讓 AI 值得放手

Lauren Tan 的方法，整理與落地建議

整理：Cab｜CabLate

核心架構：一個錯誤的五種擋法（由強到弱）

1 結構	讓錯根本做不出來（資料形狀、邊界、單一寫法）
2 機器擋	型別、lint、CI 攔截並附改法
3 跑起來驗	測試、操作實證、交出證據
4 一點點文字	規則與 skill 只留給需要判斷的事
5 人看	人工審查成本最高，放在最後一道

這份文件的觀念取自 Lauren Tan（@poteto）公開的文章、演講與開源專案 pstack，並非整理者原創。整理者只做三件事：讀完資料、用自己的話重整，再改寫成可以照著執行的規則。凡屬整理者自己的推論或補充，都標上「整理者」。她沒有參與這份文件，也沒有為內容背書；文中她的成果數字全由她自己所說，沒有經過第三方查核。各篇原文出處請看〈出處與各篇重點〉。

目錄

第一部　方法與落地建議

1. 一頁看完
2. 她的觀念，分六組
3. pstack 是什麼
4. 落地建議
5. 非程式工作怎麼用
6. 不要照搬的地方
7. 這份整理是怎麼做的

附錄　出處與各篇重點

一、長文與貼文（X）
二、演講與訪談
三、pstack 原始碼
四、沒有取得或不採用的
五、她的數字

附頁　關於整理者

> 她 她明說的，附出處　整理者 整理者的推論、類推或補充
> 自述 她自己報的數字，無第三方查核

第一部

方法與落地建議

先認幾個詞

agent：會自己讀檔、執行指令、動手修改的 AI 程式。

環境：agent 工作時碰得到的一切，包含專案裡的檔案與既有寫法、可用的工具、自動檢查與規則文件。

lint、型別、CI：這三者都是自動檢查。lint 會照規則掃描程式寫法；型別從資料結構下手，讓不合法的資料寫不出來；CI 則是每次改動後自動跑一輪檢查與測試的流程。

token：模型讀寫文字的計量單位，用得越多成本越高。

skill：寫給 agent 看的一份做法說明，用到時才載入。

pstack：她開源的那一套 skill，第 3 章會詳細介紹。文中「使用指南第 N 章」指的是 pstack 內附的十章指南；「長文 Pt. 1、Pt. 2」則是她在 X 上發表的兩篇〈The Complete Guide to pstack〉，兩者屬於不同的文件。

01.

一頁看完

> **AI 能不能放手，不看模型多聰明，看兩件事：環境讓它很難做錯，以及它做完能自己拿出證據。**

她的核心主張 她

1. 產量來自信任。信任不是盲信，是把力氣投資在 agent 工作的環境上。（AI Engineer 演講〈All Code Is Slop (Except Mine)〉）
2. 瓶頸在驗證，不在寫程式。如果 agent 沒辦法自己驗證，多開幾個對話只是多產出一堆沒檢查過的東西。（〈How I Use Cursor〉；pstack 使用指南第 6 章）
3. 先把單線做深，再把數量放廣。把還不信任的 agent 拿去平行多工，只會浪費 token、製造更多爛程式。（〈How I Use Cursor〉）
4. 每次糾正錯誤，都先問能不能改到讓它「做不出來」或「機器會擋」；文字規則排在最後，因為規則就算被跳過，也沒有任何檢查會因此失敗。（pstack `/correct`、使用指南第 9 章；AI Engineer 演講 [08:43]）
5. 環境就是 agent 的記憶。專案裡留著什麼寫法，它就會照抄什麼寫法，好的抄、壞的也照抄。（〈Verification is all you need〉；AI Engineer 演講）
6. 用程式做計畫：凡是能跑出來看結果的問題，就用原型直接回答。（長文 Pt. 2）
7. 給 agent 讀的文字要一直刪。模型變強之後，過去寫給舊模型的細節指令不必留，只保留流程就好。（pstack 版本紀錄；Matt Pocock 訪談 [63:26]）

一個錯誤的五種擋法，由強到弱 【整理者把她的兩個說法併成一張表：前四層是她的糾正去處（pstack `/correct`），最貴的一端是人工審查出自她的信任金字塔（AI Engineer 演講）】

1	**結構**	讓錯根本做不出來：定好資料形狀、目錄與邊界，一件事只留一種寫法，把容易誤用的東西整個禁掉

2	**機器擋**	靠型別、lint 與 CI 攔截，並在報錯訊息裡直接寫明該怎麼改
3	**跑起來驗**	寫測試、實際操作成品或做形式驗證，做完交出前後對照的證據
4	**一點點文字**	規則與 skill 只留給真的需要臨場判斷的事
5	**人看**	人工審查成本最高，放在最後一道

想照做，先做這三件 整理者

1. 動手前先過入口閘：先問這件事做錯了退不退得回？對不對能不能用程式驗？（規則 0）
2. 每次交代工作都寫明怎樣算完成、要看什麼證據，並儘早給 agent 一個自己驗證的固定入口（規則 4、規則 3）。她說驗證入口是整套做法裡回報最大的一件。
3. 改掉「糾正一次就補一條規則」的習慣，換成「糾正之後，從最強那一層往下找去處」（規則 1、規則 2）。

02.

她的觀念，分六組

> **這六組講的是同一件事的不同面向：怎麼把「人盯著看」換成「靠環境撐著」。**

2.1 信任是爬上去的，而且沒有捷徑

這套方法的起點，是她發現自己成了 agent 和除錯工具之間的「人肉轉接」：每個輸出都要她親眼看過，每個訊號都要她手動搬運。她於是反覆追問兩個問題：我在哪裡成了瓶頸？agent 為什麼需要我來回答這個問題？ 她（Matt Pocock 訪談 [04:53]、[37:54]）

從人工盯場到真正放手，一端是一次盯 1～5 個對話，另一端是 agent 自己合併、她事後再看。她說盯場那一階最難離開；從盯場走到放手沒有捷徑，但現成的做法能讓人更快上手。 她（〈How I Shipped 2000 PRs〉[05:34]；Maven 工作坊 [30:23]）她自己花了大約六個月，才走到讓 agent 在她睡覺時合併的程度，並說別人應該會快得多。 自述（AI Engineer 演講 [01:26]）

談到自己在這套系統裡的角色，她不喜歡「軟體工廠」這個詞，改用「米其林廚房」來比喻：主廚不再親手煮每道菜，但名字依然掛在成品上，要管的是廚房怎麼運作。 她（Matt Pocock 訪談 [11:18]）

2.2 驗證是瓶頸

在她的標準裡，「能編譯」不算證據。完成條件必須寫在第一則指令裡，回報要附上確切的指令與輸出；跑不了的項目得老實說無法確定，沒有證據卻自信滿滿的回覆反而是警訊。 她（pstack 使用指南第 6 章）

她進公司做的第一個 skill 就是驗證，讓 agent 像真人一樣把應用程式跑起來、操作、截圖、錄影並取得效能紀錄。她說，在做出這個入口之前，其他 skill 再好，她都還是 agent 和成品之間的轉接。 她（〈Loops You Can Trust〉；Matt Pocock 訪談 [16:56]）

實作上，她把驗證做成固定的小工具，不讓每個 agent 每次自己重寫腳本。她觀察到，一旦沒有固定工具，每個 agent 都會重新搭一套驗證流程，而且每次做法還都不同；在她看來，skill 只是包著工具的一層薄說明。她（Matt Pocock 訪談 [20:40]～[23:23]）

為了讓 agent 找得到路，她還搭配一份可搜尋的「功能地圖」，記錄每個功能做什麼、使用者怎麼走到那裡。她把這份地圖叫做「被具體化的記憶」，並建議每天跑一次維護。她（〈Verification is all you need〉）

不過，**驗證讓它至少寫對，不保證寫好**。她的原話是，驗證不保證 agent 能寫出好程式，只能確保它至少寫出正確的程式，而且只涵蓋你驗得到的範圍。她（Maven 工作坊 [08:07]；〈How I Shipped 2000 PRs〉[12:41]）

至於品質靠什麼守住，她在別處分開談過：靠結構設計與自動檢查，再加上抽樣審查。整理者合併再往上一層則是形式驗證，她說這是自己最近才開始嘗試的做法。她（AI Engineer 演講 [10:45]）

不管是她的形式驗證 skill 要求檢查先在故意改壞的版本上失敗一次，還是 `/correct` 規定每個新檢查都在一個真實的舊錯誤上失敗過，背後的標準都很一致：沒看過它失敗的檢查不算數。她（pstack 使用指南第 6 章；pstack `/correct`）

這套驗證流程非常花 token。她引用別人「七成 token 花在驗證」的說法，並說自己也是大部分花在這裡；在全自動模式下，單一改動會派出多個驗證 agent，但這個數量可以往下調到一個。她（AI Engineer 演講；Matt Pocock 訪談 [52:31]）

2.3 環境就是記憶，工程師變成園丁

agent 會直接從程式庫裡學怎麼寫程式。只要你在裡頭留了一個權宜之計，下一個進來的 agent 就會把它當成慣例；旁邊若還配了一段解釋它的註解，看起來就更像是刻意的設計。她（AI Engineer 演講）

所以她禁掉絕大多數註解，把想要傳達的限制放進 lint 的報錯訊息裡，讓 agent 正好在違規的那一刻讀到。她只留下四種例外：授權聲明、公開 API

說明、解釋程式本身無法表達之事的連結，以及外部相依逼出來的特殊行為。她（AI Engineer 演講；pstack 使用指南第 5 章）

同樣的道理也用在寫法規範上：一件事只留一種做法，而且讓最省力的寫法剛好就是對的寫法。她的內部框架把每個功能放進固定目錄，起因就是最早的版本曾長成幾個上萬行的大檔。她（Matt Pocock 訪談 [30:01]～[31:19]）

她也在產品程式裡禁用一個容易誤用的 API，只准框架內部使用。她（AI Engineer 演講）整理環境時，她的順序十分明確：先加一條檢查擋住新增，再清掉舊的壞範例，最後確保驗證夠強。她（AI Engineer 演講 [15:20]）她發出的大量改動裡，其實有很多都是在做這種整理，沒有開發新功能。她（Matt Pocock 訪談 [45:51]）

2.4 糾正要落在環境上

一旦發現 agent 犯錯，糾正有四級去處，由強到弱依序是：改結構、交給型別與 lint 及 CI 擋掉、寫進測試，最後才輪到文件與規則。文字排在最後，只留給需要判斷的問題。她（pstack `/correct`）

她也不急著見一個補一條規則：同一類錯誤出現兩次才算一類，單純一場怪對話只是軼事，算不上規則。她（pstack `/correct`、使用指南第 9 章）

在日常巡檢裡，她會讓一個 agent 持續找壞寫法，但只把發現記進一份清單；隔幾天回頭看，常會發現眼前列了一堆，背後其實是同一件事。直接逐一去修反而會看不到全貌。她（Matt Pocock 訪談 [47:44]）

在她眼裡，審查就是抽樣。抽樣看到問題時要修的是環境，修單一 agent 沒有意義；多個 agent 犯同一個毛病，才是該動手改環境的訊號。她（Matt Pocock 訪談 [49:14]～[51:19]）

為了不讓規則越積越肥，她會維護一張「每條規則由什麼強制」的表。規則還在、背後卻沒有東西強制、又被重複違反，就在同一次修改裡升到更強的那一層；反過來，錯已經不可能發生了，就刪掉那條規則。真的需要破例時，就把說明寫在破例的位置旁，附上理由、到期日和核准的人，絕不開整份放行的特權。她（pstack `/correct`）

2.5 怎麼下指令、怎麼做設計

下指令時，她只給五樣東西：目標、能明確判過或不過的完成檢查、想看的證據、已知事實，以及真正的限制。她不寫執行步驟，也不一開始就把自己猜的原因說出來，因為那會把 agent 的搜尋範圍縮死在你指的地方。她（pstack 使用指南第 2 章）

這個下法是針對現在的強模型；她說以前面對舊模型時會講得很細、幾乎是盯著指揮，現在已經不必了。她（長文 Pt. 2）而最常見的失敗，往往是 agent 誤讀了你的意思；當需求本身比較吵雜時，先叫它用自己的話重述一遍，把誤讀在開工這一步修掉最便宜。她（長文 Pt. 2）

談到設計，她曾說自己不相信計畫，真正的意思是她用程式來做計畫：能跑出來看的岔路就做原型並排比較；共用的東西先寫使用說明；跨模組的改動先畫出型別與介面。等設計定下來、需要拆成多個步驟時，她才寫計畫。她（長文 Pt. 2）

對於沒有實物的計畫，她不做對抗審查，因為審查的 agent 會憑空發明不會發生的風險；相對地，設計或計畫可以拿原型或直接探查現有程式來審查。她（長文 Pt. 2；pstack `/interrogate`）不過她也明確指出，多數日常改動不需要這套重流程，小而明顯的修改不必開嚴謹模式。她（pstack 使用指南第 1、4 章）

2.6 放手的順序與持續刪減

要把任務交給自動化，她把這一步排在最後，前面有四道前置關卡：你先親手做過一次，或看過 agent 做過一次；agent 手上有你用的工具與訊號；每一關都交證據、不達標就停；你讀過幾次完整過程並把重複失敗變成檢查之後，才讓它自己啟動。她（pstack 使用指南第 7 章）進入無人看管的工作前，要先寫好目標、完成條件、權限和逃生口；完成條件絕不悄悄放寬，跑了多久也從來不是完成條件。她（pstack 使用指南第 7 章）

被派出去的 agent 只認任務裡寫的門檻，不自己加關卡，也不反過來問人；它只在四種情況停下來：部署、刪資料、強制推送共用分支、對這次工作以外的人發訊息。她（pstack `worker.md`）

要注意這是她那個環境的停止清單，並非完整的權限表，你自己的紅線必須另外加進去。整理者到了判斷一個改動能不能上線時，決定者絕不能是寫它的那個 agent。她（pstack 使用指南第 6 章，上線流程）

而且驗的必須是最後交付的那個版本；中途若又改過，舊的驗證就不算數。整理者遇到長時間工作，她要求留下逐筆決定紀錄（決定、理由、證據位置、結果），結束前和實際過程核對一遍，再交另一家模型審查；最後回報結尾要放一段「需要你注意的事」，並寫明是哪個模型審的。她（pstack `/show-me-your-work`）

環境搭起來之後，給 agent 讀的文字還得持續刪減。2026 年 9 月的一筆版本紀錄就寫明，她刪掉 19 條某個模型已不需要的指示；到了 11 月中，又有兩筆紀錄把 skill 文字進一步縮短。她（pstack 版本紀錄）

整理者實測 11 月這次之後，架構 skill 從約 7,700 字元縮成約 900 字元。整理者量測但這是針對她當時的主力模型調出來的結果，換了模型就得自己重新測試。整理者

說到底，每個人都該有一套自己的環境；她建議讓 agent 讀你過去的對話紀錄，找出你反覆糾正的地方，把它們變成檢查或 skill。她（Matt Pocock 訪談 [60:04]～[64:42]）

03.

pstack 是什麼

> **pstack 是她把上面這些做法寫成的一套開源 skill；它是趁她自己手的刀，並不能直接當成標準答案。**

pstack 的原版放在 `github.com/cursor/plugins` 的 `pstack/` 目錄下。同一個儲存庫的 `harness/pstack/claude-code/` 則是官方自動產生的版本，專門給 Claude Code 安裝使用。社群裡雖然另有移植版，但內容可能落後官方。整理者查證，2026-11-18

整套工具由五個部分組成：一個作為嚴謹模式的入口、十幾份 playbook（涵蓋修 bug、做功能、重構、效能、原型、長時間自主等情境，每份各包含幾個步驟）、二十幾條原則、數十個 skill，以及一份十章的使用指南。入口本身做的事很少：先重述目標與能達成它的最小改動，接著挑一份 playbook 把步驟抄進待辦清單，最後在回覆時點名哪條原則改了哪個決定。

她的觀念	對應的 pstack 元件
驗證是瓶頸	建立與維護專案驗證 skill、量測檢查清單、形式驗證 skill、狀態圖
糾正落在環境	`/correct` 、 `/reflect` 、原則「把教訓編進結構」
用程式做計畫	原型 playbook、 `/architect` 、同題多解比較、 `/interrogate`
先懂再改	`/how` 、 `/why` 、 `/recall` 、用自己的話重述
放手與稽核	`worker.md` 、長時間自主 playbook、 `/show-me-your-work` 、skill 盲測 playbook
清掉壞範例	清爛程式、清註解、清 AI 腔文字

用之前要知道的

想拿這套工具來用之前，有三件事得先記在心上。首先，她自己說過不要盲目照用，要改成貼合自己的版本，而且同一類失敗出現兩次才需要加

skill。她（使用指南第 9 章）

談到新手怎麼起步時，她曾給過兩條路：從零開始、只把重複的失敗寫下來（2026-06 貼文）；或是裝上之後先替專案做一個驗證 skill（使用指南第 1 章）。兩條路並不矛盾，重點都在於別一次吞下整包。她；最後一句是整理者

最後要注意，pstack 的預設假設是使用者採用 GitHub PR 流程、有雲端 agent，且手上有多家模型可用；沒有這些條件的人，使用時得挑著用。整理者

04.

落地建議

> **以下十三條是整理者的建議與切法，她本人沒有列過這樣的清單；她自己寫過最接近的版本，是〈Loops You Can Trust〉結尾列出的幾個步驟。**

這份建議設想的讀者，是一個人或小團隊：平常靠 AI agent 做事，但手上沒有無限量的 token。

先看你在哪一階 整理者畫的表；兩端是她講的

0 盯場	1 能驗
一次 1～5 個對話，每個輸出都要看 **做：**規則 0（入口閘）、1（記下重複的失敗）、4（要證據）、7（說目標）；驗證先用現成的指令 **不做：**開更多 agent、上排程	agent 做完會自己跑真東西並附證據 **做：**規則 2（糾正的去處）、3（驗證入口）、5、6（清環境） **不做：**把它附了證據的檢查全部重跑一遍，改成抽樣重跑。程式本身仍要抽樣讀
2 能放	**3 自轉**
同類任務連續多次不用你糾正，結構與檢查已就位 **做：**規則 8～11（測文字、原型、平行、審查） **不做：**在沒有隔離的同一份工作區平行	事件進來自動開工，你事後看 **做：**規則 12（自動化）；持續抽樣、把新失敗變成檢查 **不做：**放寬完成條件、停止抽查

規則 0 入口閘：先問這件事能不能這樣做

【整理者，依據她在 Matt Pocock 訪談 [55:40]～[59:23] 的回答】

動手前先問兩個問題：這件事做錯了退不退得回？對不對能不能用程式驗？她在訪談裡說過，能不能放手要看你能從 agent 拿到多好的驗證；很難用程式驗證的領域很難走到那一步，這題她自己也沒有答案。把這段考量寫成兩個問題與底下的門檻，是整理者的做法。

> **驗收** 只要有任何一題的答案是否定的，就不進規則 12（讓它自己啟動、自己放行）。其餘規則照常使用；若要略過哪一條，必須寫明原因。

規則 1 每次糾正都記；重複的才加文字

先直接下指令，並打開 agent 的工具呼叫紀錄看它卡在哪裡。每次糾正都先記下來，同時問一句：能不能當下就改到讓它做不出來，或者讓機器擋掉？做得到又便宜，就立刻動手。她（AI Engineer 演講 [08:43]）

至於要寫成文字規則或新增 skill，等同一類錯誤第二次出現再補；平時先把問題記進清單，累積幾筆再看是不是同一件事。她（pstack `/correct`、使用指南第 9 章；Matt Pocock 訪談 [47:44]）

例外整理者：如果第一次出錯就看得出會造成不可逆的損害（例如寄錯人、刪錯資料、花錢），當下就加上保護。

> **驗收** 面對每一條現有規則，你都說得出它是從哪一類失敗長出來的。

規則 2 每次糾正，從最強的一層往下找去處

1. 能不能改結構，讓這個錯根本做不出來？
2. 能不能讓機器擋？報錯訊息直接寫明該怎麼改。
3. 能不能用一個行為測試或驗證步驟抓到它？
4. 以上都不行、真的需要判斷時，才寫成一句規則，附一個「錯誤長什麼樣」的例子。

每加進一個新檢查，都先確認它在當初那個真實錯誤上會失敗。一旦錯已經不可能發生，就把對應的文字刪掉。需要破例時，說明寫在違規的位置旁，附上理由與到期日。她

> **驗收** 這兩個數字必須都是零：「被重複違反、卻沒有東西強制」的規則，以及「錯已不可能發生、卻還留著」的文字。需要判斷的問題留著文字是正常的，不要為了讓表好看而硬做檢查。

規則 3 給 agent 一個自己驗證的固定入口

固定入口是一支小工具，把啟動、檢查環境、操作、取證據與收拾乾淨包在一起，再配一張功能地圖。她建議新手一開始就用她的產生器做一個出來。她（pstack 使用指南第 1 章）如果額度有限，可以這樣取捨整理者：先用現成的測試與啟動指令，看到 agent 反覆自己寫臨時腳本時，再把那件事收成固定工具。

動工前有三個成本要先算清楚整理者，依她的說明：功能地圖要持續維護，不維護就會爛掉；驗證工具必須有自己的隔離環境，不能去操作你正在用的那一份；最後，驗證本身相當吃 token。

> **驗收** 當你用模糊的話指稱一個功能時（她的例子是「左邊側欄那個東西」），agent 不用開口問就能自己走到、重現並截圖回來。

規則 4 完成＝有你能自己檢查的成品

第一則指令就寫明完成條件與要看的證據。agent 回報時要帶上：跑了什麼、輸出是什麼、修改前後的對照。每個說法都要附證據，否則就標明是量到的、推論的還是猜的；凡是它自己能跑的檢查，都不丟回給你代勞。她

同時記得，這只證明它做到了你驗的那幾件事，並不證明整件事做得好。她

> **驗收** 多數時候光看回報就能決定收下或退回；但仍要固定抽樣重跑，確認它交出來的證據是真的。

規則 5 把你不想被抄的東西清掉

先加上檢查擋住新增的壞寫法，再把舊的清掉。每件事只留一條鋪好的路讓 agent 走。她至於不該被當成範本、但本身仍有保留價值的參考資料，標清楚或移到 agent 預設讀不到的地方就好，不必刪掉。整理者

> **驗收** agent 預設會讀到的任何一份既有成品，你都願意讓它照抄。

規則 6 不要在成品裡留給下一個 agent 的叮嚀

程式裡解釋「為什麼這裡這樣繞」的註解，往往會被下一個 agent 當成繼續繞路的理由；真正要傳達的限制，應該放進檢查的報錯訊息裡。她保留的例外有四種（見 2.3）。她寫在規則檔裡的事故經過也有同樣的問題，應當一併拿掉。整理者的延伸

要分清楚，這條管的是寫給 agent 看的敘事，不是資料本身的日期與適用條件。「錯誤長什麼樣」的具體例子可以留，「誰在哪天說了什麼」這種過程紀錄不留。整理者

規則 7 指令說目標，不說做法；規則寫短

派工時只給目標、完成檢查、想看的證據、已知事實與真正的限制。遇到需求吵雜時，先叫 agent 重述一遍；不要一開始就講你猜的原因。她

至於規則與 skill，拿不準要不要留的就刪掉，只留會改變決定的句子；換了模型就重新檢查一輪，至於要往哪個方向調整，得靠實測才知道。前半她；後半整理者

規則 8 改了給 agent 讀的文字，重要的要盲測

她的完整做法包含五步：設計像真實需求的任務、讓受測的 agent 不知道自己在被測、從對話紀錄看它實際讀了什麼與做了什麼、請另一家模型當裁判，最後你自己把每份紀錄讀完。她

她的撰寫指引寫的是結構性的改動才加測試案例、主觀調整可以跳過；但她自己也說過，她幾乎每改一次 skill 都會跑一次盲測。她（pstack skill 撰寫指引；Maven 工作坊）這裡取折衷方案：只有重要的、結構性的改動才測，理由是額度限制。整理者的取捨

若想再省一點，可以用省錢版整理者：挑一到兩個真實任務，在改版前後各跑一次，只查兩件事：它有沒有打開那份文字，以及行為有沒有變。沒有第二家模型時，就由自己判並寫明是自己判的。

規則 9 能跑出來看的問題，用原型回答

遇到不確定的岔路，就做兩三個可丟掉的小版本並排比較；跨模組的改動先畫好型別與介面；共用的東西先寫使用說明。她

但要記住，「可逆」只決定值不值得做原型，不等於被授權開工。到底該不該動、往哪個方向走，依然是人的決定。整理者

規則 10 先深後廣

她的推進順序是先深後廣：還不信任的 agent 不拿去平行；有了雲端的獨立環境與可靠的驗證後，她才同時推很多條線。她（〈How I Use Cursor〉；pstack 使用指南第 7 章）整理者把這段原則寫成門檻：在還沒有隔離環境與可信的驗證之前，平行只用在同一題上（例如多個解法、多個審查者、多次重現）；兩樣條件都齊備了，才各開獨立工作區同時推多條線。整理者

被派出去的 agent 必須守住幾條紀律：門檻寫在任務裡、不自己加關卡、可逆的選擇自己做並說明，且只在她列出的四種情況停。你的環境若還有別的紅線（例如花錢），記得自己加進停止清單。前半她；最後一句整理者

規則 11 審查看證據，審的不是寫的那個，而且審最後的版本

查行為問題時，只留下重跑指令就能證明的發現；沒有抓到問題，沒有發現本身就是合法結果。她遇到權限、隱私或不可逆動作這類不適合拿真資料重跑的項目，用條文加上資料流向來核對也算證據。整理者

遇到長時間工作時，必須留逐筆決定紀錄，結束前先和實際過程核對，再交另一家模型審查，並在回報最後列出需要注意的事與審查者名稱。她

規則 12 自動化排最後，每一關都能停

照 2.6 的順序來做：第一件先親手做過或看它做過，再做成可重複的工具與檢查；工作單位切小，在最早、最便宜的階段抓錯；逃生口也要事先寫好。她另外加上一條預算上限：token 或時間到了就立刻停下來回報。整理者

05.

非程式工作怎麼用

> **這套方法能搬到非程式工作的，是「環境、範例、證據」三層；搬不過去的是「自動放行」，因為內容好不好沒辦法用程式驗。**

在這個題目上，她直接談到的有三處 她 ：領域專業比以前更重要，瓶頸變成把意圖講清楚；像醫師、律師這樣的專家只要稍懂技術就有很大優勢（Matt Pocock 訪談 [06:46]～[07:31]）。至於能不能放手，取決於領域能不能被驗證，而在難以驗證的領域該怎麼做，這題她沒有答案（同場 [55:40]～[59:23]）。另外，她的團隊也讓非工程師能夠參與軟體產品的開發（Maven 工作坊 [37:30]）。底下這張表裡的其餘項目，都是整理者對內容工作的類推。

她在程式世界的做法	內容、教材、簡報、營運的對應 整理者	把握
環境是記憶	agent 預設會讀到的舊稿就是範本。退回的版本移到封存處並記一句為什麼退；點過頭的版本放在固定範例位置。好壞成對的範例另外放在樣本夾，需要時才讀	中。文字不像程式會被自動引用，實際效果要自己觀察
lint 與 CI	只檢查查得出真假的事：連結是否活著、引用的出處存不存在、版面有沒有溢出、字數長度與格式是否符合。數字要逐筆核對來源，機器只能幫忙列出來	中
驗證入口	設一個固定入口把成品真的做出來：簡報渲染成圖、文章排成實際頁面、影片輸出畫面。**這只驗版面與格式，不驗內容對不對**	版面：高。內容：不適用
前後對照當證據	回報時附上渲染後的畫面，以及修改前後的對照	高
糾正的四級去處	多數糾正屬於判斷題，會停在第四級，這是正常的。對應的做法是累積成對範例，而不是硬做檢查	中
讀者試讀	找沒看過原稿的模型只讀成品並回答幾個問題。它答錯的地方是值得人去看的線索，不代表真人讀者一定誤讀。**這不是她的盲測**，她的盲測測的是規則改版前後 agent 的行為	中
自動放行、型別系統、形式驗證	沒有對應做法，不要硬套	—

06.

不要照搬的地方

> **她的工作條件和多數人不同，而且這些前提都是她自己親口說出來的。**

首先，她待在一間 token 不限量的公司。她（MTS 訪談 [24:39]）她說前期重構和加檢查都很花 token，要和之後省下的一起算；不是每個人都該照她的方式做。她（Maven 工作坊 [51:12]～[53:20]）

其次，她的自動合併建立在大量前置重構與一套鎖得很死的內部框架上，而那個框架並沒有開源。她至於外頭常提的高產量，她個人一個月做到上千到數千個合併，整個團隊一個月約 11,000 個、回退率約千分之五，這些都是自述（AI Engineer 演講 [13:23]～[14:06]）。而且演講主辦方的摘要註明，後來靠另一個改動修掉的問題並不算在回退裡。

這套體系本身也有明確的邊界。形式驗證目前還在實驗階段。她她也直言這套做法很難到達，並不是裝了 pstack 就會自然擁有。她（Matt Pocock 訪談 [51:19]）而且這樣嚴格受限的環境對人來說很煩，是 agent 在吸收那些麻煩。她最後別忘了，她自己的做法在十個月內就變過好幾輪：2026 年 1 月她還主張先用計畫模式把脈絡備齊，到了 9 月就改成用原型代替抽象計畫；照這個節奏，之後還會再變。前半她；判讀為整理者

07.

這份整理是怎麼做的

> **這份報告由一位主筆讀完全部原文後撰寫，五個不同廠牌的模型做了多輪交叉檢查，最後由主筆每條都回到原文核對才採用。**

這次納入整理的公開材料，包含她本人的 11 篇長文與貼文、5 場演講與訪談的逐字稿（約 3.5 小時），以及 pstack 原始碼與 155 筆版本紀錄。這是整理者找得到並順利取得的公開材料，並不是她的全部作品；沒取得的部分另列在〈出處與各篇重點〉第四節。

檢查共分四輪。第一輪由五個模型各自獨立整理她的觀念，再與主筆的版本比對；結果顯示引句都對得上原文，主筆漏掉的補了回來，先前主筆誤把幾則別人的貼文記成她的話，也在這一輪被抓出來更正。第二輪請五個模型各自攻擊主筆的初稿，共提出 107 條意見。

第三輪把意見去掉署名後發回五家互評，由主筆逐條裁決並改寫。第四輪由五個模型審查改寫後的成品，各抽查數十處篇名、章節與時間點等出處；抽到的錯都已更正，例如把一句話掛錯了篇名，或是把整理者自己量的數字誤標成她說的，但這一輪沒有逐條全查。

來回檢查裡抓到最嚴重的幾個錯包括：把「可以不逐行讀產出」放得太早、把她的盲測寫得比她本人還要嚴格、漏掉「不要問人」的停止條件、把「需要注意的事」在回報裡的位置寫反（她放在結尾）、把渲染截圖當成內容驗證，以及把兩句分開的話併成一句卻標成她說的。最後也要說明這份整理的限制：模型互評通過並不代表正確；幾場演講使用的是機器轉寫的逐字稿，個別字詞可能有誤；還有一場付費訪談與一集 podcast 沒有取得。

想把這套做法帶進團隊？

如果你想把驗證入口、環境整理與 AI 協作規範帶進團隊日常工作，可以找 Cab 做企業內訓、AI 諮詢或陪跑。完整介紹與聯絡方式見書末〈關於整理者〉，或造訪 cablate.com。

附錄

出處與各篇重點

以下每一篇與每一場都是 Lauren Tan（@poteto）的作品。這裡只提供連結，以及整理者用自己的話重述的重點摘要，不放原文全文；需要引用時請回到原文確認。這些重點代表整理者的理解，可能有誤；她沒有參與這份文件，也沒有為內容背書。文中提到的成果數字皆為她自己所說，未經第三方查核。取得日期：2026-11-18。

> 讀法建議：想最快抓到整體輪廓，可先看 A4〈Loops You Can Trust〉和 T1 AI Engineer 演講；想直接照著做，則看 pstack 指南第 6、9 章。

一、長文與貼文（X）

代號	日期	標題
A0a	2026-01-22	deepcoding
A0b	2026-01-23	Coding is Dead, Long Live Engineering
P1	2026-04-25	貼文：平行的價值在深不在廣
A1	2026-05-25	How I Use Cursor
P3	2026-06-12	貼文：start with nothing
A4	2026-06-24	Loops You Can Trust
P2	2026-08-20	貼文：every team needs a gardener
A2	2026-08-31	The Complete Guide to pstack Pt. 1（Verification is all you need）
A3	2026-09-09	The Complete Guide to pstack Pt. 2
P4	2026-09-21	貼文：how i shipped 2,500 PRs（附影片，即 T4）
A5	2026-09-25	Coding is dead, long live Coding
L1	約 2026-08	LinkedIn 貼文：Cloud Agents and Cursor Harness Improvements

A0a deepcoding：把「先寫再說」的順序倒過來，先把脈絡與指引放進去，再讓模型動手寫。理由是模型會依眼前的內容繼續生成，放任它自己亂找，只會把

技術債一起讀進來。這時她還主張先用計畫模式；和九個月後的說法對照，能看出她的做法一直在變。

A0b Coding is Dead, Long Live Engineering：瓶頸已經不在模型本身，而在於你給它的規則、工具、脈絡與下指令的方式。自動化之後，工作並沒有消失，而是轉變成不斷重整這座工廠。

P1：平行的價值在於同一個問題上多路並進，不在同時處理很多不相干的事；限制是人能記住多少。

A1 How I Use Cursor：把 agent 當成一個聰明、容易失憶但很好教的新人來帶。如果沒有嚴謹的做法，它就會只顧著討好你，或者用猜的。瓶頸在驗證；在還不能把一件事從頭到驗證都交給它之前，還談不上自動化；對還不信任的 agent 開平行，只是在浪費。這篇同時宣布 pstack 開源，並介紹她的維護流水線：收回報、重現問題、修正、跑驗證、附上前後影片，最後開草稿。

P3 start with nothing：不要一開始就套用別人的整包設定。從零開始，先看 agent 在哪裡失敗，只把重複出現的失敗寫下來。

A4 Loops You Can Trust（建議先讀）：這篇是整套想法的完整版。她借用生產管理的「限速步驟」概念，找出流程裡最慢的那一步，而這一步通常是等人驗證。她也寫出自己第一個驗證 skill 的由來：agent 自信地講沒有根據的話，她的結論是 agent 要看得到人看的訊號、用得到人用的工具。文中也詳細談了信任曲線、每一關都能叫停、在最早的階段抓錯，以及結尾給出的幾個起步步驟。

P2 gardener：每個團隊都需要一個「園丁」，持續清掉會被 agent 照抄的壞寫法；架構是長出來並持續被修剪的。

A2 pstack 指南 Pt. 1：驗證 skill 是基礎設施，而不只是一個單純的 skill。給 agent 工具，不只是說明文字：用一支小工具把操作與除錯包起來。功能地圖是被具體化的記憶；做技術選型時，也要考慮 agent 好不好驗證。

A3 pstack 指南 Pt. 2：想監督一個寫程式比你強的 agent，靠的是讓它用你能檢查的話來解釋。最常見的兩種失敗，是意圖沒講清楚以及脈絡不夠。她其實也有做計畫，只是用程式來做：寫原型、先寫使用說明、先畫出介面；她不對抽象計畫做對抗審查。

A5 Coding is dead, long live Coding：模型變強之後，以前需要人手把手做的細部指揮就成了多餘，人的工作往上移到釐清意圖與打造環境。

L1：她把工作流程區分為內圈（agent 朝你的意圖動手做事）與外圈（從各處把新資訊帶回來），把兩圈接起來才放得大。

二、演講與訪談

代號	日期	內容	長度
T1	2026-11	AI Engineer 演講：All Code Is Slop (Except Mine)	17 分
T2	2026-08	MTS 訪談：Are Agents About to Replace Software Engineering?	28 分
T3	2026-10-03	Matt Pocock 直播訪談	65 分
T4	2026-09	How I Shipped 2000 PRs Last Month	38 分
T5	2026-08-12	Maven 工作坊：How Cursor Turned AI Agents Into Better Engineers	59 分

T5 在 YouTube 上另有兩支重新剪輯的轉載，內容相同。以下各場的時間點皆以整理者取得的字幕為準；其中 T1 到 T3 為機器轉寫，前後可能差幾秒。

T1 All Code Is Slop (Except Mine)（建議先看）：這場以「倒置的信任金字塔」為主軸，最寬的一層是程式庫設計與靜態檢查，往下是執行期驗證與形式驗證，最窄的是文字規則，人工審查成本最貴。她指出權宜之計配上註解會變成慣例，所以禁掉絕大多數註解，改把指引放進 lint 的報錯訊息。清理的順序是先擋新增、再清舊的、再把驗證做強，內圈可靠了才放大。

- `02:48` 文字那一層只給需要判斷的事
- `08:43` 每次糾正都先問能不能改善程式庫
- `10:03` 功能地圖讓模糊的指稱也找得到
- `15:20` 先 lint、再清舊的、再確保驗證

T2 MTS 訪談：這場多半為產品示範。值得記下來的一點，是她承認自己有不限量的 token，前期重構值不值得是投資報酬問題。

- `24:39` 關於 token 與投資報酬

T3 Matt Pocock 訪談：這是所有場次裡談得最完整的一場。

- `04:53` 她曾是 agent 與除錯工具之間的人肉轉接
- `06:46` 領域專業比以前更重要，瓶頸是把意圖講清楚
- `11:18` 用米其林廚房取代軟體工廠的比喻
- `16:04` 工具箱裡最重要的一個 skill 是驗證

- `20:40` 把決定論的部分抽成工具，skill 只是一層薄說明
- `26:00` 工程師的新工作是整理環境
- `30:01` 內部框架只留一種做法
- `35:18` 內圈與外圈
- `45:51` 大量改動是整理，不是新功能
- `47:44` 發現問題先記進清單，隔幾天看模式
- `49:14` 審查是抽樣；修環境，不修單一 agent
- `51:19` 這很難到達，不是裝了 pstack 就有
- `52:31` 合併後才看；驗證很吃 token，可以調小
- `55:40` 能不能放手取決於領域能不能驗證；難驗證的她沒有答案
- `60:04` 讀自己過去的對話紀錄，把反覆的糾正變成檢查或 skill
- `63:26` skill 會越來越短

T4 How I Shipped 2000 PRs Last Month：這場完整講解信任曲線，從一次盯幾個對話一路走到自動合併。驗證只保證正確；品質靠結構與檢查。

- `05:34` 盯場那一階最難離開
- `12:41` 驗證談的是正確性

T5 Maven 工作坊：可視為同一套內容的教學版，多了用盲測維護 skill、改動要小、token 與投資報酬，以及讓非工程師也能貢獻這幾個段落。

- `08:07` 驗證不保證寫得好，只保證至少寫對
- `30:23` 沒有捷徑

三、pstack 原始碼

- 原版位址：https://github.com/cursor/plugins/tree/main/pstack （整理時的版本：2026-11-16）
- 官方 Claude Code 版：放在同一個儲存庫的 `harness/pstack/claude-code/`
- 使用指南：`pstack/docs/guide/` 共十章。第 2 章談怎麼下指令、第 4 章用程式做設計、第 6 章談驗證、第 7 章談無人看管、第 9 章談改成你自己的版本、第 10 章整理常見陷阱。
- 和本報告最相關的檔案：`skills/correct`（糾正的四級去處）、`skills/poteto-mode/worker.md`（派出去的 agent 的規矩）、`skills/show-me-your-work`（決定紀錄）、`skills/interrogate`（對抗審查）、`skills/create-verification-skill`（建立驗證入口）。

- 她更早的前身專案：`github.com/poteto/brainmaxxing`、`github.com/poteto/noodle`。

四、沒有取得或不採用的

- Behind the Craft Podcast（2026-09-27）完整集為付費內容，沒有取得。
- YC Startup Podcast（2026-11-14）沒有找到可用的音源。
- pstack 指南第三篇截至取得日尚未發表。
- 她長文裡內嵌了三則別人的貼文（@cursor_ai、@systematicls、@ariccio），並非她的話，本報告沒有引用。
- 網路上的二手整理只拿來當成找原文的線索，不當成她的原話。

五、她的數字

以下數字全部是她自己說的，沒有第三方查核：個人產出從一個月約 1,000 個合併（2026-08）升到約 2,500 個（2026-09），再到約 3,000 個；整個團隊一個月約 11,000 個合併、回退率約千分之五（2026-11）。主辦方的摘要註明，後來用另一個改動修掉的問題不算在回退裡。

整理與出版說明

關於整理者

> Cab 在第一線教 AI 落地：從農會推廣人員教到外商工程團隊，自己每天用 Claude Code 工作。提供企業內訓、講座、工作坊，也接 AI 諮詢與陪跑。

連結與服務窗口

官網	https://cablate.com/
文章	https://cablate.com/articles/
電子報	https://cablate.com/newsletter/
企業內訓	https://cablate.com/services/enterprise/
諮詢	https://cablate.com/services/consulting/
陪跑	https://cablate.com/services/coaching/
Threads	https://www.threads.com/@cab_late
GitHub	https://github.com/cablate

整理：Cab｜CabLate（cablate.com）
資料取得與整理日期：2026-11-18。本書內容觀念與開源作品源自 Lauren Tan（@poteto），未經其參與或背書。