

Claude Code הלכה למעשה: טיפים וטריקים מהאיש שבנה אותו

Boris מ-Anthropic עובר בסדר מדורג מ-Codebase Q&A, דרך כלים, הקשר ושיתוף
בצוות, ועד SDK והרצה מקבילית, ומסביר למה בחרו ב-CLI ולא ב-IDE

Boris (Anthropic) · הרצאה באירוע Anthropic, ובשקופית מופיע המיתוג Code w/ Claude · הועלה מחדש ב-X על ידי
Komal Prajapati · 2026-10-08
הרצאה ו-Q&A · משך: כ-27 דקות · X
סיכום אוטומטי — AI Briefing (aibriefing.dev)

על המסמך הזה

זו הרצאה מעשית של Boris, member of technical staff ב-Antropic. הוא מציג את עצמו כמי שיצר את Claude Code. ההרצאה ניתנה באירוע של Anthropic, ובשקופיות מופיע המיתוג Code w/ Claude. היא לא עוסקת בתיאוריה או בהיסטוריה. זו רשימה מסודרת של הרגלי עבודה: איך מגדירים את הסביבה, במה כדאי להתחיל, איך מחברים את הכלים של הצוות, איך נותנים ל-agent הקשר, ואיך מגיעים לשימוש מתוך סקריפטים ולהרצה מקבילית. בסוף יש כעשר דקות של שאלות מהקהל. הכיתוב של מי שהעלה את הסרטון ל-X מציג אותו כסדנת prompting של 24 דקות. בפועל זו הרצאה על Claude Code, והיא נמשכת כ-27 דקות.

הטיעון המרכזי פשוט: Claude Code הוא כלי agentic כללי, שלא מכתיב workflow. הערך שמפיקים ממנו תלוי בשלושה דברים: מתחילים בשאלות על הקוד ולא בעריכה, נותנים למודל דרך לבדוק את העבודה של עצמו, ומשקיעים בהקשר משותף (MCP, CLAUDE.md, הרשאות) שנשמר ב-git ומשותף עם כל הצוות. לאורך ההרצאה Boris חוזר על כך שרוב היכולות לא כתובות ב-system prompt אלא נובעות מהמודל עצמו.

איך לקרוא את המסמך

- 1. **תקציר מנהלים:** הטענות המרכזיות עם חותמות זמן, ומה לא נאמר.
- 2. **הגדרה ונקודת פתיחה:** התקנה, פקודות setup, ולמה מתחילים ב-Codebase Q&A.
- 3. **כלים ו-workflows:** עריכת קוד, חיבור CLI ו-MCP של הצוות, תכנון לפני קוד ולולאות משוב.
- 4. **הקשר ושיתוף בצוות:** CLAUDE.md, ההיררכיה slash commands, enterprise/global/project, והרשאות.
- 5. **multi-Claude Keybindings, SDK:** קיצורי מקלדת, p-claude כ-Unix utility, והרצת כמה מופעים במקביל.
- 6. **שאלות מהקהל:** בטיחות של bash, multimodal, למה CLI ולא IDE, ושימוש בתוך Anthropic.
- 7. **מה עושים עכשיו:** צעדים מעשיים וטבלת שאלות פתוחות. אחריו מילון מונחים.

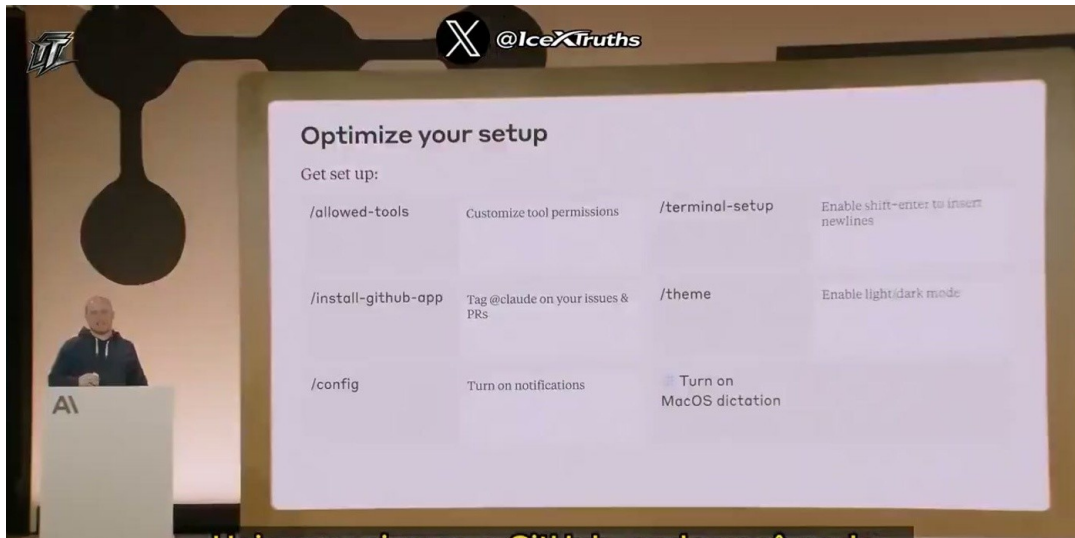
הערות מקור המסמך הופק אוטומטית מהתמלול של ההקלטה ומהשקופיות שנראות בה. התמלול נוצר ב-ASR, ולכן שמות ומונחים עלולים להשתבש. למשל, Claude Code מתומלל לסירוגין כ-QuadCode או Cloud Code, וה-CLI הדמיוני שמופיע בשקופית כ-barley תומלל כ-burly. הציטוטים מועתקים מילה במילה מהתמלול ונבדקו מול חותמת הזמן שלהם. שם המשפחה של הדובר לא נאמר בהקלטה. העותק ב-X כולל כתוביות בפורטוגזית וסימן מים של מעלה צד שלישי. לא נוסף מידע ממקורות חיצוניים.

1. תקציר מנהלים

ההרצאה בנויה כסולם: מתחילים בשאלות על הקוד, ממשיכים לעריכה ולכלים, משם להקשר משותף ובסוף לאוטומציה והרצה מקבילית. בכל שלב שלב Boris נותן prompts מוחשיים ומראה את קבצי התצורה הרלוונטיים. ההמלצות החשובות הן גם הכי פחות טכניות: לבקש תוכנית לפני קוד, לתת למודל דרך לראות את התוצאה, ולכתוב הקשר פעם אחת לכל הצוות.

- **מתחילים ב-Codebase Q&A ולא בעריכה:** כך מלמדים עובדים חדשים ב-Anthropic. לדבריו, onboarding טכני התקצר משבועיים-שלושה ליומיים-שלושה [4:31].
 - **אין אינדוקס ואין העלאה של הקוד:** הקוד נשאר מקומי, אין מסד נתונים מרוחק ואין זמן הקמה. לדבריו גם לא מאמנים מודלים גנרטיביים על הקוד [4:31].
 - **היכולות באות מהמודל ולא מה-system prompt:** חיפוש ב-git history או ביצוע commit, push ו-PR לא מוגדרים בהנחיות. המודל פשוט יודע לעשות אותם [6:02], [9:03].
 - **תוכנית לפני קוד:** בפיצ'ר גדול, הדרך הפשוטה לקבל את מה שרוצים היא לבקש brainstorm ותוכנית ולחכות לאישור. לא צריך plan mode, מספיק לבקש [7:33].
 - **לולאת משוב משפרת דרמטית את התוצאה:** unit tests, צילומי מסך ב-Puppeteer או ב-iOS simulator מאפשרים למודל לחזור על העבודה, ואחרי 2-3 סבבים התוצאה "often it gets it almost perfect" [10:35].
 - **CLAUDE.md קצר עדיף על ארוך:** הקובץ נטען להקשר בתחילת כל session. קובץ ארוך מבזבז הקשר ובדרך כלל לא עוזר [12:07].
 - **הקשר משותף יוצר אפקט רשת:** אם לא יודעים מאיפה להתחיל, כדאי להתחיל ב-shared project context שנשמר ב-Enterprise policy [16:44]. git מאפשרת גם לאשר אוטומטית פקודות וגם לחסום URL-ים [15:13].
 - **claude-p הוא Unix utility:** ה-SDK הוא בדיוק מה ש-Claude Code עצמו משתמש בו. ב-Anthropic מריצים אותו ב-Cl, בטיפול באירועים וב-[21:21], [19:51]. pipelines.
 - **CLI במקום IDE, מתוך הימור על קצב השיפור של המודל:** לדבריו יש סיכוי טוב שעד סוף השנה אנשים כבר לא ישתמשו ב-IDE, ולכן לא משקיעים בשכבות [25:55]. UI. כ-80% מהעובדים הטכניים ב-Anthropic משתמשים בכלי מדי יום [25:55].
- מה לא נאמר**
- **אין מדידה מאחורי המספרים:** הקיצור מ-2-3 שבועות ל-2-3 ימים בזמן ה-onboarding וה-80% שימוש יומי הם הצהרות של הדובר. לא הוצגו נתונים, הגדרות או שיטת מדידה.
 - **אין עלויות:** לא נאמר דבר על צריכת tokens, תמחור או מגבלות שימוש, גם לא בהקשר של כמה sessions במקביל או CI.
 - **אין כשלים:** חוץ מההערה שפיצ'ר גדול לפעמים יוצא לא נכון, אין דוגמאות למה שנשבר. אין הדגמה חיה, ואין השוואה לכלים אחרים.
 - **תחזית בלי ביסוס:** הטענה שעד סוף השנה לא ישתמשו ב-IDE נשענת רק על כך שב-Anthropic רואים את המודל משתפר מקרוב.
 - **זו לא סדנת prompting:** הכיתוב של מי שהעלה את הסרטון מתאר סדנה כללית על prompting ל-Claude. התוכן הוא Claude Code בלבד.

2. הגדרה ונקודת פתיחה: Codebase Q&A

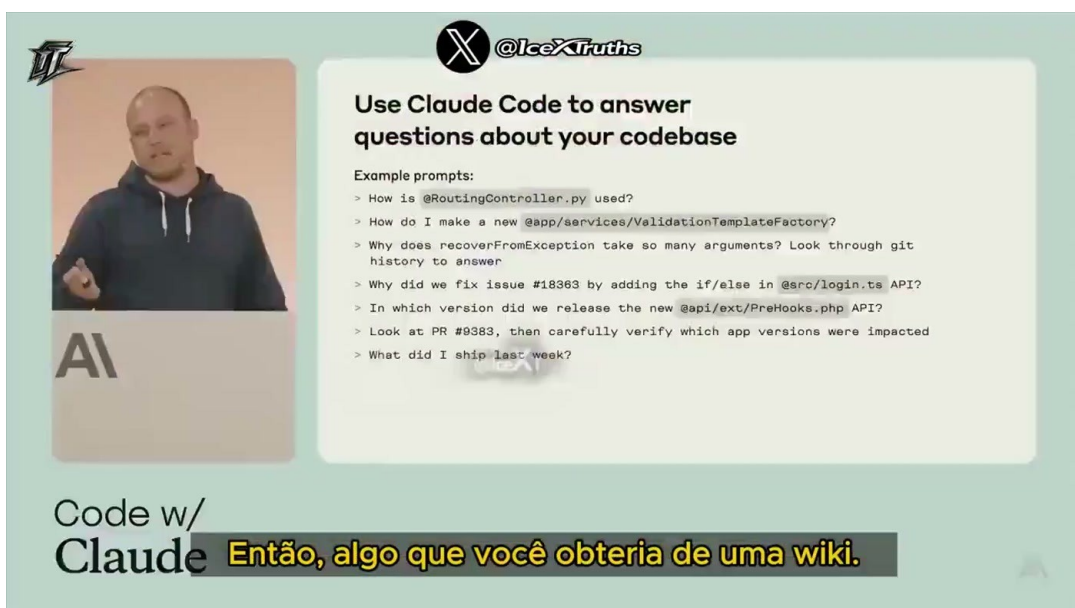


השקופית `/allowed-tools`: `Optimize your setup` להתאמת הרשאות, `/install-github-app` לתיג @claude ב-issues וב-PRs, `/config` להתראות, `/theme`, `/terminal-setup` והפעלת הכתבה קולית ב-macOS

Boris פותח בהתקנה (`@anthropic-ai/claude-code` `npm install -g`, צריך רק Node.js) ומגדיר את הכלי: "It's fully agentic". הוא לא נועד להשלים שורה, אלא לבנות פיצ'רים, לכתוב קבצים שלמים ולתקן באגים. הוא עובד מול כל IDE וכל טרמינל, מקומית, דרך SSH או בתוך `tmux`. כיוון שזה כלי כללי, הוא לא מכון את המשתמש ל-workflow מסוים, וזה גם מה שמקשה על ההתחלה.

צעדי ההגדרה הראשונים: `terminal setup` בשביל `Shift+Enter` לשורה חדשה, `/theme`, התקנת ה-GitHub app שהוכרזה לדבריו באותו יום, והגדרת `allowed tools` כדי לא לאשר שוב ושוב אותן פעולות [1:31], [3:01]. טיפ אישי שלו: רוב ה-prompts שלו לא מוקלדים אלא מוכתבים בהקלטה של macOS. זה גם מוביל ל-prompts מפורטים יותר.

ההמלצה הראשונה מעל הכול היא Codebase Q&A, כלומר לשאול שאלות על הקוד בלי לערוך אותו. השקופית מראה prompts לדוגמה: איך משתמשים במחלקה מסוימת, למה פונקציה מקבלת כל כך הרבה ארגומנטים (עם הנחיה לחפש ב-git history), באיזו גרסה שוחרר API, ו-"What did I ship last week?". את השאלה האחרונה Boris מריץ בעצמו כל יום שני לקראת ה-standup השבועי [6:02]. התשובות הן לא רק חיפוש טקסט: הכלי מחפש דוגמאות שימוש ומסכם את ההיסטוריה של ה-commits ושל ה-issues שהם מקושרים אליהם.



השקופית `Use Claude Code to answer questions about your codebase`: רשימת prompts לדוגמה עם הפניות @ לקבצים, מספרי issues ו-PR, ושאלה על מה שנשלח בשבוע האחרון

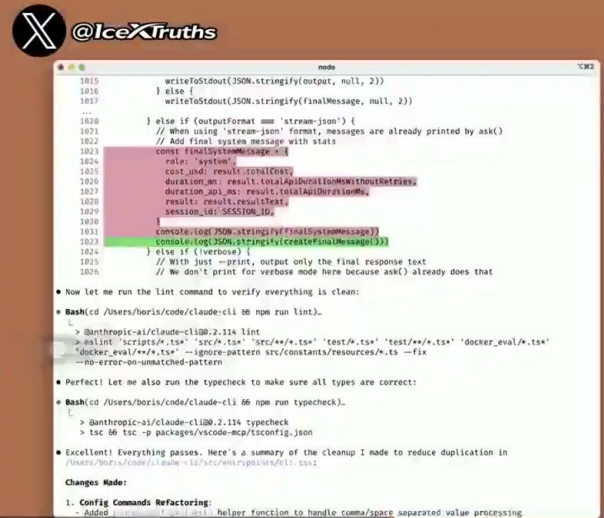
Boris [4:31] And so at Anthropic, onboarding used to take about two or three weeks for technical — hires. It's now about two or three days

3. עריכת קוד, כלים ו-workflows



Use tools to get things done

- Claude Code ships with a dozen tools out of the box. Tools are what makes Claude Code so powerful.
- Built-in tools: bash, file search, file listing, file read and write, web fetch and search, TODOs, sub-agents



Qualquer tipo de agente, sabe?

השקופית Use tools to get things done: הכלי מגיע עם כתיסר כלים מובנים, ביניהם bash, חיפוש ורשימת קבצים, קריאה וכתיבה, web sub-agents ו- fetch, TODOs, search, לצידה מוצג diff של עריכה

בשלב העריכה, לפי Boris, סט הכלים קטן: עריכת קבצים, הרצת bash וחיפוש. המודל משרשר ביניהם בעצמו כדי לחקור את הקוד, לחשוב על כיוונים ובסוף לערוך [7:33]. השקופית מפרטת יותר, "a dozen tools", כולל sub-agents ו- TODOs. הכשל שהוא מתאר: מבקשים פיצ'ר של 3,000 שורות, ולפעמים מה שנבנה הוא "not at all the thing that you want". הפתרון הוא לבקש brainstorm ותוכנית ולחכות לאישור לפני כתיבת קוד: "So just say, 'before you write code, make a plan. That's it.' [9:03] commit, push, הרבה הוא, עוד צירוף שהוא משתמש בו הרבה, הוא

המודל קורא את ה-log git ומסיק ממנו בעצמו את פורמט ה-commit. השלב הבא הוא לחבר את הכלים של הצוות. יש שני סוגים: כלי bash, כלומר CLI פנימי שמסבירים למודל להריץ עם --help כדי ללמוד אותו, וכלי MCP. בשני המקרים, כלי שמשתמשים בו הרבה כדאי לתעד ב-CLAUDE.md כדי שייזכר בין [9:03] sessions. בשקופית מופיעים barley CLI ו-barley_server כדוגמה דמיונית, ו-Boris מדגיש שזה לא כלי אמיתי.

Plug in your team's tools

Tell Claude about your bash tools

> Use the barley CLI to check for error logs in the last training run. Use -h to check how to use it.

Tell Claude about your MCP tools

```
$ claude mcp add barley_server -- node myserver
```

> Use the barley MCP server to check for error logs in the last training run

Mas este é um padrão comum que seguimos na Anthropic.

השקופיות Plug in your team's tools: שמנחה להשתמש ב-barley CLI (עם -h) כדי לבדוק לוגים של שגיאות, והפקודה claude mcp add barley_server -- node myserver לחיבור שרת MCP

Boris מציג שלושה workflows. הראשון הוא חקירה, תכנון ובקשת אישור. השניים האחרים חזקים במיוחד לדבריו, וכולם מבוססים על אותו עיקרון: לתת למודל דרך לבדוק את העבודה שלו. אפשר לתת לו unit tests, integration tests, צילום מסך ב-Puppeteer או צילום מה-iOS simulator. עם mock של UI הוא מגיע בניסיון הראשון ל-"pretty good", ואחרי שניים-שלושה סבבים כמעט למושלם. בלשונו: "So the trick is give it some sort of tool that it can use for feedback to check its work" [10:35]. השקופית Tell Claude to use tools your way מוסיפה prompts כמו בקשה להציע כמה תיקונים ל-issue ולממש את זה שנבחר, זיהוי edge cases שלא מכוסים בבדיקות עם think hard, והרצת שלושה agents במקביל לסיעור מוחות.

4. הקשר: CLAUDE.md, היררכיה ושיתוף עם הצוות

Give Claude more context

Pull in context on demand:

- ~/.claude/commands/foo.md Type /user:foo
- project-root/
 - .claude/commands/foo.md Type /project:foo
 - a/
 - commands/foo.md Type @a
 - CLAUDE.md Type /project:a:foo
 - foo.py Pulled in on demand
- foo.py Type @a/foo.py

```
✓ .claude
  ✓ commands
    ↳ create-release-pr.md
    ↳ fix-github-issue.md
    ↳ get-feedback.md
    ↳ label-github-issues.md
    ↳ lint.md
  {} settings.json
  {} settings.local.json
```

Ou pode ser incluído no seu projeto.

השקופיות Give Claude more context: מתי כל מקור הקשר נטען. וגם ~/.claude/commands ו-foo.py בפרויקט הופכים ל-foo.py או ל-foo.py/project:foo, אזכור @ מוסיף קבצים, ו-CLAUDE.md בתיקיות מקוננות נטען לפי הצורך. מימין דוגמת תיקיית .claude/commands

ככל שהמודל מקבל יותר הקשר, ההחלטות שלו טובות יותר. הדרך הפשוטה ביותר היא קובץ CLAUDE.md בתיקיית השורש של הפרויקט. הוא נטען אוטומטית, ולמעשה נכנס לתור הראשון של המשתמש בכל session. את CLAUDE.md שומרים ב-git ומשתפים עם הצוות. CLAUDE.local.md הוא אישי ולא נשמר ב-git. מה כדאי לכתוב בו: פקודות bash נפוצות, כלי MCP, החלטות ארכיטקטורה, קבצים חשובים, style guide. ההנחיה המפורשת: "Try to keep it pretty short, because if it gets too long, it's just gonna use up a bunch of context, and it's usually not that useful." [12:07] קבצי CLAUDE.md בתיקות משנה נטענים רק כשהמודל עובד בתיקייה הזו. ארגון יכול להחזיק קובץ ברמת enterprise שנטען לכל המשתמשים [13:39].

Slash commands נשמרים ב-`claude/commands`, בתיקיית הבית או בפרויקט. הדוגמה של Boris: התיג האוטומטי של issues ב-repo של Claude Code עצמו הוא slash command שרץ בתוך [13:39] GitHub Action. הוא גם ממליץ להעביר את ההקשר דרך prompt improver, ולשאלו לכל פריט: למי הוא מיועד, האם הוא נטען תמיד או לפי הצורך, ואם הוא משותף או אישי. לדבריו, כוונן נכון [15:13] "will improve performance dramatically".

	Enterprise policy (shared)	Global (just me)	Project (shared)	Project (just me)
Memory	/Library/Application Support/ClaudeCode/CLAUDE.md	~/.claude/CLAUDE.md	CLAUDE.md	CLAUDE.local.md
Slash commands	-	~/.claude/commands/	.claude/commands/	-
Permissions	/Library/Application Support/ClaudeCode/policies.json	~/.claude/settings.json	.claude/settings.json	.claude/settings.local.json
MCP servers	-	claude mcp	.mcp.json	claude mcp

Cada empresa é diferente.

מטריצת Share with your team: לכל אחד מ-Memory, Slash commands, Permissions, MCP servers, הנתב ברמות Enterprise policy, Global, Project (shared) ו-Project (just me). למשל CLAUDE.md מול CLAUDE.local.md, .claude/settings.json מול .settings.local.json, .mcp.json.

Boris עצמו מכנה את המטריצה "kind of an insane matrix". ההיררכיה חלה לא רק על memory אלא גם על הרשאות ועל שרתי MCP. ב-enterprise policy אפשר לאשר אוטומטית פקודת בדיקה שכל העובדים מריצים, וגם לחסום URL כך שאף עובד לא יוכל לעקוף את החסימה [15:13]. קובץ mcp.json. שנשמר ב-repo גורם לכל מי שמריץ את הכלי שם לקבל הצעה להתקין את השרתים. ההמלצה למי שלא יודע מאיפה להתחיל: "I would recommend start with shared project context" [16:44]. ההיגיון הוא אפקט רשת: עבודה קטנה של אדם אחד מועילה לכל הצוות. /memory מציג את כל קבצי ה-memory שנטענים ומאפשר לערוך אותם, ו-# מוסיף פריט לזיכרון שבוחרים. הדוגמה מתוך Anthropic: ב-repo של ה-web וה-apps נשמר שרת Puppeteer MCP, וכל מהנדס יכול להריץ בדיקות end-to-end ולצלם מסכים בלי להתקין אותו בעצמו [16:44], [18:17].

multi-Claude-1 Keybindings, SDK .5



Interlude: Keybindings

Key combos for common tasks:

1 Shift+tab to auto-accept edits

2 # to create a memory

3 ! to enter bash mode

4 @ to add a file/folder to context

5 Esc to cancel

6 Double-esc to jump back in history, --resume to resume

7 ctrl+r for verbose output

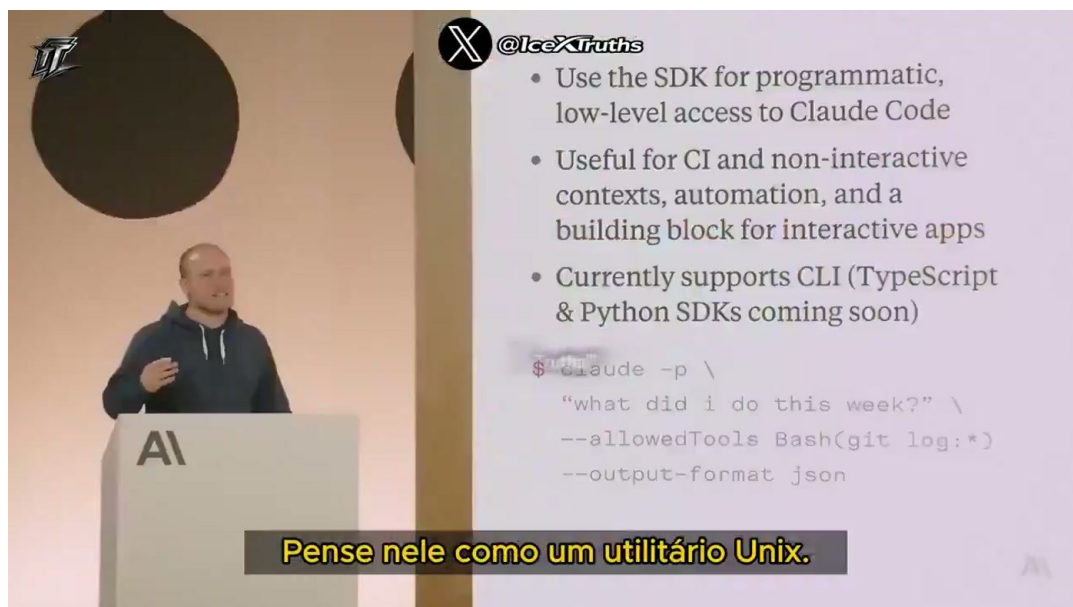
8 /vibe

Se quiser, continue com um 'um' ou '---'.



השקופית Interlude: Keybindings, שמונה קיצורים: Shift+Tab ל- auto-accept, # לזיכרון, ! ל- bash mode, @ להוספת קובץ, Esc לעצירה, Esc כפול לחזרה בהיסטוריה, Ctrl+R ל- verbose output ו- /vibe

קיצור	מה הוא עושה, לפי ההרצאה
Shift+Tab	מצב auto-accept edits: עריכות מאושרות אוטומטית, פקודות bash עדיין דורשות אישור. שימושי כשהמודל על המסלול הנכון או כשהוא עובד על בדיקות [18:17]
#	שומר הנחיה בזיכרון ומכניס אותה אוטומטית ל- CLAUDE.md
!	מריץ פקודת bash מקומית. הפקודה והפלט נכנסים להקשר והמודל רואה אותם בתור הבא
@	מוסיף קובץ או תיקייה להקשר
Esc	עוצר בבטחה בכל רגע בלי לפגוע ב- session, למשל כדי לתקן שורה אחת מתוך הצעה של 20 [19:51]
Esc כפול / --continue / --resume	חזרה אחורה בהיסטוריה, והמשך של session קודם
Ctrl+R	מציג את הפלט המלא, כפי שהמודל רואה אותו בהקשר שלו
/vibe	מופיע בשקופית בלבד. בהרצאה לא הוסבר



השקופית על ה-SDK: גישה תכנותית ו-low-level ל-Claude Code, מתאים ל-CI ולהקשרים לא אינטראקטיביים, כרגע CLI בלבד ו-SDK ל-TypeScript ול-Python בדרך. דוגמה: `claude -p \\'what did i do this week?\' \\' --allowedTools Bash(git log:*) --output-format json`

Boris מסביר שה-SDK הוא בעצם הדגל ק-, והוא "exactly the same SDK" ש-Claude Code עצמו משתמש בו [19:51]. שקופית נוספת מציגה את השכבות: האפליקציה או ה-agent של המשתמש, מעליהם Claude Code SDK, ומתחתיו ה-API של Anthropic, Bedrock או Vertex. נותנים prompt, רשימת allowed tools שיכולה לכלול פקודות bash ספציפיות, ופורמט פלט JSON או streaming JSON. אפשר להזרים אליו קלט ב-pipe (למשל git status) ולחלץ את התוצאה עם jq, לקרוא לוג ענק מ-GCP bucket, או למשוך נתונים מ-Sentry. בלשונו: "It's like a super intelligent Unix utility." [21:21]. הוא מוסיף שבנושא הזה "barely scratched the surface". לסטן ה-SDK המעמיק הוא מפנה לדובר בשם Sid, והשם כאן מבוסס על התמלול בלבד.

הפרק האחרון הוא Boris multi-Claude. מכנה את עצמו "quad normie": לרוב הוא מריץ מופע אחד ועוד כמה טאבים ל-repos שונים. לדבריו, power users בתוך Anthropic ומחוצה לה מחזיקים כמה checkouts של אותו repo, או git worktrees לבידוד, ומתחברים ל-sessions דרך SSH ו-tmux כדי להריץ כמה מופעים במקביל. הוא מציין ש-Anthropic עובדת על דרך פשוטה יותר לעשות את זה [21:21], [22:51].

6. שאלות מהקהל

החלק הקשה ביותר במימוש, לדבריו, היה להפוך הרצת bash לבטוחה: "Bash is inherently pretty dangerous" [22:51] and it can change system state in unexpected ways. מצד שני, אישור ידני של כל פקודה הורס את הפרודוקטיביות, ולא כולם מריצים בתוך Docker. הפתרון שבחרו כולל שלושה חלקים: זיהוי פקודות, read-only static analysis שקובעת אילו פקודות אפשר לשרשר בבטחה, ומערכת הרשאות מדורגת עם allow list ו-block list ברמות שונות [24:23].

לשאלה על תמונות: הכלי "fully multimodal" מהתחלה. אפשר לגרור תמונה לטרמינל, לתת נתיב לקובץ או להדביק. ה-workflow שלו הוא לגרור mock, לבקש מימוש ולתת שרת Puppeteer כדי שהמודל יבדוק את עצמו [24:23]. לשאלה למה CLI ולא IDE הוא נותן שתי סיבות. הראשונה: ב-Anthropic משתמשים ב-VS Code, Zed, Vim ו-Xcode, Emacs, והטרמינל הוא המכנה המשותף. השנייה: הקצב שבו המודל משתפר. לשאלה על ML הוא עונה שחוקרים משתמשים בו מדי יום, כולל כלי ה-notebook לעריכה ולהרצה של notebooks [25:55].

Boris [25:55] And so I think there's a good chance that by the end of the year, people aren't using IDEs anymore. And so we want to get ready for this future, and we want to avoid overinvesting in UI and other layers on top

7. מה עושים עכשיו

- לעשות onboarding דרך שאלות: מי שמכניס צוות לכלי צריך להתחיל ב-Codebase Q&A, ב-git history וב-issues לפני כל עריכה. כך לומדים איפה הגבול בין מה שעובד בניסיון אחד לבין מה שדורש ליווי.
- להגדיר תוכנית לפני קוד כברירת מחדל: בכל משימה בגודל פיצ'ר, לבקש חקירה, תוכנית ובקשת אישור לפני שכותבים שורה.
- לבנות לולאת משוב לכל תחום: בדיקות, צילומי מסך ב-Puppeteer או ב-simulator. כל דבר שמאפשר למודל לראות את התוצאה ולחזור עליה.
- לכתוב CLAUDE.md קצר ולשמור אותו ב-git: פקודות, style guide וקבצים מרכזיים. העדפות אישיות עוברות ל-CLAUDE.local.md.
- לשמור mcp.json. והרשאות משותפות ב-repo: כך כל מהנדס מקבל אותם כלים ואותם אישורים. בארגון כדאי לבדוק enterprise policy גם לחסימות.
- להתחיל להשתמש ב-p- claude ב-pipelines: ב-CI, בניתוח לוגים ובתגובה לאירועים, עם allowedTools מצומצם ופלט JSON.

שאלה פתוחה	למה היא חשובה
איך נמדד הקיצור ב-onboarding משבועות לימים?	זה המספר החזק ביותר בהרצאה, והוא לא מבוסס על מדידה שהוצגה
מה עולה ריצה של כמה sessions במקביל, או של claude p- ב-CI?	לא נאמר דבר על tokens, עלויות או מגבלות
עד כמה static analysis של bash עומדת מול פקודות משורשרות מורכבות?	התשובה תיארה את הגישה, לא את הגבולות או את הכשלים שלה
מה הגודל הנכון ל-CLAUDE.md?	ההנחיה היא לשמור על קובץ קצר, אבל לא ניתן סף או דוגמה מלאה
מה עושה /vibe?	מופיע בשקופית ה-keybindings בלי הסבר
האם התחזית על סוף עידן ה-IDE התממשה?	זו הצהרה עתידית מתוך Anthropic, בלי נתונים שתומכים בה

8. מילון מונחים

מונח	פירוש
Agentic	כלי שמבצע משימה שלמה בעצמו, בוחר כלים ומשרשר צעדים, ולא רק משלים שורה
Codebase Q&A	שאלות שאלות על הקוד, ההיסטוריה וה-issues, בלי לערוך
CLAUDE.md / CLAUDE.local.md	קובץ הקשר שנטען אוטומטית בכל session. הראשון משותף ונשמר ב-git, השני אישי
MCP / .mcp.json	פרוטוקול לחיבור כלים חיצוניים, וקובץ התצורה שלו בפרויקט
Slash commands	פקודות מותאמות שנשמרות כקבצי markdown תחת claude/commands.
Enterprise policy	תצורה ארגונית שנאכפת על כל העובדים: memory, הרשאות, חסימות
Auto-accept edits	מצב שבו עריכות קבצים מאושרות אוטומטית ופקודות bash עדיין דורשות אישור
Claude Code SDK / claude -p	הפעלה לא אינטראקטיבית של הכלי מסקריפט, CI או pipeline, עם פלט JSON
One-shot	משימה שהמודל משלים נכון בניסיון הראשון
REPL / interactive mode	עבודה אינטראקטיבית צעד אחר צעד מול המודל
Puppeteer	כלי לשליטה בדפדפן, משמש להרצת בדיקות ולצילומי מסך שהמודל בודק
Git worktrees	כמה תיקיות עבודה של אותו repo, לבידוד בין מופעים שרצים במקביל
tmux / SSH	ריבוי sessions בטרמינל וחיבור מרוחק, בשביל הרצה מקבילית
Multimodal	יכולת לקבל גם תמונות, למשל mock של UI, ולא רק טקסט