AGENTS 101 - המדריך המלא למתחיל בקלוד קוד ובקודקס

עשרה פרקים בעברית, בגובה העיניים, עם אנלוגיה לכל מושג ודוגמה לכל פיצ'ר. בלי לדעת קוד.

אביתר יצחק · E.Y. AI · גרסה 1.0 · 17.08.2026 ·
evyataryizhak.com/he/guides/agents-101

תוכן העניינים

1 **היסודות: מה זה Claude Code, ולמה זה לא רק למתכנתים**
8 דקות קריאה

2 **זיכרון, קונטקסט ומגבלות שימוש**
10 דקות קריאה

3 **סקילים: המתכונים במגירה**
8 דקות קריאה

4 **סוכנים, סאב-אייג'נטים, פלאגינים והוקים**
6 דקות קריאה

5 **חיבורים: MCP, קונקטורים, API, ארטיפקטים ותוסף הכרום**
6 דקות קריאה

6 **קלוד וקודקס ביחד על אותו פרויקט (Handoff)**
6 דקות קריאה

7 **מוח שני: קבצי md, תיקיות, Obsidian, ומקומי מול דרייב**
8 דקות קריאה

8 **קלוד בענן, בטלפון ובטלגרם**
7 דקות קריאה

9 **טיפים וטריקים עדכניים (יולי-אוגוסט 2026)**
5 דקות קריאה

10 **שימושים: יומיומיים ומתקדמים (מתכונים)**
5 דקות קריאה

למה המדריך הזה

ביקשתי סרטון, הלכתי לשתות שייק, וכשחזרתי קלוד אמר: מוכן.

זה הרגע שבו הבנתי שמשהו השתנה. לא בגלל שהמודל חכם יותר (הוא כן). בגלל שנתתי לו **תיקייה, כללים וכלים** - ובמקום תשובה קיבלתי **תוצאה**. קובץ. בתיקייה. שנשאר גם מחר.

אבל זה לא שקלוד יוצא מהקופסה ישר עם כל היכולות האלה. יש פה שפה חדשה: סקילים, MCP, קונטקסט, סשנים, הוקים, worktrees, Dispatch, רוטינות. הרבה מילים באנגלית שנשמעות מפחידות ובעצם מסתירות רעיונות פשוטים.

המדריך הזה הוא התרגום. בעברית, בגובה העיניים, עם אנלוגיה לכל מושג ודוגמה לכל פיצ'ר. הוא בנוי על סדנת "Keep the Agent Running" שהעברתי באוניברסיטת תל אביב לסטודנטים למחקר, על חצי שנה של עבודה יומיומית עם קלוד קוד וקודקס, ועל בדיקה מול התיעוד הרשמי נכון לאוגוסט 2026.

למי: למי שכבר מדבר עם ChatGPT/Claude ורוצה לעבור מ"תשובות" ל"עבודה". לא צריך לדעת קוד. כן צריך לדעת להגדיר מטרה, לתת הקשר, לבחור גבולות ולבדוק תוצאה.

מה מקבלים בסוף: לא רק הבנה - סביבת עבודה שאפשר לחזור אליה.

איך לקרוא

- **מתחילים?** פרקים 1, 2, 3, 10 לפי הסדר. זה השבוע הראשון.
- **כבר עובדים ורוצים לסדר?** 2 (זיכרון), 6 (קלוד+קודקס), 7 (מוח שני).
- **רוצים קלוד בטלפון?** ישר לפרק 8.
- **רק לדעת מה חדש?** פרק 9.

בכל פרק: אנלוגיה ← מה זה בפועל ← דוגמה קונקרטית ← "מה זה נותן לי?"

המילון הקצר (לפני שצוללים)

מונח	במילה שלי	הפרק
Agent	עובד שמקבל יעד ועושה כמה צעדים עד תוצר	1
Session	פגישת עבודה אחת. שולחן אחד	1
Context window	השולחן. כמה אפשר לפרוש בו-זמנית	2
Usage	התקציב. כמה שעות עבודה יש בחודש	2
CLAUDE.md / AGENTS.md	הדף שמודבק על המסך של העובד	2, 6
Memory	הפנקס האישי של העובד	2
Skill	מתכון במגירה	3

מונח	במילה שלי	הפרק
Sub-agent	עוזר בחדר אחר, שולחן משלו	4
Worktree	עותק של המשרד לכל עובד	4
Plugin	ארגז שמביא כמה חלקים ביחד	4
Hook	חוק בית אוטומטי (מזגן שנדלק לבד)	4
MCP	השקע המשותף	5
Connector	הדלת עם הידית המוכנה	5
API	מפתח לשער	5
Artifact	דף שיוצא מהשיחה ועומד לבד	5
Cloud session	העובד עבר לעבוד ממשרד אחר	8
Remote Control	שלט לעובד שיושב אצלכם	8
Dispatch	מזכירה שמנתבת משימות מהטלפון	8
Channel (Telegram)	ווקי-טוקי לסשן	8
Routine	שעון מעורר עם רשימת משימות	8
Second brain	הארכיון שלכם, ב-md, בגיט	7

הכלל שמעל כולם

הצ'אט זמני. הקבצים זוכרים. מה שחשוב - נכנס לקובץ. מה שחוזר - הופך לסקיל. מה שנעשה - נרשם בלוג. ותמיד: **יש קובץ? יש מקור? מסומנת אי-ודאות? נשמר log?**

יאללה, תפסו מקום.

היסודות: מה זה Claude Code, ולמה זה לא רק למתכנתים

> **"Claude Code לא אומר שצריך לדעת קוד. זו סביבת עבודה עם תיקייה, כלים ותוצרים." (זה המשפט שפתחתי איתו את הסדנה בתל אביב. הוא עדיין נכון.)**

1.1 מתשובה לתוצאה

עד היום ביקשנו מ-AI **תשובה**. "תסביר לי את המאמר". "תכתוב לי פוסט". והתשובה נעלמה בהיסטוריית הצ'אט.

עכשיו אנחנו מבקשים **תוצאה**. "צור טבלה מכל המאמרים בתיקייה, שמור אותה כ-CSV, ותכתוב לי מה חסר." ובסוף יש קובץ. אמיתי. בתיקייה. שאפשר לפתוח מחר.

זה כל ההבדל בין **Chatbot** ל-**Agent**:

- Chatbot: מהיר, נוח, לא צריך כלום. מצוין לשאלה אחת.
- Agent: מקבל יעד, מסתכל מה יש, בוחר כלים, עושה כמה צעדים, בודק שיצא תוצר. עובד גם כשהמשימה נמשכת שלושה ימים ועוברת בין 40 קבצים.

האם Agent זה מודל אחר? לא בהכרח. זו צורת עבודה סביב המודל. אותו קלוד, רק שנתנו לו תיקייה, כלים, וזיכרון.

האנלוגיה: חלון שיחה זוכר שיחה. סביבת עבודה מחזיקה פרויקט. חלון שיחה זה כמו לדבר עם מישהו בטלפון; סביבת עבודה זה שהוא יושב במשרד שלכם, עם גישה לתיקיות.

1.2 ממה בנוי Agent (הציור שנחזור אליו כל המדריך)

```
מטרה  <-  הקשר  <-  תכנית  <-  כלים  <-  הרשאות  <-  תוצר
Goal      Context    Plan       Tools     Permissions  Output
```

חלק	השאלה שהוא עונה עליה	דוגמה
Goal	מה צריך להיות נכון בסוף?	"טבלה עם עמודות: מקור, שיטה, ממצא, מגבלה" (ולא "עזור לי בסקירה")
Context	מה הסוכן יודע ורואה עכשיו?	הקבצים שפתח, ה-CLAUDE.md, ההודעות. "אם הסוכן לא פתח את הקובץ, מבחינתו הוא לא על השולחן"
Plan	מה הוא עומד לעשות לפני שהוא נוגע ב-20 קבצים?	"אקרא X, אשנה Y, אעצור ב-Z"
Tools	במה הוא יכול להשתמש?	קריאת קבצים, טרמינל, דפדפן, יומן גוגל
Permissions	מה מותר לו לעשות בלי לשאול?	לקרוא - כן. למחוק - לשאול

חלק	השאלה שהוא עונה עליה	דוגמה
Output	מה נשאר בסוף שאפשר לפתוח ולבדוק?	קובץ, diff, log. "אם אין קובץ שאפשר לפתוח, המשימה עוד לא הסתיימה"

למה זה חשוב: אם Agent לא עובד טוב, בדרך כלל אחד משלושת החלקים לא ברור. במקום לכתוב פרומפט ארוך יותר - בודקים שישה חלקים.

1.3 מפת המוצרים: איפה אני נמצא בכלל?

אנתרופיק בנתה כמה "חדרים" לאותה צורת עבודה. אפליקציית Claude לדסקטופ (מק/ווינדוס) מכילה שלושה טאבים:

טאב	מרכז העבודה	מתי
Chat	השיחה	שאלה אחת, לחשוב בקול, לבדוק פסקה. אין סיבה לפתוח פרויקט שלם
Cowork	המשימה והתוצר	עבודה ארוכה עם קבצים, קונקטורים ותוצר, בלי טרמינל. פה גר גם **Dispatch** (פרק 8)
Code	התיקייה	שליטה ישירה: קבצים, פקודות, מה קורה על המחשב. **זה קלוד קוד.**

ובנוסף:

- **Claude Code CLI** - אותו מנוע, בטרמינל. מה שהיה קודם. יש כמה דברים שרק שם (agent teams, `--print` לסקריפטים).
- **Claude Code on the web** (claude.ai/code) - סשנים בענן.
- **אפליקציית המובייל** - לראות סשני ענן, Remote Control, Dispatch.
- **Codex** - הגרסה של OpenAI. מצב עבודה אייג'נטי בתוך אפליקציית ChatGPT (הם איחדו את אפליקציית Codex לתוך ChatGPT ביולי), וגם CLI ו-IDE. אותם עקרונות: תיקייה, הוראות, סקילים, כלים, הרשאות, תוצר. (פרק 6)

"צריך שליטה ישירה בקבצים ופקודות? Code. צריך תוצר ידע ארוך בלי טרמינל? Cowork." אין מנצח.

מה נותן לי הטאב Code שאין בצ'אט: תיקייה אמיתית, סשנים מקבילים, סקילים ופלאגינים, MCP, טרמינל מובנה, עורך קבצים, דפדפן פנימי, פריוויו של אפליקציות, ובעיקר: **הכל נשאר**.

1.4 הסשן הראשון (5 דקות)

1. פותחים את אפליקציית Claude ← טאב **Code**.
2. **Environment:** Local (המחשב שלכם). יש גם Cloud ו-SSH, לא עכשיו.
3. **Project folder:** בוחרים תיקייה. **ספציפית**, לא את כל Downloads. "זה הגבול הראשון של המשימה: תיקייה, לא המחשב." (בפרויקט גיט כל סשן חדש מקבל אוטומטית עותק מבודד - worktree - כדי ששני סשנים לא ידרכו אחד על השני.)

4. **Model:** מהתפריט ליד כפתור השליחה. Sonnet ליומיום, Opus או Fable 5 לחשיבה כבדה. אפשר להחליף באמצע.
5. **Permission mode:** מתחילים ב-**Plan** או **Manual**.
6. כותבים את הבקשה הראשונה. ההמלצה שלי לבקשה ראשונה, תמיד:

```
תסתכל על התיקייה. אל תשנה כלום.
תסביר לי במילים פשוטות מה יש כאן, איזה קבצים נראים חשובים,
ומה היית מציע לעשות קודם.
```

"זה פרומפט קטן, אבל הוא משנה את היחסים. אני לא מבקש תשובה. אני מבקש מהסוכן להתמצא בסביבה."

1. אחרי שהוא ענה - `/init`. קלוד כותב CLAUDE.md ראשוני לתיקייה. מעכשיו יש לפרויקט "דף על המסך" (פרק 2).

1.5 מצבי הרשאה - הרשאה היא החלטה, לא חלון שסוגרים מהר

זה הדבר שהכי חשוב לי שתצאו איתו מהפרק. כשקלוד שואל "מותר לי להריץ את הפקודה הזאת?" - זו לא הפרעה. זו החלטה.

מצב	מה קורה	מתי
Plan	קורא, חוקר, מציע תוכנית. **לא משנה כלום.**	תחילת כל משימה לא טריוויאלית. הכי זול, הכי בטוח
Manual (ברירת מחדל)	שואל לפני כל עריכה ופקודה	כשלומדים, או בתיקייה עם דברים אמיתיים
Accept edits	עריכות קבצים - אוטומטי. פקודות אחרות - שואל	אחרי שאישרתם כיוון ויש גיבוי (git)
Auto	קלוד שני בודק כל פעולה ומאשר מה שנראה בטוח	Pro/Max. משימות הפיכות
Bypass permissions	לא שואל כלום	רק סביבה מבודדת, עם גיבוי, בלי מידע רגיש. לא תרגיל חובה

מחליפים: בורר המצב בפינה (באפליקציה) או Shift+Tab (בטרמינל).

הרמזור של החומרים: ירוק - פומבי או סינתטי (תנו חופש). צהוב - אנונימי, באישור. אדום - מזהה, רגיש, של לקוח (Manual, וחושבים פעמיים).

כלל הזהב: "פעולה אוטומטית טובה משאירה אחריה ראיות" - קובץ, diff, log, ציטוט. לא בודקים אם התשובה "נשמעת חכם". בודקים: **יש קובץ? יש מקור? מסומנת אי-ודאות? נשמר log?**

1.6 מה זה סשן, ואיך גורמים לסשנים לדבר

סשן = שיחה שמורה, קשורה לתיקייה, עם ההיסטוריה שלה, נקודות שמירה (checkpoints) של קבצים, והמודל שנבחר. נשמר במחשב שלכם (`~/.claude/projects/...`, כ-30 יום כברירת מחדל).

מה חשוב להבין: כל סשן מתחיל עם שולחן ריק. הידע לא עובר בין סשנים דרך השיחה. הוא עובר דרך **קבצים** (CLAUDE.md, זיכרון, KICKOFF) - פרק 2.

פעולות על סשנים

- **להמשיך:** `/resume` (בוחרים מהרשימה), או בסרגל הצד באפליקציה. בטרמינל `claude --continue` = הסשן האחרון בתיקייה.
- **שם:** `/rename הצעה-לשרון` . שם = ידית לחזור אליה. סשן בלי שם מקבל כותרת אוטומטית.
- **לפצל:** `/branch` - עותק של השיחה לסשן חדש, המקור נשאר. "רוצה לנסות כיוון אחר בלי להרוס".
- **לחזור אחורה:** `/rewind` - גם הקוד וגם השיחה חוזרים לנקודת שמירה.
- **לפנות שולחן:** `/clear` (הישן נשמר), `/compact` (מסכם).
- **שאלת צד:** `/btw` או Cmd+; - שאלה שמשתמשת בקונטקסט של הסשן אבל לא נכנסת אליו.

איך סשנים מדברים אחד עם השני (כדי לא להסביר הכל מההתחלה)

יש ארבע דרכים, מהפשוטה למתוחכמת:

1. **קובץ משותף.** הכי פשוט, הכי אמין. סשן א' כותב `HANDOFF.md` ("מה עשיתי, מה נשאר, איפה הקבצים"), סשן ב' קורא. אצלי: `KICKOFF*.md` בכל פרויקט + `TASKS.md` + `AGENT_LOG.md` (שורה לכל פעולה, כדי שהסשן הבא - וגם קודקס - יידע מה קרה).
2. **CLAUDE.md + זיכרון.** דברים קבועים ← CLAUDE.md. דברים שקלוד למד ← הזיכרון האוטומטי. שני סשנים מאותה תיקייה חולקים את שניהם.
3. **סשנים מדברים ישירות (אפליקציית הדסקטופ).** אפשר לבקש בשפה חופשית: "איזה סשן נגע בהצעת המחיר?", "מה סיכם הסשן של האתר?", "תגיד לסשן של האנליטיקס שהסכמה השתנתה". קלוד רואה את 20 הסשנים האחרונים באפליקציה, קורא מה עשו, ושולח הודעה (שמופיעה שם ככרטיס עם קישור חזרה). לפני שהוא מארכב סשן - שואל אתכם. גם בטרמינל אפשר `@` על שם סשן (2.1.232+).
4. **סאב-אייג'נטים ו-workflows.** בתוך סשן אחד, קלוד שולח עוזרים לחדרים אחרים ומקבל תוצאות. (פרק 4)

מה שאני עושה בפועל: סשן Hub אחד ב-Downloads (ניהול, יומן, משימות קטנות). סשן נפרד לכל פרויקט עמוק, שנפתח מתיקיית הפרויקט כדי שה-KICKOFF ייטען. וכל סשן משמעותי מסתיים בשורה ב-AGENT_LOG.md. ככה גם קלוד וגם קודקס וגם אני-של-מחר יודעים מה קרה.

1.7 שני התווים שצריך להכיר: `/` ו-`@`

`/` = מגירת הכלים. מקלידים לוכסן בתחילת השורה ורואים מה אפשר להפעיל: פקודות מובנות (`/init`, `/clear`, `/model`, `/compact`, `/context`, `/resume`, `/plan`, `/loop`, `/schedule`), סקילים שלכם (`/ey-ai-proposal`), סקילים מפלאגינים (`/telegram:configure`). "המטרה אינה לזכור פקודות אלא להרגיש איפה הן נמצאות."

`@` **= על מה בדיוק.** במקום "המסמך ההוא", מקלידים `@` ובוחרים קובץ או תיקייה. `@MEETING_SUMMARY.md` מכניס את הקובץ לשולחן. `@תיקייה/` מראה מה יש בה. `@` על שם סאב-אייג'נט מפעיל אותו. `@` על שם סשן אחר. גם גרירה של קובץ לחלון, או הדבקת צילום מסך (Cmd+V) עובדים.

האם `@` נותן הרשאה חדשה? לא. הוא רק מפנה לקובץ שכבר בתחום הגישה של הסשן.

רשימת הפקודות שבאמת משתמשים בהן ביומיום

פקודה	למה
`/init`	לייצר CLAUDE.md לתיקייה חדשה
`/plan`	לעבור למצב תכנון
`/model`	להחליף מודל
`/effort`	רמת מאמץ (low/medium/high/xhigh/max)
`/clear` `/compact` `/context`	ניהול השולחן
`/resume` `/rename` `/branch` `/rewind`	ניהול סשנים
`/memory`	לפתוח את קבצי הזיכרון וה-CLAUDE.md
`/usage`	כמה נשאר מהמכסה
`/mcp` `/plugin` `/skills` `/agents`	ניהול תוספים
`/loop` `/schedule`	לולאות ותזמונים
`/code-review` `/security-review` `/simplify`	ביקורת על שינויים
`/export`	לשמור את השיחה לקובץ טקסט
`/doctor`	כשמשהו לא עובד
`/btw`	שאלת צד

1.8 מה זה נותן לי? (הסיכום שלי לפרק)

- אתם לא לומדים "כלי למתכנתים". אתם לומדים לתת ל-AI **תיקייה, כללים וכלים**, ולקבל **תוצרים** במקום תשובות.
- ששת החלקים (מטרה, הקשר, תכנית, כלים, הרשאות, תוצר) הם רשימת הבדיקה כשמשהו לא עובד.
- Plan קודם. Manual בהתחלה. הרשאה = החלטה.
- הצ'אט זמני. הקבצים זוכרים.
- `/` מה להפעיל, `@` על מה.

זיכרון, קונטקסט ומגבלות שימוש

> **הפרק הזה עונה על השאלה שכולם שואלים אחרי שבוע: "למה קלוד שוכח? ולמה נגמר לי השימוש?"**

2.1 האנלוגיה שמסדרת את הכל: הפרילנסר עם האמנזיה

תדמיינו שיש לכם עובד מבריק. באמת מבריק. אבל כל בוקר הוא מגיע למשרד בלי לזכור שום דבר מאתמול.

מה עושים עם עובד כזה?

1. **מדביקים לו דף על המסך** - "ככה עובדים פה, אלה הכללים, זה מבנה התיקיות". זה **CLAUDE.md**.
2. **נותנים לו פנקס אישי** שהוא כותב בו לעצמו הערות - "אביתר לא אוהב em dash", "הפרויקט הזה משתמש ב-Heebo". זה **הזיכרון האוטומטי (auto-memory)**.
3. **יש לו שולחן עבודה** - עליו הוא פורש את הניירות שהוא עובד עליהם עכשיו. השולחן סופי. כשהוא מתמלא, צריך לפנות. זה **חלון הקונטקסט (context window)**.
4. **יש לו שעות עבודה** - הוא לא עובד 24/7 בחינם. זה **מגבלות השימוש**.

זהו. כל השאר בפרק הזה זה פירוט של ארבעת הדברים האלה. בסדר?

2.2 חלון הקונטקסט - השולחן

מה זה בעצם: כל מה שקלוד "רואה" ברגע נתון. ההודעות שלכם, התשובות שלו, קבצים שהוא קרא, פלט של פקודות שהוא הריץ, ה-CLAUDE.md, הזיכרון. הכל יושב על השולחן. השולחן נמדד ב**טוקנים** (בערך 3/4 מילה באנגלית לטוקן, בעברית קצת פחות יעיל).

כמה גדול השולחן (אוגוסט 2026)

- ברירת מחדל: **200 אלף טוקנים** (Fable 5, Opus, Sonnet).
- יש מצב מורחב של **מיליון טוקנים** למנויי Pro/Max (בטרמינל: `claude --context 1m`; באפליקציה: מהגדרות המודל). זה שולחן ענק, אבל שולחן ענק גם עולה יותר לכל הודעה, כי כל פעם שאתם שולחים משהו קלוד "קורא מחדש" את כל השולחן.

מה יושב על השולחן עוד לפני שאמרתם מילה

- ה-system prompt של קלוד קוד (בערך 4 אלף טוקנים, נסתר).
- CLAUDE.md (כל הרמות - גלובלי, פרויקט, תיקייה).
- MEMORY.md (200 השורות הראשונות).
- רשימת הסקילים והכלים (רק השמות + תיאור קצר, לא הגוף).
- שמות כלי ה-MCP.

איך רואים מה תופס מקום: מקלידים `/context`. מקבלים "מפת חום" צבעונית של מי אוכל כמה. אם אתם רואים ששרת MCP אחד לוקח 20 אלף טוקנים ואתם לא משתמשים בו - זה הרגע לכבות אותו.

מה קורה כשהשולחן מתמלא: Compaction

קלוד לא קורס. הוא עושה **דחיסה אוטומטית (auto-compact)**:

1. קודם זורק פלטים ישנים של כלים (התוצאה של `ls` מלפני שעה, למשל).
2. אם עדיין צפוף - מסכם את השיחה לפסקה ומתחיל "שולחן נקי" עם הסיכום.

מה שורד דחיסה: CLAUDE.md של שורש הפרויקט (נטען מחדש מהדיסק), העריכות האחרונות, ההודעות האחרונות שלכם. **מה נעלם:** הנחיות שאמרתם בעל פה בתחילת השיחה. זה הלקח החשוב ביותר בפרק: **מה שחשוב שישרוד - נכנס לקובץ, לא לצ'אט.**

פקודות שכדאי להכיר

פקודה	מה עושה	מתי
`/context`	מראה מה תופס מקום	כשמרגישים שקלוד "כבד" או שהחשבון עולה
`/compact`	דחיסה ידנית עכשיו	לפני שעוברים לתת-משימה חדשה בתוך אותו סשן
`/compact תשמור את ההחלטות העיצוב`	דחיסה עם הנחיה מה לשמור	כשיש משהו קריטי בשיחה
`/clear`	שולחן נקי לגמרי (הסשן הישן נשמר, אפשר לחזור אליו)	בין משימות שלא קשורות
`/btw שאלה` (או Cmd+;)	"שאלת צד" שלא נכנסת לשיחה הראשית	"רגע, מה זה הקובץ הזה?" בלי ללכלך את השולחן
`/rewind`	לחזור אחורה לנקודת שמירה (גם קוד וגם שיחה)	קלוד עשה בלגן, רוצים לחזור 3 צעדים

דוגמה קונקרטית: אתם עובדים על הצעת מחיר. באמצע אתם נזכרים שצריך לבדוק משהו באנליטיקס של האתר. אל תעשו את זה באותו סשן. `/clear`, או פשוט סשן חדש. אחרת כל הודעה בהצעת המחיר "סוחבת" את כל נתוני האנליטיקס על השולחן.

2.3 מגבלות שימוש - שעות העבודה

איך זה עובד במנוי Pro/Max (אוגוסט 2026)

- יש **חלון מתגלגל של 5 שעות** ויש **מכסה שבועית**.
- המכסה **משותפת** לקלוד קוד, לצ'אט הרגיל ול-Cowork. כלומר, אם שרפתם את הבוקר על יצירת תמונות בצ'אט - נשאר פחות לקלוד קוד.
- החלפת מודל (`/model`) **לא** מאפסת את המכסה. זו אותה מכסה.
- כשנגמר: הודעה "hit your 5-hour limit" ותאריך איפוס. אפשר להפעיל **usage credits** (תשלום לפי שימוש מעבר למנוי) מ-`/usage-credits`.

איך בודקים כמה נשאר: `/usage` (בטרמינל) או אייקון השימוש בפינת האפליקציה. `/insights` מייצר דוח HTML על ההרגלים שלכם (מה אוכל לכם את המכסה).

טריקים למתוח את המכסה (מהחשוב לפחות)

1. **שולחן נקי בין משימות.** `/clear` או סשן חדש. הסיבה: כל הודעה שולחת את כל השולחן. שולחן של 150 אלף טוקנים = כל "כן, תמשיך" שלכם עולה כמו לקרוא ספר.
2. **מודל לפי משימה.** `/model` - Sonnet לרוב העבודה היומיומית (מהיר, זול במכסה), Opus/Fable לתכנון, ארכיטקטורה, כתיבה רגישה. Haiku לדברים מכניים.
3. **רמת מאמץ.** `/effort low` למשימות פשוטות (פחות "חשיבה", פחות טוקנים). `/effort high` או יותר כשצריך באמת. יש גם `/fast` - מהיר יותר, אבל שימו לב שהוא עולה יותר לטוקן, לא פחות.
4. **Plan mode לפני ביצוע.** (Shift+Tab בטרמינל / בורר המצב באפליקציה). קלוד קורא ומתכנן בלי לערוך. אתם מאשרים תוכנית אחת במקום לתקן 5 ניסיונות. חוסך המון.
5. **סאב-אייג'נטים לעבודה מלוכלכת.** "תריץ את הבדיקות ותביא לי רק את הכשלים" - הפלט הענק נשאר בשולחן של הסאב-אייג'נט, לשולחן שלכם חוזרת פסקה. (פרק 4)
6. **פרומפטים ספציפיים.** "תקן את הבאג בטופס יצירת הקשר ב-contact.tsx" ולא "תשפר את האתר". פרומפט מעורפל = קלוד סורק חצי פרויקט.
7. **לכבות MCP שלא בשימוש.** `/mcp` מראה כמה כל שרת עולה בטוקנים. הפיגמה והבלנדר לא צריכים להיות מחוברים כשאתם כותבים פוסט.
8. **סקילים במקום CLAUDE.md ענק.** הנחיות שרלוונטיות רק לפעמים - עדיף שיהיו סקיל (נטען רק כשצריך) ולא ב-CLAUDE.md (נטען תמיד). (פרק 3)
9. **לא לצאת מסשן פעיל סתם.** יש cache של שעה על השולחן. אם חזרתם תוך שעה - זול. אחרי שעה - קלוד "קורא הכל מחדש" במחיר מלא.
10. **חלוקת עבודה עם קודקס.** יש לכם שני מנויים? קודקס לעבודות המכניות הארוכות, קלוד לחשיבה. (פרק 6)

מה שאני עושה בפועל: סשן Hub אחד ב-Downloads לניהול, וסשן נפרד לכל פרויקט עמוק. ככה שום סשן לא הופך למפלצת של 400 אלף טוקנים.

2.4 CLAUDE.md - הדף שמודבק על המסך

מה זה: קובץ טקסט (Markdown, פרק 7) שקלוד קורא **אוטומטית בתחילת כל סשן**. הוא לא "מוטמע" בקלוד, הוא נטען כהודעה ראשונה. כלומר קלוד קורא ומשתדל לציית. לא 100% אכיפה, אבל קרוב.

איפה הוא גר (מהכללי לספציפי)

רמה	מיקום	למה זה טוב
גלובלי (אני)	`~/.claude/CLAUDE.md`	דברים שנכונים בכל פרויקט: "אני אביתר, לא מתכנת, כותב בעברית, אף פעם לא em dash"
פרויקט	`CLAUDE.md` בשורש התיקייה	"ככה בנוי הפרויקט הזה, אלה הכללים שלו" (עולה לגיט, משותף לצוות)

רמה	מיקום	למה זה טוב
מקומי לפרויקט	`CLAUDE.local.md`	הערות פרטיות שלא עולות לגיט
תת-תיקיות	`תיקייה/CLAUDE.md`	נטען רק כשקלוד נוגע בקבצים שם. מושלם לפרויקטים גדולים
כללים לפי נושא	`.claude/rules/*.md`	קבצי כללים קטנים. אפשר לתת להם `paths:` כדי שייטענו רק לקבצים מסוימים

הכלל שלי: בתיקיית `Downloads/` יש לי CLAUDE.md אחד גדול שהוא "מפת ניווט ראשית" - מי אני, מבנה התיקיות, פרוטוקול פתיחת סשן, איפה כל דבר. בכל תיקיית פרויקט יש KICKOFF.md שקלוד קורא כשהוא נכנס לשם. זה עובד לי מצוין כבר חצי שנה.

דוגמה קונקרטית ל-CLAUDE.md של פרויקט לא-קודי (הצעת מחיר ללקוח):

```
# פרויקט: הצעת מחיר לסטודיו 0304

## מה זה
סדנת "Keep the Agent Running" לשרון קשר: איש קשר. איש בת"א. מיתוג בת לסטודיו.

## קבצים חשובים
- MEETING_SUMMARY.md – סיכום שיחת ההיכרות (מקור האמת)
- Proposal_*.html – ההצעה עצמה. יש להשתמש בסקיל ey-ai-proposal

## כללים
- הצעות מחיר תמיד בגוף שני ("אתם"), אף פעם לא בגוף שלישי
- לא להמציא מסלולי מחיר שלא ביקשתי
- אין em dash. מקף רגיל עם רווחים.
- לפני יצירת קובץ – לבדוק שהוא לא קיים

## מצב נוכחי
[ ] שיחת ספק [ ] נחתמה [ ] הצעה נשלחה
```

טיפים

- **קצר.** מתחת ל-200 שורות. CLAUDE.md ארוך = יותר טוקנים בכל הודעה + פחות ציות. פרטים גדולים - לקובץ נפרד ולקשר אליו.
- **ספציפי.** "תריץ `npm test` לפני commit" ולא "תבדוק את הקוד".
- `/init` מייצר CLAUDE.md התחלתי אוטומטית. `/memory` פותח לעריכה את כל קבצי הזיכרון וה-CLAUDE.md שנטענו.
- **ייבוא:** אפשר לכתוב `@docs/style.md` בתוך CLAUDE.md וזה נטען איתו (עד 4 רמות). לא חוסך טוקנים, רק מסדר.
- **הכי חשוב:** מה שקלוד עשה טוב היום ואתם רוצים גם מחר - תגידו לו "תוסיף את זה ל-CLAUDE.md". זה איך "מחנכים" את קלוד. אחרי חודשיים - "קלוד שלי כבר מחונך".

ואיפה AGENTS.md נכנס לתמונה?

AGENTS.md זה אותו רעיון בדיוק, רק בקודקס (ובכלים אחרים: Cursor, Copilot). פירוט מלא בפרק 6. השורה התחתונה: קלוד קוד קורא CLAUDE.md, קודקס קורא AGENTS.md, ואפשר לגרום לאחד להצביע על השני.

2.5 הזיכרון האוטומטי - הפנקס האישי

מה זה: קלוד כותב לעצמו הערות בין סשנים, בלי שתבקשו. "אביתר מעדיף שהכפתורים יהיו מתחת לטקסט", "הפקודה לבנייה זה `npm run build`", "הפרויקט הזה נכשל כש...".

איפה זה גר

```
~/.claude/projects/<שם-התיקייה-שעבדתם-בה>/memory/
├── MEMORY.md          <- האינדקס. נטען בכל סשן (25 / שורות ראשונות בלבד 200KB)
├── feedback_*.md      <- קובץ לכל עובדה. נקראים לפי הצורך
├── project_*.md
└── reference_*.md
```

איך זה עובד בפועל: בתחילת סשן קלוד רואה את MEMORY.md (רשימת שורות: "כותרת - רמז"). כשמשהו רלוונטי, הוא פותח את הקובץ הספציפי. זה אותו רעיון של "progressive disclosure" כמו בסקילים: אינדקס קטן תמיד, פרטים רק כשצריך.

מה זה נותן לכם: אחרי חודש עבודה, קלוד "יודע" אתכם. לי יש היום 76 קבצי זיכרון בסשן ה-Hub: איך אני אוהב הצעות מחיר, שאסור em dash, מה חשבון הבנק העסקי, איך למשוך אנליטיקס מהאתר. אני לא מסביר את זה יותר.

השאלה הגדולה: זיכרון אחד או זיכרון לכל פרויקט?

התשובה הקצרה: **אתם לא בוחרים, התיקייה בוחרת.** הזיכרון נשמר לפי התיקייה שממנה פתחתם את הסשן. פתחתם מ-`Downloads/` - זיכרון אחד. פתחתם מ-`Downloads/E.Y.AI/לקוחות/אמיר/` - זיכרון אחר, ריק.

אז מה עושים? **ההמלצה שלי, אחרי חצי שנה:**

1. **תיקיית Hub אחת** (אצלי `Downloads/`) שממנה פותחים את רוב הסשנים הכלליים. שם נצבר הזיכרון "מי אתם": העדפות, סגנון, כללי ברזל. זה הזיכרון היקר.
2. **פרויקטים עמוקים** נפתחים מתוך תיקיית הפרויקט (סשן נפרד). שם הזיכרון צובר דברים של הפרויקט: "הבאג הזה קורה כש...", "הלקוח רוצה X".
3. **מה שצריך להיות נכון בשני המקומות** - לא סומכים על הזיכרון. שמים ב-**קובץ** שהסשן ייקרא: CLAUDE.md גלובלי (`~/.claude/CLAUDE.md`), או ב"מוח השני" (פרק 7). הזיכרון האוטומטי הוא של הכלי (קלוד); הקבצים הם שלכם ועוברים גם לקודקס.
4. **מדי פעם לעשות ניקיון.** `/memory` ← פותחים MEMORY.md ← מוחקים שורות ישנות. קלוד עצמו מתריע כשמתקרבים ל-200 שורות.

טיפ: אפשר לומר לקלוד בסוף סשן "תשמור לזיכרון ש...". ואפשר לבקש "מה אתה זוכר עליי?" והוא יקריא. שקיפות מלאה, הכל Markdown.

כיבוי: `/memory` ← toggle, או `"autoMemoryEnabled": false` ב-settings של פרויקט (למשל בפרויקט של לקוח רגיש).

2.6 סיכום הפרק - מה נכנס לאן

סוג מידע	איפה שמים	למה
"ככה עובדים איתי" (קבוע)	`~/.claude/CLAUDE.md`	נטען תמיד, בכל פרויקט
"ככה עובד הפרויקט הזה"	`CLAUDE.md` בתיקיית הפרויקט	נטען כשנכנסים לפרויקט
מה שקלוד למד לבד	auto-memory (הוא מנהל)	אתם רק מנקים מדי פעם
מצב עבודה: מה נעשה, מה הבא	`KICKOFF.md` / `HANDOFF.md` / `TASKS.md`	שורד החלפת סשן ואפילו החלפת כלי
ידע ארוך טווח	המוח השני (פרק 7)	ניטרלי לכלי, לנצח
הנחיה חד-פעמית	הצ'אט	ונעלמת. וזה בסדר.

הפואנטה: **הצ'אט הוא זמני. הקבצים הם הזיכרון.** מי שמפנים את זה - מפסיק להסביר לקלוד מההתחלה.

סקילים: המתכונים במגירה

> **Skill הוא לא עוד פרומפט יפה. יש לו Trigger, קלט, שלבים, פלט ובדיקת סיום.**

3.1 מה זה סקיל, בשפה של מטבח

יש לכם מתכון מוצלח לשקשוקה. פעם ראשונה כתבתם אותו על פתק. מאז, כל פעם שמישהו אומר "שקשוקה" - שולפים את הפתק, לא ממציאים מחדש.

סקיל זה הפתק. **דרך עבודה שאפשר להפעיל שוב.** תיקייה עם קובץ אחד חובה, `SKILL.md`, שמסביר:

- **מתי** להשתמש בו (ה-trigger - "כשאביתר אומר הצעת מחיר / תכין הצעה ל-...")
- **איזה קלט** צריך (סיכום פגישה, שם לקוח)
- **מה השלבים**
- **מה נחשב תוצר תקין** (קובץ HTML בעיצוב שלי, בגוף שני, בלי em dash)

ואופציונלית: תיקיית `scripts/` (קוד שהסקיל מריץ), `references/` (מסמכי עזר שנקראים לפי הצורך), `assets/` (תבניות, לוגו).

האם סקיל חייב לכתוב קוד? לא. רוב הסקילים שלי הם טקסט: איך לכתוב, מה לבדוק, איך זה נראה בסוף.

מה ההבדל מפרומפט שמור? סקיל יכול לכלול קבצים, דוגמאות, בדיקות וסקריפטים - ו**נטען רק כשצריך**.

הסטנדרט פתוח. נקרא Agent Skills (agentskills.io). אותו SKILL.md עובד בקלוד קוד, בקודקס, ב-Cursor ובעוד עשרות כלים. אצלי 20 סקילים משותפים לקלוד ולקודקס דרך symlink. קובץ אחד, שני עובדים.

3.2 איך נראה SKILL.md (דוגמה אמיתית שלי, מקוצרת)

```
---
name: ey-ai-proposal
description: Generate branded E.Y. AI price proposals (הצעות מחיר) in Hebrew RTL HTML.
  Use ANY time the user asks to create/draft a price quote, הצעת מחיר, proposal for AI
  consulting, workshops, training. Trigger on "תכין הצעת מחיר", "צור הצעה", "הצעה ל-
[client]".
  Takes a meeting summary as input, produces a styled HTML the user prints to PDF.
---

# E.Y. AI Proposal

## Inputs
- MEETING_SUMMARY.md או בריף חלקי. אם חסר מחיר - ברירות מחדל מהקטלוג, לא לשאול.

## Workflow
1. קרא את הקטלוג (references/pricing.md) ואת הביו.
2. חקור את אתר הלקוח (WebFetch) לסעיף "עליכם".
3. כתוב בגוף שני. אל תמציא מסלולים שלא ביקשו.
4. צור HTML מהתבנית (assets/template.html). שמור ב-E.Y.AI/לקוחות/<לקוח>/.

## Guardrails
- אין em dash. אין "אשמח לעבוד איתכם". סוגר = תוצאה ללקוח + צעד הבא שאני מוביל.

## Completion check
- הקובץ נפתח בדפדפן? הכותרות בעברית? המחירים תואמים לקטלוג?
```

שימו לב לשלושה דברים:

1. ה-**description** היא הכי חשובה. זה מה שקלוד קורא תמיד (בערך 100 טוקנים לסקיל). לפיה הוא מחליט אם להפעיל. תיאור עמום = הסקיל לא יופעל, או יופעל בטעות.
2. **הגוף** נטען רק כשהסקיל מופעל. אפשר להיות מפורט.
3. **/references** נקראים רק אם צריך. אפס עלות עד אז. זה נקרא progressive disclosure, וזו הסיבה שאפשר להחזיק 50 סקילים בלי שהשולחן יתמלא.

3.3 איך מפעילים

- **אוטומטית:** אתם כותבים "תכין הצעת מחיר לשרון" ← קלוד רואה שה-description מתאים ← טוען את הסקיל.
- **ידנית:** `/ey-ai-proposal` (בקלוד) או `$ey-ai-proposal` (בקודקס). סקיל מפלאגין: `/שם-הפלאגין:שם-הסקיל`.
- **לראות מה יש:** `/skills`, או `+` ← Slash commands באפליקציה. אצלי יש גם סקיל `list-skills` שמדפיס את כל הרשימה עם הטריגרים.

איפה סקילים גרים

מיקום	למי
`~/.claude/skills/<שם>/SKILL.md`	לכל הפרויקטים שלכם (סשנים מקומיים)
`.claude/skills/<שם>/SKILL.md`	לפרויקט הזה (עולה לגיט, משותף לצוות)
בתוך פלאגין	מגיע עם ההתקנה
`~/.codex/skills/` או `.agents/skills/`	קודקס (אותו קובץ, symlink)
סשני ענן	הסקילים שמופעלים בחשבון claude.ai, לא התיקייה המקומית

דגלים ב-frontmatter שכדאי להכיר: `allowed-tools` (אילו כלים מאושרים מראש), `disable-model-invocation: true` (רק ידני, לא אוטומטי - טוב לסקילים "מסוכנים" כמו שליחת מיילים), `context: fork` (רץ בשולחן נפרד).

3.4 סקיל / סאב-אייג'נט / CLAUDE.md / MCP - מתי מה?

זו הטבלה מהסדנה, והיא עדיין הכי טובה שיש לי:

מושג	השאלה שהוא עונה עליה	דוגמה
Agent	מי מבצע ומקבל החלטות בתוך גבולות?	סוכן ביקורת ציטוטים
Skill	איך מבצעים את המשימה שוב באותה דרך?	תהליך הצעת מחיר
Connector / MCP	לאיזה שירות יש גישה?	Google Drive, Figma
Plugin	אילו רכיבים מתקינים יחד כחבילה?	טלגרם = MCP + סקילים + הגדרות

"Agent הוא העובד, Skill הוא שיטת העבודה, Connector הוא הדלת לשירות חיצוני, Plugin הוא הארגז שמביא כמה חלקים ביחד."

ו-CLAUDE.md? זה הדף על הקיר. כללים שנכונים **תמיד** בפרויקט. אם הכלל רלוונטי רק לפעמים ("כשכותבים פוסט ללינקדאין...") - זה סקיל, לא CLAUDE.md. חוסך טוקנים בכל הודעה.

3.5 סקילים שכדאי להכיר

הסקילים שלי (דוגמאות מה-GenOS, לתת לכם רעיונות)

סקיל	מה הוא עושה	הלקח
`ey-ai-proposal`	הצעת מחיר ממותגת מסיכום פגישה	סקיל שמקודד **את הטעם שלי** (מחירים, טון, מה אסור)

סקיל	מה הוא עושה	הלקח
`workshop-to-deliverables`	הקלטת סדנה ← סיכום HTML + מייל ללקוח	תהליך שחוזר כל שבועיים ← סקיל
`carousel-machine`	קרוסלה לאינסטגרם מנושא + התמונה שלי	סקיל עם assets (7 תבניות)
`linkedin-post-skill` , `twitter-thread-skill`, `instagram-reels-creator`	תוכן לפי הפלטפורמה	סקיל לכל "פורמט"
`nano-banana-prompt` , `seedance-2-5` , `hyperframes`	פרומפטים לתמונה / וידאו / וידאו מ-HTML	סקיל = ידע על כלי חיצוני שמשתנה
`hebrew-rtl-best-practices`	RTL נכון באתרים	ידע טכני שנשכח
`link-to-knowledge`	קישור מיוטיוב/X ← ידע מסודר במוח השני	"סקיל קליטה"
`knowledge-to-skill`	קובץ ידע ← SKILL.md מסודר	**מטא-סקיל: סקיל שמייצר סקילים**
`process-to-pack`	תהליך שבניתי ← סקיל + מניפסט + פרומפט + רילס + thread	תהליך ← מוצר
`list-skills`	"מה הסקילים שלי?"	כשיש 50, צריך אינדקס

הדפוס: כל דבר שעשיתי פעמיים וארצה פעם שלישית - הופך לסקיל. וכל דבר שקלוד עשה טוב במיוחד - "תארוז את זה לסקיל".

סקילים רשמיים של אנתרופיק (github.com/anthropics/skills)

סקיל	למה
docx / xlsx / pptx / pdf	לייצר ולערוך וורד, אקסל, פאוורפוינט, PDF כמו שצריך (מגיעים מובנים גם ב-claude.ai)
skill-creator	לבנות סקיל חדש, לשפר קיים, לבדוק שהתיאור מפעיל נכון. **תתחילו מזה**
frontend-design	ממשקים יפים בלי "מראה AI גנרי"
mcp-builder	לבנות שרת MCP משלכם
webapp-testing	לבדוק אפליקציית ווב עם דפדפן
canvas-design, **brand-guidelines**, **internal-comms**	עיצוב, מותג, תקשורת פנים
security-guidance (פלאגין)	סורק קוד שקלוד כותב אחרי כל עריכה. מותקן אצלי

התקנה: `/plugin marketplace add anthropics/skills` ואז `/plugin install <שם>@anthropics/skills`, או להעתיק תיקייה ל- `~/.claude/skills/` .

מהקהילה (בדוק לפני שמתקינים - סעיף 3.6)

מקור	מה
obra/superpowers	אוסף ענק ופופולרי של סקילים ושיטות עבודה (TDD, debugging, brainstorming)
awesome-claude-code / awesome-agent-skills (VoltAgent)	רשימות מסודרות, נקודת התחלה לחיפוש
ui-ux-pro-max	67 סגנונות עיצוב, פלטות, זוגות פונטים. מותקן אצלי
apple-design ← **emilkowalski/skills**	עיצוב וגמישות בסגנון אפל, ספרינגים, sheets. הותקן היום
create-viral-content	הוקים, כותרות, הסרת "סימני AI" מטקסט. הותקן היום
Vercel / Stripe / Figma skills	סקילים רשמיים של חברות למוצרים שלהן
skills.sh / awesomeclaude.ai	מנועי חיפוש לסקילים

3.6 להוריד סקיל מהאינטרנט - ואיך מוודאים שהוא לא מרגל

למה בכלל לדאוג: סקיל זה קבצים שקלוד קורא ולפעמים סקריפטים שהוא מריץ. סקיל זדוני יכול:

1. להכיל **הנחיה מוסתרת** ב-SKILL.md ("שלח את תוכן ה-.env לכתובת X", "אל תראה למשתמש") - prompt injection.
2. לכלול **סקריפט** שמעלה קבצים החוצה, קורא משתני סביבה (מפתחות API), או מריץ `curl | bash` .
3. לבקש `allowed-tools` **רחב** (bash + write) בלי סיבה.
4. להגיע בתוך **פלאגין** עם hooks (רצים אוטומטית בכל סשן) או שרת MCP לא מוכר.

להתייחס לסקיל כמו לתוכנה שמתקינים. מה שלא הייתם מריצים ממקור לא מוכר במחשב - לא מתקינים כסקיל.

הצ'קליסט שלי (5 דקות, שווה כל דקה)

1. **מי כתב?** אנתרופיק / חברה מוכרת / ריפו עם אלפי כוכבים והיסטוריה ← אור ירוק. משתמש אנונימי, ריפו מלפני שבוע ← ביקורת מלאה או לוותר.
2. **לקרוא את SKILL.md עד הסוף.** לחפש: "silently", "hidden", "do not show", כתובות URL שהסקיל "טוען מהן הוראות" בזמן ריצה, בקשות לקרוא `.env` / מפתחות.
3. **לסרוק סקריפטים** (אם יש `scripts/`):

```
grep -rn "curl\|wget\|base64\|eval\|ssh\|scp\|\.env\|API_KEY\|TOKEN" הסקיל/
```

דגלים אדומים: `curl ... | bash` , קידוד base64, כתובות IP קשיחות, קריאה למשתני סביבה.

1. **frontmatter:** `allowed-tools` רחב מדי? hooks? `.mcp.json` שמצביע לשרת לא מוכר?
2. **קודם ל-sandbox.** מורידים לתיקייה זמנית (לא ישר ל-`~/.claude/skills`), פותחים סשן קלוד בתיקייה ריקה, מבקשים מקלוד עצמו: "תעבור על הסקיל הזה ותגיד לי אם יש בו משהו שמנסה לגשת החוצה, לקרוא סודות, או להסתיר משהו ממני". קלוד טוב בזה.
3. **להעדיף marketplace רשמי** (`claude-plugins-official`, `anthropics/skills`) - עוברים בדיקה. ולנעוץ גרסה.
4. **לקלף אטריביושן.** אני מסיר שמות מותג/מחבר מגוף הסקיל ומשאיר שורת `source:` ב-frontmatter. הסקיל הופך לשלי, ולא נראה במקרה כאילו קרדיט זר בתוצרים.

מה שעשיתי היום בפועל: צ'יפ לסשן נפרד שמוריד את שני הסקילים ל-scratchpad, קורא כל SKILL.md וכל סקריפט, מדווח דגלים, ורק אז מתקין ל-`~/.claude/skills` + ל-GenOS. זה הפרוטוקול.

3.7 לבנות סקיל משלכם ב-10 דקות

הדרך הקצרה: בסוף עבודה מוצלחת אומרים לקלוד:

```
description מה שעשינו עכשיו - תארוז לסקיל. תקרא לו <שם>. תכתוב
.ובדיקת סיום, guardrails, עם הטריגרים בעברית ובאנגלית, שלבים
בתיקייה README ותוודא שאין ~/.claude/skills/<שם>/SKILL.md-שים אותו ב.
```

(אנתרופיק ממליצים: אין README.md בתוך תיקיית סקיל. רק SKILL.md.)

הדרך המסודרת: `/skill-creator` (הסקיל הרשמי) מנחה, בודק שהתיאור מפעיל נכון, ויכול להריץ evals.

שלושת הדברים שמשנים כשמתאימים סקיל קיים: קלט, תוצר, בדיקת הצלחה. הכי קל להתחיל מסקיל של מישהו אחר ולשנות את השלושה.

איך יודעים אם סקיל טוב? הוא מגדיר קלט, שלבים, תוצר ובדיקות. אם הוא רק מבטיח "לכתוב מצוין" בלי שיטת עבודה - הוא פרומפט, לא סקיל.

טיפ אחרון: תנו לסקילים לעבוד עבור הקהילה. אצלי כל סקיל חדש הוא גם יחידת לימוד עתידית. "הסקיל האמיתי הוא החזון שלכם" - הסקיל הוא רק איך מבצעים אותו שוב.

סוכנים, סאב-אייגנטים, פלאגינים והוקים

> "Agent מנהל משימה. Sub-agent מקבל חלק קטן ומוגדר ממנה."

4.1 סוכן וסאב-אייג'נט - המנהל והעוזרים

האנלוגיה: אתם מנהלים פרויקט. יש לכם עוזר ראשי (הסשן שלכם). לפעמים הוא שולח מישהו לחדר אחר: "לך תבדוק את כל הציטוטים בביבליוגרפיה ותחזור עם רשימת בעיות". העוזר בחדר האחר עובד על **שולחן משלו** (קונטקסט נפרד), ומחזיר רק את השורה התחתונה.

זה **סאב-אייג'נט**. ולמה זה מעולה:

1. **הפלט הענק נשאר בחדר האחר.** ריצת בדיקות של 3,000 שורות, סריקה של 200 קבצים - לשולחן שלכם חוזרת פסקה.
2. **מקביליות.** "תחקור את התיקון של האתר, את מבנה הנתונים ואת האנליטיקס - במקביל, שלושה סוכנים נפרדים". קורה בו-זמנית.
3. **גבולות.** סוכן ביקורת שמקבל רק Read (בלי Write) לא יכול לשבור כלום.

"תפקיד אחד, מטרה אחת, תוצר אחד." סאב-אייג'נט טוב בודק מקורות ומחזיר רשימה. הוא לא מנהל את כל הדוקטורט.

מתי לא: כשהמשימות תלויות זו בזו. שני סוכנים שעורכים את אותו קובץ במקביל = בלגן. "רק כשאפשר להפריד את העבודה בלי לאבד שליטה."

סוכנים מובנים

- **Explore** - חיפוש מהיר, קריאה בלבד. "איפה בקוד מטופל הטופס?"
- **Plan** - מחקר לפני תוכנית.
- **general-purpose** - עובד כללי עם כל הכלים.
- **claude-code-guide** - עונה על "איך עושים X בקלוד קוד?" מהתיעוד. שימושי מאוד למתחילים: פשוט תשאלו.

להגדיר סוכן משלכם

קובץ ב-`.claude/agents/<שם>.md` (פרויקט) או `~/.claude/agents/` (גלובלי):

```
---
name: proposal-reviewer
description: בודק הצעות מחיר של E.Y. AI. השתמש כשמבקשים "תבדוק את ההצעה" לפני שליחה.
tools: Read, Grep, Glob
model: sonnet
---
אתה מבקר הצעות מחיר. תבדוק: גוף שני, אין em dash, מחירים תואמים לקטלוג,
אין מסלולים שלא ביקשו, ה-CTA תחזיר רשימת בעיות עם שורות. "אשמח לעבוד איתכם" לא.
```

מפעילים ב-`@proposal-reviewer` או "תשתמש ב-proposal-reviewer על הקובץ". יש גם `/agents` לניהול. שדות שימושיים: `tools` (רשימת היתר), `disallowedTools`, `model`, `permissionMode`, `memory:` `project` (זיכרון משלו), `isolation: worktree` (עותק נפרד של הריפו).

Fork: `/subtask נסח בדיקות לשינויים עד עכשיו` - סאב-אייג'נט שיורש את **כל** השולחן שלכם (לא מתחיל מאפס). טוב כשההקשר חשוב, יקר יותר.

Workflows (מתקדם)

כשיש הרבה עבודה מקבילית - "תעבור על 30 קבצים ותבדוק כל אחד משלוש זוויות" - קלוד יכול לבנות **workflow**: סקריפט קצר שמפעיל צי סוכנים, אוסף תוצאות, מאמת. באפליקציה רואים את זה בפאנל ה-tasks. המילה `ultracode` בפרומפט אומרת לקלוד "מותר לך להשתמש בכמה סוכנים שצריך" (זה יקר, שמרו למשימות שוות).

מה זה נותן לי?

- כל משימה "מלוכלכת" (בדיקות, סריקה, מחקר רחב) ← סאב-אייג'נט. השולחן נשאר נקי, המכסה נמתחת.
- סוכן-מבקר קבוע לכל תוצר שיוצא ללקוח.

4.2 Worktrees - עותק של המשרד לכל עובד

בפרויקטי גיט, אפליקציית הדסקטופ נותנת **לכל סשן חדש עותק מבודד של התיקייה** (git worktree, בתוך `.claude/worktrees/`). סשן א' עורך את הכותרת, סשן ב' משכתב את הפוטר - אף אחד לא דורך על השני. בסוף עושים merge.

בטרמינל זה אופציונלי: `claude --worktree`. סאב-אייג'נטים יכולים לקבל `isolation: worktree` גם הם.

מתי זה קריטי: שני סשנים (או קלוד וקודקס) על אותו פרויקט. פרק 6.

מתי לא צריך: תיקייה של מסמכים בלי גיט. (וגם אז - `git init` שווה, פרק 7.)

4.3 פלאגינים - הארגז שמביא כמה חלקים ביחד

פלאגין = חבילה שמתקינים פעם אחת ומקבלים: סקילים + סוכנים + הוקים + שרתי MCP + הגדרות. "Plugin הוא האריזה, לא העובד."

דוגמה שנגעתם בה היום: `telegram@claude-plugins-official` הביא שרת MCP (מדבר עם טלגרם), סקיל `/telegram:access`, פקודת `/telegram:configure`. שלושה חלקים, התקנה אחת.

מאיפה: marketplaces. הרשמי - `claude-plugins-official` (אנתרופיק). יש בו: code-review, frontend-design, skill-creator, plugin-dev, ralph-loop, session-report, וקבוצת "external": telegram, discord, imessage, github, playwright, context7, linear, asana, serena, security-guidance.

איך

```
/plugin                                    <- מנהל הפלאגינים (או + -> Plugins באפליקציה)
/plugin install security-guidance@claude-plugins-official
/plugin marketplace add anthropics/skills <- להוסיף marketplace
/reload-plugins                            <- אחרי שינוי
```

סקילים מפלאגין נקראים `שם-פלאגין:שם-סקיל/` כדי שלא יתנגשו.

פלאגין משלכם = תיקייה עם `.claude-plugin/plugin.json` (שם, תיאור, גרסה) + `skills/`, `agents/`, `hooks/`, `.mcp.json`. בודקים עם `claude --plugin-dir ./my-plugin`. זו הדרך לארוז את כל ה-GenOS שלי לחבילה שמישהו אחר מתקין בפקודה אחת - וזה בדיוק מה שאעשה לקהילת הלימוד.

אזהרה: פלאגין = הרשאות רחבות (הוקים רצים אוטומטית, MCP מתחבר החוצה). אותו צ'קליסט אבטחה מפרק 3.6, רק יותר.

4.4 הוקים (Hooks) - חוקי הבית האוטומטיים

האנלוגיה: מזגן שנדלק לבד ב-25 מעלות. לא צריך לבקש. הוק = פקודת מחשב שרצה **אוטומטית** באירוע מסוים. בניגוד ל-CLAUDE.md (קלוד "משתדל" לציית), הוק **תמיד** קורה. דטרמיניסטי.

אירועים: לפני כלי (`PreToolUse`), אחרי כלי (`PostToolUse`), תחילת סשן, סוף סשן, כשקלוד מחכה לכם (`Notification`), יצירת worktree.

דוגמאות שימושיות למי שלא מתכנת

- **פינג כשקלוד מחכה לי:** הוק על Notification שמשמיע צליל / שולח הודעה. הלכתם לשתות שייק - תדעו לחזור.
- **גיבוי אוטומטי:** SessionEnd ← `git commit -am "session end"`.
- **לסנן פלט ארוך:** PreToolUse על Bash - כשמריצים בדיקות, להוסיף `| grep FAIL` כדי שקלוד יראה רק כשלים. חוסך טוקנים.
- **חסימה:** PreToolUse שמונע `rm -rf` או כתיבה לתיקיית `raw/`.
- **security-guidance:** הפלאגין של אנתרופיק הוא בעצם 6 הוקים שסורקים כל עריכת קוד.

איפה: `~/.claude/settings.json` (גלובלי) או `.claude/settings.json` (פרויקט):

```
{
  "hooks": {
    "Notification": [{ "hooks": [{ "type": "command", "command": "afplay
/System/Library/Sounds/Glass.aiff" }] }]
  }
}
```

או פשוט: "תוסיף לי הוק שמשמיע צליל כשאתה מחכה לי" - קלוד יודע לכתוב אותו (יש לו סקיל `update-config`). יש גם הוקים מסוג `prompt` - "האם הפקודה הזאת בטוחה? אשר או דחה" - קלוד קטן שופט.

כלל אצבע: אם אתם אומרים "מעכשיו, כל פעם ש-X קורה תעשה Y" - זה הוק, לא זיכרון. זיכרון קלוד יכול לשכוח. הוק לא.

4.5 הרשאות מפורטות (settings.json)

במקום לאשר כל פעם, אפשר לרשום מראש:

```
{
  "permissions": {
    "allow": ["Bash(git status)", "Bash(npm test)", "Read", "Edit"],
    "deny": ["Bash(rm -rf*)", "Read(.env)"]
  }
}
```

יש סקיל מובנה `/fewer-permission-prompts` שסורק את ההיסטוריה שלכם ומציע רשימת allow.
`settings.local.json` = הגדרות אישיות שלא עולות לגיט.

4.6 סיכום: העובדים, הארגזים והחוקים

דבר	הוא...	דוגמה
Session	העוזר הראשי, שולחן אחד	"בוא נכתוב את ההצעה"
Sub-agent	עוזר בחדר אחר, שולחן משלו	"תבדוק את הציטוטים ותחזור"
Agent (מוגדר)	עוזר עם תפקיד קבוע וכלים מוגבלים	proposal-reviewer
Workflow	צי סוכנים לפי סקריפט	ultracode על 30 קבצים
Worktree	עותק של המשרד	שני סשנים על אותו אתר
Skill	מתכון במגירה	ey-ai-proposal/
Plugin	ארגז: סקילים + סוכנים + הוקים + MCP	telegram
Hook	חוק בית אוטומטי	צליל כשקלוד מחכה
CLAUDE.md	הדף על הקיר	"אין em dash"

חיבורים: MCP, קונקטורים, וAPI, ארטיפקטים ותוסף הכרום

> "עד עכשיו הסוכן עובד בתוך התיקייה. MCP מאפשר לו לפתוח דלת למערכת נוספת."

MCP 5.1 - השקע המשותף

האנלוגיה: לפני שהיה תקן לשקעי חשמל, כל מכשיר הגיע עם תקע משלו. MCP (Model Context Protocol) הוא **התקן**. במקום שכל שירות ימציא דרך אחרת לדבר עם AI, MCP נותן מבנה משותף: אילו כלים יש, מה הם מקבלים, מה מחזירים.

בשפה פשוטה: שרת MCP הוא "מתאם" שאומר לקלוד: "אני יודע לקרוא את היומן שלך, ליצור אירוע, ולמחוק אירוע. הנה איך קוראים לי." קלוד לא צריך ללמוד את ה-API של גוגל. הוא רואה שלושה כלים.

האם MCP זה API? MCP משתמש ב-API מאחורי הקלעים, אבל הוא שכבה שמציגה יכולות לסוכן בפורמט משותף. **מקומי או ענן?** שניהם: `npx ...` מריץ שרת קטן על המחשב שלכם, `https://...` פונה לשרת רחוק (עם OAuth).

מה יש היום (2026): GitHub, Figma, Notion, Slack, Google Drive/Calendar/Gmail, Playwright (דפדפן), Supabase, Cloudflare, Context7 (תיעוד ספריות), Blender, Consensus (מאמרים), NotebookLM, כלי מדיה (fal, Magnific), ועוד מאות. אצלי מחוברים בין השאר: פיגמה, קנבה, גאמה, יומן, ג'ימייל, דרייב, בלנדר, Netlify, HuggingFace, Consensus, מנוע מדיה.

איך מוסיפים (טרמינל)

```
claude mcp add --transport http figma https://mcp.figma.com/mcp        # שרת רחוק
claude mcp add notebooklm -- npx -y notebooklm-mcp@latest              # שרת מקומי
claude mcp list
```

scopes: `local` (רק אני, רק כאן), `project` (קובץ `.mcp.json` בריפו, לצוות), `user` (כל הפרויקטים שלי). `/mcp` בתוך סשן מראה מה מחובר, מאפשר להתחבר (OAuth), ולכבות. באפליקציה: `+` ← Connectors (זה בדיוק MCP עם ממשק גרפי).

קודקס: אותם שרתים, ב-`~/.codex/config.toml` תחת `[mcp_servers.<שם>]`, או `codex mcp add`.

שני דברים שכדאי לדעת

- **טוקנים.** כל שרת MCP מוסיף את שמות הכלים שלו לשולחן. היום זה "deferred" (רק שמות, הסכמות נטענות בשימוש), אבל 15 שרתים עדיין מרגישים. `/context` יראה. לכבות מה שלא בשימוש.
- **אבטחה.** שרת MCP = קוד שרץ עם הרשאות. שרת לא מוכר יכול לקרוא קבצים או להוציא מידע. אותם כללים: מקור מוכר, לקרוא מה הוא מבקש, OAuth ולא מפתחות ידניים בקבצים.

מה זה נותן לי? "תבדוק ביומן מתי פנוי לשרון, תציע 3 זמנים, ותכתוב לה מייל." שלוש דלתות (יומן, ג'ימייל, קובץ סיכום) בבקשה אחת.

5.2 קונקטורים - הדלת עם הידית המוכנה

Connector = שרת MCP עם **הגדרה גרפית**: לוחצים, מתחברים עם גוגל/סלאק/פיגמה, סיימתם. "MCP הוא התקן, Connector הוא המימוש."

איפה: באפליקציית הדסקטופ `+` ← Connectors (סשנים מקומיים ו-SSH; בענן מגדירים בזמן יצירת רוטינה). ניהול: Settings ← Connectors. ב-claude.ai (הצ'אט הרגיל) יש ספריית קונקטורים באותו חשבון, וטאב Cowork שואב מאותה הגדרה ("Customize"), שמסונכרנת דרך החשבון ולא מ-`~/.claude`.

מה חשוב: אחרי חיבור בודקים שלושה דברים: **מה אפשר לקרוא, מה אפשר לשנות, מה דורש אישור.** ולפני חיבור כותבים שלושה דברים: מה הוא חוסך, איזו הרשאה נדרשת, מה ה-fallback. אם אין תשובה טובה - לא מחברים.

הכלל שחוזר: מה שקונקטור מחזיר הוא **נקודת כניסה** למקור (כותרת, קישור), לא הראיה עצמה. Consensus מוצא מאמרים; הוא לא מכניס אותם אוטומטית לטבלה. אתם פותחים ובודקים.

קודקס: ה"פלאגינים" של קודקס (gmail, calendar, drive, figma, canva, github, vercel...) הם בערך קונקטורים + סקילים. אצלי מופעלים 26.

5.3 API ו-Agent SDK - למי שרוצה לבנות, לא רק להשתמש

"API זה מפתח לשער שמאפשר לי לייבא טכנולוגיה ממקור אחד לאפליקציה שלי." (זה מהרצאה שלי מ-2025, עדיין עומד.)

מתי אתם צריכים API ולא מנוי?

- כשאתם בונים **מוצר** שמישהו אחר משתמש בו (בוט וואטסאפ ללקוח, אתר שמייצר תוכן, אוטומציה שרצה בשרת 24/7).
- כשאתם רוצים לשלם לפי שימוש ולא מכסה.

המנוי (Pro/Max) = קלוד קוד, הצ'אט, Cowork, האפליקציות. יש מכסה. **API** (console.anthropic.com) = מפתח, משלמים לפי טוקנים, בלי חלונות של 5 שעות. קלוד קוד עצמו יכול לרוץ גם על מפתח API במקום login (למי שרוצה לעקוף מכסות בכסף).

מודלים באוגוסט 2026 (למי שכותב קוד): `claude-fable-5` (החזק ביותר), `claude-opus-5`, `claude-sonnet-5` (היחס הכי טוב), `claude-haiku-4-5` (מהיר וזול). כולם עם קונטקסט של עד מיליון, prompt caching (חלק זול בהרבה כשחוזרים על אותו הקשר), Batch API (חצי מחיר לעבודה לא דחופה).

Claude Agent SDK (Python / TypeScript) = **קלוד קוד כספרייה**. אותו לולאת סוכן, אותם כלים (קריאה, כתיבה, טרמינל, חיפוש), אותם הוקים וסקילים - אבל אתם מארחים. זה מה שהייתי משתמש בו לבנות סוכן וואטסאפ ללקוח: Agent SDK + Twilio + Supabase. יש גם **Managed Agents** - אנתרופיק מארחת את הסוכן בשבילכם.

מה לא צריך לדעת: אם אתם רק משתמשים בקלוד קוד - לא צריך API בכלל. אם צריך - יש סקיל `claude-api` מובנה שיודע את כל המחירים והפרמטרים; פשוט תשאלו את קלוד "תבנה לי בוט טלגרם קטן עם Agent SDK".

5.4 ארטיפקטים - התוצר שעומד בפני עצמו

מה זה: "במקום להשאיר הכל בתוך השיחה, מקבלים חלון נפרד שאפשר לערוך ולשתף." בצ'אט ובקלוד קוד: קלוד יכול **לפרסם דף** (HTML או Markdown) לכתובת ב-claude.ai. פרטי כברירת מחדל; אתם מחליטים אם לשתף.

מתי: דוח, מדריך, דשבורד, השוואה - כל דבר שקורא-אחר צריך לפתוח בלי להתקין כלום. "תפרסם את סיכום האנליטיקס כארטיפקט ותן לי לינק לשלוח לצוות."

מגבלות שכדאי לדעת: הדף מבודד (CSP): לא יכול לקרוא לשרתים חיצוניים, לא פונטים מ-CDN, לא מצלמה/מיקרופון. הכל inline. עד 16MB. יש "capabilities" מתקדמות (נתונים חיים, מצב משותף) למי שיש. אפליקציית וב אמיתית עם מצלמה? Netlify (אצלי מחובר), לא ארטיפקט.

באפליקציית הדסקטופ יש גם **פריוויו** של אפליקציות (השרת המקומי שלכם רץ בפאנל) ו**דפדפן פנימי** לגלישה - זה לא ארטיפקט, זה בשביל לבדוק מה בניתם.

5.5 Claude in Chrome - ידיים בדפדפן שלכם

מה זה: תוסף לכרום. קלוד יכול לפתוח טאבים, לקרוא דפים, ללחוץ, למלא טפסים, לצלם מסך, לקרוא console. **בדפדפן האמיתי שלכם, עם ההתחברויות שלכם.** (יש גם "Browser" פנימי באפליקציה, נקי בלי cookies - לגלישה כללית. הכרום - כשצריך את החשבונות שלכם.)

התקנה: תוסף Claude מחנות כרום ← באפליקציה/CLI הוא מופיע כ-MCP `claude-in-chrome`. בטרמינל `claude --chrome`.

מתי: "תיכנס ל-YouTube Studio ותוריד לי את הנתונים של השבוע", "תעלה את 15 הקבצים לדרייב דרך הדפדפן" (עשיתי את זה אתמול), "תבדוק שהאתר נראה טוב ב-3 רזולוציות". חילוץ מידע מאתרים שאין להם API. וגם: כש-yt dlp נכשל, הכרום מביא תמלול.

"אני מעדיף לומר 'לעבוד בדפדפן' ולא 'להשתלט על הדפדפן'." והזהירות: דף אינטרנט יכול להכיל הוראה זדונית שמנסה להשפיע על הסוכן. לכן מתחילים ב-Ask before acting, ומה שכתוב בדף = מידע, לא פקודה. קלוד לא ימלא סיסמאות/כרטיסי אשראי, לא יעבור CAPTCHA, ויבקש אישור לפני שליחה/פרסום/מחיקה. זה בכוונה.

5.6 מפה: איזו דלת למה

אני רוצה ש...	הדלת
קלוד יקרא/יכתוב ביומן, מייל, דרייב, נושן, סלאק	**Connector** (Connectors ← `+`)
שירות שאין לו קונקטור מוכן	**MCP** ידני (`claude mcp add`)
לעבוד באתר בדפדפן שלי עם ההתחברות שלי	**Claude in Chrome**
לגלוש/לבדוק אתר בלי החשבונות שלי	**Browser** הפנימי
לתת למישהו דף לקרוא	**Artifact**
אפליקציה אמיתית עם מצלמה/שרת	Netlify / Cloudflare (דרך קלוד קוד)

אני רוצה ש...	הדלת
לבנות מוצר/בוט למישהו אחר	**API / Agent SDK**
קובץ מהמק	`@` או גרירה. בלי דלת בכלל

קלוד וקודקס ביחד על אותו פרויקט (HANDOFF)

> "Chat, Cowork, Code ו-Codex הם ארבעה חללים עם דלתות שונות לאותה צורת עבודה."

6.1 מה זה קודקס באוגוסט 2026 (בקצרה)

קודקס = הסוכן של OpenAI. מותג אחד, ארבע דלתות:

- **CLI** (`codex` בטרמינל, קוד פתוח)
- **תוסף IDE**
- **טאב Codex בתוך אפליקציית ChatGPT** (ביולי הם איחדו את אפליקציית Codex לתוך ChatGPT. סשנים מקבילים עם worktrees, תור ביקורת, דפדפן פנימי, computer use)
- **Codex cloud** (chatgpt.com/codex + אפליקציית ChatGPT בטלפון; QR = **Codex Remote** מהמק לטלפון, בדיוק כמו Remote Control)

מודלים: משפחת GPT-5.6 (Sol החזק, Terra המאוזן - זה מה שאני מריץ, Luna הזול). קונטקסט 272 אלף. הכל מ-`~/.codex/`. הגדרות ב-`~/.codex/config.toml`.

העקרונות זהים: תיקייה, הוראות, סקילים, כלים, הרשאות, תוצר. מי שמכיר קלוד קוד מבין קודקס בעשר דקות. אותו SKILL.md עובד בשניהם.

למה שניהם? כי כל אחד טוב במשהו אחר ביום נתון, כי שני מנויים = כפול מכסה, וכי שני מוחות על אותה בעיה תופסים יותר. אני: קלוד לחשיבה, כתיבה, ניהול, מחקר; קודקס לבנייה מכנית ארוכה ולביקורת. אצלי Bootch/reelsmith נבנה עם קודקס וקלוד הוא הסוקר החיצוני.

6.2 CLAUDE.md מול AGENTS.md - הדף על הקיר, בשני כתבי יד

AGENTS.md = ה-CLAUDE.md של קודקס. אותו רעיון: קובץ Markdown שנטען אוטומטית. הוא גם סטנדרט פתוח (Cursor, Copilot, Gemini CLI קוראים אותו).

	Claude Code	Codex
קובץ	`CLAUDE.md` (+ `CLAUDE.local.md`, `.claude/rules/`)	`AGENTS.md` (+ `AGENTS.override.md` שדורס זמנית)
גלובלי	`~/.claude/CLAUDE.md`	`~/.codex/AGENTS.md`
מיזוג	מהשורש למטה, הקרוב מנצח, תת-תיקיות נטענות עצלנית	מהשורש למטה, הקרוב מנצח
תקרה	אין (המלצה: <200 שורות)	32KB לכל הקבצים יחד (`project_doc_max_bytes`)

	Claude Code	Codex
ייבוא	`@קובץ`	אין. אבל `project_doc_fallback_filenames = ["CLAUDE.md"]` אומר לו לקרוא CLAUDE.md כשאין AGENTS.md
התחלה	`/init`	`/init`
זיכרון אוטומטי	`~/.claude/projects/<פרויקט>/memory/`	`~/.codex/memories/` (גלובלי, `/memories`)

חשוב: קלוד קוד **לא** קורא AGENTS.md לבד. קודקס **לא** קורא CLAUDE.md לבד. ו"אז AGENTS.md הוא Agent?" לא. זה קובץ טקסט. השם מבלבל.

הפתרון: קובץ אחד, שני קוראים

אופציה א' (ההמלצה הרשמית של אנתרופיק): AGENTS.md הוא הקנוני, ניטרלי לכלי. CLAUDE.md מכיל שורה אחת + תוספות לקלוד:

```
@AGENTS.md

## Claude Code
השתמש ב-plan mode לכל שינוי בתיקיית output/.
```

זה מה שעשיתי בערכת המשתתפים של TAU: CLAUDE.md רזה שמצביע על AGENTS.md.

אופציה ב' (מה שיש לי ב-Downloads): CLAUDE.md הוא הקנוני (מפת ניווט ענקית). לקודקס - AGENTS.md של 5 שורות: "תקרא את CLAUDE.md קודם, אותם כללים חלים עליך", או שורה ב-`config.toml`: `project_doc_fallback_filenames = ["CLAUDE.md"]`. שימו לב ל-32KB.

אופציה ג': symlink (`ln -s AGENTS.md CLAUDE.md`). עובד במק, לא בווינדוס, ואי אפשר תוספות לקלוד.

מה לא לעשות: לתחזק שני קבצים שונים ביד. הם יתפצלו תוך שבועיים. זה הכשל מספר אחת שמדווחים עליו.

כלים לגשר: `/import` בקלוד (2.1.213+) מעתיק חד-פעמית הגדרות של קודקס (AGENTS.md, MCP, סקילים). ובאפליקציית ChatGPT יש מאוגוסט "Import from Claude Code / Cowork / Cursor" עם סנכרון.

6.3 סקילים וזיכרון - משותפים או נפרדים?

סקילים: משותפים. אותו SKILL.md. קודקס מחפש ב-`~/.codex/skills`, `.codex/skills`, `~/.agents/skills`, `.agents/skills`. אצלי:

```
ln -s ~/.claude/skills/hyperframes ~/.codex/skills/hyperframes
```

20 כאלה. בערכת TAU שמתי את אותם 7 סקילים גם ב-`.claude/skills` וגם ב-`.agents/skills`. "לא צריך לבחור ולא צריך למחוק." מפעילים: קלוד `/שם`, קודקס `$שם` (`/skills` לרשימה, `$skill-installer` להוריד

מהקטלוג של OpenAI).

זיכרון: נפרד, וזה בסדר. לקלוד יש זיכרון לפי תיקייה; לקודקס זיכרון גלובלי ב-`~/.codex/memories/` (עם MEMORY.md, סיכומי סשנים, ואפילו `.git` פנימי; מופעל אצלי: `[features] memories = true`). שניהם Markdown, שניהם קריאים. **מה שצריך להיות בשני המוחות - לא סומכים על אף זיכרון. שמים בקובץ בפרויקט.** (פרק 2, ופרק 7.)

Handoff 6.4 - איך מוסרים עבודה בין קלוד לקודקס (ובין סשן לסשן)

האנלוגיה: שני פרילנסרים על אותו פרויקט, שלא נפגשים אף פעם. מה צריך כדי שזה יעבוד? פנקס משותף, תיקייה משותפת, וכללי ברזל.

הפנקס המשותף - שלושה קבצים

1. ** `KICKOFF_*.md` / `HANDOFF.md` **(מצב חי, נכתב מחדש כל סשן): מטרה, מה נעשה, מה הבא, חסמים, החלטות,** נקודת עצירה לאישור**. הנוסח שקודקס עצמו שמר עליי בזיכרון: "כשמבקשים handoff, לתת מסמך אחד מסודר, copy-paste-able, עם נתיבים, אילוצים, בדיקות קבלה ועצירת אישור."
2. **`TASKS.md`** - רשימת משימות אחת. `[ ]` / `[x]`. מקור אמת יחיד.
3. **`AGENT_LOG.md`** - append-only. שורה לכל פעולה משמעותית: `תאריך שעה | סוכן | פעולה | | סטטוס`. כל סוכן קורא את 5-10 השורות האחרונות **לפני** שהוא עושה משהו שיכול להתנגש (אירוע ביומן, יצירת קובץ, טיוטת הודעה).

למה זה נולד אצלי: ב-16.05 קלוד וקודקס יצרו לי שניהם אירוע ביומן לאותה פגישה. פעמיים. מאז יש AGENT_LOG וכללי ברזל: **חפש לפני שאתה יוצר, רשום מיד אחרי שיצרת.**

כללי הברזל (לשים ב-AGENTS.md/CLAUDE.md של שניהם)

1. לפני כתיבה חיצונית (יומן, מייל, קובץ) - **חפש קודם**. `list_events` עם מילת מפתח, Glob לנתיב.
2. בדוק את טבלת "פעילים" ב-AGENT_LOG.
3. פעל, ואז **שורת לוג מיד** (לא בסוף הסשן).
4. פעולות **אידמפוטנטיות**: שמות קבצים דטרמיניסטיים (`MEETING_SUMMARY_2026-05-27_realclicks.md`), "ערוך אם קיים, אחרת צור".
5. **נתיב בעלות** לכל סוכן בסשן: קלוד = hub/תכנון, קודקס = בנייה ב-worktree. כתוב ב-HANDOFF.
6. מה שחייב לשרוד החלפת כלי - **בקובץ בריפו**, לא בצ'אט.

git הוא החוזה

- שניהם עושים commit, עם שם הסוכן בהודעה (`[claude]` , `[codex]`). הסוקר קורא `git log -p` במקום צ'אט.
- עבודה מקבילית ← **worktrees** (שתי האפליקציות יוצרות לבד; בטרמינל `git worktree add ../proj-codex codex/feature`). שני סוכנים לא נוגעים באותו קובץ בו-זמנית.

- קודקס בכלל נחמד יותר בתיקיית גיט (מצב Auto במקום read-only, `codex exec` מסרב לתיקייה בלי גיט). לכן: **`git init` על הכל**, גם על תיקיית Markdown.

תבניות עבודה שעובדות

- **בונה / מבקר:** קלוד בונה, קודקס מבקר (`codex review` או `$review-agent`). או הפוך. אני עושה את זה על reelsmith.
- **ספק-קודם:** אחד כותב בדיקות/קריטריוני קבלה, השני מממש.
- **הצלבה:** אותה משימה בשני ענפים, משווים גישות.
- **"קלוד מבין, קודקס מבצע":** קלוד חוקר ומתכנן, קודקס עושה 40 קבצים מכניים.

לקרוא אחד לשני ישירות (מתקדם)

- **קודקס מתוך קלוד:** `claude mcp add codex --scope user -- codex mcp-server` (קודקס הופך לשרת MCP עם כלים `codex` ו-`codex-reply`). או פשוט מהטרמינל של קלוד: `codex exec -s workspace-write -a never -o /tmp/out.md "האחרון ותרשום באגים diff-תבדוק את ה"` ואז קלוד קורא את הקובץ.
- **קלוד מתוך קודקס:** `claude mcp serve` (קלוד כשרת MCP), ב-config.toml: `[mcp_servers.claude-code] command="claude" args=["mcp","serve"]`. או `claude -p "..." --output-format json` מהטרמינל של קודקס.
- יש לי סקיל בקודקס `claude-export-context` שקורא `/export` של קלוד ומזקק אותו לזיכרון של קודקס.

6.5 מה שיש לי בפועל (כדי שתראו שזה לא תיאוריה)

```
Downloads/
├── CLAUDE.md          <- מפת הניווט (קלוד)
├── AGENTS.md          <- משימה: 5 שורות "תקרא" ריק כרגע! CLAUDE.md")
├── TASKS.md           <- מקור אמת למשימות
├── AGENT_LOG.md       <- כל פעולה של כל סוכן, מאז מאי
├── GenOS/             <- המוח השני (פרק 7), שניהם קוראים
└── E.Y.AI/לקוחות/X/KICKOFF_X.md   <- handoff לכל פרויקט
```

- `~/.codex/skills/` - 20 symlinks ל-`~/.claude/skills/`.
- קודקס עם זיכרון פעיל, מודל terra, 26 פלאגינים.
- reelsmith: קודקס בונה, קלוד סוקר.

משימה שיצאה מהפרק: למלא את `~/.codex/AGENTS.md` ואת `Downloads/AGENTS.md` בכללי הברזל. (נוסף ל-TASKS.)

6.6 מה זה נותן לי?

- לא מסבירים פעמיים. הקובץ מסביר.
- לא יוצרים אירוע פעמיים. הלוג עוצר.
- שני מוחות, שתי מכסות, אותה תיקייה, אפס בלגן - אם יש פנקס, גיט וכללים.

מוח שני: קבצי md, תיקיות, Obsidian, ומקומי מול דרייב

> "התיקייה היא הבית של העבודה." וגם: "הסיומת .md לא יוצרת זיכרון קסום."

7.1 מה זה קובץ md (ולמה כל העולם האייג'נטי בנוי עליו)

בשפה פשוטה: קובץ טקסט רגיל. פותחים אותו ב-Notes או TextEdit ורואים טקסט קריא. רק שכמה סימני פיסוק אומרים "עיצוב": `#` כותרת, `-` תבליט, `**מודגש**`, `[קישור]`. מכונות (GitHub, Obsidian, קלוד, קודקס) מציגות אותו יפה; בני אדם קוראים אותו גם גולמי.

למה לא Word? Word טוב למסמך מוגמר. md טוב להוראות, תיעוד וזיכרון - כי אפשר לקרוא, להשוות גרסאות (diff), לערוך בשקיפות, ובעיקר: **מודלים קוראים וכותבים אותו באופן טבעי.** לכן CLAUDE.md, AGENTS.md, SKILL.md, README.md, MEMORY.md - כולם .md.

צריך לדעת Markdown? מספיק כותרת, רשימה, קישור. הנה כל מה שצריך:

```
# כותרת ראשית
## כותרת משנה
### תת-כותרת

**מודגש** *נטוי* `קוד` ~~מחוק~~

- תבליט
- רשימה שבוצעה [x]
- [ ] משימה פתוחה

1. ממוספר
   - מקונן

[טקסט](https://...)  ![תמונה](image.png)

> ציטוט

---

| עמודה | עמודה |
|---|---|
| א | ב |
```

תוספות שהסוכנים אוהבים: **frontmatter** בראש הקובץ (`---` / `name:` / `description:` / `---`) - זה איך SKILL.md וקבצי זיכרון מתייגים את עצמם. `[[קישור-פנימי]]` (Obsidian). `<!-- הערה -->` (קלוד מתעלם ממנה בקונטקסט).

טיפ עברית: ל-md אין כיוון. התצוגה מזהה לבד לפי פסקה. שורות מעורבות עם מספרים ונתיבים לפעמים מתהפכות - שימו נתיבים ומספרים ב-`backticks`.

איך פותחים md יפה

איפה	האופציות (מהפשוטה)
מק	Quick Look עם רווח בפיינדר - צריך תוסף: `brew install --cask qlmarkdown` (חינם). עורכים: **Obsidian** (חינם, ההמלצה), Typora (WYSIWYG, בתשלום), iA Writer (יפה, RTL טוב), Marked 2 (מציג ומתעדכן חי כשקלוד עורך - שילוב מנצח), VS Code / Cursor (Cmd+Shift+V לפריוויו). וגם: קלוד דסקטופ מציג md בפאנל, ואפליקציית ChatGPT גם
ווינדוס	Obsidian, Typora, VS Code, MarkText (חינם), Notepad++ עם תוסף PowerToys (פריוויו בסייר)
טלפון	Obsidian mobile (סנכרון דרך iCloud/Obsidian Sync/git), iA Writer, אפליקציית GitHub (מציגה כל md בריפו)
טרמינל	`glow file.md`
להמיר	"תהפוך את זה ל-docx/PDF" (הסקילים הרשמיים) או `pandoc`

7.2 מוח שני - איפה, איך, ולמה

למה: הצ'אט מת בסוף הסשן. הזיכרון האוטומטי הוא של הכלי (קלוד או קודקס) ושל המחשב הזה. **מוח שני** הוא ידע שהוא **שלכם**: ניטרלי לכלי, greppable, diffable, נייד, קריא לבני אדם. עובר איתכם גם כשמחליפים כלי, כשמחשב נשרף, כשמצטרפת שותפה.

מה: תיקייה מקומית של קבצי md. זהו. אין API, אין מסד נתונים, אין מנוי. קלוד וקודקס הם "סוכני קבצים" - תיקייה היא הזיכרון המשותף הכי זול שיש.

איפה: ריפו גיט על הדיסק (`~/brain/`, או אצלי `Downloads/GenOS/`), אופציונלית פתוח כ-vault של Obsidian, אופציונלית מגובה ל-GitHub פרטי / Time Machine / iCloud - **כגיבוי, לא כתיקיית עבודה** (סעיף 7.5).

איך זה נראה אצלי (GENOS)

```
GenOS/
├── CLAUDE.md          <- הוראות למי שנכנס לכאן
├── BRAIN.md           <- מפת ניתוב: לאיזה סקיל/סוכן/זיכרון הולכים לכל סוג משימה
├── A-agents/          <- הגדרות סוכנים
├── B-brain/           <- ידע: אנליטיקס, מחירים, דוגמאות (מה שנקלט מקישורים)
├── C-core/            <- זהות: voice-dna.md, icp-profile.md (הכי חשוב. נקרא לפני כל תוכן)
├── M-memory/          <- learning-log.md, decisions.md, feedback.md
├── O-output/          <- תוצרים
└── T-tools/skills/    <- 50 סקילים
```

"תיקיות לפי תפקיד + מפת ניתוב." זו שיטה. יש אחרות:

- **PARA** (Projects / Areas / Resources / Archive) - פשוט, טוב למי שמתחיל.
- **Zettelkasten-lite** - פתקים אטומיים עם `[[קישורים]]` ודף אינדקס. סוכנים אוהבים כי קישורים = גרף שהמודל עוקב אחריו.
- **LLM Wiki** (הדפוס שקרפתי פרסם באפריל, ורוב תבניות ה-Obsidian+Claude ב-2026 מיישמות): `raw/` (זורקים מקורות) ← הסוכן כותב דפים ב-`wiki/` ומקשר ← `index.md` + `log.md`. סכמה אחת ב-

CLAUDE.md מגדירה פורמט דף, שמות, קישורים, ושלוש פעולות: **ingest / query / lint**. שאלה שהתשובה עליה היא מושג חדש = דף חדש. זה בדיוק מה שהסקיל `link-to-knowledge` שלי עושה: קישור נכנס ← ידע מסודר ב-B-brain/knowledge/.

איך מתחילים (בלי לחפור)

1. תיקייה. `git init`.
2. `CLAUDE.md` + `AGENTS.md` (5 שורות: "זה המוח שלי. דפים ב-md. כל דף עם כותרת ותאריך. תקשר בין דפים. אל תמחק, תארכב.").
3. `inbox/` לזרוק פנימה. `notes/` למה שעוכל. `index.md` שקלוד מתחזק.
4. פעם בשבוע: "תעבור על inbox, תעכל, תעדכן index, תגיד לי מה סותר מה" (lint).

זהו. אחרי חודש יש לכם מוח.

7.3 למה אנשים אוהבים Obsidian (ולמה זה מתחבר מושלם לקלוד קוד)

- **זה רק קבצים.** ה-vault של Obsidian = תיקייה של md. אין נעילה, אין פורמט קנייני. `cd vault && claude` - אפס אינטגרציה.
- **קישורים וגרף.** `[[דף]]` יוצר קישור; יש תצוגת גרף שמראה **מה הסוכן חיבר**. backlinks. פתאום רואים את המוח.
- **תוספים.** אלפים. Dataview (שאילתות על הפתקים), Templater, Web Clipper (לקלוט דפי אינטרנט ל-inbox), Calendar.
- **אופליין ובטלפון.** אפליקציות מק/ווינדוס/iOS/אנדרואיד. סנכרון דרך iCloud/גיט/Obsidian Sync.
- **חינם** לשימוש אישי.

הזרימה הטיפוסית (2026): פתק יומי ← קלוד סורק inbox, מתייק, מקשר, מעדכן אינדקס. שבועי - lint (סתירות, יתומים). "journal:" בצ'אט ← קובץ מתוארך. שואלים שאלה, קלוד כותב את הסינתזה בחזרה ל-vault. אתם קוראים ב-Obsidian, קלוד כותב בטרמינל, אותם קבצים.

האם חייבים Obsidian? לא. VS Code, Cursor, או פיינדר עם QLMarkdown עושים את העבודה. Obsidian פשוט הכי נעים לבני אדם.

חלופות ואיפה הן מתאימות

מאגר	גישת סוכן	טוב ל	חסרון
md מקומי (גיט)	כלי קבצים, ילידי	המוח הראשי, שני הסוכנים	הגיבוי עליכם
Obsidian	אותם קבצים	ממשק אנושי מעל המוח	אין
Notion	MCP / פלאגין	צוותים, טבלאות	לא מקומי, אין grep, צריך רשת
Google Drive/Docs	MCP לקריאה	מסמכים של לקוחות, שיתוף תוצרים	סעיף 7.5

מאגר	גישת סוכן	טוב ל	חסרון
NotebookLM	MCP	שאלות מעוגנות על PDFים	לא מוח שכותבים אליו. "NotebookLM מעגן, קלוד מבצע"

7.4 מבנה תיקייה נוח לכל פרויקט (השלד שלי)

עובד לקוד ולא-קוד: סדנה, הצעה, מחקר, סרטון.

```
הפרויקט/
├── README.md              # מסך אחד: מה זה, למי, סטטוס, איך פותחים (בני אדם קודם)
├── AGENTS.md              # הבריף לסוכנים (ניטרלי): מטרה, כללים, מפת תיקיות, מותר/אסור
├── CLAUDE.md              # "@AGENTS.md" + הערות לקלוד (או symlink)
├── HANDOFF.md / KICKOFF.md# מצב חי: נעשה / הבא / חסמים / החלטות / עצירת אישור
├── TASKS.md               # [ ] / [x], שורה למשימה
├── AGENT_LOG.md           # append-only | סטטוס | פעולה | סוכן | תאריך |
├── docs/ או research/      # סיכומי פגישות, החלטות, מחקר (YYYY-MM-DD_נושא.md)
├── raw/ או sources/       # קלטים מקוריים. לא נוגעים. (read-only)
├── assets/                # לוגו, פונטים, רפרנסים
├── work/ או src/          # קבצי עבודה / קוד
├── output/                # מה שהלקוח רואה. לא עורכים מקורות פה
├── .claude/
│   ├── settings.json      # הרשאות והוקים לפרויקט
│   ├── skills/…/SKILL.md
│   ├── agents/…md
│   └── rules/*.md
├── .agents/skills/        # לקודקס (symlink ל-.claude/skills)
├── .mcp.json              # MCP של הפרויקט
├── .gitignore
└── .env.example           # שמות של סודות, לא ערכים
```

בערכת TAU השתמשתי במספור: `00_admin, 01_questions, 02_sources/raw, 03_methods, 04_data/{raw,processed}, 05_analysis, 06_writing, 07_outputs, 08_submission, 99_archive`. "שמות התיקיות עונים מראש על השאלה איפה המקור ואיפה התוצר." "raw נשאר ללא שינוי, חסר מסומן, וכל טענה מקבלת מקור."

מוסכמות שמות

- קבצי שליטה באותיות גדולות (README, AGENTS, CLAUDE, HANDOFF, TASKS, AGENT_LOG) - ממוינים למעלה, ברור שהם "מטא".
- kebab-case לסקילים/סוכנים (`ey-ai-proposal`). עברית לתיקיות תוכן - כן, עובד מצוין (כל ה-Downloads שלי בעברית).
- **תאריך ISO בהתחלה** לכל מה שנצבר: `2026-08-17_kickoff.md`. ממוין נכון לנצח.
- רעיון אחד לקובץ. AGENTS/CLAUDE מתחת ל-200 שורות. פרטים ← docs/ עם קישור.
- **אף פעם לא קבצים משוחררים בשורש.** לכל תוצר יש בית. (הכלל מספר אחת ב-CLAUDE.md שלי.)

git גם ללא-קוד: `git init` ביום הראשון. היסטוריה של כל עריכה של כל סוכן, `git diff` כביקורת, worktrees למקביליות, קודקס במצב נחמד. remote פרטי ב-GitHub = גיבוי + גישה מהטלפון.

.gitignore חובה: `.env` , `*.key` , `secrets/` , `CLAUDE.local.md` , `.claude/settings.local.json` , `node_modules/` , `.DS_Store` , וידאו כבד. **אף פעם לא מפתחות API ב-AGENTS.md/CLAUDE.md/סקילים** - שני הסוכנים בולעים אותם לזיכרון. במקום זה: קובץ registry שאומר "המפתח של X נמצא ב-Y" (אצלי `api-keys-` `registry.md`).

7.5 לעבוד מקומי או בגוגל דרייב?

התשובה הקצרה: מקומי + גיט, ודרייב **לקרוא ולפרסם**, לא לעבוד.

למה לא תיקיית דרייב מסונכרנת כתיקיית עבודה

- **latency:** הסוכן כותב, דרייב מעלה, מחשב אחר רואה אחר כך. שני סוכנים = "conflicted copy" כפולים.
- **גיט נשבר** בתוך תיקיות מסונכרנות (נעילות, קבצי `desktop.ini` , קבצים שמתרוקנים ל-stub ב"Optimize storage" - הסוכן עושה grep ומוצא כלום).
- היה גם באג היסטורי של איבוד קבצים בדרייב דסקטופ. לא צריך יותר מזה.

מה כן

- העבודה בתיקייה מקומית עם `git init` , גיבוי ל-GitHub פרטי (או Time Machine).
- **Drive MCP / קונקטור** כדי **לקרוא** מסמכים שלקוחות משתפים ("תקרא את הסינופסיס בדרייב ותבנה תיקיית פרויקט" - עשיתי אתמול), ו**לפרסם** תוצרים גמורים (PDF לתיקיית הלקוח).
- אם צוות אנושי חייב לחיות בדרייב - עותק העבודה של הסוכן נשאר מקומי, ומסנכרנים בכוונה (העתקה או `rclone`), אף פעם לא שני סוכנים כותבים לאותה תיקיית דרייב במקביל.
- אותו היגיון ל-Dropbox / OneDrive / iCloud Drive.

חריג סביר: תיקיית סקרין-שוטים או מדיה שרק נקראת - בסדר גם מדרייב.

7.6 מה זה נותן לי?

- md = השפה המשותפת של האדם, קלוד וקודקס. שלושה קוראים, קובץ אחד.
- מוח שני = תיקייה בגיט. Obsidian מעל, אם רוצים. שום מנוי.
- שלד תיקייה קבוע = הסוכן יודע איפה הכל, אתם יודעים איפה הכל, והסשן הבא מתחיל מהמקום הנכון.
- דרייב לקרוא. מקומי לעבוד.

קלוד בענן, בטלפון ובטלגרם

> **"ביקשתי סרטון, הלכתי לשתות שייק וכשחזרתי קלוד אמר: מוכן." הפרק הזה הוא איך עושים את זה גם כשאתם לא ליד המחשב בכלל.**

8.1 ארבע דרכים לעבוד עם קלוד כשאתם לא מול המק

יש פה בלבול גדול כי אנתרופיק הוציאה ארבעה פיצ'רים שנשמעים דומה. האנלוגיה:

פיצ'ר	האנלוגיה	מה רץ איפה
Cloud session (סשן ענן)	שלחתם את העובד לעבוד ממשרד של אנתרופיק	המחשב שלכם יכול להיות כבוי. העבודה קורית על מחשב וירטואלי בענן, על עותק של הפרויקט (מ-GitHub או העלאה)
Remote Control (שלט רחוק)	אתם בחוץ, העובד יושב במשרד שלכם, ואתם מדברים איתו דרך אפליקציית Claude בטלפון	הכל רץ אצלכם במק (הקבצים, ה-MCP, הסקילים). המק חייב להישאר דלוק
Dispatch	מזכירה שמקבלת ממכם משימות בוואטסאפ ומחליטה למי במשרד להעביר	טאב Cowork באפליקציית הדסקטופ. שולחים משימה מהטלפון, Dispatch פותח סשן קוד או עושה בעצמו
Channels (טלגרם / דיסקורד / iMessage)	ווקי-טוקי ישיר לסשן קלוד קוד ספציפי	סשן מקומי במק, אתם כותבים לבוט בטלגרם, ההודעה נופלת לסשן, התשובה חוזרת לטלגרם

ועוד אחד שלא קשור לטלפון אבל תמיד מתערבב:

| **Routines** (רוטינות / משימות מתוזמנות) | שעון מעורר עם רשימת משימות | רץ לבד לפי לו"ז (בענן או מקומית) בלי שאתם בכלל שם |

עכשיו אחד-אחד.

8.2 סשן ענן (Claude Code on the web)

מתי: ריפקטור גדול, ריצת בדיקות ארוכה, "תעבור על 40 קבצים ותתקן", כל דבר שאתם רוצים לסגור את המחשב באמצע.

איך

- באפליקציית הדסקטופ: בתחילת סשן בוחרים **Environment: Cloud** במקום Local. אפשר גם להתחיל מקומי, ואז מתפריט "Continue in..." לשלוח את הסשן לענן עם סיכום השיחה (צריך working tree נקי בגיט).
- בטרמינל: `claude --cloud "תתקן את הבאג בהתחברות"`.
- באתר: claude.ai/code. ובאפליקציית המובייל של Claude רואים את סשני הענן הפעילים ואפשר לדבר איתם.

מה צריך לדעת

- הענן משכפל את הריפו מ-**GitHub** (הענף הנוכחי). יש לכם commits מקומיים שלא דחפתם? הענן לא יראה אותם. `git push` קודם.
- אין GitHub? עובד גם. קלוד "אורז" את התיקייה המקומית ומעלה (עד 100MB).
- סקילים בענן = הסקילים שמופעלים בחשבון claude.ai שלכם, לא `~/.claude/skills`. פלאגינים - צריך להצהיר ב-`.claude/settings.json` של הריפו.
- זה יורד מאותה מכסת שימוש של המנוי.
- `claude --teleport <session-id>` מושך סשן ענן חזרה לטרמינל שלכם.

דוגמה שלי: אתר evyatar-95 - "תעבור על כל 12 החלונות, תוודא שכל אחד עומד בכללי הנגישות בקובץ accessibility.md, ותפתח PR". שולח לענן, סוגר לפטופ, בערב יש PR.

8.3 Remote Control - השלט

מתי: רוצים את **הסביבה המלאה שלכם** (הקבצים, ה-MCP של הפיגמה, הסקילים) אבל להיות בספה עם הטלפון.

איך

1. במק: `claude --remote-control` (או `/remote-control` בתוך סשן).
2. לוחצים רווח ← מופיע QR. סורקים עם הטלפון ← נפתח באפליקציית Claude (או בדפדפן).
3. מעכשיו אפשר לכתוב לסשן מהטלפון, מהדפדפן ומהטרמינל - הכל אותו סשן, מסונכרן.

מגבלות: המק חייב להישאר ער (טיפ: `caffeinate -i claude --remote-control`). אם קלוד מבקש אישור הרשאה ואתם לא בטרמינל - הסשן מחכה. פתרון: מצב Auto/acceptEdits, או Channels (בטלגרם אפשר לאשר). זמין לכל התוכניות (Pro/Max/Team/Enterprise) כ-research preview.

8.4 Dispatch - המזכירה

מה זה בדיוק (מהתיעוד הרשמי, אוגוסט 2026): שיחה קבועה עם קלוד שגרה ב**טאב Cowork** של אפליקציית הדסקטופ. אתם שולחים ל-Dispatch משימה (מהטלפון, דרך אפליקציית Claude), והוא מחליט איך לטפל בה:

- משימת פיתוח ("תתקן את הבאג בלוגין", "תעדכן תלויות", "תריץ בדיקות") ← Dispatch **פותח סשן Claude Code** בעצמו. הסשן מופיע בסרגל של טאב Code עם תג **Dispatch**.
- מחקר, עריכת מסמכים, אקסלים ← נשאר ב-Cowork.
- אפשר גם לבקש במפורש: "תפתח סשן קלוד קוד ותתקן...".
- כשהוא מסיים או צריך אישור ← **push notification** לטלפון.
- אם Computer Use מופעל, גם סשני Dispatch יכולים להשתמש בו (אישורי אפליקציות פגים אחרי 30 דקות).

דרישות: אפליקציית הדסקטופ פתוחה במק + מנוי Pro או Max (לא זמין ב-Team/Enterprise). ההגדרה והצימוד לטלפון - דרך מאמר העזרה של Dispatch (support.claude.com, מאמר 13947068).

מתי זה מנצח: כשאתם לא רוצים לחשוב "לאיזה סשן לשלוח". "תכין לי סיכום של הפגישה שהקלטתי ותשלח לשרון" - Dispatch מנתב.

Channels 8.5 - טלגרם (וזה מה שהתקנו לך)

מה זה: פלאגין רשמי של אנתרופיק שמחבר **בוט טלגרם** לסשן קלוד קוד שרץ במק. אתם כותבים לבוט, ההודעה מגיעה לסשן כאירוע, קלוד עונה בחזרה בטלגרם. הוא יכול גם לשלוח קבצים ותמונות, להגיב באימוג'י, ולערוך הודעה ("עובד..." ← "מוכן").

למה זה הפתרון שאני ממליץ עליו למי שלא מתכנת: אין שרת, אין קוד, אין ngrok. בוט מ-BotFather + פלאגין + פקודה אחת. עשר דקות.

מה כבר עשינו (בסשן הזה, 17.08)

- הותקן **Bun** (הריצה של שרת הפלאגין): `brew install oven-sh/bun/bun`.
- הותקן הפלאגין: `claude plugin install telegram@claude-plugins-official`.
- נכתב סקריפט הפעלה: `~/.claude/channels/telegram/start_telegram_claude.sh` (מפעיל את קלוד עם הערוץ ומשאיר את המק ער).

מה נשאר לך (5 דקות, זה מחייב אותך כי זה טוקן סודי)

1. ליצור בוט: בטלגרם, לפתוח שיחה עם `@BotFather`, לשלוח `/newbot`. הוא שואל שם תצוגה (למשל "Claude של אביתר") ושם משתמש שמסתיים ב-`bot` (למשל `evyatar_claude_bot`). הוא מחזיר **טוקן** שנראה כמו `123456789:AAHfiq...`. להעתיק את כולו.

2. להזין את הטוקן (בתוך סשן קלוד קוד בטרמינל):

```
/telegram:configure 123456789:AAHfiq...
```

זה כותב את הטוקן ל-`~/.claude/channels/telegram/.env`. (אפשר גם לכתוב את הקובץ ידנית: שורה אחת `TELEGRAM_BOT_TOKEN=...`.)

3. להפעיל את קלוד עם הערוץ

```
~/.claude/channels/telegram/start_telegram_claude.sh
```

(או ידנית: `claude --channels plugin:telegram@claude-plugins-official`. אפשר להעביר תיקיית פרויקט כפרמטר לסקריפט; ברירת המחדל היא Downloads.)

4. צימוד: לשלוח לבוט הודעה כלשהי בטלגרם. הוא מחזיר קוד בן 6 תווים. בסשן קלוד:

```
/telegram:access pair ABC123
```

5. לנעול: שאף אחד אחר לא יקבל קוד צימוד:

```
/telegram:access policy allowlist
```

זהו. מעכשיו: "תבדוק מה מצב הפריסה של האתר", "תכתוב לי טיוטת מייל ללקוח מהסיכום בתיקייה X", "צלם לי את הגרף מהאנליטיקס ותשלח" - מהטלפון.

דברים שחשוב לדעת

- **המק חייב להישאר דלוק** ועם הסשן פתוח. הסקריפט מריץ `caffeinate` כדי שלא יירדם. אם המחשב נכבה - הבוט שותק.
- **תמונות** שאתם שולחים נשמרות ב-`~/.claude/channels/telegram/inbox/` וקלוד יכול לקרוא אותן. טלגרם דוחס תמונות; רוצים מקור - לשלוח כקובץ.
- **אין היסטוריה:** ה-Bot API של טלגרם לא נותן לקרוא הודעות ישנות. הבוט רואה רק מה שמגיע מעכשיו.
- **הרשאות:** אם קלוד מבקש אישור לפעולה, הערוץ יכול להעביר לכם את השאלה. לעבודה בלי הפרעות: מצב acceptEdits/Auto, ולא `--dangerously-skip-permissions` (זה לחדר בידוד בלבד).
- **קבוצות:** אפשר לצרף את הבוט לקבוצה (מגיב רק ל-@אזכור כברירת מחדל). פירוט ב-ACCESS.md של הפלאגין.
- **בדיקה בלי טלגרם:** `claude --channels plugin:fakechat@claude-plugins-official` ← localhost:8787, צ'אט מדומה בדפדפן.
- **אלטרנטיבות:** דיסקורד (`discord@claude-plugins-official`, טוב לצוות), iMessage (מק בלבד, `imessage@claude-plugins-official`, כותבים לעצמכם).

מה זה שונה מ-Remote Control?

Remote Control = לראות ולנהל את **הסשן עצמו** מאפליקציית Claude (עם כל ההיסטוריה). טלגרם = ווקי-טוקי: הודעה נכנסת, תשובה יוצאת, ואתם בסביבה שאתם ממילא חיים בה. אני משתמש בשניהם: טלגרם למשימות קצרות "כשעולה לי רעיון", Remote Control כשאני רוצה לראות מה הוא עושה.

8.6 Routines ומשימות מתוזמנות - השעון המעורר

יש שלוש רמות:

רמה	איפה רץ	מתי להשתמש	איך
`/loop`	בתוך סשן פתוח	"תבדוק כל 5 דקות אם הפריסה נגמרה"	`/loop 5m בדוק את סטטוס הדיפלוי`. נעלם כשסוגרים
משימות מתוזמנות מקומיות	אפליקציית הדסקטופ (Settings ← Routines ← Local)	"כל בוקר ב-8, תסכם לי את התיקייה Downloads ותסדר קבצים" (יש לי כזה: `downloads-daily-cleanup`)	רץ רק כשהאפליקציה פתוחה. כל ריצה = סשן חדש. פספס - משלים כשהמחשב מתעורר

רמה	איפה רץ	מתי להשתמש	איך
Routines בענן	claude.ai/code/routines	"כל יום ראשון בבוקר: תעבור על PRs שנפתחו השבוע ותכתוב סיכום". גם טריגר API (webhook) או אירוע GitHub	`/schedule "כל יום ב-9",` או `...` מהאתר. רץ גם כשהלפטופ סגור

מה חשוב: רוטינה בענן רצה **בלי אישורים**. כלומר צריך מראש לתת לה קונקטורים וכלים. והיא לא דוחפת ל-main מוגן - היא פותחת ענף `claude/*` ואתם מאשרים PR.

דוגמאות שעובדות

- "כל בוקר: תקרא את AGENT_LOG.md ואת TASKS.md ותשלח לי בטלגרם 5 דברים שפתוחים היום." (זה שילוב של רוטינה + ערוץ.)
- "כל שישי: דוח אנליטיקס שבועי לאתר לפי המתכון בקובץ analytics_pull.md."
- חד-פעמי: `/schedule "מחר ב-14:00 תזכיר לי לשלוח את ההצעה לשרון"`.

8.7 אז מה לבחור? (טבלת החלטה)

אני רוצה...	פתרון
לשלוח משימה קצרה מהטלפון ולקבל תשובה	**טלגרם**
לראות את הסשן המלא ולנהל אותו מהטלפון	**Remote Control**
שמישהו "ינתב" משימות מבלי שאני אחשוב לאיזה סשן	**Dispatch**
עבודה ארוכה שתמשיך גם כשהמחשב כבוי	**Cloud session**
שמשהו יקרה לבד כל יום/שבוע	**Routine**
הכל ביחד: רוטינה שמדווחת לי לטלגרם	רוטינה מקומית + ערוץ טלגרם באותו מק

8.8 מה שנשאר פתוח לך מהפרק הזה

- [ ] BotFather ← טוקן ← `/telegram:configure` ← allowlist ← pair (5 דקות)
- [ ] להתקין את אפליקציית Claude בטלפון (אם עוד לא) ולנסות Remote Control פעם אחת
- [ ] להחליט אם רוצים Dispatch (טאב Cowork ← Dispatch ← צימוד לטלפון)

טיפים וטריקים עדכניים יולי-אוגוסט 2026

> **מה שהשתנה בחודשיים האחרונים, ומה שאנשים גילו שעובד. נכון ל-17.08.2026, גרסת Claude Code 2.1.224 אצלי (2.1.233 בחוץ). דברים פה משתנים כל שבוע - אם משהו לא עובד, `/help` או `claude --help` יגידו את האמת.**

9.1 חדש באפליקציית הדסקטופ (מהתיעוד הרשמי, אוגוסט 2026)

1. **שאלת צד בלי ללכלך:** `Cmd+;` או `/btw` פותח side chat שרואה את כל הסשן אבל לא נכנס אליו. "רגע, מה עושה הקובץ הזה?" בלי לשלם על זה בכל הודעה אחר כך.
2. **שני סשנים זה לצד זה:** Cmd+קליק על סשן בסרגל = נפתח בפאנל שני. `Cmd+\` סוגר. Ctrl+Tab מדפדף.
3. **סשנים מדברים:** "איזה סשן נגע ב-X?", "תגיד לסשן של האתר ש...". קלוד רואה 20 סשנים אחרונים של האפליקציה, קורא, שולח (מופיע ככרטיס עם קישור). שואל לפני ארכוב. גם בטרמינל `@שם-סשן` (2.1.232+).
4. **צ'יפים של משימות:** כשקלוד מזהה משהו מחוץ לסקופ - הוא מציע צ'יפ. קליק = סשן חדש ב-worktree משלו, הסשן הנוכחי ממשיך. (ככה פתחתי לך היום את התקנת שני הסקילים.)
5. **Continue in... Cloud:** מהסשן המקומי, שולחים לענן עם סיכום השיחה. דוחף ענף, יוצר סשן ענן. צריך working tree נקי.
6. **מעקב PR:** האפליקציה מראה סטטוס PR ו-CI בתוך הסשן.
7. **Computer use:** קלוד יכול לפתוח אפליקציות ולעבוד עליהן (מק ווינדוס). דפדפנים = צפייה בלבד (לזה יש Chrome), טרמינלים = קליק בלבד. אישור לכל אפליקציה, לסשן.
8. **worktree אוטומטי לכל סשן** בפרויקטי גיט. אין יותר "שני סשנים דרכו אחד על השני".
9. **פאנל Tasks:** רואים סאב-אייג'נטים, פקודות רקע ו-workflows בזמן אמת. Tasks ← Views.
10. **Dispatch מהטלפון** לטאב Cowork ← פותח סשן קוד עם תג. (פרק 8)

9.2 חדש ב-CLI ובמנוע (changelog יולי-אוגוסט)

1. `claude --resume <id>` **חוצה פרויקטים** (2.1.223+): מחפש קודם בפרויקט הנוכחי, אחר כך בכל המחשב.
2. **סאב-אייג'נטים ב-fork כברירת מחדל** (2.1.232): יורשים קונטקסט מבודד ונקי יותר.
3. `--worktree` **עם GitLab MR** (2.1.233).
4. **אזהרות תקרת הוצאה** מדויקות יותר ב-usage (2.1.225).
5. `/import` (2.1.213+): מייבא הגדרות מקודקס/כלים אחרים (AGENTS.md, MCP, סקילים) חד-פעמית.
6. **Fable 5:** המודל הכללי החזק ביותר של אנת'רופיק (הראשון במשפחת Claude 5, מעל Opus), זמין בקלוד קוד. `/model fable` . שימו לב למכסה - Sonnet ליומיום.

9.3 טריקים שחוסכים מכסה (הכי מבוקש)

1. `/effort` - low למכניקה, high/xhigh לחשיבה. חשיבה = טוקנים. אל תשלמו על חשיבה בשביל "תשנה את הצבע לכחול".
2. `/fast` - מהיר פי ~2.5 אבל יקר יותר לטוקן. למשימות שאתם מחכים להן בזמן אמת, לא לרקע.
3. `/insights` - דוח HTML על ההרגלים שלכם. תגלו שאתם שורפים 40% על סשן אחד שלא ניקיתם.
4. `/context` לפני שמתלוננים "קלוד איטי". פעמיים מתוך שלוש: MCP מיותר או שולחן של 180 אלף.
5. `/compact תשמור X` במקום `/compact` עירום - אחרת הוא בוחר מה לשמור.
6. **Plan mode + הודעה אחת ארוכה** > עשר הודעות קצרות. כל הודעה = כל השולחן מחדש.
7. **סשן ענן למשימות ארוכות** = לא תופס את הטרמינל, אותה מכסה, ולפעמים זול יותר כי אין "עוד הודעה קטנה" שלכם באמצע.
8. **הוק סינון פלט:** `PreToolUse` על Bash שמוסיף `| grep -E "FAIL|ERROR" | head -50` לריצות בדיקות. פלט של 5,000 שורות הופך ל-50.
9. **subscription vs API:** מנוי Max נגמר? אפשר להריץ קלוד קוד על מפתח API בתשלום לפי שימוש עד האיפוס. לא זול, אבל לא נעצרים ביום דדליין.

9.4 טריקים של עבודה חכמה

1. **"תבדוק את זה כאילו אתה מישהו אחר":** `/code-review` (עד `ultra` - סוכנים מרובים בענן) ו-`/security-review`. גם על HTML של הצעת מחיר. מבקר שני תמיד מוצא משהו.
2. `/loop 10m נגמר הרינדור בדוק אם` - לולאה בתוך סשן. `/loop` בלי מרווח = קלוד קובע קצב לבד.
3. `/schedule "תזכיר לי מחר ב-14:00 X"` - חד-פעמי, לא נספר במכסת הרוטינות.
4. `ultracode` במילה אחת בפרומפט = הרשאה לקלוד לפרוס workflow של סוכנים. למשימות ענק. אל תשימו בכל הודעה.
5. `/rewind` לפני שאתם מתווכחים. "חזור לפני שנגעת ב-header" ומתחילים נקי. יותר זול מלתקן.
6. `/branch` לנסות שני כיוונים עיצוביים על אותו בסיס בלי לאבד את המקור.
7. **צילום מסך = הפרומפט הטוב ביותר.** Cmd+Shift+4, Cmd+V לחלון. "ככה זה נראה, ככה אני רוצה". חוסך 200 מילים.
8. `claude-code-guide` - סוכן מובנה שקורא את התיעוד. "איך אני עושה X?" ← תשובה מהמקור, לא מהזיכרון של המודל.
9. `/fewer-permission-prompts` סורק היסטוריה ומציע allow-list. אחרי שבוע - חצי מהחלונות נעלמים.
10. `caffeinate -i claude ...` כשמשאירים סשן לרוץ עם טלגרם/Remote Control. אחרת המק נרדם והבוט שותק.
11. `CCR_FORCE_BUNDLE=1 claude --cloud` - סשן ענן על תיקייה שאין לה GitHub (מעלה עד 100MB).
12. `claude --teleport <id>` - להביא סשן ענן חזרה לטרמינל.
13. **פלאגין** `ralph-loop` (רשמי): "תמשיך עד ש-X נכון" בלולאה עם בדיקות. למשימות שאפשר למדוד.
14. `security-guidance` מותקן = כל קוד שקלוד כותב עובר סריקה. אין סיבה לא.

15. **AGENTS.md ריק = בזבוז.** אם יש לכם קודקס, 5 שורות ב-`~/.codex/AGENTS.md` עם כללי הברזל שלכם. (משימה שלי מהיום.)

9.5 מה הקהילה מדברת עליו (יולי-אוגוסט 2026)

- **"Full Claude Code Course for Beginners - freeCodeCamp"** (אוגוסט) - קורס מלא, כולל סקילים וזרימות עבודה. טוב למי שרוצה 4 שעות רצופות.
- **Miles Deutscher (X)** - thread "Claude Code is going mega viral": מסנן את ה-1% ששווה ללמוד. שני חלקים.
- **Geeky Gadgets (X)** - 50 טיפים ליומיום, הרבה על `/context` ו-`/compact`.
- **Claude Code Tips (2026 Edition)" (X) +35"** - ממתחיל לפאוור-יוזר.
- **Nick Saraev / Edmund Yong (YouTube)** - הערוצים הנצפים ביותר על קלוד קוד ברבעון הזה.
- **קרפתי - LLM Wiki** (אפריל, עדיין הכי מדובר) - המוח השני שהסוכן מתחזק. פרק 7.
- **Codex 0.147** (7.8): `--approve-for-me` (מודל מאשר אישורים במקומכם), פלאגינים ניידים. **Codex Remote GA** (25.6). **איחוד Codex לתוך ChatGPT** (9.7). GPT-5.4 יורד מ-ChatGPT ב-31.8.
- **agentskills.io** - הסטנדרט לסקילים אומץ ע"י 40+ כלים. סקיל אחד, כולם.

איפה עוקבים: code.claude.com/docs/en/changelog (רשמי, כל גרסה), r/ClaudeAI, וה-X של אנתרופיק. אצלי: הכל נכנס דרך `link-to-knowledge` ל-B-brain.

9.6 הכלל האחרון

טיפ טוב הוא כזה שאחרי שבוע אתם לא זוכרים שהוא היה טיפ - הוא פשוט איך אתם עובדים. תבחרו שלושה מהרשימה. לא ארבעים.

שימושים: יומיומיים ומתקדמים (מתכונים)

> "היום לא בנינו Agent מושלם. בנינו סביבת עבודה שה-Agent יכול להבין, שיטה שחוזרת על עצמה ותוצר שאפשר לפתוח ולבדוק."

כל מתכון: מה מבקשים, מה מקבלים, מה החלק שאנשים מפספסים. אתם מוזמנים להעתיק את הפרומפטים כמו שהם.

10.1 יומיומיים (למי שרק התחיל)

1. לסדר תיקייה

```
תסתכל על התיקייה. אל תזיז כלום עדיין. תציע לי מבנה תיקיות הגיוני,
תגיד לי איזה קבצים נראים כפולים או זבל, ותשאל אותי על מה שלא ברור.
```

אחרי שאישרתם: "יאללה, תבצע. ואל תמחק - תעביר ל-_למחיקה." (ככה סידרתי 31 אלף קבצים ב-Downloads. עם CLAUDE.md שמסביר את הכללים לפעם הבאה.)

2. סיכום פגישה ← משימות ← יומן

```
תכתוב סיכום פגישה (החלטות, משימות, מי אחראי, תאריכים) - הקלטה_או_תמלול@.md
שמור כ-MEETING_SUMMARY_<לקוח>_<תאריך>.md. תוסיף את המשימות ל-TASKS.md.
לפני שאתה יוצר אירועים ביומן - תבדוק שהם לא קיימים.
```

מפספסים: את השורה האחרונה. (פרק 6.)

3. הצעת מחיר / מסמך ללקוח

"תכין הצעת מחיר לשרון לפי הסיכום" ← הסקיל `ey-ai-proposal` נטען, יוצא HTML ממותג. אצלכם: תבנו סקיל אחד למסמך שאתם מייצרים הכי הרבה. פעם אחת קשה, לנצח קל.

4. מייל / פוסט בקול שלכם

"תכתוב מייל תשובה ללקוח על בסיס @MEETING_SUMMARY.md, בטון שלי, קצר." שיהיה `voice-dna.md` שהוא קורא קודם. בלי זה תקבלו ChatGPT גנרי.

5. "מה זה הקובץ הזה?"

`/btw מה זה הקובץ config.toml ולמה הוא חשוב?` - שאלת צד. לומדים תוך כדי, בלי לזהם.

6. לקרוא PDF ארוך ולהוציא טבלה

```
@מאמר.pdf - שמור .עמוד ,ראיה ,טענה :טבלה תבנה CSV. סמן 'not reported' אם אין.
.אל תמציא. אם משהו לא בטוח - כתוב את זה
```

7. מדריך/מסמך שיוצא לאנשים

"תפרסם את זה כארטיפקט" ← לינק. או "תהפוך ל-docx" (הסקיל הרשמי).

8. תמונה/וידאו/פרזנטציה

"נאנו בננה: פרומפט לתמונה של..." / "תבנה מצגת מ-@(pptx) "brief.md" / "סרטון 20 שניות מהטקסט הזה" (hyperframes). כולם סקילים. הרעיון: **הסקיל יודע את הכלי, אתם יודעים מה אתם רוצים.**

9. יומן/מיילים

עם קונקטורים: "מה יש לי היום? תכין לי בריף עם המיילים הפתוחים מאתמול." אחר כך זה הופך לרוטינה של 07:00.

10. "מה זוכרים עליי?"

`/memory`, או "מה אתה זוכר על איך אני אוהב לעבוד?" - וגם "תשכח את X" / "תוסיף ל-CLAUDE.md ש-Y".

10.2 מתקדמים

11. פרויקט חדש מאפס, מסודר מהיום הראשון

```
לפי השלד X צור לי תיקיית פרויקט בשם: README, AGENTS.md, CLAUDE.md (מייבא AGENTS),
HANDOFF.md, TASKS.md, AGENT_LOG.md, docs/, raw/, output/, .claude/skills/, .gitignore.
git init + commit ב .ראשון–AGENTS.md חפש לפני שאתה יוצר (כללי ברזל ,תכתוב: מטרה
.רשום ללוג אחרי), מבנה תיקיות. אל תוסיף מפתחות
```

(פרק 7.4.) הפעם הראשונה 3 דקות. אחר כך - סקיל.

12. שני סשנים על אותו פרויקט (או קלוד + קודקס)

- סשן א' (Hub): "תכתוב HANDOFF.md עם מה שהחלטנו ומה קודקס צריך לבנות, כולל בדיקות קבלה ועצירת אישור."
- סשן ב' (או קודקס): "תקרא HANDOFF.md ו-AGENTS.md. תעבוד ב-worktree. אחרי כל שלב - שורה ב-AGENT_LOG."
- בסוף, סשן א': "תבדוק את מה שקודקס עשה. `git log -p` ותגיד לי מה חסר."

13. סוכן-מבקר קבוע

`.claude/agents/reviewer.md` (פרק 4) עם Read בלבד. לפני כל תוצר ללקוח: "`@reviewer` תעבור על ההצעה". זה ה-second pair of eyes שאין לפרילנסר.

14. קליטת ידע ← מוח שני ← סקיל

קישור מיוטיוב/X ← `link-to-knowledge` (מסכם ל-B-brain/knowledge). אחרי 3-4 כאלה על נושא: `knowledge-to-skill` ← SKILL.md. עשיתי את זה עם Seedance, עם HyperFrames, עם קרוסלות. **הזרימה: תוכן שצרכתם ← ידע ← יכולת.**

15. עבודה מהטלפון (יום עבודה שלם בלי לפתוח לפטופ)

בוקר: הסקריפט `start_telegram_claude.sh` רץ על המק. במהלך היום, בטלגרם: "תבדוק את האנליטיקס של השבוע ותשלח לי גרף", "תכין טיוטת תשובה ל-X ותשמור בטיוטות", "מה סטטוס ה-TASKS?". ערב: Remote Control לראות מה קרה. (פרק 8.)

16. רוטינה יומית שמדווחת

רוטינה מקומית 08:00: "קרא AGENT_LOG (10 אחרונות), TASKS (פתוחות), היומן להיום. כתוב בריף של 7 שורות ושלח לטלגרם." בדקתם ידנית פעמיים לפני שתזמנתם. "בריף בוקר הוא Routine טובה רק אחרי שהרצנו אותה ידנית ובדקנו את הפלט."

17. אתר/אפליקציה - מבנייה לפריסה

Plan mode ← "תבנה" ← פריוויו בפאנל (האפליקציה מריצה את השרת) ← "תצלם ב-3 רזולוציות ותתקן מה שנשבר" ← `/code-review` ← "תפרוס ל-Netlify". כל אחד מהצעדים - הודעה אחת. (ככה נבנה evyataryizhak.com, סבבים של "נדחף".)

18. מחקר רחב במקביל

"תחקור את X משלוש זוויות (טכנית, עסקית, מה הקהילה אומרת) בשלושה סאב-אייג'נטים במקביל, כל אחד כותב קובץ ב-research/, ואז תסנתז." זה בדיוק איך המדריך הזה נכתב היום: חמישה סוכני מחקר, חמישה קבצים, ואז כתיבה.

19. עבודה בדפדפן עם החשבונות שלכם

Claude in Chrome: "תיכנס ל-YouTube Studio, תוריד את דוח 28 הימים, שמור ל-analytics/". או העלאת 27 קבצים לדרייב דרך הממשק (עשיתי אתמול כי ה-MCP לא תמך בהעלאה). מתחילים ב-Ask before acting.

20. תוצר שמלמד

בסוף כל סשן משמעותי: "תכתוב case study של מה עשינו לפי התבנית `_BRICK_TEMPLATE.md`". אצלי כל סשן = לבנה בקורס. אצלכם = תיעוד שהעתיד-שלכם יגיד עליו תודה.

10.3 בדיקת ההצלחה (לכל מתכון)

לפני שאומרים "מעולה, תודה":

- **יש קובץ?** (לא רק תשובה יפה בצ'אט)
- **יש מקור?** (מאיפה זה בא)

- **מסומנת אי-ודאות?** ("not reported", "לא בטוח")
- **נשמר log?** (שורה ב-AGENT_LOG / commit)

"אם רק דבר אחד נשאר בזיכרון של המשתתפים, שתהיה השאלה הזו."

10.4 השבוע הראשון (תוכנית ל-7 ימים)

יום	עושים	תוצר
1	פותחים תיקייה קטנה בטאב Code, פרומפט read-only, `/init`	CLAUDE.md ראשון
2	`/plan` על משימה אמיתית, מאשרים, מבצעים ב-Accept edits	קובץ שאתם באמת צריכים
3	סקיל ראשון: "תארוז מה שעשינו אתמול לסקיל"	SKILL.md
4	קונקטור אחד (יומן או דרייב). "מה אפשר לקרוא, מה לשנות, מה דורש אישור"	בריף מהיומן
5	HANDOFF.md + TASKS.md + AGENT_LOG.md. סשן שני שקורא אותם	הסשן השני לא שאל כלום
6	טלגרם (5 דקות) או Remote Control	הודעה מהספה, תשובה מהמק
7	`/insights` + `/memory`. מנקים, מעדכנים CLAUDE.md	קלוד קצת יותר מחונך

בשבוע השני: Obsidian מעל המוח, סוכן-מבקר, רוטינת בוקר, קודקס במקביל.

"**תנו לעבודה שלכם סביבת עבודה שאפשר לחזור אליה.**" זהו. זה כל המדריך במשפט אחד.
