לחשוב מחדש על Best Practices של Claude Code

המודלים השתפרו, ומה שהיה בעבר ההגדרה הנכונה מעכב אותם היום. הבקרה שהמפתח עשה ידנית עוברת למודל, ל-harness ולסביבת הריצה.

Lydia Hallie · AI Engineer · 12.11.2026

משך: כ-18 דקות · YouTube

סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

Lydia Hallie מצוות Claude Code ב-Anthropic מעבירה כאן הרצאה קצרה של כ-18 דקות במסגרת AI Engineer. בגלל תקלות טכניות בהרצאה החיה היא הקליטה אותה מהמחשב בבית. השאלה שבמרכז ההרצאה: איך צריך להיראות setup של Claude Code כשהמודלים (Opus 5.5 ומודל חדש שנשמע כמו Fable 5.2) מסוגלים לרוץ שעות, להבין את ה-codebase ולאמת את העבודה של עצמם.

הטענה המרכזית: ה-best practices הישנים לא נעלמו, הם רק עברו למקום אחר. בעבר המפתח הכווין את ה-session, אישר פקודות, אמר לא, בדק את הפלט והתחיל כל session בעצמו. היום כל אחד מהתפקידים האלה עובר למודל, ל-harness (הרשאות ו-hooks) או לסביבה (sandbox, cloud session ו-CI). ההדגמה בנויה כמסלול: מתחילים מ-session ישן בסגנון תחילת השנה ומשדרגים אותו צעד אחר צעד, עד שמגיעים ל-orchestrator שמנהל עשרות sessions.

איך לקרוא את המסמך

- **1. תקציר מנהלים:** הטענות העיקריות עם המספרים, ומה לא נאמר.
- **2. מלולאה לניהול צי:** איך session נראה בתחילת השנה ואיך הוא נראה היום ב-Anthropic.
- **3. ברירות מחדל:** Opus 5.5, ניקוי CLAUDE.md והגדרת auto mode בשפה טבעית.
- **4. בטיחות:** deny rules, sandbox ו-cloud session, ולמה כל שכבה לבדה לא מספיקה.
- **5. אימות:** נקודות ביקורת, hooks עם exit code 2, ו-CI עם review מדורג לפי סיכון.
- **6. Orchestrator:** session ראשי אחד שמפצל את העבודה ל-threads.
- **7. מה עושים מחר:** צעדים מעשיים ושאלות פתוחות.
- **8. מילון מונחים:** המונחים באנגלית שמופיעים במסמך.

הערת מקור המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקפים שלה. התמלול נוצר בזיהוי דיבור (ASR), ולכן שמות ומונחים עלולים להיות משובשים: למשל ClockCode/Cloud במקום Claude Code, ושם המודל Fable ושם הפקודה dr.promptaudit לא אומתו. הציטוטים מועתקים מילה במילה מהתמלול, כולל שיבושי ה-ASR, ונבדקו מול חותמת הזמן שלהם. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

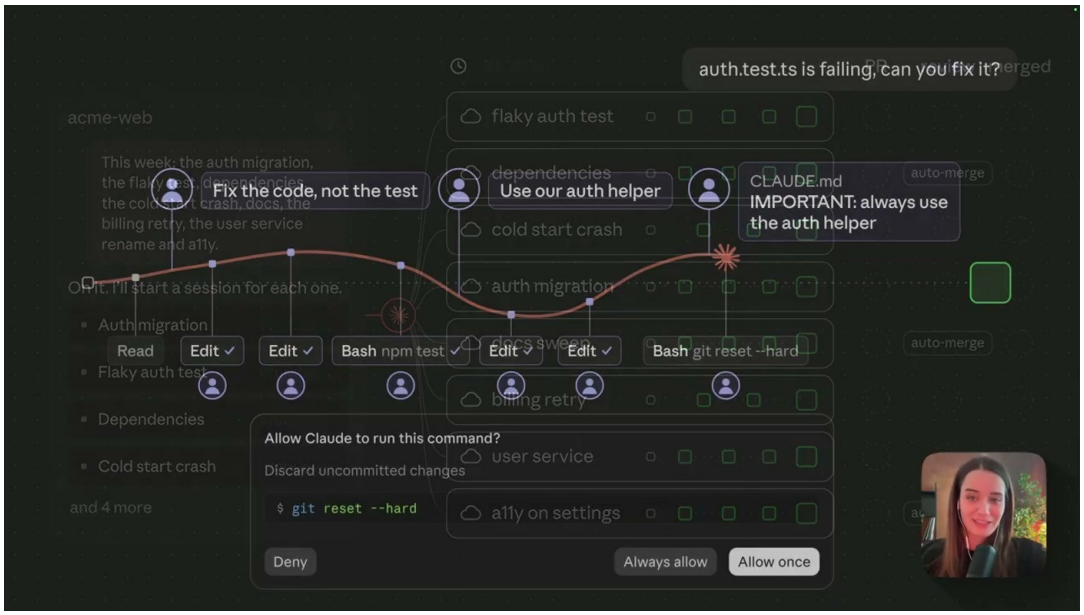
ההרצאה ממפה מחדש את תפקיד המפתח. הוא כבר לא עומד בלולאה של כל פקודה, אלא מנהל עשרות sessions. מה שמחזיק את ההרצאה הוא שתי השאלות שהיא שואלת מהרגע שהאדם יוצא מהלולאה: איך מבטיחים שהקוד איכותי, ואיך מבטיחים שהריצה בטוחה. לכל שאלה היא נותנת שכבות הגנה מוחשיות.

- **הקצב השתנה בסדר גודל:** לפי פוסט של Anthropic מאוקטובר, מספר השינויים ביום עלה מכ-2,500 במאי ליותר מ-30,000 בספטמבר. המפתחים הפכו ל"מנהלים" של עשרות sessions.
- **CLAUDE.md רזה יותר עובד טוב יותר:** הוראות כמו IMPORTANT! כבר לא נחוצות ויכולות להוציא את המודל מהמסלול: יותר tool calls ותוצאה פחות טובה. צריך להשאיר רק מה שאי אפשר לראות בקוד.
- **auto mode הוא ברירת המחדל מאוגוסט:** classifier נפרד מאשר פקודות בטוחות וחוסם פקודות לא ודאיות. אפשר להגדיר אותו ב-settings בשפה טבעית: repos, buckets ו-domains מהימנים.
- **המודל טוב מבני אדם, אבל זו לא ערובה:** במחקר מאוגוסט רק כ-14% מהאנשים זיהו פקודה מסוכנת בעליל, ו-auto mode חסם 89%. ועדיין כ-1 מכל 10 עברה. ערובה אמיתית אפשר לאכוף רק ב-harness.
- **deny rules נבדקות לפני ה-classifier:** אם tool call תואם deny rule, ה-classifier לא רואה אותו בכלל. המבחן מה הופך לחוק: אם Claude יטעה פעם אחת, אפשר להחזיר את זה אחורה?
- **הרשאות לא מגבילות תופעות לוואי:** npm test מריץ קוד של תלויות שלא אתה כתבת. sandbox ברמת מערכת ההפעלה מגביל קבצים, תיקיות ו-hosts, כולל child processes.
- **cloud session סוגר את מה שה-sandbox משאיר פתוח:** מכונה נקייה עם עותק טרי של ה-repo, בלי .env, בלי מפתחות SSH ובלי GitHub token. הרשת מוגבלת כברירת מחדל ל-package registries ול-GitHub.
- **אימות: נקודות ביקורת במקום review של כל השינוי בסוף:** hooks על עריכה, על סיום task ועל סיום session. סקריפט שנכשל צריך להחזיר exit code 2, אחרת ה-task מסומן כהושלם.
- **CI הוא השכבה החשובה ביותר בסקייל:** CI autofix, merge when ready ו-review ראשון על ידי Claude בהקשר נקי. כמה review אנושי יש תלוי ברמת הסיכון: docs בלי אדם, שינוי רגיל עם reviewer אחד, שינוי מסוכן עם שניים.

מה לא נאמר

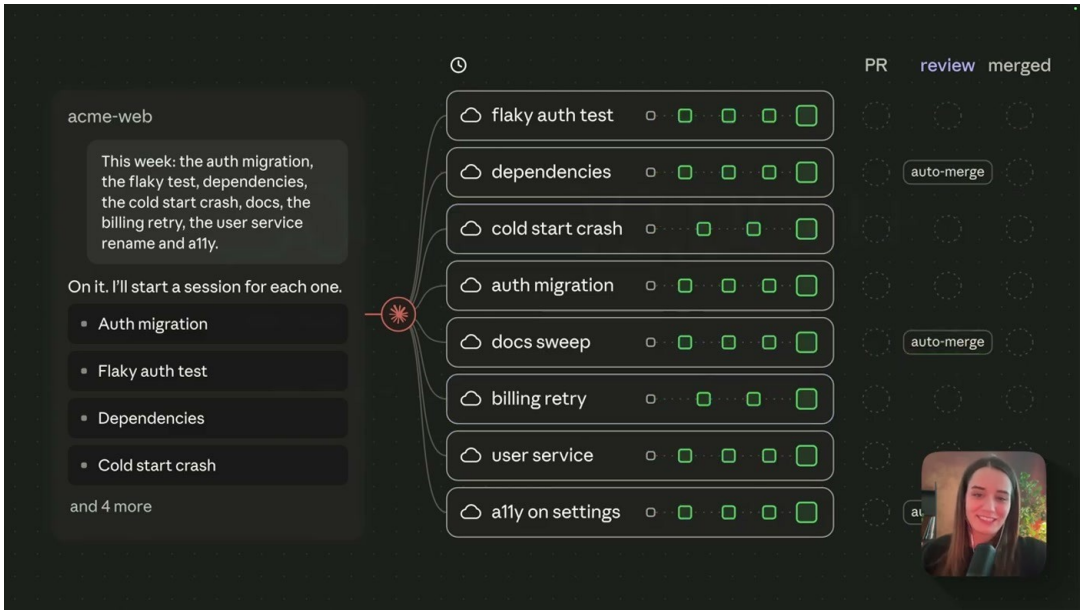
- **אין נתוני איכות:** הקפיצה ל-30,000 שינויים ביום מוצגת בלי שיעור באגים, שיעור reverts או מדד איכות כלשהו.
- **פרטי המחקר חסרים:** לא נאמר כמה פקודות נבדקו, איך הוגדרה פקודה מסוכנת בעליל ומה הייתה רמת ה-false positives של auto mode.
- **אין עלויות:** אין שום התייחסות לעלות של עשרות sessions שרצים שעות, או של orchestrator עם מודל נפרד.
- **Claude Code projects עדיין בבטא:** החזון של orchestrator מוצג בלי מגבלות ידועות ובלי נתוני הצלחה.
- **שם הפקודה לבדיקת הוראות לא ברור:** ה-ASR שמע dr.promptaudit. השם המדויק לא אומת.

2. מלולאה לניהול צי: איך sessions נראו ואיך הם נראים היום

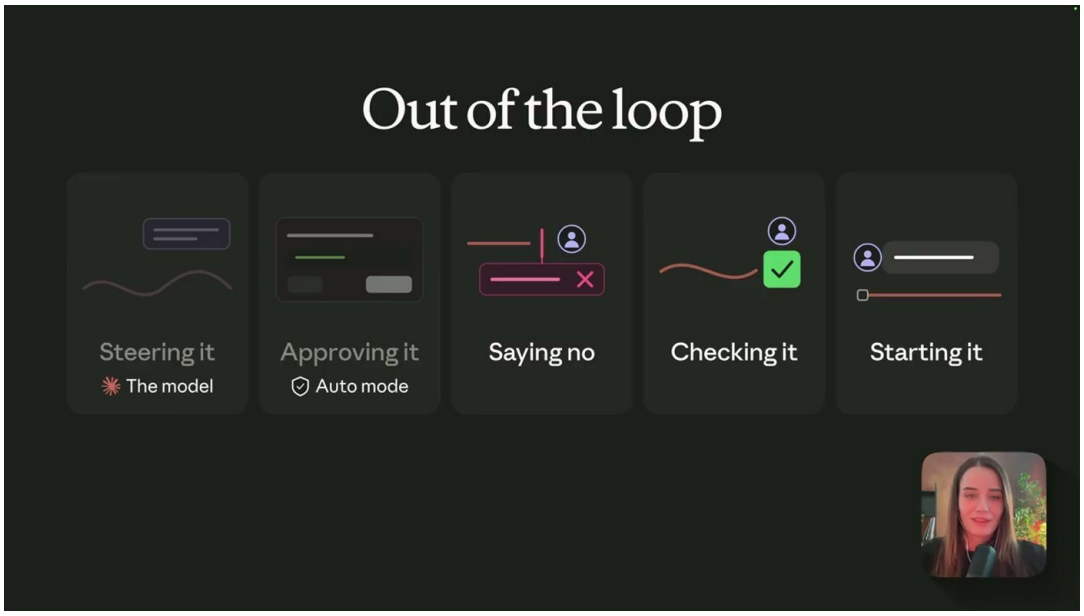


session בסגנון תחילת השנה: תיקוני משתמש חוזרים ("Fix the code, not the test"), הערת IMPORTANT ב-CLAUDE.md ודיאלוג אישור ידני לפקודת git reset --hard

בתחילת השנה עבדו עם Hallie Opus 4.5. מתארת אותו כמודל טוב לזמנו, אבל כזה שעצר מוקדם, רץ זמן קצר ודרש תיקונים ותזכורות חוזרים. בגלל זה קובצי CLAUDE.md התמלאו ב-IMPORTANT וסימני קריאה. auto mode עוד לא היה קיים, ולכן היו רק שתי אפשרויות: לאשר הכל ידנית או להשתמש ב-dangerously skip permissions. האדם היה בלולאה כמעט בכל צעד.



session היום: הוראה אחת ("I'll start a session for each one") מתפצלת לשמונה sessions מקבילים, כמו flaky auth test, auth migration, docs sweep, billing retry ו-auto-merge, כל אחד עם שלבי אימות ו-



Out of the loop: חמשת התפקידים שהמפתח מילא, ומי לוקח אותם: Steering עובר למודל, Approving ל-auto mode, ואחריהם Saying no, Checking it ו-Starting it

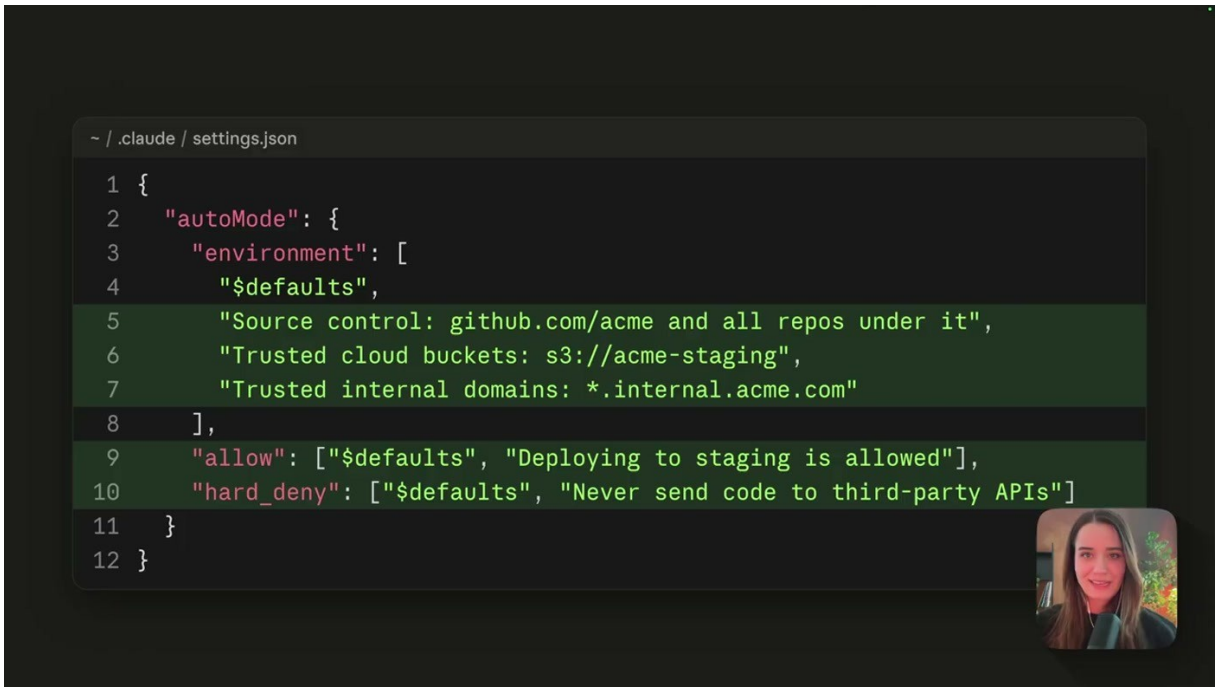
היום, לפחות ב-Anthropic, מריצים עשרות sessions במקביל. הם רצים שעות, מאמתים את התוצאות של עצמם, וחלקם אפילו מתזמרים sessions אחרים. בכל מקום שבו האדם יוצא מהלולאה, משהו אחר צריך להיכנס: המודל, ה-harness או הסביבה. כאן נשאלות שתי השאלות שמלוות את כל ההמשך: האם הקוד איכותי, והאם הוא רץ בבטחה.

3. ברירות מחדל: מודל, CLAUDE.md ו-auto mode

הצעד הראשון הוא פשוט להשתמש בברירות המחדל של היום: Opus 5.5 במקום Opus 4.5, ו-auto mode במקום אישור ידני. Opus 5.5 מבין טוב יותר את הכוונה ואת ה-codebase, ולכן צריך להכוון אותו הרבה פחות באמצע ה-session.

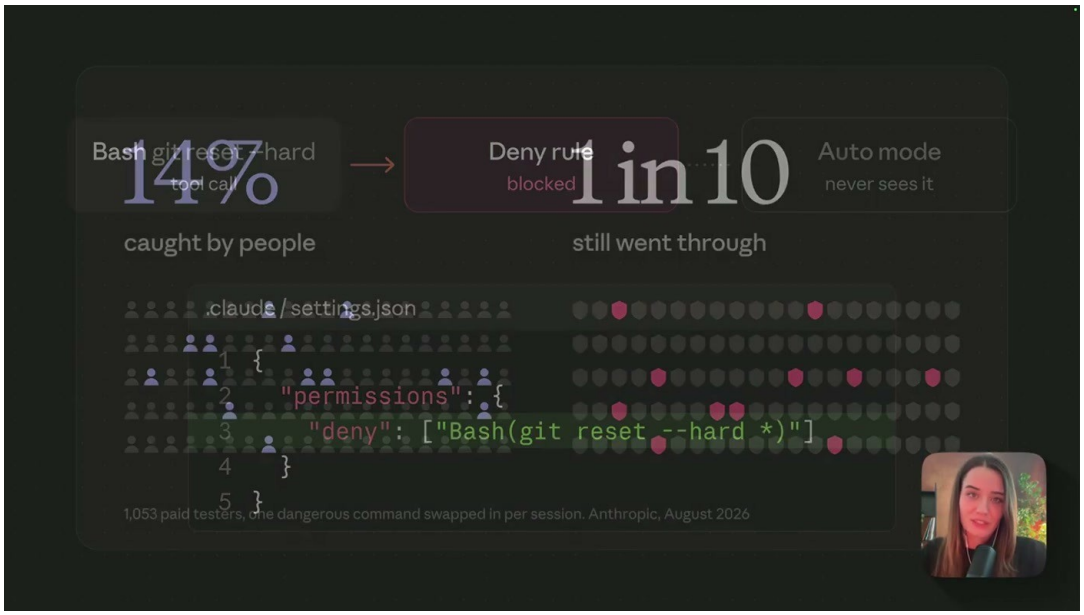
השינוי הזה מחייב גם לנקות את ההוראות. הוראות בסגנון IMPORTANT או never do this! כבר לא נחוצות, ולדבריה הן עלולות להוציא את המודל מהמסלול ולהוביל ליותר tool calls ולתוצאה גרועה יותר. הכלל: "So nowadays, your CloudMD file should only include the things that Cloud cannot see from your code.". דוגמה למה שכן כדאי להשאיר: service שעדיין משתמש במקור legacy שאין לו הפניה ישירה בקוד. מה שלא צריך: איזה stack משתמשים בו ואיפה לחפש. היא מזכירה פקודה (ה-ASR שמע dr.promptaudit, השם לא אומת) שבודקת את CLAUDE.md ואת ה-skills ומסמנת מה המודל החדש כבר לא צריך.

auto mode מריץ classifier נפרד על כמעט כל מה שהוא לא קריאה או עריכה של קבצים בפרויקט. הוא מאשר פקודות בטוחות כמו הרצת טסטים או התקנת תלויות, וחוסם כשהוא לא בטוח. הבעיה: כברירת מחדל הוא סומך רק על התיקייה וה-repo הנוכחיים, ולכן push ל-repo של deploy ייחסם, והאדם חוזר ללולאה. אם עדיין יש הרבה אישורים גם ב-auto mode, הפתרון הוא להגדיר ב-settings בשפה טבעית אילו repos, buckets ו-domains מהימנים, ומה תמיד מותר ומה אסור. הטקסט הזה הוא מה שנשלח ל-classifier.



*קובץ claude/settings.json עם בלוק autoMode: environment מגדיר בשפה טבעית את github.com/acme, את s3://acme-staging ואת *.internal.acme.com. כמהימנים; allow כולל "Deploying to staging is allowed"; hard_deny כולל "Never send code to third-party APIs"*

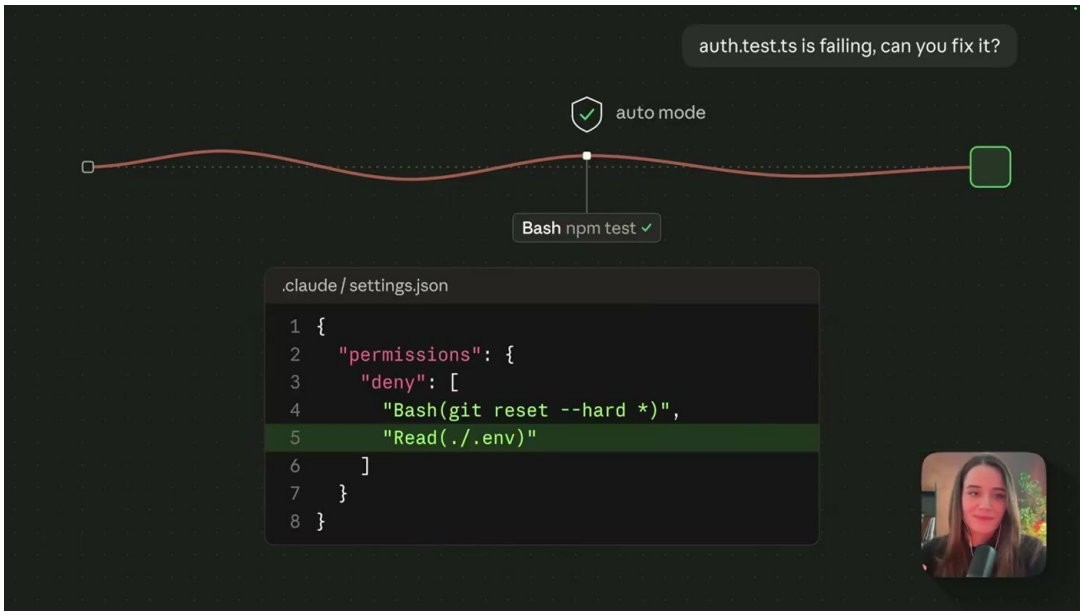
4. בטיחות: deny rules, sandbox ו-cloud session



*תוצאות המחקר: רק 14% מהפקודות המסוכנות נתפסו על ידי אנשים, ו-1 מכל 10 עדיין עברה ב-auto mode. ברקע, deny rule ל-Bash(git reset --hard *) ב-settings.json*

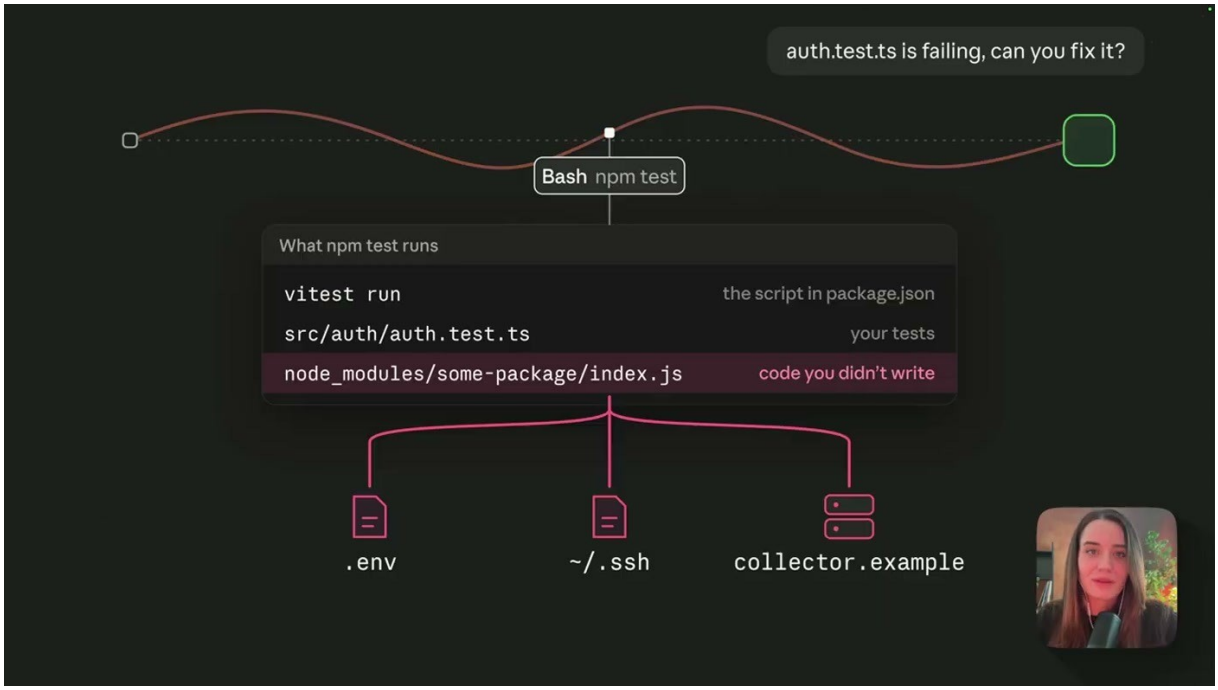
אחרי הצעדים הקודמים יציאה מהלולאה כבר קלה. השאלה הקשה יותר, במילים שלה: "But the harder question now is how do we make sure it's okay that we're not there anymore?". במחקר ש-Anthropic פרסמה באוגוסט רק כ-14% מהאנשים זיהו פקודה מסוכנת בעליל, ו-auto mode חסם 89%. הוא טוב מאיתנו, אבל כ-1 מכל 10 פקודות עדיין עברה. ולכן: "if you do want the guarantee, you can only enforce that in the harness.".

כאן ה-best practice הישן של permissions עדיין רלוונטי. Claude Code בודק deny rules לפני ה-classifier, ו-tool call שתואם deny rule בכלל לא מגיע ל-classifier. כדי להחליט מה צריך להיות חוק היא מציעה מבחן פשוט: "if clot gets this wrong once can i take it back". קוד שגוי אפשר לעשות לו revert, ולכן אפשר להשאיר אותו למודל. מחיקה של שינויים שלא עברו commit, או קריאה של מפתח שנכנס לתמלול, הן דברים שצריך לחסום בחוק.



*deny rules ב-claude/settings.json: Bash(git reset --hard *). ו-Read(./.env). ה-session עובר ב-auto mode ופקודת npm test מאושרת*

Lydia Hallie having deny rules is still the best way to make sure clot actually never does — something



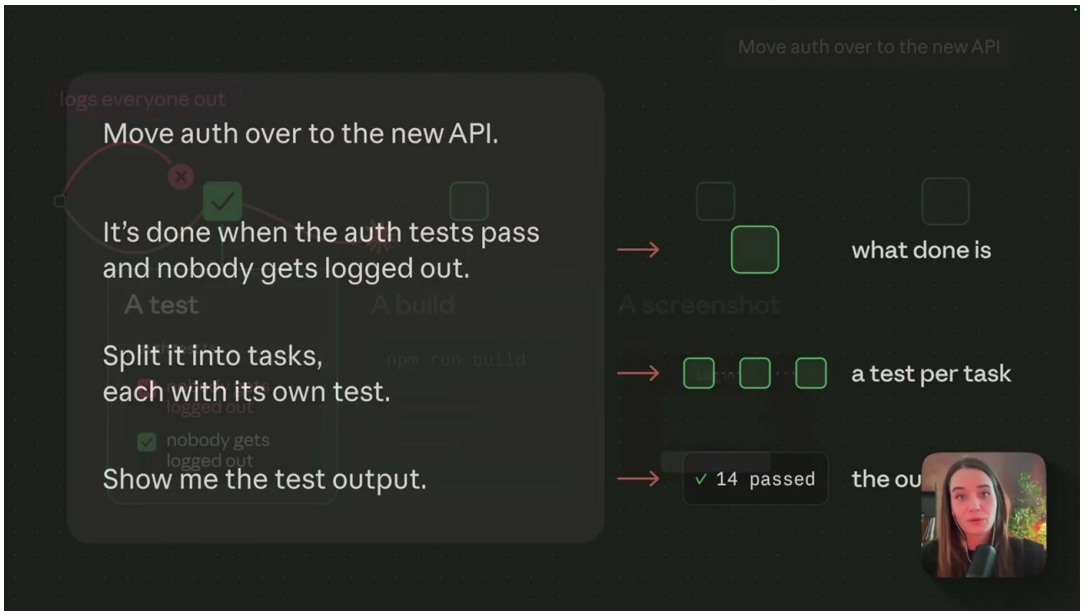
מה npm test מריץ בפועל: vitest run (הסקריפט ב-package.json), src/auth/auth.test.ts (הטסטים שלך) ו-node_modules/some-package/index.js (קוד שלא אתה כתבת), שיכול להגיע ל-.env, ל-~/.ssh ול-collector.example

הבעיה שהשקף מדגים: npm test היא פקודה רגילה ו-auto mode יאשר אותה. אבל היא מריצה קוד של תלויות, ולפעמים גם סקריפט ש-Claude כתב לבדיקה מהירה. אם אחד מהם compromised, הוא יכול לקרוא את .env או את מפתחות ה-SSH ולשלוח אותם לשרת זר. גם אדם בלולאה היה מאשר את זה. "permissions only decide if a tool call runs but they cannot limit what it can reach once it's running."

לכן מגבילים את הסביבה. sandbox מקומי נותן שליטה בקבצים שהפקודה יכולה לקרוא, בתיקיות שהיא יכולה לכתוב אליהן וב-hosts שהיא יכולה להגיע אליהם. מערכת ההפעלה אוכפת את זה, כולל על child processes.
כשה-sandbox פעיל, אותו deny rule עובר למערכת ההפעלה, ותלות שמנסה לפתוח את .env נחסמת. כברירת מחדל ה-sandbox עדיין יכול לקרוא את רוב המכונה, ולכן מומלץ להוסיף חוק גם ל-~/.ssh.

ה-sandbox מכסה רק פקודות shell. הקבצים של Claude, שרתי MCP וה-hooks רצים מחוצה לו, על המכונה שלך. הפתרון המלא הוא cloud session: מכונה משלו עם עותק טרי של ה-repo, שממשיכה לרוץ גם כשהלפטופ סגור. שם אין.env, אין מפתחות SSH ואין GitHub token, כי הם לא ב-git. מפתח API נדרש מוגדר כ-network secret, כך שהוא נמצא בשימוש בלי שיישלח למודל. הגישה לרשת מוגבלת כברירת מחדל ל-package registries ול-GitHub. עם זאת "commit": "this machine is disposable, but the things that leave it aren't" שנדחף או הודעה שנשלחה דרך connector. לכן גם בענן ממשיכים עם auto mode ועם deny rules שנמצאים ב-repo עצמו, כדי לתפוס force push או secret בתוך commit.

5. אימות: נקודות ביקורת, hooks ו-CI

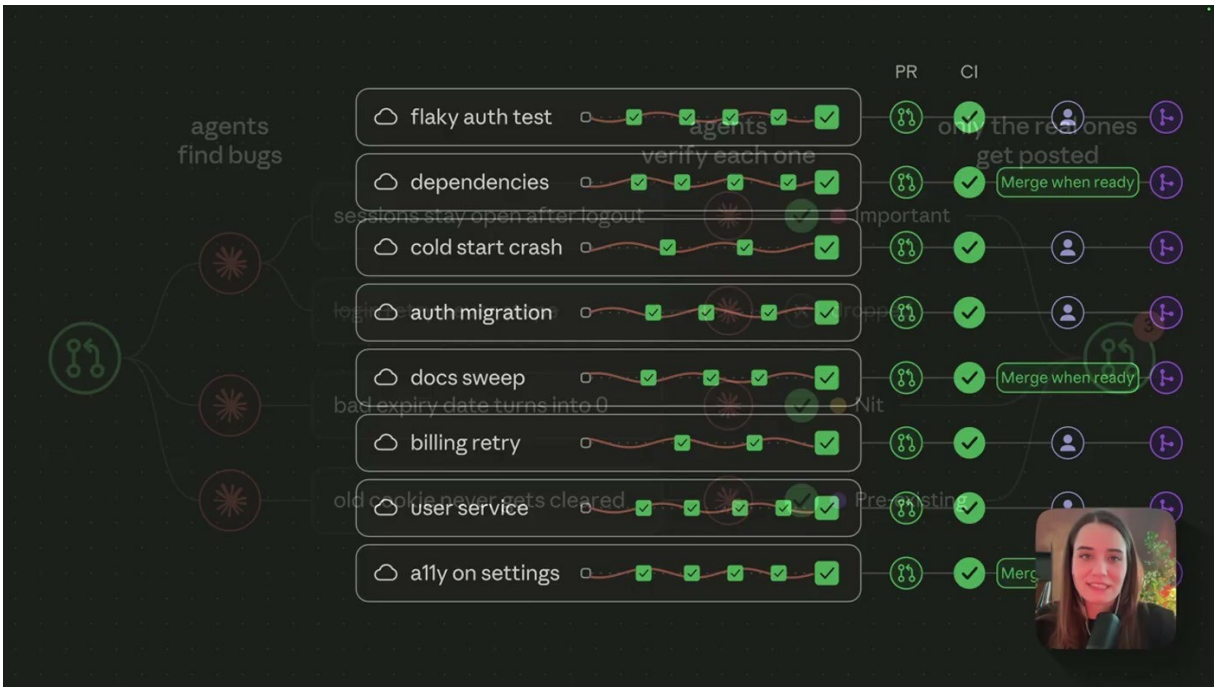


דוגמת prompt לאימות: "Move auth over to the new API. It's done when the auth tests pass and nobody gets logged out. Split it into tasks, each with its own test. Show me the test output" (לצד הסימונים: 14 passed, a test per task, the output, what done is)

כשהאדם היה בלולאה, הוא ידע שהעבודה טובה כי ראה כל שינוי. כש-Claude עובד שעות ברקע, הידיעה הזו נעלמת, ואף אחד לא יקרא כל קובץ בסוף. הדוגמה שלה היא העברת auth ל-API חדש. עם אותו prompt בדיוק, ריצה אחת עשויה להשאיר את ה-API הישן, ריצה אחרת להעביר רק התחברויות חדשות, ושלישית לנתק את כל המשתמשים. כל אחת מהן נראית סבירה. הפתרון הוא נקודות אימות לאורך הדרך: כל דבר שמחזיר pass או fail, למשל טסטים, build או screenshot. אותן בודקים, וזה קצר בהרבה מ-review של השינוי כולו.

יש שלוש דרכים להגדיר את זה. הקלה ביותר היא ב-prompt: מגדירים איך נראה done, מבקשים לפצל ל-tasks שלכל אחד טסט משלו, ומבקשים להראות את הפלט, כי לא היית שם לראות אותו. prompt לא מבטיח שהמודל יעשה את זה, ולכן מה שחייב לרוץ מגדירים כ-hooks: linter על כל עריכת קובץ, הטסטים הרלוונטיים לפני סימון task כהושלם (נגעת ב-auth? טסטי auth צריכים לעבור), וה-test suite המלא לפני סיום ה-session. כשמשהו נכשל, Claude חוזר לתקן עד שהבדיקה עוברת.

מלכודת רוב הכלים יוצאים עם exit code 1 כשטסט נכשל, אבל במקרה כזה Claude Code מסמן את ה-task כהושלם בכל זאת. הסקריפטים של ה-hooks צריכים להחזיר exit code 2. לדבריה זה מה שבאמת שולח את Claude חזרה לתקן.



שמונה sessions מקבילים, כל אחד עם שרשרת בדיקות ירוקות, עמודות PR ו-CI, תיוג Merge when ready, ושכבת agents find bugs שבודקת כל PR

כש-session אחד כבר אמין, פותחים עוד כמה. לכל אחד מכונה משלו, ולכן כמעט לא צריך להתעסק ב-worktrees. בשלב הזה, היא אומרת, "the most important thing to have in place here now is your CI": כל session פותח בסוף PR, וה-CI מריץ את אותן בדיקות על כולם, לא משנה איך ה-session הוגדר. עם CI autofix, Claude רואה בדיקה שנכשלה וחוזר לתקן. עם merge when ready, GitHub ממזג את ה-PR כשכל הבדיקות עוברות. זו לא ברירת המחדל, וזה תלוי ברמת הסיכון.

איך זה עובד אצל הצוות שלה: כל PR עובר קודם review של Claude. לא של ה-session שכתב אותו, אלא של instance עם הקשר נקי, ולצידו קבוצה נפרדת של agents שבודקת כל באג שנמצא. כמה review אנושי נדרש תלוי בסיכון: docs או copy לא צריכים אדם, שינוי מוצר רגיל צריך reviewer אחד, שינוי מסוכן אולי שניים.

6. Orchestrator: session אחד שמנהל את כולם

לדבריה, לשם הדברים הולכים, ו-Anthropic כבר עובדת כך יותר ויותר: גם את ה-sessions כבר לא מתחילים ידנית. Hallie מודה שהיא גרועה ב-context switching. במקום לנהל שמונה sessions, מדברים עם session ראשי אחד, ה-orchestrator. הוא מפצל את העבודה, פותח session לכל task (נקראים threads), עוקב אחרי מה שחוזר ומחזיר לעבודה כל מה שלא באמת גמור. זה זמין היום ב-Claude Code projects, שעדיין בבטא.

ה-orchestrator לא כותב קוד בעצמו, ולכן הוא צריך מודל שטוב בהחלטה מה לעשות ובשיפוט של מה שחוזר. כאן היא מציינת את Fable (שם לפי ה-ASR, לא אומת). ה-threads צריכים מודל שטוב בביצוע, כמו Opus 5.5, ואפשר להגדיר את שני המודלים בנפרד. יתרון נוסף: ב-Claude Code projects ניהול ה-context אוטומטי, ולא צריך לחשוב מתי לעשות compact או clear. הסיכום שלה: כל ה-best practices הישנים עדיין קיימים, רק במקום אחר. המודל מכווין, auto mode מאשר, deny rules והסביבה מגבילים, ו-tests, hooks ו-CI בודקים. נשאר לאדם להתחיל את ה-session הראשון ולעשות review למה שיוצא.

7. מה עושים מחר

- **משימה גדולה מהרגיל:** לפי ההמלצה שלה, "take one thing that's way bigger than you normally give Claude", ולראות איפה ה-session נתקע.
- **לנקות את CLAUDE.md:** להסיר IMPORTANT, סימני קריאה ותיאורי stack, ולהשאיר רק מה שאי אפשר להבין מהקוד.
- **להגדיר auto mode בשפה טבעית:** repos, buckets ו-domains מהימנים, וחוקי allow ו-hard_deny ב-settings.json.

- **deny rules לכל מה שאי אפשר להחזיר אחורה:** git reset --hard, קריאה של.env, ולהכניס אותן ל-repo כדי שילוו גם cloud sessions.
- **להפעיל sandbox או לעבור ל-cloud session:** להגדיר network access, secrets ו-setup script לסביבה.
- **hooks עם** linter **exit code 2:** על עריכה, טסטים ממוקדים על סיום task, suite מלא על סיום session.
- **לבדוק שה-CI תופס את מה שחשוב לכם:** ולשקול CI autofix ו-merge when ready לשינויים בסיכון נמוך.
- **לתעד איפה עדיין צריך אתכם:** לדבריה, כל מקום שבו עדיין נשארתם בלולאה הוא הדבר הבא להעביר ל-Claude.

שאלה פתוחה	למה זה חשוב
מה שיעור ה-false positives של auto mode במחקר?	חסימות מיותרות מחזירות את האדם ללולאה. ה-89% לבדם לא מספרים את כל הסיפור.
האם הקפיצה ל-30,000 שינויים ביום שמרה על איכות?	לא הוצג שום מדד באגים או reverts לצד הנפח.
מה העלות של עשרות sessions שרצים שעות?	הנושא לא הוזכר, ובארגונים הוא משפיע ישירות על אימוץ.
איך מגדירים בפועל רמת סיכון ל-PR?	החלוקה ל-docs, רגיל ומסוכן הוצגה בלי קריטריונים.
מה מגבלות Claude Code projects בבטא?	ה-orchestrator מוצג כחזון, בלי נתוני הצלחה.
מה שם הפקודה המדויק לבדיקת הוראות?	ה-ASR שמע dr.promptaudit. צריך לאמת בתיעוד.

8. מילון מונחים

מונח	משמעות
auto mode	מצב הרשאות שבו classifier נפרד מחליט אם לאשר פקודה, במקום אישור ידני. ברירת המחדל מאוגוסט.
classifier	מודל נפרד שמסווג כל פקודה כבטוחה או לא ודאית, לפי ברירות המחדל וההגדרות בשפה טבעית.
harness	שכבת התוכנה סביב המודל (Claude Code עצמו) שאוכפת הרשאות ומריצה hooks.
deny rule / hard_deny	חוק חסימה ב-settings.json שנבדק לפני ה-classifier, כך שה-classifier לא רואה פקודות שנחסמו.
sandbox	בידוד ברמת מערכת ההפעלה שמגביל קבצים, תיקיות ו-hosts לפקודות shell ולתהליכים שהן מפעילות.
cloud session	session שרץ על מכונה ייעודית בענן עם עותק טרי של ה-repo, גם כשהלפטופ סגור.
network secret	מפתח API שמוגדר בסביבה ונמצא בשימוש בלי שיישלח למודל.
hooks	סקריפטים ש-Claude Code מריץ בנקודות מוגדרות: עריכה, סיום task, סיום session.
exit code 2	קוד יציאה שגורם ל-Claude Code לחזור ולתקן. exit code 1 מסמן את ה-task כהושלם.
CI autofix	יכולת של Claude לזהות בדיקת CI שנכשלה ולחזור לתקן.
merge when ready	מיזוג אוטומטי של PR ב-GitHub כשכל הבדיקות עוברות.
orchestrator / threads	session ראשי שמפצל עבודה ושופט תוצאות, ו-sessions משניים שמבצעים את ה-tasks.
compact / clear	פעולות ידניות לניהול context window, שב-Claude Code projects קורות אוטומטית.
CLAUDE.md	קובץ הוראות פרויקט שנטען ל-session. היום מומלץ לכלול בו רק מה שאי אפשר לראות בקוד.