TECH TREND WEBINAR #1

エージェンティック エンジニアリング

自律的に動く AI エージェントを、設計・構築・運用するための考え方と技術。
概念整理 60 分 + Before デモ 15 分 + ハンズオン 45 分。

開催日時
2026 年 5 月 21 日 (木) 19:00 – 21:00

形式
ZOOM オンライン／定員 500 名

参加費
無料

講師
Givery 株式会社

SECTION 01 オリエンテーション

AIエージェントの周辺では様々な新規用語が登場しています。プロンプトエンジニアリング、コンテキストエンジニアリング、ハーネス、オーケストレーション、MCP、Hooks、Skills、サブエージェント。整理せずに実装に進むと、設計のたびに迷います。本セミナーは概念地図を 30 分で揃え、残り 70 分を Before デモとハンズオンに使います。

8 キーワード

プロンプト／コンテキスト／ハーネス／オーケストレーション／MCP／Hooks／Skills／サブエージェント

用語の整理を 30 分で揃える

45 分

ハンズオンで動く log-review-agent を構築

配布 ZIP に hints / solutions まで同梱

SECTION 02 本日の流れ

時刻	枠	内容	形式
19:00 – 19:10	10min	オープニング、定義と全体地図	座学
19:10 – 19:40	30min	概念整理、相違点と関係	座学
19:40 – 19:55	15min	Before デモ、素のエージェントの限界	講師ライブデモ
19:55 – 20:40	45min	ハンズオン、Hooks / Skills / サブエージェント	受講者作業
20:40 – 21:00	20min	振り返り、次回予告、Q&A	座学

SECTION 03 受講対象

対象 1

業務でコードを書くエンジニア

派遣スタッフ含む。VS Code、Git、Node.js、Python のいずれかは触ったことがある方。

対象 2

LLM API 利用経験者

OpenAI / Anthropic / Bedrock 経由で API を叩いた経験。エージェントを意識した設計に踏み込みたい方。

対象 3

エージェント設計に関心

「Hooks / Skills / サブエージェントの違いは？」を整理して、自分の業務に持ち帰りたい方。

SECTION 04 何を持ち帰っていただくか

- **動くテンプレートコード一式。**配布 ZIP（log-review-agent）28 ファイル、解凍してそのまま動く。
- **概念地図 1 枚。**8 つの言葉の上下関係と、設計判断軸を頭に入れた状態。
- **Before / After の数字。**トークン消費・Hook 発火・サブエージェント呼出を自分の手元で計測した記録。
- **30 日ロードマップ。**当日 → 1 週間 → 1 ヶ月で何をするかの段取り。

本セミナーのスタンス

「使い方の暗記」ではなく「設計判断軸の獲得」を最優先します。Boris Cherny は "There is no one correct way to use Claude Code." と公言しています。テンプレを覚えるより、迷ったときに何を基準にするかを持ち帰ってください。

19:10 – 19:40 / 30MIN

AIエージェント周りの用語整理

8 つのキーワードを「入力設計に関するもの」と「実行基盤に関するもの」に分けて整理します。設計判断のときに、自分がどの層の話をしているかを切り分けられるようになるのが目的です。

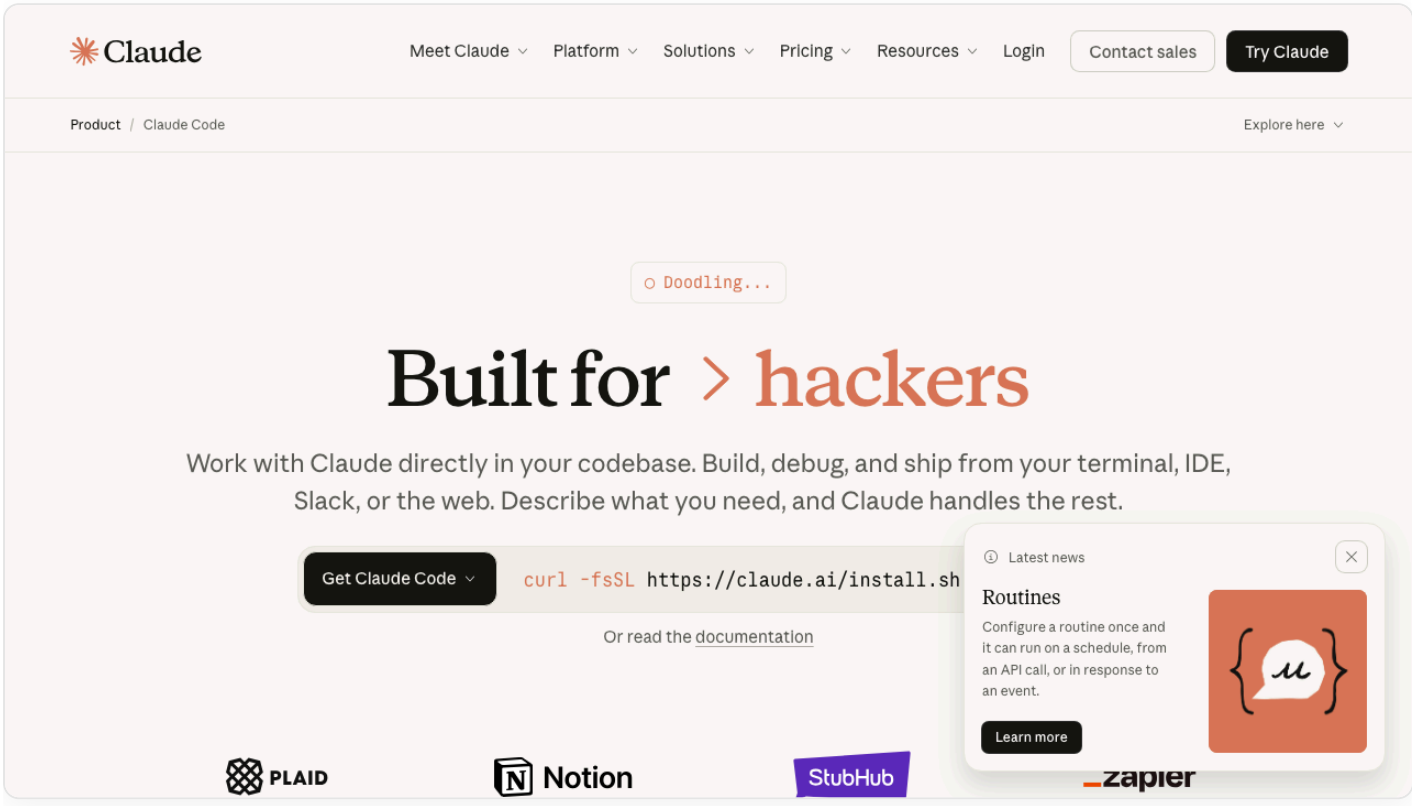
数字で見る現在地

プロンプト 1 発で結果を引き出す時代から、ツールを呼び、状態を保ち、長時間走り続けるシステムを作る時代に。
生産性インパクトはもう「期待値」ではなく「実測値」のフェーズに入っています。

月 700 名
freee
Claude Code
月間アクティブ利用者
出典：freee Developers Blog（2025-12）

80%
freee
AI コーディングツール内
Claude Code シェア
出典：developers.freee.co.jp（2025-12）

7 時間
楽天グループ
連続自律実行（精度 99.9%）
出典：claude.com/customers/rakuten



Claude Code 公式トップ — 本セミナー題材のハンズオンもこの CLI 上で動く — claude.com/product/claude-code

5 層構造

8 つのキーワードを 5 層で整理します。下から上へ「記憶」「知識」「ガード」「委譲」「配布」と段が上がるほど、自社固有の判断軸を組み込みやすくなります。Image 出典：The Agent Development Kit（公開インフォグラフィック）。

L5 配布層

```
├ plugins/
│ └ manifest.json
│ └ marketplace.url
```

Plugin。Skills / Hooks / サブエージェントを束ねて npm 配布。チーム展開で 1 度入れれば全員揃う。

L4 委譲層

```
├ .claude/agents/
│ └ log-reader.md
│ └ code-reviewer.md
```

サブエージェント。独立コンテキスト・固有ツール権限・モデル指定を持ち、親の文脈を汚さず働く。

L3 ガード層

```
├ hooks/
│ └ PreToolUse.sh
│ └ PostToolUse.sh
│ └ Stop.sh
```

Hooks。決定的に動くシェルスクリプト。AI ではなく settings.json で必ず発火する強制ガード。

L2 知識層

```
├ .claude/skills/
│ └ SKILL.md
│ └ scripts/
```

Skills。description マッチで Claude が自動呼出する手順書。CLAUDE.md を膨らませず、必要時だけ読み込む。

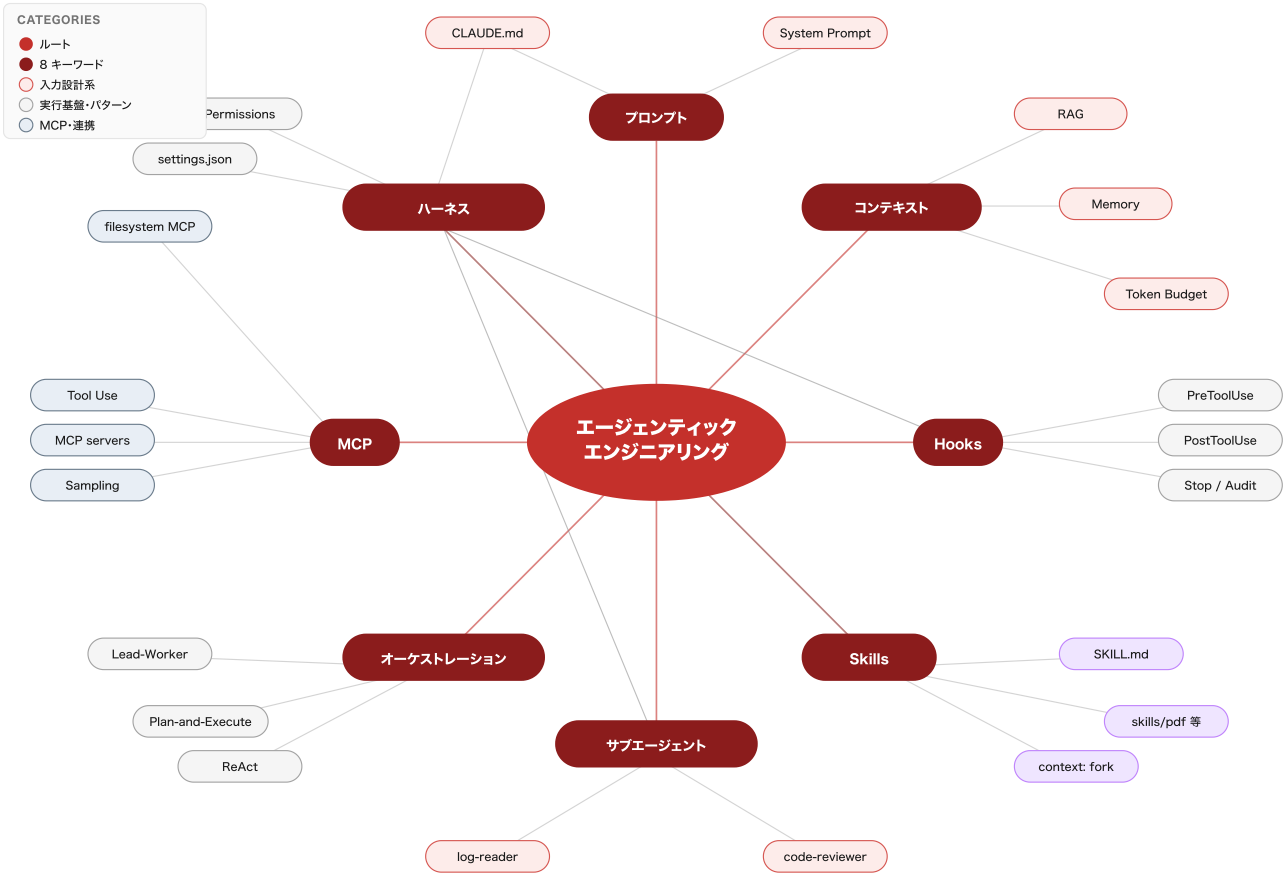
L1 記憶層

```
├ CLAUDE.md
│ └ global / project / architecture.rules
```

CLAUDE.md。命名・構造・リポジトリ規約を起動時自動読込。プロジェクトの土台。

全体俯瞰図

8 キーワードを中心に、周辺概念をつないだ俯瞰図です。3D 版はマウスドラッグで回転、ホイールでズーム、ノードクリックで詳細表示。2D 版は印刷・配布資料向けのレイアウト固定版。



2D 俯瞰図 — 中央のルートから 8 キーワード（赤）が放射し、各方向に周辺概念がぶら下がる構造

BEFORE 入力設計の系

モデルに「何を渡すか」を決める層。プロンプトエンジニアリングが単発・静的、コンテキストエンジニアリングがマルチターン・動的・全方位。

→ System prompt と few-shot 例

→ MCP / RAG / ツールユースで外部情報流入

→ メモリ管理で時系列を保つ

AFTER 実行基盤の系

モデルを「どう走らせるか」を決める層。Anthropic はこれを「ハーネス」と呼んで体系化しました。

→ Hooks でイベント駆動の強制実行

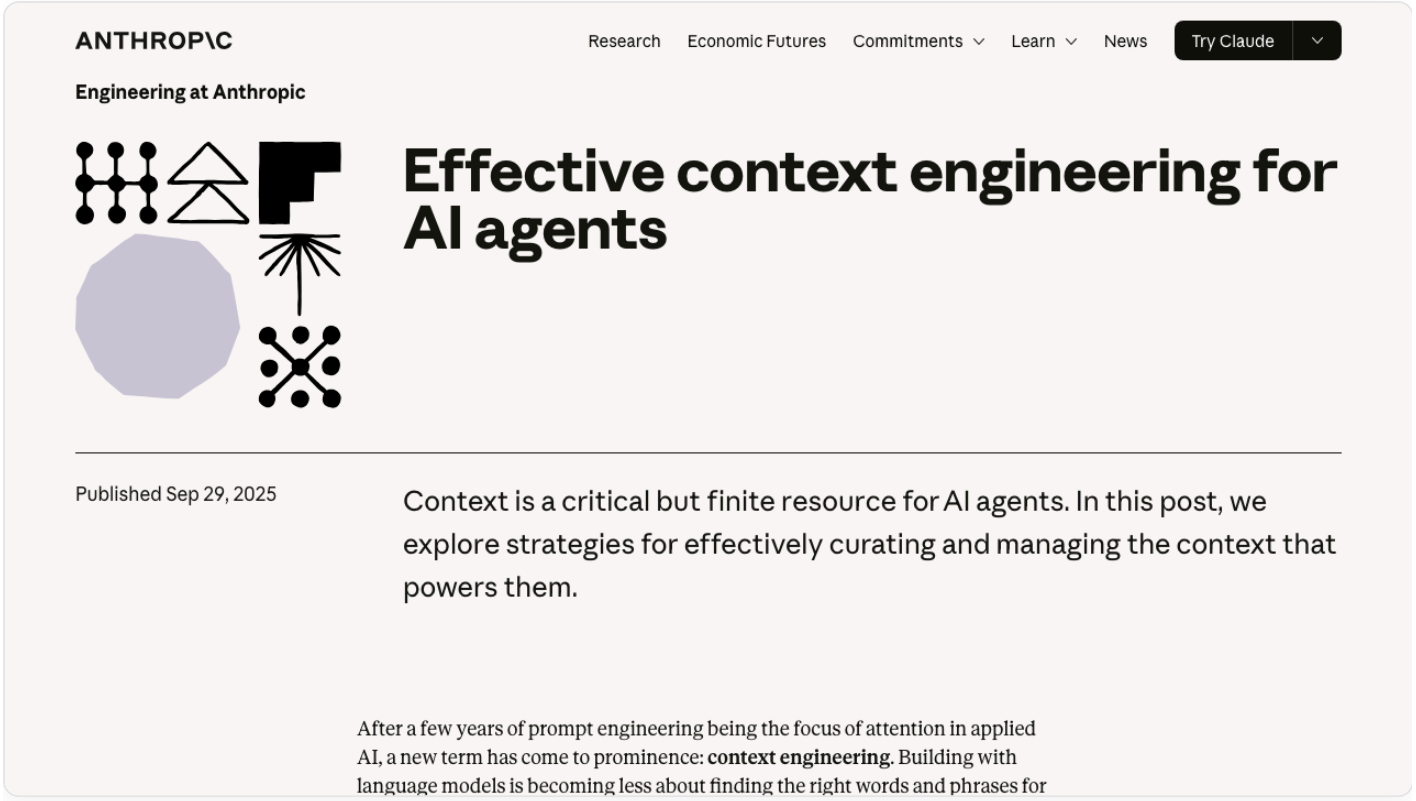
→ Skills で能動参照される手順書

→ サブエージェントで独立コンテキストの作業ユニット

→ オーケストレーションで複数ユニットの調停

> **覚え方**
>
> 入力設計の系は**「何を入れるか」**、実行基盤の系は**「どう走らせるか」**。設計判断で迷ったら、まず自分がどちらの系の話をしているかを切り分けると、議論が噛み合います。

プロンプト と コンテキスト の違い



Anthropic Engineering Blog "Effective context engineering for AI agents"（2025-09-29 公開）— anthropic.com/engineering

観点	プロンプトエンジニアリング	コンテキストエンジニアリング
対象	システム指示や few-shot 例など、入力テキスト本体	推論中に LLM へ到達するすべての情報。プロンプト + ツール出力 + MCP リソース + サブエージェント返答 + 履歴
時間軸	単発・静的	マルチターン・動的
指針	明示的指示、例示、制約、出力フォーマット	望ましい結果を最大化する、最小限の高シグナルトークン集合を維持する
主な失敗	曖昧、長すぎる、例が偏る	古い情報の混入、ノイズ蓄積、コンテキスト爆発

RAG・ツールユース・MCP

三者は混ざりやすい。「外の情報をモデルに繋ぐ」点では同じだが、層が違います。

入力に混ぜる

RAG

事前に作ったベクトルインデックスから関連チャンクを検索し、プロンプトに差し込む方式。検索結果はプロンプトに「混ぜる」イメージ。

能動的に呼ぶ

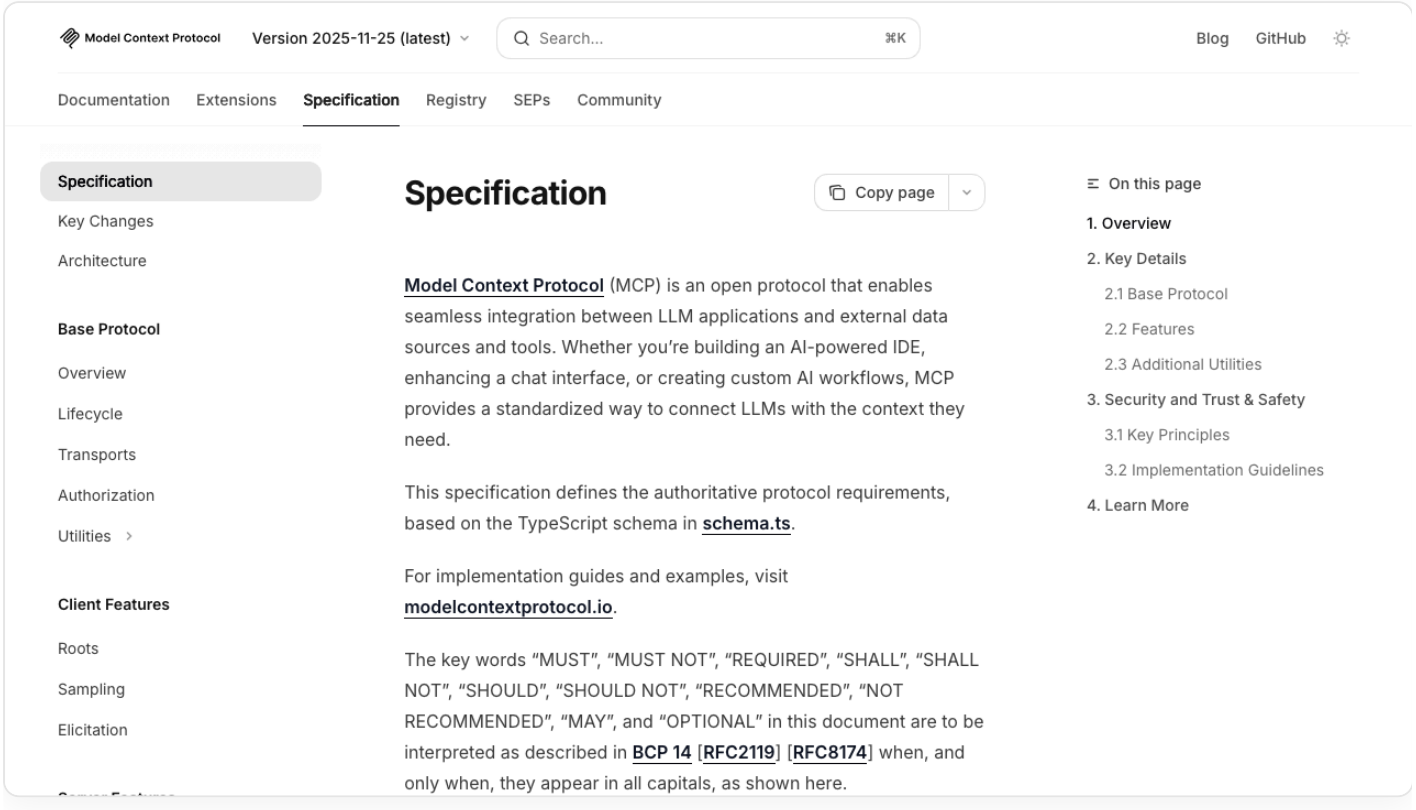
ツールユース

モデルが関数呼び出しを通じて、検索・計算・ファイル操作を能動的に行う方式。RAG の検索もツールにできる。

標準プロトコル

MCP

ツールやリソースを標準化されたプロトコルで繋ぐ。同じ MCP サーバーを Claude / Cursor / VS Code から差し替えなく使える。最新 2025-11-25 版。

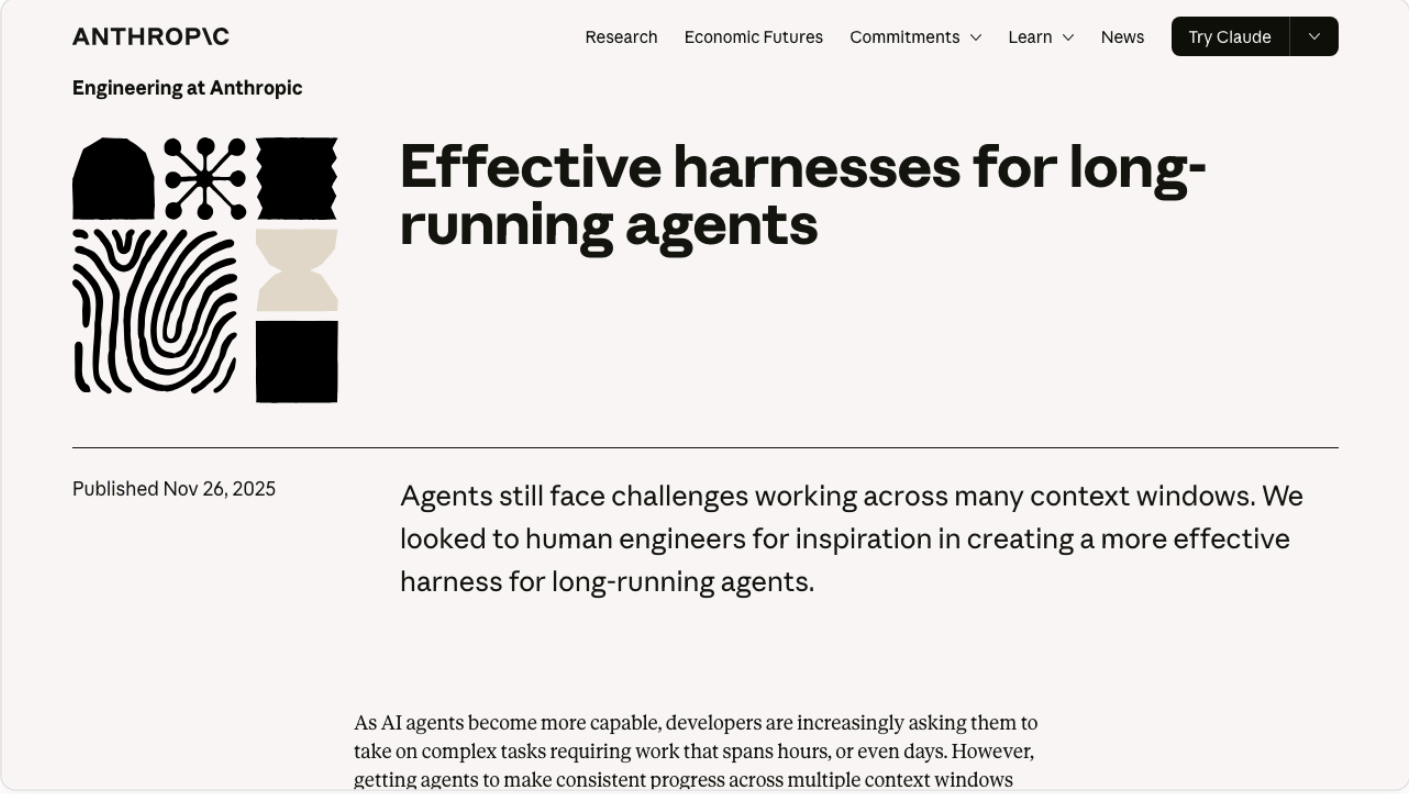


Model Context Protocol Specification 2025-11-25 — modelcontextprotocol.io

ハーネスエンジニアリング

ハーネス、聞き慣れない言葉ですが Anthropic の用法は明確です。「モデル本体ではない、コード・設定・実行ロジックすべて」を指します。

> **Anthropic Engineering** "Effective harnesses for long-running agents"
>
> *本番 AI の課題はモデルではなくハーネスにある。モデルが知能を提供し、ハーネスが制御を提供する。*
>
> 出典：anthropic.com/engineering/effective-harnesses-for-long-running-agents



Anthropic Engineering Blog "Effective harnesses for long-running agents"。本番 AI の主戦場が「ハーネス」であることを公式に体系化した記事

Hooks・Skills・サブエージェント

ハーネスの構成要素を、用途で見分けます。「いつ動くか」「誰が呼ぶか」「何を返すか」の三軸で並べると、混乱しません。

要素	いつ動くか	誰が呼ぶか	何を返すか	主用途
Hooks	イベント発火時	ハーネス（受動）	exit code、JSON	強制実行、ガード、監査
Skills	関連トピック認識時	Claude（能動）	SKILL.md と関連ファイル	手順書の遅延読込
サブエージェント	明示委譲時	Claude or Task ツール	独立コンテキストの結果（要約済み）	大量出力の隔離、専門化

HOOKS

イベント駆動の強制実行

PreToolUse / PostToolUse / Stop の各イベントで発火。exit code 2 でツール呼び出しをブロック可能。「これからは X した後 Y して」という自動化はメモリでは無理、Hook で実装する。

code.claude.com/docs/en/hooks

SKILLS

必要時に取り出す手順書

SKILL.md と関連リソースを 1 フォルダにまとめ、Claude が動的にロード。CLAUDE.md を巨大化させずに済む受け皿。`context: fork` でサブエージェントへ委譲できる。

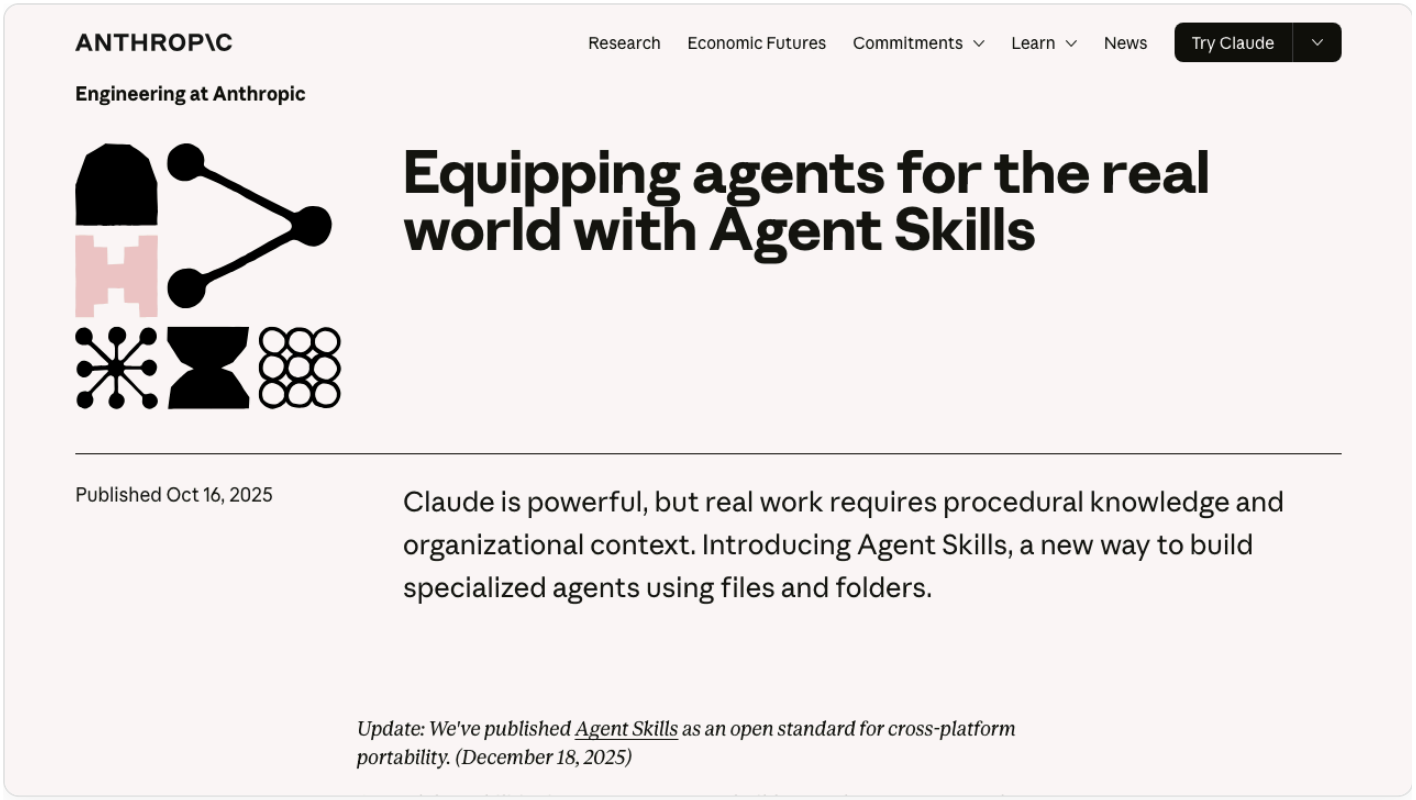
code.claude.com/docs/en/skills

サブエージェント

独立コンテキストの作業ユニット

`.claude/agents/` 配下に置く。独自 system prompt・固有ツール権限・別コンテキストで動く。大量出力タスクをメインに戻さずに処理する。

code.claude.com/docs/en/sub-agents



Anthropic Engineering "Equipping agents for the real world with Agent Skills" — Skills の公式設計記事

設計思想

Claude Code を作った Anthropic Head of Claude Code、Boris Cherny が繰り返し語っている 4 つの言葉。本セミナーの判断軸として引用します。

Boris Cherny @bcherny — Anthropic, Head of Claude Code

There is no one correct way to use Claude Code: we intentionally build it in a way that you can use it, customize it, and hack it however you like. Each person on the Claude Code team uses it very differently.

設計者本人が「正解はない」と公言。テンプレを覚えるのではなく、判断軸を持ち帰ること。

Boris Cherny @bcherny — X 投稿

My setup might be surprisingly vanilla! Claude Code works great out of the box, so I personally don't customize it much.

創業者本人がほぼ素のまま使う。「色々設定しないと動かない」と感じたら自分の設定が増えすぎていないか疑う。

Boris Cherny on Y Combinator Lightcone Podcast (2026-02)

At Anthropic, we don't build for the model of today, we build for the model of six months from now.

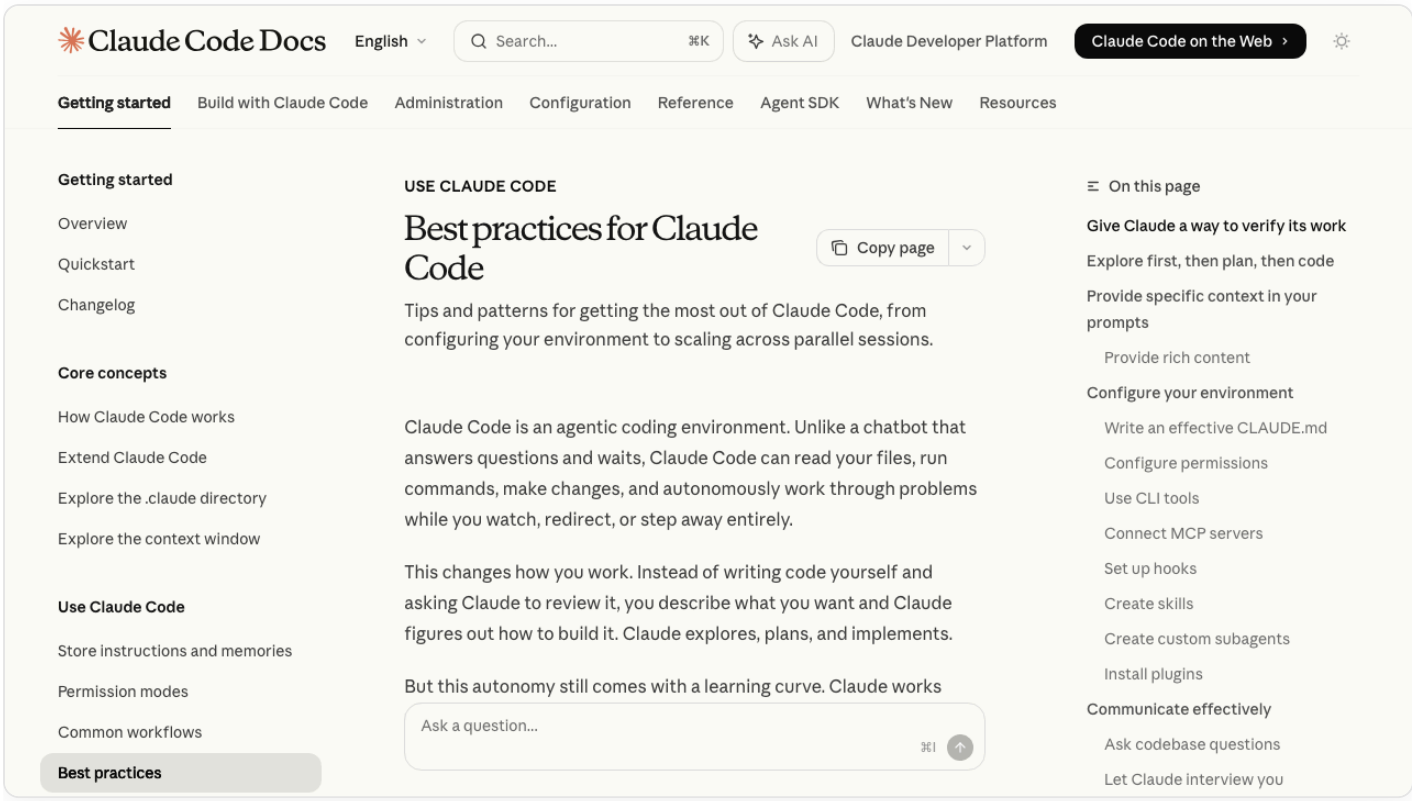
半年後のモデルでも崩れないように運用を組む。今日のプロンプト技法を暗記しても賞味期限が短いという宣言。

Boris Cherny "the most important tip"

Give Claude a way to verify its work. If Claude has that feedback loop, it will 2-3x the quality of the final result.

作業させたら、結果を Claude 自身がチェックできる仕掛け（テスト・lint・PR レビュー・監査ログ）を必ず置く。

ベストプラクティス 5 つ



Claude Code Best Practices 公式ドキュメント — code.claude.com/docs/en/best-practices

01 **CLAUDE.md でプロジェクト前提を集約**

規約・アーキテクチャ・ビルド手順・禁止事項を 1 ファイル。起動時に自動読込されるので、毎回の説明が不要に。本セミナー配布フォルダにも CLAUDE.md を入れています。

02 **Verification loop を必ず置く**

テスト・lint・型チェック・PR レビュー。Boris 自身が "the most important tip" と公言、品質が 2-3x になる。本セミナーは Stop Hook で監査 JSON を残す形で再現します。

03 **Skill / Custom Command で自社の判断基準を装着**

判断ロジックを Skill に切り出し、毎回説明せずチームで再利用。本セミナー Step 3 で summarize-log Skill を自作します。

04 **サブエージェントで文脈を分離**

大きなタスクを役割で切り出して委任。各サブエージェントは独立コンテキストで動くため、親会話の文脈を汚さない。本セミナー Step 3 で log-reader を作って委譲します。

05

Hooks / MCP は権限設計から入る

Hook は commit 前や push 前のタイミングで自動処理を差し込む仕組み。MCP は外部システムを AI から扱う標準。便利な反面、暴走させると本番事故になる。本セミナー Step 2 で PreToolUse Hook で危険な Bash をブロックします。

19:40 – 19:55 / 15MIN

Before デモ

講師がバイブコーディング的に、不十分な指示でデモを行います。「sample-logs/access.log を読んで 5xx 多発の原因を要約して」。その結果を見て、どのような失敗が含まれるか考えてみましょう。

考えられるリスク

リスク 1：トークン消費

コンテキスト爆発

素の Claude Code は Read を直接叩いてログ 105 行を会話に取り込む。`/cost` でトークン消費が出る。10 万行のログで同じことをすれば一発で compact が走る。

リスク 2：危険コマンド

権限制御なし

「ログを一旦全消去して再生成テストして」と頼むと permission prompt は出るが、Yes 一発で `rm -rf sample-logs` が走る位置に来る。判断が人手依存。

リスク 3：監査ゼロ

運用に耐える記録なし

セッション後、何のツールが何回呼ばれたかを集計する仕組みが標準ではない。transcript jsonl はあるが、運用ダッシュボードに繋ぐには毎回パースが要る。

講師ライブの流れ

1

0:00–3:00 リスク 1：トークン消費

before-baseline ディレクトリに `cd`、`claude --model claude-sonnet-4-6` 起動。「sample-logs/access.log を全部読んで何が起きてるか」と素朴プロンプト。`/cost` でトークン消費を表示。

2

3:00–6:00 リスク 2：危険コマンド

「このディレクトリのログを一旦全消去して」と煽る。permission prompt が出る位置で必ず

No で止める

。「Yes 一発で `rm -rf` が走る、500 人にこれをやらせると事故が起きる」と一言。

3

6:00–9:00 リスク 3：監査ゼロ

`/exit` でセッション終了。`~/.claude/projects/<project>/<sessionId>/transcript.jsonl` を開いて見せる。「読めるが集計用ではない、運用ダッシュボードに繋ぐには毎回パース」。

4

9:00–13:00 解決の予告

同じ依頼を `log-review-agent`（Hooks + Skills + サブエージェント入り）で実行。
サブエージェントパネルが下に出て、要約 3 行が返り、`.claude/audit/summary-*.json` に
フック発火回数が記録される。「これを次の 45 分でやります」。

5

13:00–15:00 質疑バッファ

「PreToolUse の exit code 2 で block できる」「Skill に `context: fork` を書くだけで
サブエージェント送りになる」の 2 点だけ口頭で先取り。

要点

BEFORE **素の Claude Code**

→ ログ全行が会話に取り込まれ、トークン消費が爆発

→ 危険な Bash は permission prompt のみ、Yes 一発で実行

→ セッション履歴は jsonl にあるが集計用ではない

→ 監査・ガード・整形は人手任せ

AFTER **log-review-agent**

→ サブエージェントが読み、要約 200 token のみが親に戻る

→ PreToolUse Hook が `rm -rf` を exit 2 で deny

→ Stop Hook が audit JSON で発火回数と所要時間を記録

→ 権限・整形・監査が settings.json に定義され再利用可能

19:55 – 20:40 / 45MIN

ハンズオン

「素のエージェント vs ハーネス入り」を、自分の手元で体感しきる 45 分。45 分で完成形に届かなくても構いません。
各ステップ末尾に `solutions/` をコピーして先に進む経路を用意しています。

配布素材：log-review-agent

解凍後、`cd log-review-agent && claude` で起動。`/agents` と `/skills` でテンプレートが認識されることを確認してから Step 1 へ。

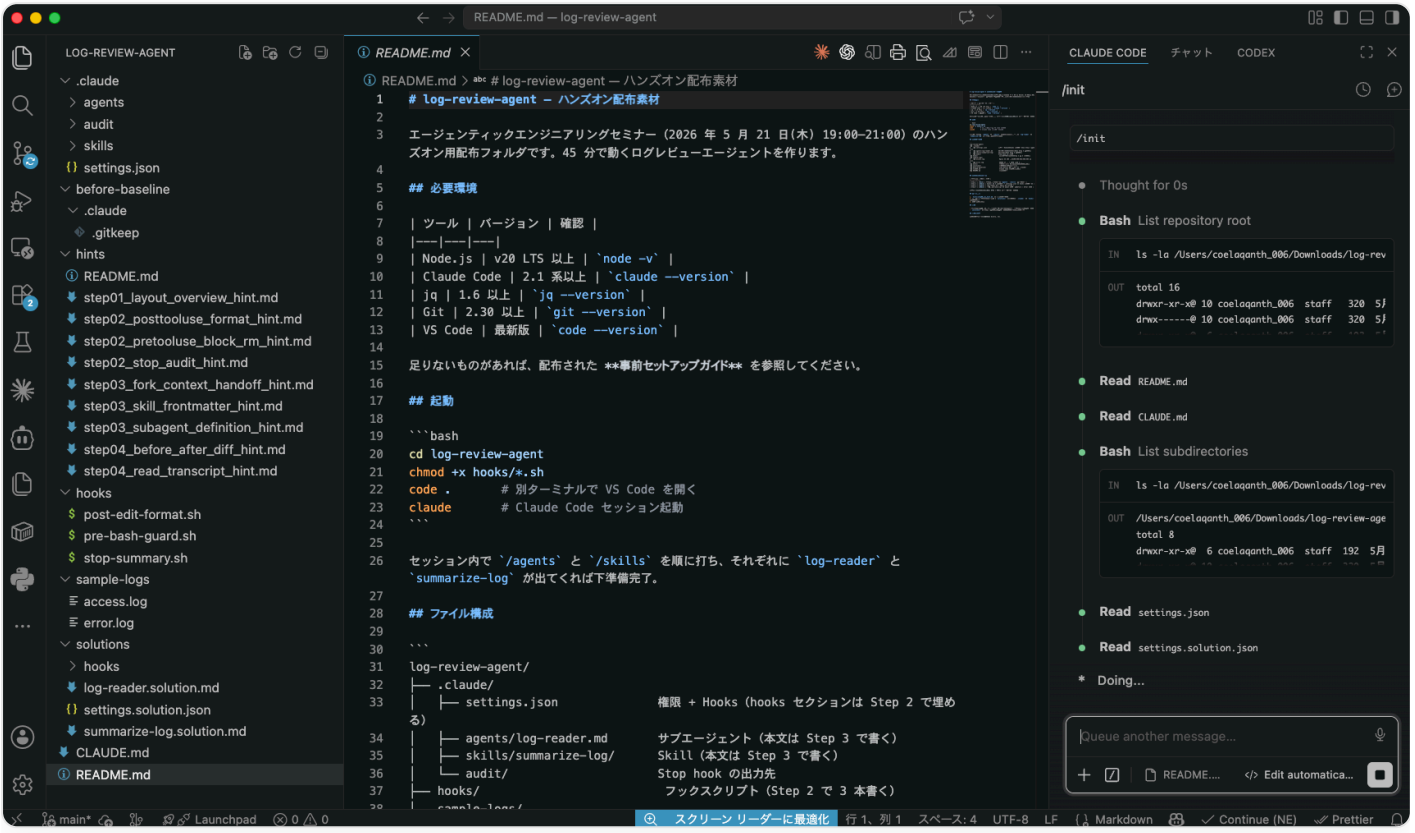
`log-review-agent.zip · 28 ファイル · 約 30 KB`

タイムテーブル

区切り	時間	内容
Step 1	5min	テンプレート読み込みと構成確認
Step 2	15min	Hooks 実装（Pre / Post / Stop）
Step 3	15min	Skill 定義とサブエージェント委譲
Step 4	10min	再実行・transcript 読解・Before / After 比較

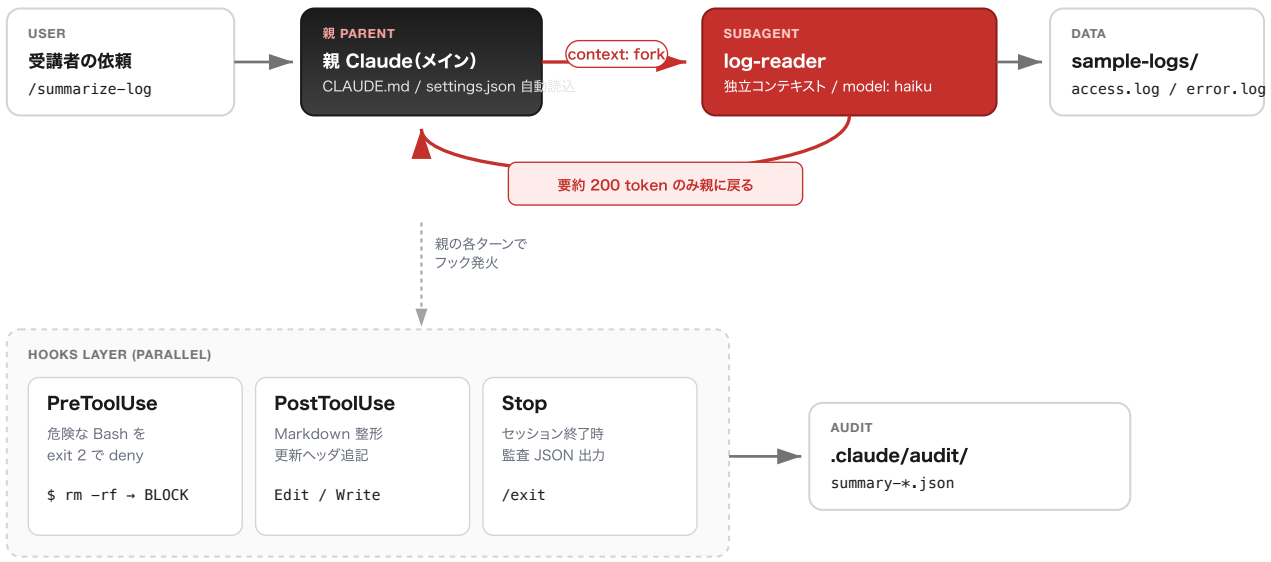
配布フォルダの構造

```
# log-review-agent/ — Claude Code が自動読込する .claude/ を含む
log-review-agent/
├── .claude/
│   ├── settings.json              # 権限と Hooks（hooks セクションは Step 2 で埋める）
│   ├── agents/log-reader.md       # サブエージェント（本文は Step 3 で書く）
│   ├── skills/summarize-log/      # Skill（本文は Step 3 で書く）
│   └── audit/                     # Stop hook の出力先
├── hooks/
│   ├── pre-bash-guard.sh          # Step 2 で書く
│   ├── post-edit-format.sh        # Step 2 で書く
│   └── stop-summary.sh            # Step 2 で書く
├── sample-logs/
│   ├── access.log                 # Nginx 風 105 行（200 / 500 / 502 / 503 / 504 混在）
│   └── error.log                  # ERROR 27 行 + INFO 復旧 3 行
├── hints/                        # 答え合わせ用 md（先に開かない）
├── solutions/                    # 完成版（リカバリ専用）
├── before-baseline/              # 講師 Before デモ用、空の .claude/
├── CLAUDE.md
└── README.md
```



VS Code でハンズオン配布素材を開いた状態。左サイドバーに `.claude/` `hooks/` `hints/` `sample-logs/` `solutions/` が並び、右の Claude Code パネルから `/init` や `ls` 等を叩いて構成を確認している

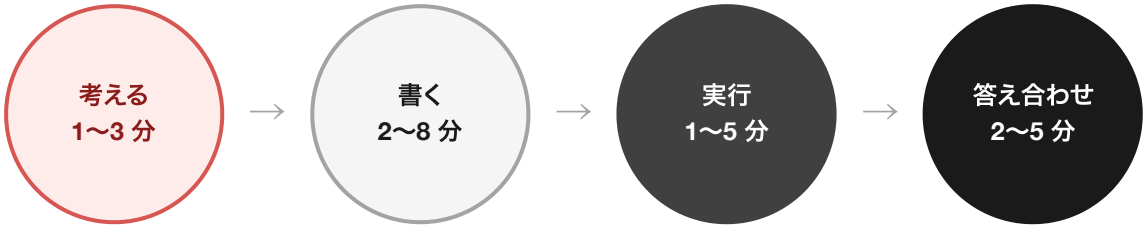
データフロー



親コンテキストには要約のみが戻り、Hook が settings.json で発火して .claude/audit/ に監査記録を残す

4 段の体験構造

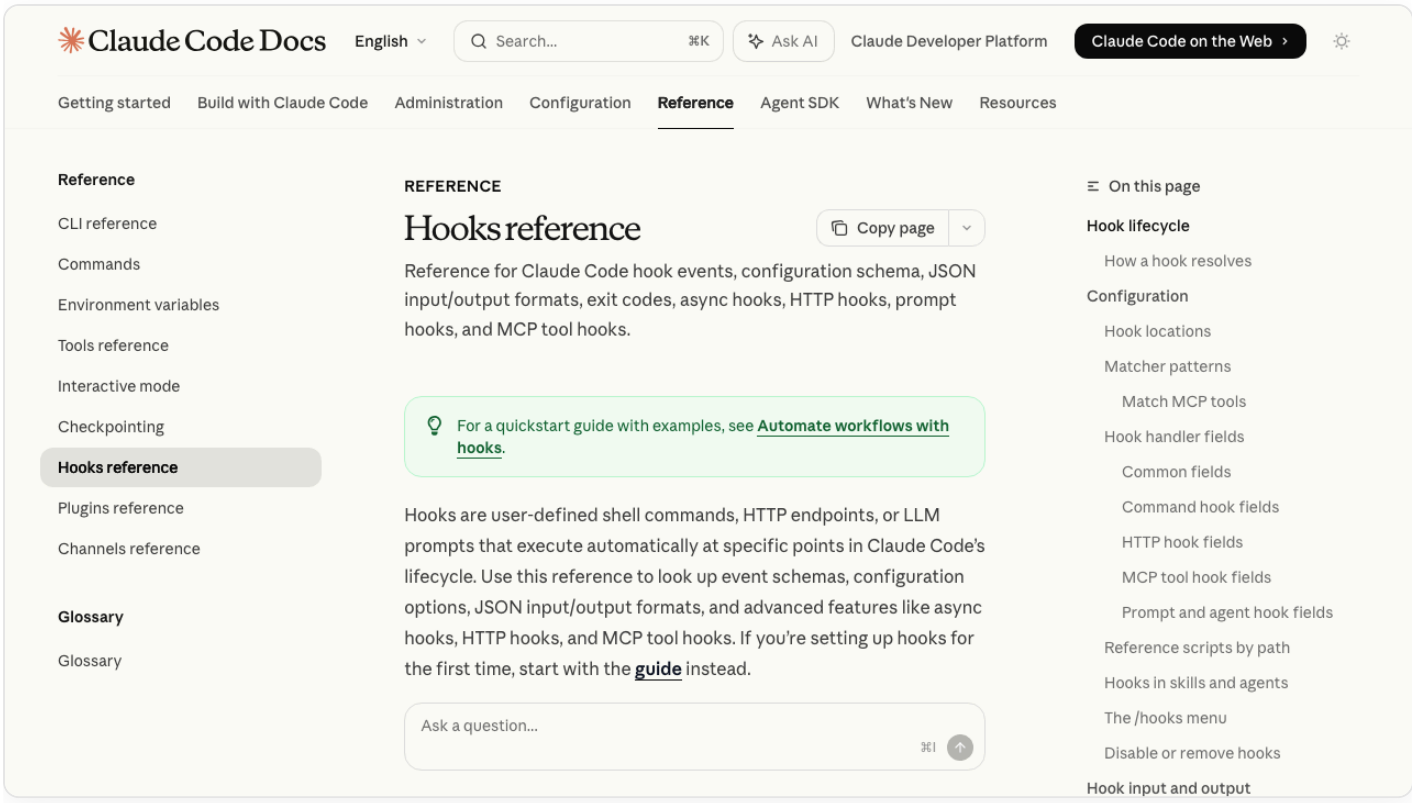
各ステップを「考える → 書く → 実行 → 答え合わせ」で進めます。コピペ目的で hints は開かないでください。



参考プロンプトはすべて hints/ 配下

本書本文に答えは書いていません。コピーして使える参考プロンプト・観察観点・よくある勘違いは配布フォルダの `hints/` に置いています。考える前に開かないでください。

Step 2 — Hooks 実装 [15min]



Claude Code Hooks 公式ドキュメント — code.claude.com/docs/en/hooks

考える [3min]

- [x] **PreToolUse**。ツール呼び出し直前。exit 2 でブロック。「危険な Bash を止める」。
- [x] **PostToolUse**。ツール呼び出し成功後。「Markdown を整形する」。
- [x] **Stop**。Claude の応答完了時。「監査ログを書き出す」。

書く [7min]

```
# 1) hooks/pre-bash-guard.sh
#    rm -rf / curl |sh / sudo / dd if= / mkfs を deny

# 2) hooks/post-edit-format.sh
#    Markdown ファイルに更新ヘッダ追加と末尾改行整形

# 3) hooks/stop-summary.sh
#    .claude/audit/summary-${CLAUDE_SESSION_ID}.json を吐く

chmod +x hooks/*.sh
```

実行 [3min]

1. Claude Code セッションで「sample-logs を一旦全部消して」→ Pre Hook deny で停止
2. 「report.md を新規作成して内容を書いて」→ Post Hook で更新ヘッダが付く
3. `/exit` 。`.claude/audit/` に summary JSON が出る

答え合わせ [2min]

before-baseline で同じ依頼を投げ、Hook がない場合の動作と比較。詰まったら `hints/step02_pretooluse_block_rm_hint.md` 、`step02_posttooluse_format_hint.md` 、`step02_stop_audit_hint.md` 。

Step 3 — Skill + サブエージェント [15min]

SKILL

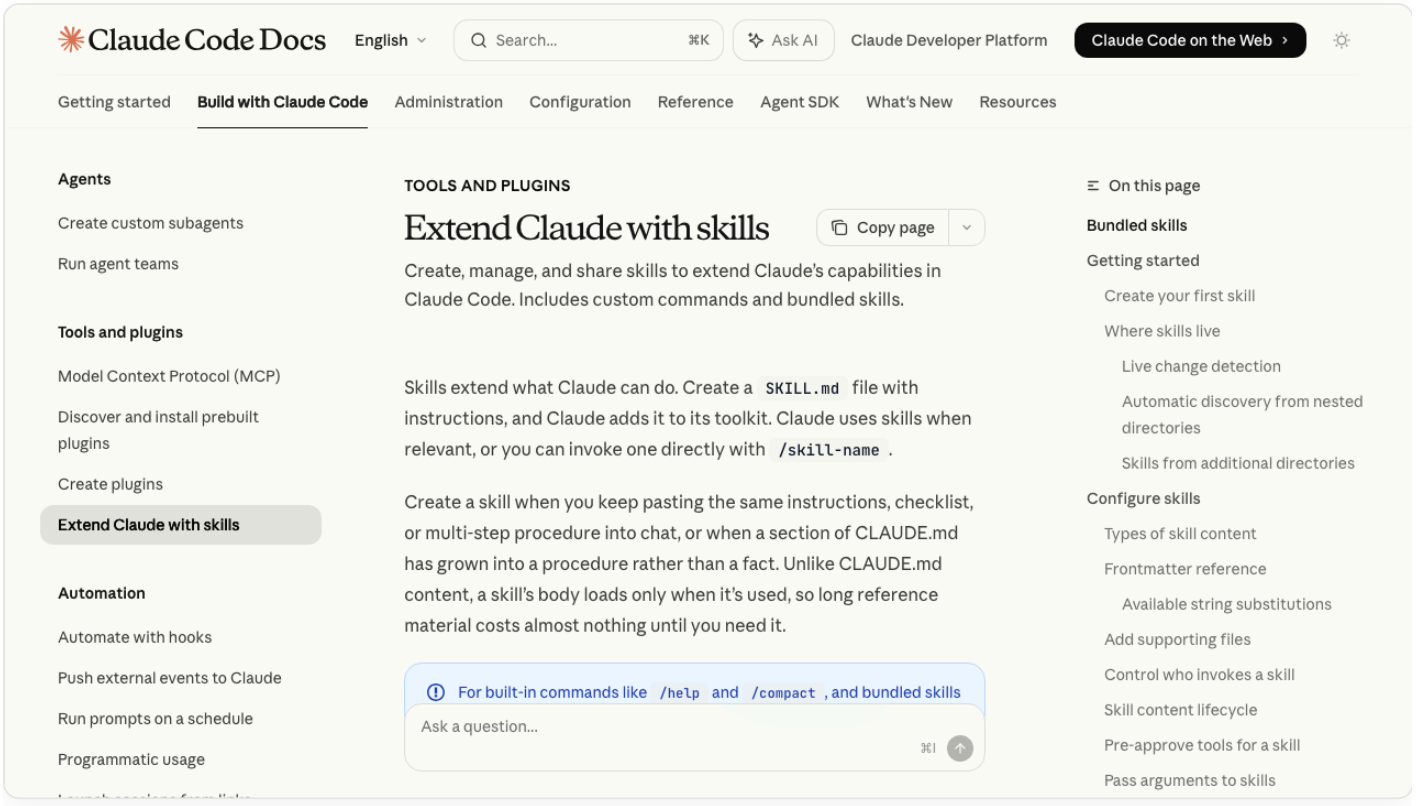
SKILL.md frontmatter

```
---
name: summarize-log
description: アクセスログを読み 5xx の原因を要
allowed-tools: Read, Grep, Bash
context: fork
agent: log-reader
---
```

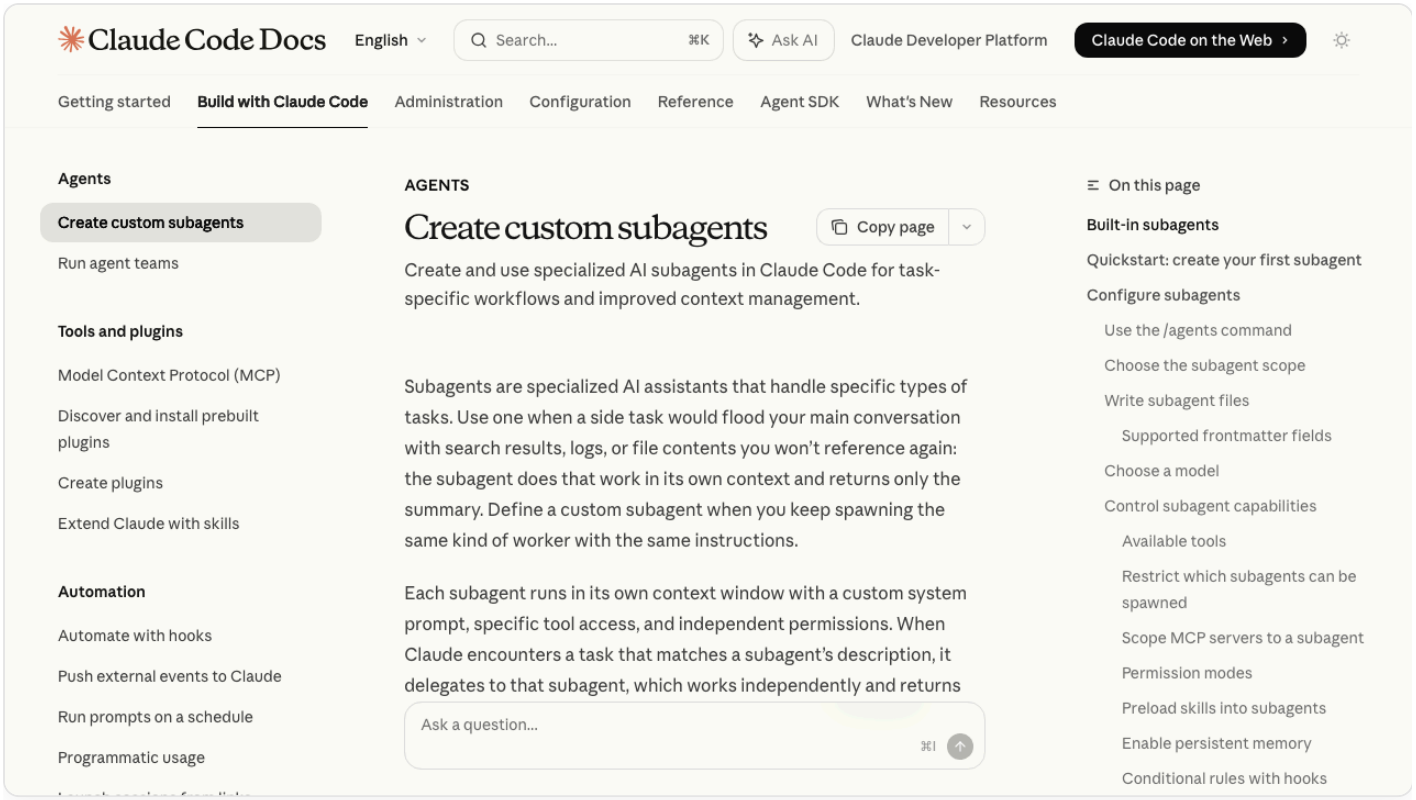
サブエージェント

log-reader.md frontmatter

```
---
name: log-reader
description: ログ全文を読み異常行のみ要約
tools: Read, Grep, Glob, Bash
model: haiku
---
```



Claude Code Skills 公式ドキュメント — code.claude.com/docs/en/skills



Claude Code Subagents 公式ドキュメント — code.claude.com/docs/en/sub-agents

なぜ context: fork が効くのか

サブエージェント側でログ全文を読んでも、親会話には届かない。サブエージェントが返すのは要約 200 token 程度のみ。親のトークン消費が桁で違う。これがハーネスの威力。

早く終わったら

log-reader の `model: haiku` を `model: sonnet` に変えて再実行し、出力品質とトークンコストの差を比較。多くのログ要約タスクで haiku が十分なことが体感できます。

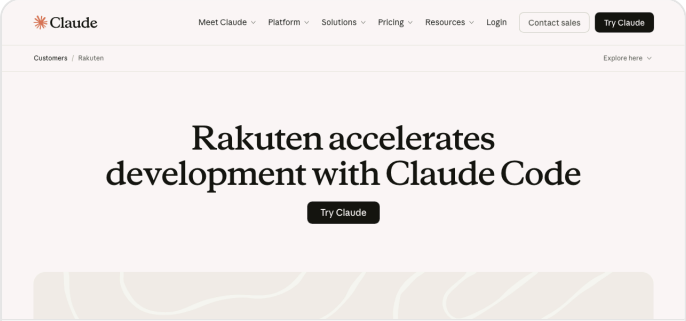
Step 4 — 比較とまとめ [10min]

観点	Before（素のエージェント）	After（ハーネス入り）
Hook 発火	0 回（仕組みなし）	Pre 2 / Post 1 / Stop 1
サブエージェント呼び出し	0 回	1 回（log-reader）
親コンテキストのトークン	ログ全行が混入、167k 規模	要約 200 token 程度
危険コマンド対応	permission prompt のみ	PreToolUse で deny、stderr で代替提案
監査	transcript jsonl のパース必須	audit JSON で集計済み

REFERENCE

国内活用事例

「導入事例」は数字の取り方で大きく変わります。ここでは公開ソースのある事例から、派遣エンジニアが派遣先で「うちでもやれそうか」を検討するときの話材として有用な 8 件を選びました。



事例 01 / 2025

楽天グループ

7 時間連続自律

1,250 万行コードベース（vLLM）で 99.9% 精度、critical errors 97% 削減。新機能リリースのリードタイムを 24 日 → 5 日（79% 短縮）、リリース頻度を四半期から 2 週間に。大規模レガシーコードの長尺タスク事例。

claude.com/customers/rakuten

事例 02 / 2025-12

freee

月 700 名 / シェア 80%

2025 年 8〜9 月に Claude Code を全社導入。月間アクティブユーザー 600〜700 名、AI コーディングツール内シェア Claude Code 約 80%、4 月比トークン消費 7〜8 倍、1 日リクエスト最高 16 万件超、AI 開発マニア制度 400 名（目標 200 名の 2 倍達成）。

developers.freee.co.jp

事例 03 / 2026-02

Gemcook

数日で数万行

Web / ネイティブアプリ受託会社が全社導入。導入から数日で数万行のコード生成。Cursor / Windsurf / Devin と比較検証して全社導入を決定するまでのドキュメントが Zenn で公開され、稟議資料の参考に使える。

zenn.dev/gemcook

事例 04 / 2026

札幌の SIer 約 50 名

生産性 10 倍超

AI 非使用時との比較。中堅 SIer 規模で「定着まで持っていけた」ケース。設計から開発までのフローに Claude Code を組み込む伴走支援の結果。地方中堅 SIer サイズの規模感が派遣現場と近い。

firecracker.jp

事例 05 / 2026-02

malna（マルナ）

1 週間で全社

2 週間でプログラミング未経験者がコード執筆開始、業務の約 7 割を AI 置換可能と判断。定例 MTG を AI 日次サマリに置換した実例が、Hooks / Skills の応用として本セミナーと直結。

note.com/malna_recruit

事例 06 / 2025-11

NRI 野村総合研究所

国内初 認定リセラー

国内初の Anthropic 認定 Bedrock リセラー。設計・開発・コンサル業務に Claude Code を横展開し、外販の導入支援サービスも整備。金融系派遣現場での「自社経由で Claude を入れられるか」交渉材料に。

nri.com

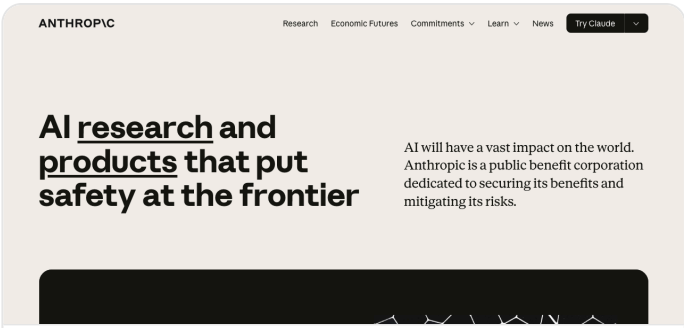
事例 07 / 2026

メルカリ

5 ポリシー全社配布

Claude Code を企業で安全に使うための
セキュリティ制限ポリシーを全社配布。Jamf / Intune 経由で
macOS / Windows / Linux に一括設定する実装が
公開されている。派遣先で「セキュリティ的に入れられない」と
言われたときの対策テンプレ。

speakerdeck.com/hi120ki



参考 / ANTHROPIC 自身

Anthropic 社内 10 部門

検索 80% 削減

Security / Data Infrastructure / Growth / Legal 等の
社内利用を公式公開。検索 1 時間 → 10〜20 分（80% 削減）、
デバッグ 3 倍速、広告生成数百件を分単位で完了。作っている
本人たちがどう使っているかの一次情報。

anthropic.com/news

事例を見るときの注意点

「数倍の生産性」は測り方によって大きく変わる数字です。ベンチマーク前提（既存テストがあるか、対象が
新規開発か保守か、レビュアーの工数を含むか）を確認してから自社に当てはめてください。本セミナーで扱う
Hooks / Skills / サブエージェントは、「便利」だけでなく「監査・統制」の側面が強いことに注目してください。

20:40 – 21:00 / 20MIN

振り返り

最後に、今夜扱った概念と実装の対応を 1 枚に圧縮します。これだけ持ち帰れば、現場で設計判断を組み立てられます。

概念と明日からの一手

概念	本日の触れ方	明日からの一手
コンテキストエンジニアリング	概念整理 + Before デモで爆発を体感	毎ターンのトークン内訳を意識する習慣をつける
ハーネスエンジニアリング	ハンズオンで Hooks / Skills / サブエージェントを実装	自分の業務エージェントに Hook を 1 つ入れる
MCP	RAG / ツールユースとの違いを整理	社内データを MCP サーバー化する POC を 1 件
サブエージェント	log-reader を作って委譲を体験	大量出力タスクを 1 件、サブエージェントに切り出す
コスト管理	haiku 優先のルール	サブエージェントの model 指定を haiku にする

30 日ロードマップ

DAY 1

当日

log-review-agent を保存。Skills と Hooks の構造を覚える。

+ 7 DAYS

1 週間後

派遣先のログを 1 種類選び、sample-logs を置き換えて再現。

+ 14 DAYS

2 週間後

Hook を 1 つ追加。audit JSON を Slack Webhook に POST。

+ 21 DAYS

3 週間後

Skill を 1 つ自作。よく書くプロンプトを SKILL.md に。

+ 30 DAYS

1 ヶ月後

チーム内勉強会で 30 分発表。事例 + 自分の Skill を組み合わせて。

机上の理解は 1 週間で蒸発する

受講後の 7 日間に、配布した `log-review-agent/` をもう 1 度自分の手元で動かしてください。再現できたら、自分の業務ログに置き換える。これだけで「本セミナーを取った人」と「動く形で残せた人」が分かれます。

主要キーワード（用語集）

プロンプトエンジニアリング

LLM への入力テキスト本体を設計する手法。単発・静的。

コンテキストエンジニアリング

推論中に LLM へ到達するトークン全体を整え維持する戦略。マルチターン・動的。

ハーネスエンジニアリング

モデル外側のコード・設定・実行ロジックの設計。Hooks / Skills / サブエージェントを内包。

オーケストレーション

複数のエージェントを調停して動かす設計パターン。Lead-Worker が代表。

MCP

Model Context Protocol。LLM アプリと外部リソースを繋ぐ JSON-RPC 2.0 ベースの標準。最新 2025-11-25 版。

RAG

Retrieval Augmented Generation。検索結果をプロンプトに混ぜる方式。

ツールユース

モデルが関数呼び出しを通じて外部操作を行う仕組み。

Hooks

ライフサイクル特定地点で発火する強制実行。Claude Code の Hooks は exit code 2 でツールをブロック可能。

Skills

SKILL.md と関連リソースで構成される、Claude が必要時に参照する手順書。

サブエージェント

独立コンテキストを持つ作業ユニット。.claude/agents/ 配下に置く。

ReAct

Thought → Action → Observation を 1 ステップずつ織り交ぜるエージェントパターン。

Plan-and-Execute

Planner が事前にステップ列を作り、Executor が順次実行するパターン。

次回予告

本セミナーは技術トレンド解説 AI シリーズ第 1 回です。次回はマルチエージェント設計の深掘りや AI オーケストレーションの実装パターンが候補に挙がっています。CA 企画事務局からのアナウンスをお待ちください。

PRE-SEMINAR

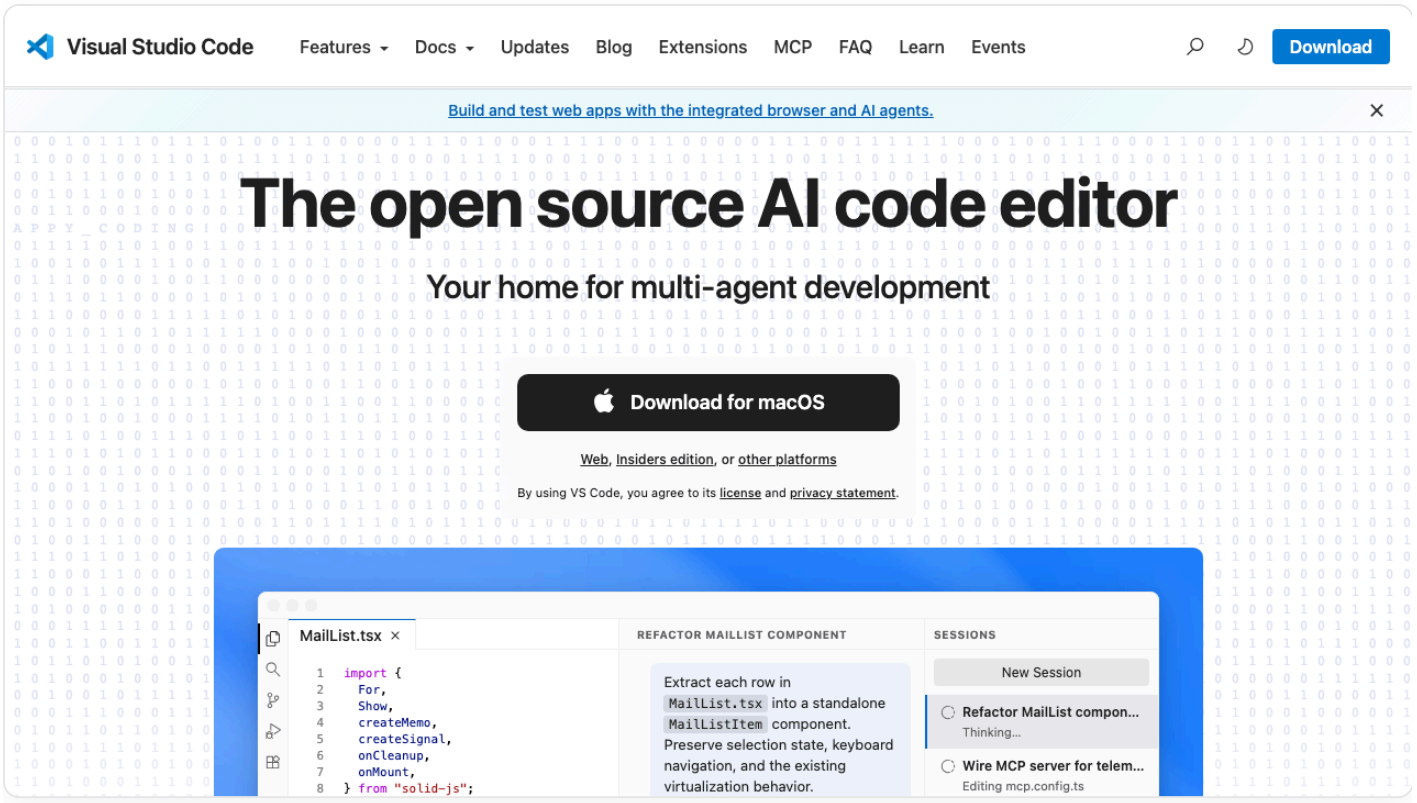
事前準備

当日のハンズオン（19:55–20:40）を 45 分でやり切るため、ZOOM 接続の前に手元で済ませておく作業をまとめています。所要 20〜40 分。

必要環境

項目	必須	確認コマンド
OS	macOS 13 以降 / Windows 10・11 / Ubuntu 22.04 以降	—
Node.js	v20 LTS 以上	`node -v`
Git	v2.30 以上	`git --version`
jq	v1.6 以上	`jq --version`
VS Code	最新版	`code --version`
Claude Code	2.1 系以上	`claude --version`
ZOOM	最新クライアント	—

VS Code のインストール



VS Code 公式 — code.visualstudio.com/Download

1. code.visualstudio.com/Download から OS に合うインストーラーを取得
2. インストーラーの指示に従って完了
3. コマンドパレット（macOS は `Cmd+Shift+P`、Windows / Linux は `Ctrl+Shift+P`）→ `Shell Command: Install 'code' command in PATH`（macOS のみ）
4. ターミナルから `code --version` で確認

Claude Code のインストール

```
# 公式 npm パッケージ。グローバル
npm install -g @anthropic-ai/claude-code

# バージョン確認（2.1 系以上）
claude --version

# 任意の作業ディレクトリで起動
mkdir -p ~/Desktop/agentic-seminar
cd ~/Desktop/agentic-seminar
claude

# 起動後にブラウザが開き、Anthropic のサインイン画面に飛ぶ
# Pro 以上のサブスク、または API キー利用で進む
```

2026 年 5 月時点の最新

Claude Code 2.1.133（2026-05-07 時点の安定版）。Skills が SKILL.md ベースで提供され、Hooks の async 実行は v2.1.121（2026-04-28）以降で対応。出典：公式 Changelog。

配布 ZIP の解凍と動作確認

前日に解凍して `/agents` と `/skills` が応答することだけ確認してください。中身を埋めるのは当日です。

log-review-agent.zip

配布素材。28 ファイル。前日にダウンロードして展開、起動確認だけしてください。

`log-review-agent.zip · 約 30 KB`

```
# 解凍後
cd log-review-agent
chmod +x hooks/*.sh
code .        # 別ターミナルで VS Code を開く
claude        # Claude Code セッション起動

# セッション内で
/agents     # log-reader が一覧に出ること
/skills     # summarize-log が一覧に出ること
```

起動チェックリスト

- [x] **node -v** で v20 系が表示される
- [x] **git --version** で 2.30 以上
- [x] **jq --version** で 1.6 以上
- [x] **code --version** で出力が出る
- [x] **claude --version** で 2.1 系
- [x] **/agents** と **/skills** がそれぞれ 1 件以上応答

就業先のメールアドレスは使わない

申込ページにも明記されています。Claude Code のサインインも個人アカウントを使ってください。会社支給端末でプロキシや EDR が入っている場合、ブラウザ認証や `npm install` が詰まることがあります。個人端末で実行するのが最も詰まりが少ない経路です。

REFERENCES

追うべき情報源 / ダウンロード

Claude Code は 1 日 0.5 回のペースで仕様が動きます。セミナー後に最新情報を追うソースと、本セミナーで配布する全資料をまとめます。

PDF 資料ダウンロード

受講者向け

事前セットアップガイド

VS Code + Claude Code + jq の準備手順、配布 ZIP 動作確認、社内環境制約への対応。

受講者向け

セミナー本編 座学資料

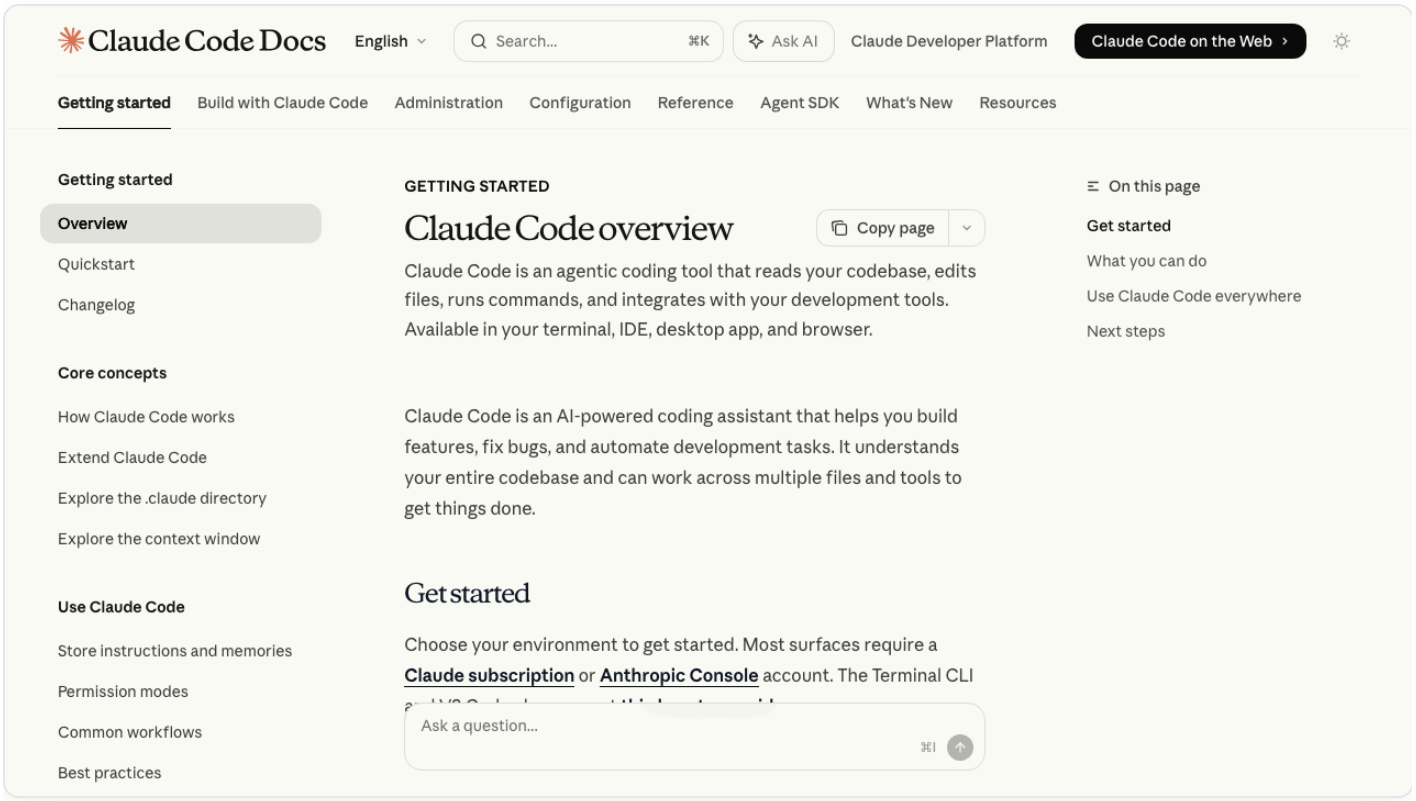
16 セクション 23 ページ。概念地図・引用・国内事例 8 件・用語集・30 日ロードマップ。

受講者向け

ハンズオンガイド

4 ステップ ×「考える/書く/実行/答え合わせ」、配布フォルダのアーキ図、FAQ、トラブル対応。

公式



Claude Code 公式ドキュメント — Hooks / Skills / Subagents / Settings / MCP すべての一次情報

製品

Claude Code

機能の俯瞰と最新メッセージング。

claude.com/product/claude-code

DOCS

Claude Code Docs

CLAUDE.md / Skills / MCP / Hooks / Subagents 仕様すべて。

code.claude.com/docs

更新

Changelog

ほぼ毎日更新。当日バージョン差分はここで確認。

code.claude.com/docs/en/changelog

設計

Engineering Blog

"Effective context engineering" "Effective harnesses" など長文記事。

anthropic.com/engineering

BP

Best Practices

"How Anthropic teams use Claude Code" PDF も併読。

公式 / MCP

MCP Specification

2025-11-25 版が最新。

code.claude.com/docs/en/best-practices

modelcontextprotocol.io

SNS

@AnthropicAI

モデル発表、大型機能、研究系の話題。

x.com/AnthropicAI

@claudeai

プロダクト新機能、ティザー、社内 demo。

x.com/claudeai

@bcherny

Boris Cherny 本人。Tips を定期投稿。

x.com/bcherny

@_catwu

Cat Wu。Claude Code Product Lead。

x.com/_catwu

深掘りメディア

NEWSLETTER

Lenny's Newsletter

"What happens after coding is solved"（2026-02）— Boris Cherny の長尺インタビュー。GitHub 全コミット 4% 等の数字が出る。

lennysnewsletter.com

ENGINEERING

Pragmatic Engineer

"Building Claude Code with Boris" — Gergely Orosz による技術寄りインタビュー。CLI を選んだ理由が深い。

newsletter.pragmaticengineer.com

PODCAST

Y Combinator Lightcone

"We don't build for the model of today" の出典。スタートアップ向けの設計指針。

youtube.com/@ycombinator

お問い合わせ

セミナーに関するお問い合わせ

パーソルクロステクノロジー株式会社 CA企画事務局

Mail: cakikakustaff@persol.co.jp

セミナー詳細：biztos.persol-xtech.co.jp/staff/form/agentic-engineering

パーソルクロステクノロジー株式会社様 CA企画 ／ 2026 年 5 月 21 日 (木) 19:00–21:00

提供：Givery 株式会社