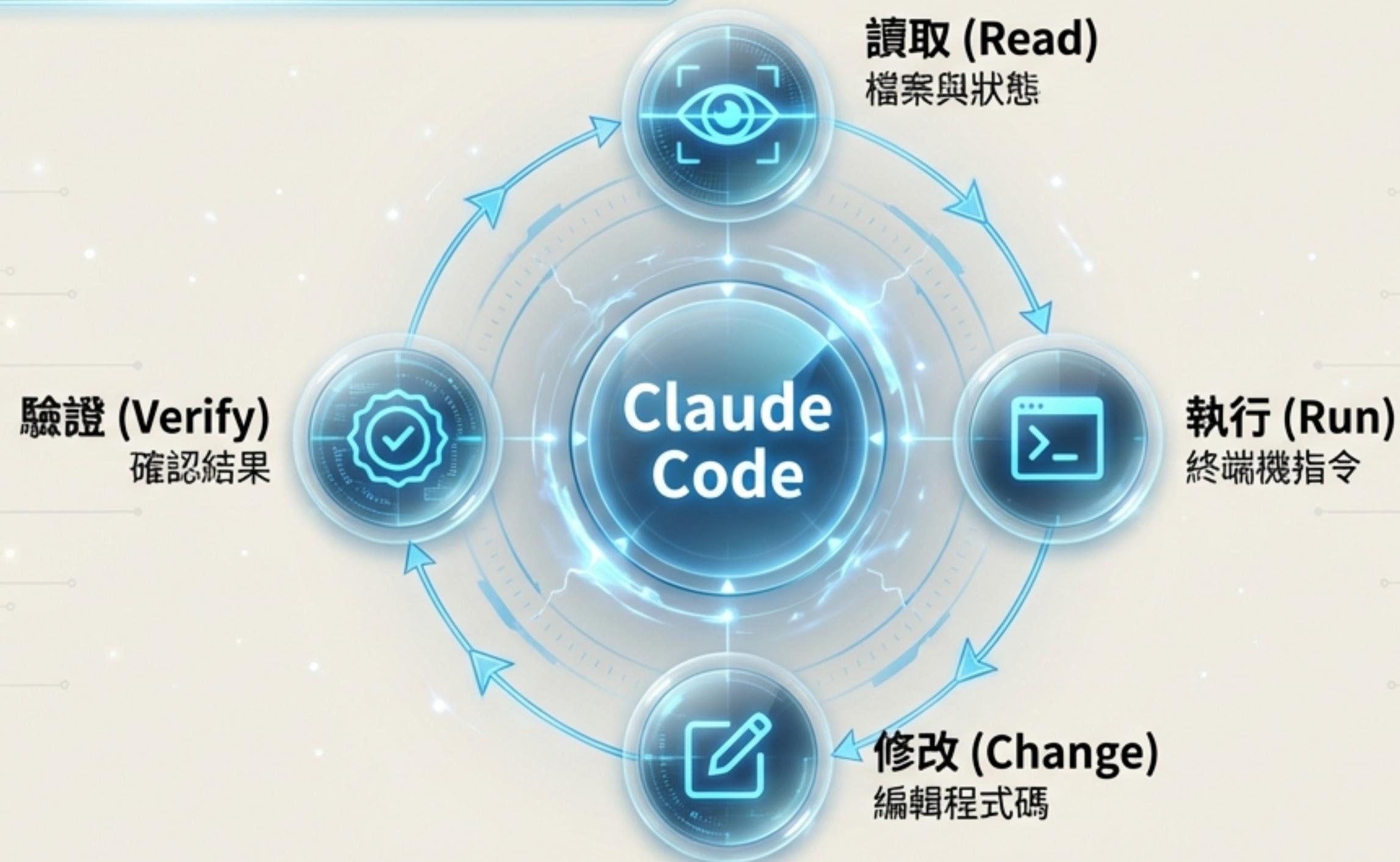
核心限制：脈絡視窗 (Core Constraint)



隨著脈絡填滿，效能將會下降。

- 脈絡視窗是您最珍貴的資源。
- 單次除錯或探索可能會消耗數萬個 Token。
- 當視窗將滿時，Claude 可能會開始「遺忘」指令或產生錯誤。

代理迴圈機制 (The Agentic Loop)



Claude Code 是一個代理編碼環境 (Agentic Coding Environment)。它不只是回答問題，而是會主動解決問題：從探索、規劃到實作。

專案準則：CLAUDE.md

File: CLAUDE.md

Code Style

- Use ES modules (import/export)
- Prefer functional programming patterns

持久性脈絡
(Persistent Context)

Workflow

- Run single tests, not full suite
- Always run lint before commit

Coding Style 規範

常用 Bash 指令

保持精簡。若移除該規則不會導致錯誤，請刪除它。

保持 簡。若移除該規則不會導致錯誤，請刪除它。過長的規則檔會導致 Claude 忽略重要指令。

位置：專案根目錄或 ~/.claude/CLAUDE.md

環境安全與權限配置 (Security & Permissions)

System Configuration

允許清單 (Allowlist)

[ON]

自動授權常用指令 (如 npm test, git commit)。指令：/permissions

沙箱模式 (Sandbox)

[ON]

限制檔案系統與網路存取，隔離風險。
指令：/sandbox

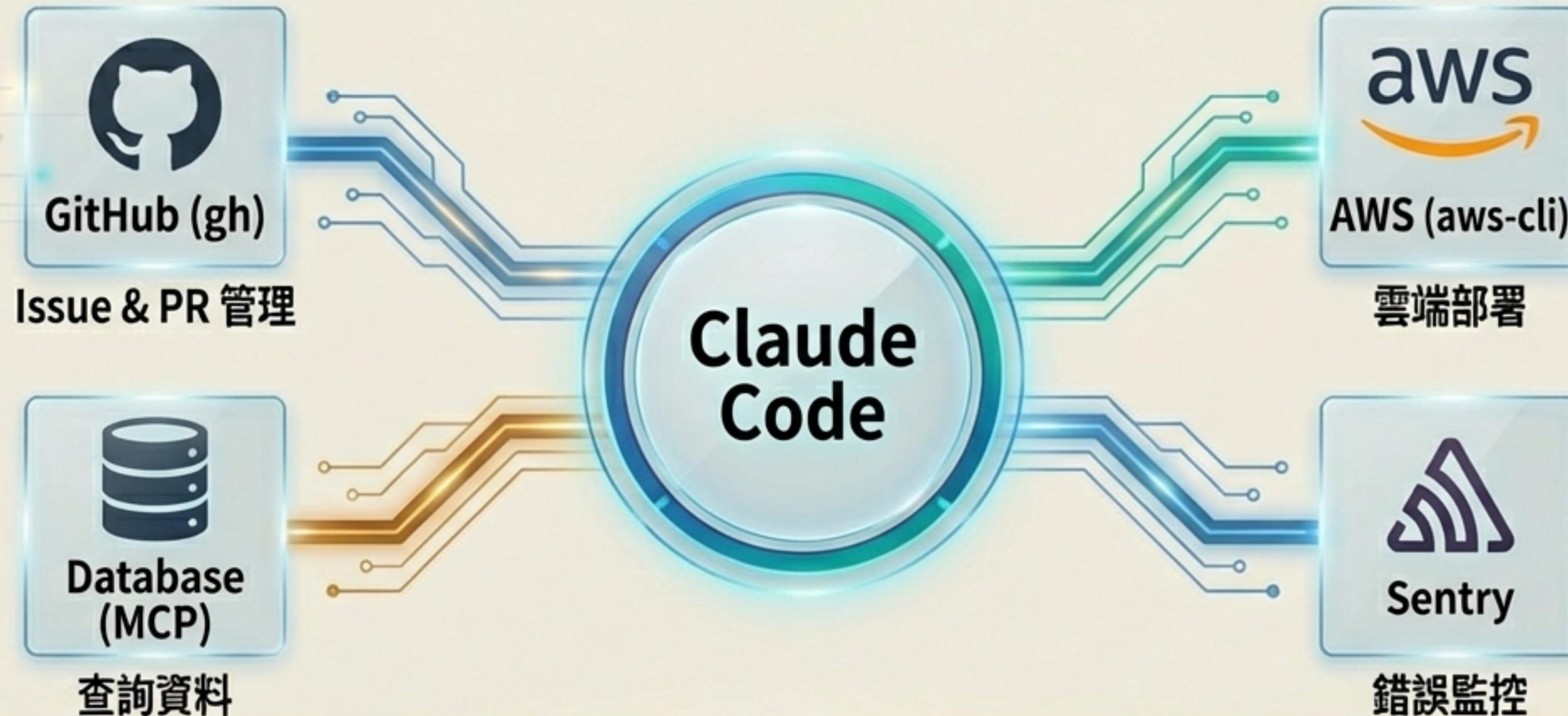
危險模式 (Dangerously Skip)

[WARNING]

跳過所有權限檢查。
僅限無網路容器中使用。

平衡安全與速度。使用允許清單減少干擾，使用沙箱防止資料外洩。

工具整合：CLI 與 MCP (Tool Integration)



賦予 Claude 自行獲取脈絡的能力 (Fetch Context)，而非手動複製貼上。
CLI 工具讓 Claude 能直接操作外部服務；MCP 伺服器擴展其能力邊界。

黃金法則：驗證機制 (Verification Mechanism)

Blind Execution (盲目執行)



僅生成程式碼
(Code Only)

Claude 無法確認程式碼是否有效，人類成為除錯瓶頸。

Verified Execution (驗證執行)



程式碼 + 測試案例
(Code + Tests)

Claude 執行測試、比對截圖、確認 Build 成功。

若沒有明確的驗證標準，你就會成為除錯的瓶頸。要求 Claude 『修復錯誤並確認測試通過』。

分階段執行策略 (Phased Execution)

探索 (Explore)

Plan Mode

讀取檔案，
不修改。

規劃 (Plan)

Plan Mode

產出實作計畫
(Implementation Plan)。

實作 (Implement)

Normal Mode

依據計畫編碼
與測試。

提交 (Commit)

Normal Mode

建立 PR 與
Commit
Message。

先規劃，後編碼。使用 Plan Mode 降低風險，避免 Claude 跳過思考直接寫出錯誤的程式碼。

精準指令工程 (Precision Prompting)

Terminal Input

① Before / 模糊指令

Context: Low

> 新增測試 (add tests)

Terminal Input

② After / 精準指令

Context: High

> 為 foo.py 新增測試，涵蓋使用者登出的邊界案例，避免使用 Mock。參考既有的 Auth 測試模式。

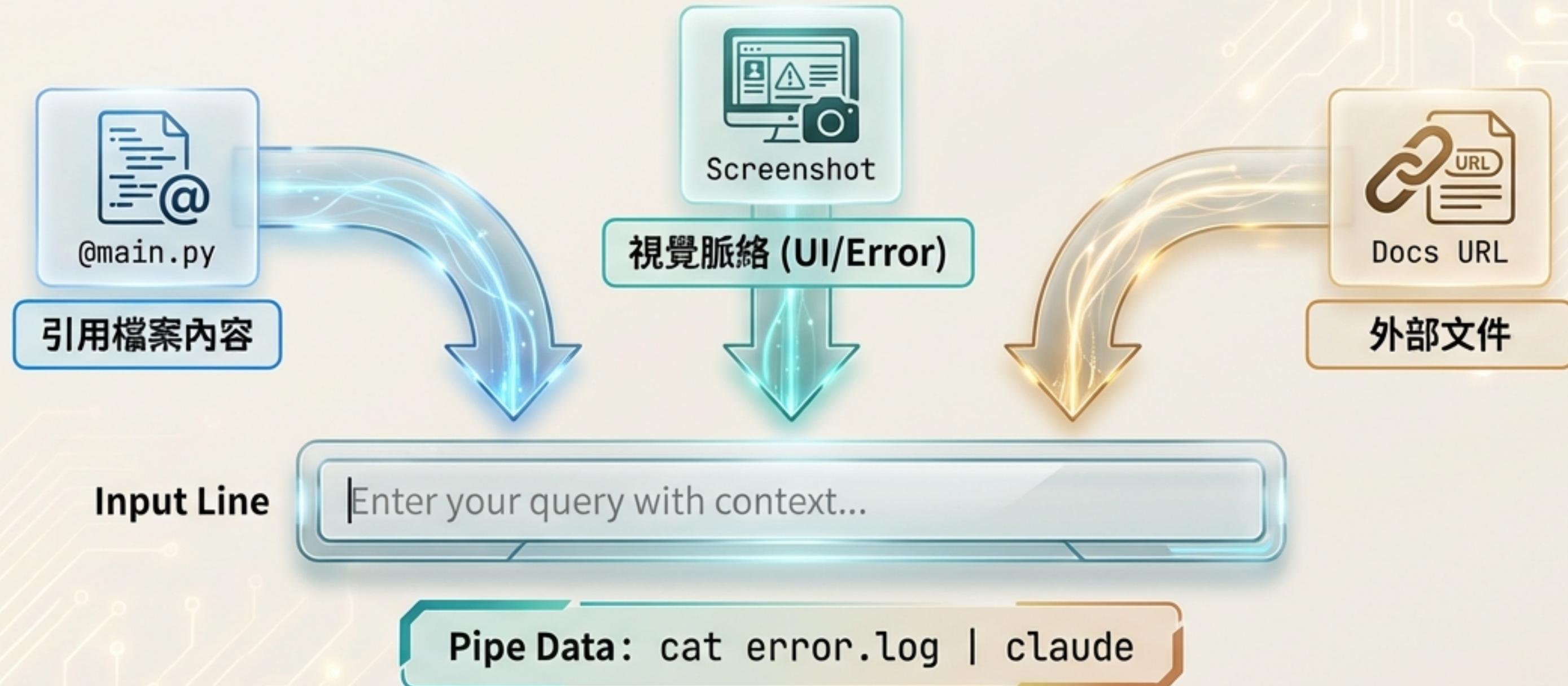
🎯 定義範圍：明確指出檔案名稱與情境。

📘 指向來源：引導 Claude 查閱 Git History 或文件。

✳️ 參考模式：要求「參考現有的實作模式」。

🐞 描述症狀：提供具體錯誤訊息而非僅說「修復 Bug」。

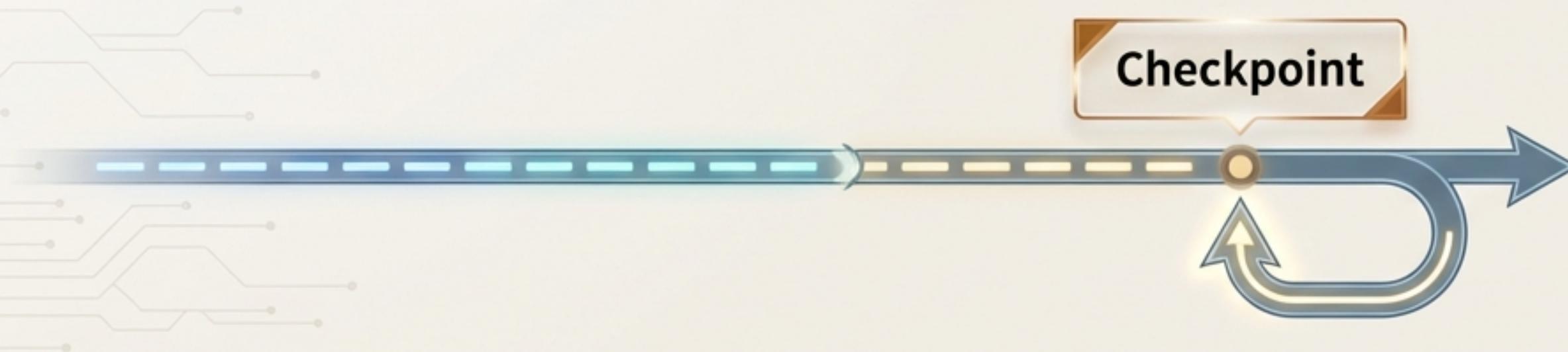
豐富脈絡注入 (Rich Context Injection)



使用 '@' 提及檔案。直接貼上設計圖或錯誤截圖。善用 Pipe 將終端機輸出傳送給 Claude。

路徑修正與回溯 (Course Correction)

路徑修正指令 (JetBrains Mono)



雙擊 ESC = Rewind (回溯)
= JetBrains Mono

II 快速中斷

按 ESC 停止 Claude 的動作。

◀ 時光回溯

使用 /rewind 回到錯誤發生前的狀態。

⌚ 清除脈絡

任務完成後，使用 /clear 重置視窗。



不要陷入錯誤迴圈 (Death Spiral)。若修正兩次仍失敗，請立即回溯並優化指令。

溝通策略：反向面試 (Reverse Interview)

為了開發這個功能，我需要確認：

1. 權限管理的邏輯？
2. 是否有特定的 Edge Case 需要考慮？



Claude



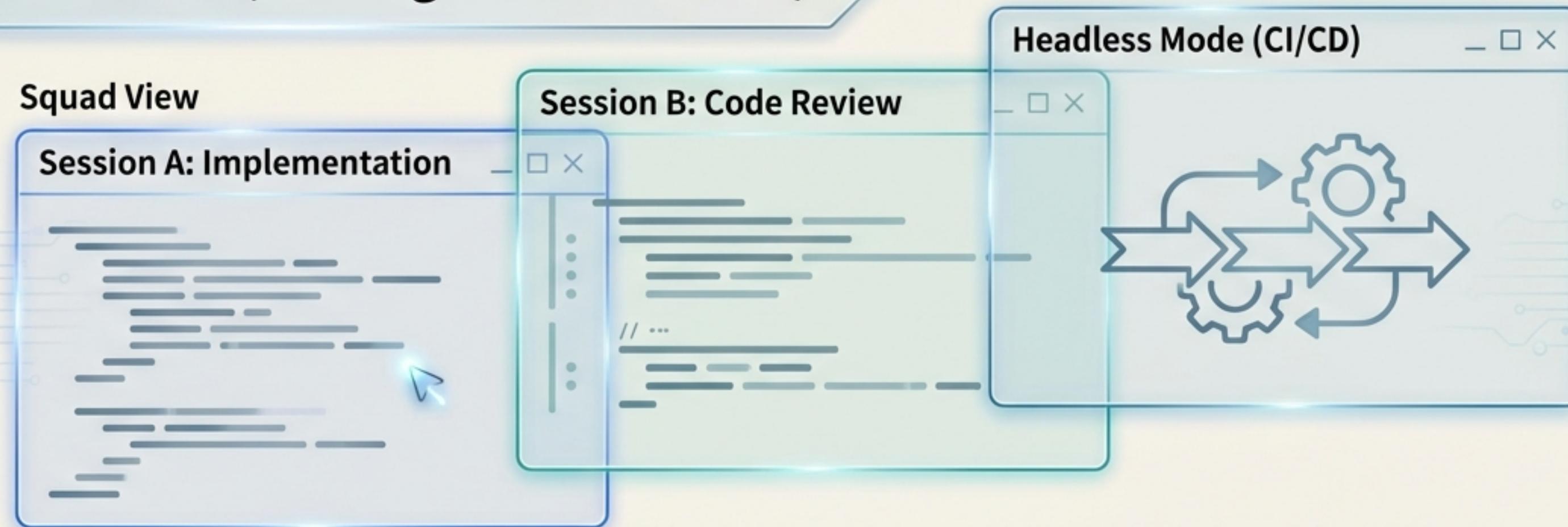
User

權限邏輯請參考 @auth.ts。
Edge Case 請考慮 token 過期的狀況。

將 Claude 視為資深工程師 (Senior Engineer)。

在開發大型功能前，要求 Claude 使用 `AskUserQuestion` 工具面試你，
找出你沒考慮到的技術盲點。

規模化與自動化 (Scaling & Automation)



```
claude -p "Migrate this file" --output-format json
```

- **平行作業**

同時開啟多個 Session，分別進行「實作」與「Code Review」。

- **無頭模式 (Headless)**

整合至 CI/CD 或 Pre-commit hook。

- **大規模扇出 (Fan-out)**

透過腳本批量處理數千個檔案的遷移任務。

技能與子代理 (Skills & Subagents)

Select Class

[Selected] Security Reviewer (Subagent)

[] API Specialist (Subagent)

[] Fix Issue Workflow (Skill)

委派研究任務給子代理 (Subagents)，讓它們在獨立的脈絡中閱讀大量文件並回報摘要，不占用主視窗空間。

```
.claude/  
└─ skills/ (SKILL.md)  
└─ agents/ (SecurityReviewer.md)
```

常見失誤與總結 (Debrief & Summary)

Mission Report

⚠ Avoid (避免) ⚠

- 大雜燴對話 (The Kitchen Sink)：混合無關任務。→ 解法：`/clear`
- 無限修正 Loop：錯誤堆疊。→ 解法：`/rewind`
- 信任即發布：未經測試。→ 解法：要求驗證。

★ Final Advice (最終建議)

培養直覺。觀察何時該詳細規劃 (Plan Mode)，何時該讓 Claude 自由探索 (Normal Mode)。

