Givery, Inc.

HANDS-ON GUIDE · 2026 年 5 月版

ハンズオンガイド

パーソルクロステクノロジー株式会社様 ／ 45 分でログレビューエージェントを作る

提供	Givery 株式会社
配布先	パーソルクロステクノロジー株式会社 御中
所要	45 分（19:55–20:40）
題材	log-review-agent（アクセスログ要約エージェント）
版数	v1.0（2026-05-09）

Hooks／Skills／サブエージェントを 1 つの動くコードで体験します。配布 ZIP の `log-review-agent/` を開いて手順に従ってください。

SECTION 01

1. 本日のゴールと進め方

「素のエージェント vs ハーネス入りエージェント」を、自分の手元で体感しきります。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。

到達点 1　PreToolUse Hook で危険コマンドが止まることを目視

到達点 2　Skill 経由でサブエージェントに委譲し、要約だけが親に返ることを目視

到達点 3　Stop Hook が監査 JSON を吐き、Before / After のトークン消費差を数字で確認

1.1 各ステップの体験は 4 段で進めます

考える

何を書くべきかを自分の頭で予想する。1〜3 分。

書く

自分の言葉でファイルに書く。2〜8 分。コピペ目的で hints は開かないでください。

実行する

Claude Code に投げて、出力と挙動を観察する。1〜5 分。

答え合わせ

外部の `hints/stepNN_xxx_hint.md` を開いて、自分のと並べて比べる。2〜5 分。

> **参考プロンプトはすべて hints/ 配下**
>
> 本書本文には答えを書いていません。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。

SECTION 02

2. タイムテーブル

区切り	時間	内容	本書セクション
Step 1	[5min]	テンプレート読み込みと構成確認	SECTION 04
Step 2	[15min]	Hooks 実装（Pre / Post / Stop）	SECTION 05
Step 3	[15min]	Skill 定義とサブエージェント委譲	SECTION 06
Step 4	[10min]	再実行・transcript 読解・Before / After 比較	SECTION 07

各ステップの最後 1 分は「ここまで詰まっている人は `solutions/` をコピーして先へ」と判断する時間です。考えるのは終わってから取り戻せます。

SECTION 03

3. 配布フォルダの構造

ハンズオン中はこの構造のまま VS Code で開いてください。

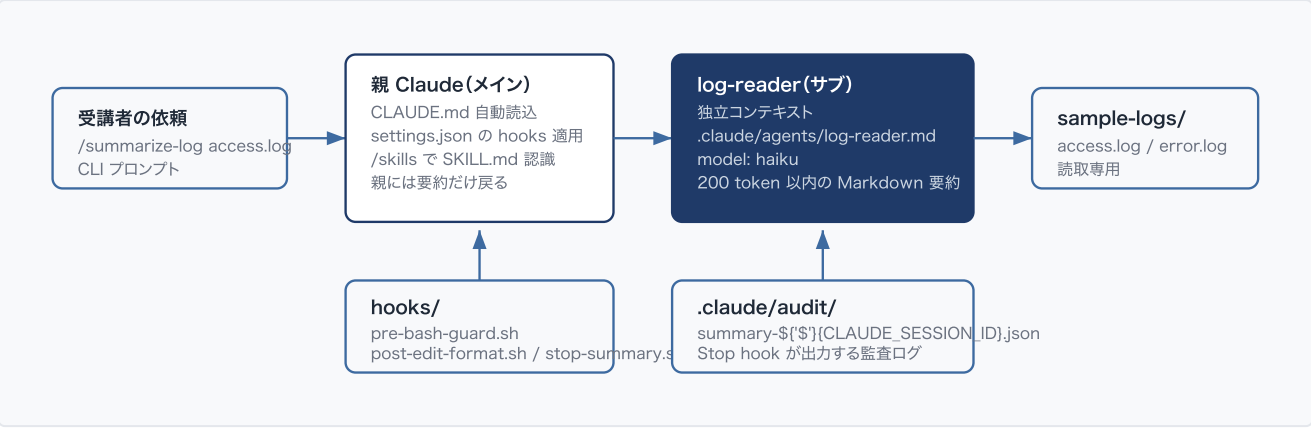


図 1：log-review-agent のデータフロー。親 Claude → log-reader(fork)→ sample-logs。Hook が親側の各イベントで発火し、監査と統制を司る

```
log-review-agent/
├── .claude/
│   ├── settings.json          # hooks 雛形（受講者が当日埋める）
│   ├── agents/
│   │   └── log-reader.md      # サブエージェント定義
│   ├── skills/
│   │   └── summarize-log/
│   │       └── SKILL.md       # Skill 定義（context: fork で委譲）
│   └── audit/                 # Stop hook の出力先（空、当日生成）
├── 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             # 5xx 混じり Nginx 風 200 行
│   └── error.log              # スタックトレース 3 件 80 行
├── hints/                     # 答え合わせ用 md（先に開かない）
│   ├── README.md
│   ├── step01_layout_overview_hint.md
│   ├── step02_pretooluse_block_rm_hint.md
│   ├── step02_posttooluse_format_hint.md
│   ├── step02_stop_audit_hint.md
│   ├── step03_skill_frontmatter_hint.md
│   ├── step03_subagent_definition_hint.md
│   ├── step03_fork_context_handoff_hint.md
│   ├── step04_read_transcript_hint.md
│   └── step04_before_after_diff_hint.md
├── solutions/                 # 完成版（リカバリ用、開封禁止）
│   ├── settings.solution.json
│   ├── log-reader.solution.md
│   ├── summarize-log.solution.md
│   └── hooks/
├── before-baseline/           # 講師 Before デモ用、空の .claude/
├── CLAUDE.md
└── README.md
```

solutions/ は最終手段

解凍直後に開かないでください。先に答えが目に入ると、考えて書く時間が形骸化します。詰まって 90 秒で進めない、と判断したらコピーで前進してください。

SECTION 04

4. Step 1 — テンプレート読み込みと構成確認 [5min]

配布フォルダの中身が「Hooks / Skills / サブエージェントのどれに対応するか」を頭の中で対応づけます。手を動かすより、目で構造をつかむフェーズ。

4.1 考える [1min]

VS Code のサイドバーで `.claude/` 配下を眺めて、agents/ と skills/ と settings.json のうち、どれが「Hooks」「Skills」「サブエージェント」に対応するかを頭の中でマッピングしてみてください。`hooks/` ディレクトリは別にあります。

4.2 書く [1min]

ターミナルでプロジェクトルートに移動して `claude` を起動。最初に下のコマンドを順に叩いてください。

```
cd log-review-agent
claude

# Claude Code 内のプロンプトで
/agents
/skills
```

4.3 実行する [2min]

`/agents` の出力に `log-reader` が、`/skills` の出力に `summarize-log` が並んでいることを確認します。並んでいなければ `.claude/` の場所が間違っているか、Claude Code のバージョンが古い可能性があります。

4.4 比較する [1min]

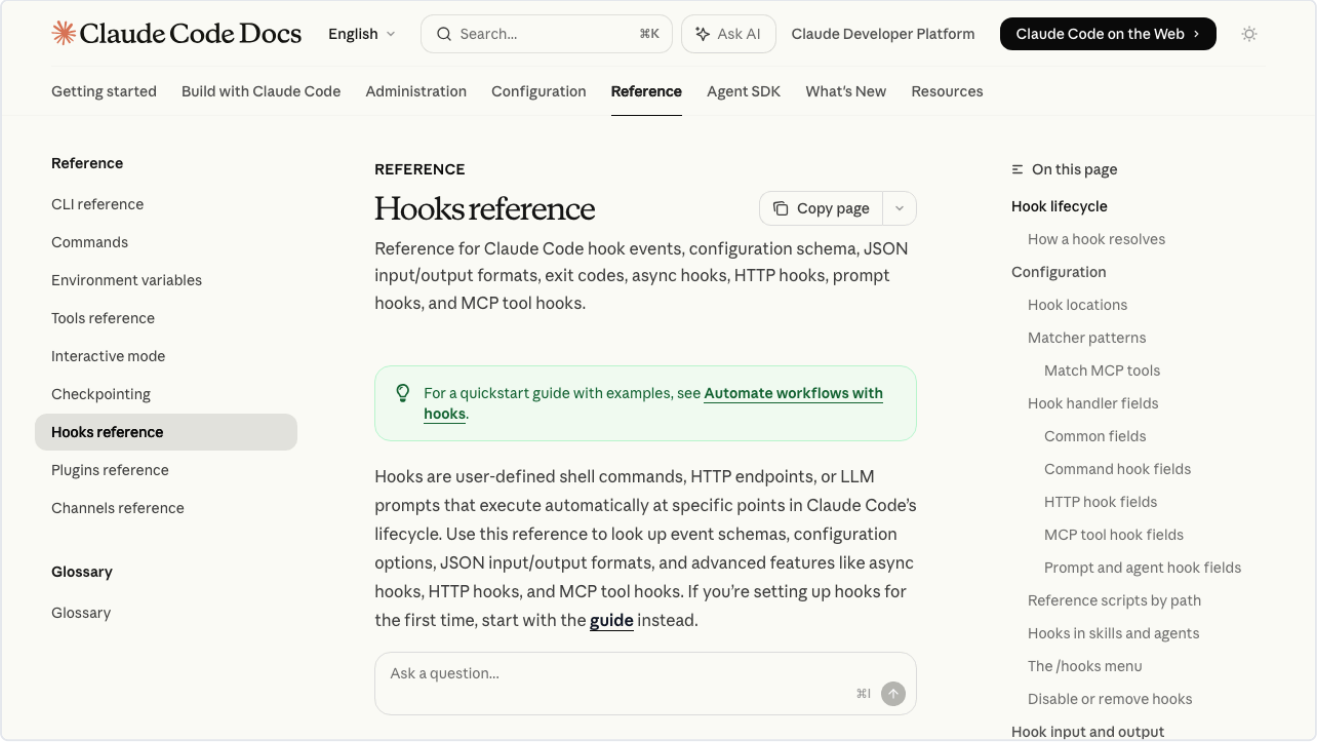
講師が画面共有で「素の `before-baseline/` ディレクトリ」と並べて、`/agents` と `/skills` の出力差を見せます。`.claude/` の有無で何が違うか、を 1 分で確認してください。

> **ヒント参照**
>
> ここで詰まったら `hints/step01_layout_overview_hint.md` 。プロジェクト構造を Claude 自身に説明させるプロンプトが置いてあります。

SECTION 05

5. Step 2 — Hooks 実装 [15min]

3 つの Hook を書いて、Claude Code に「使う前に止める」「使った後で整える」「終わったら集計する」の 3 種類を仕込みます。



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

5.1 考える [3min]

Pre
PreToolUse。ツール呼び出し直前に発火。exit 2 で呼び出しをブロック。「危険な Bash を止める」用途。
Post
PostToolUse。ツール呼び出し成功後に発火。「Markdown を整形する」「変更を Slack に流す」用途。
Stop
Stop。Claude の応答完了時に発火。「監査ログを書き出す」用途。
matcher を何にするか（Bash / Edit|Write / 全部）を考えてください。

5.2 書く [7min]

3 つのファイルを書きます。中身の細部は自分で考えて書いてみてください。

```
# 1) hooks/pre-bash-guard.sh
#    Bash の呼び出しが rm -rf や curl |sh を含むとき exit 2 で止める
#    (stdin から JSON を読み、jq で tool_input.command を見る)

# 2) hooks/post-edit-format.sh
#    Edit/Write 直後にレポート対象 .md があれば
#    末尾改行と "更新: $(date)" のヘッダを足す

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

chmod +x hooks/*.sh
```

続いて `.claude/settings.json` の hooks セクションを埋めます。matcher と command の対応を意識して書いてください。

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": ["Read(./sample-logs/**)", "Grep", "Glob"],
    "deny": ["Bash(rm -rf *)", "Read(./.env)"]
  },
  "hooks": {
    "PreToolUse":  [ /* matcher: "Bash" / command で pre-bash-guard.sh */ ],
    "PostToolUse": [ /* matcher: "Edit|Write" / command で post-edit-format.sh */ ],
    "Stop":        [ /* command で stop-summary.sh */ ]
  }
}
```

5.3 実行する [3min]

1
Claude Code セッション内で「sample-logs を一旦全部消して」と頼んでみる。Pre Hook の deny が効くと、Claude は別アプローチに切り替えるか、停止します。
2
「report.md を新規作成して内容を書いて」と頼む。書き込み後に Post Hook の整形が効いていればファイル末尾に更新ヘッダが付きます。
3
`/exit` でセッションを閉じる。`.claude/audit/` 配下に summary JSON が出ます。

5.4 比較する [2min]

`before-baseline/` でも同じ依頼を投げてみます。Hook がないので「全消去」は permission prompt 1 発で素通りしうる、Markdown 整形もされない、監査ログも残らない。差を目視で確認したら Step 3 へ。

> **期待される結果**
> 「sample-logs を一旦全部消して」と頼むと **blocked by pre-bash-guard.sh: rm -rf 系を禁止** と stderr に出て、Claude が別アプローチに切り替えます。`.claude/audit/` 配下に `summary-*.json` が生成されます。

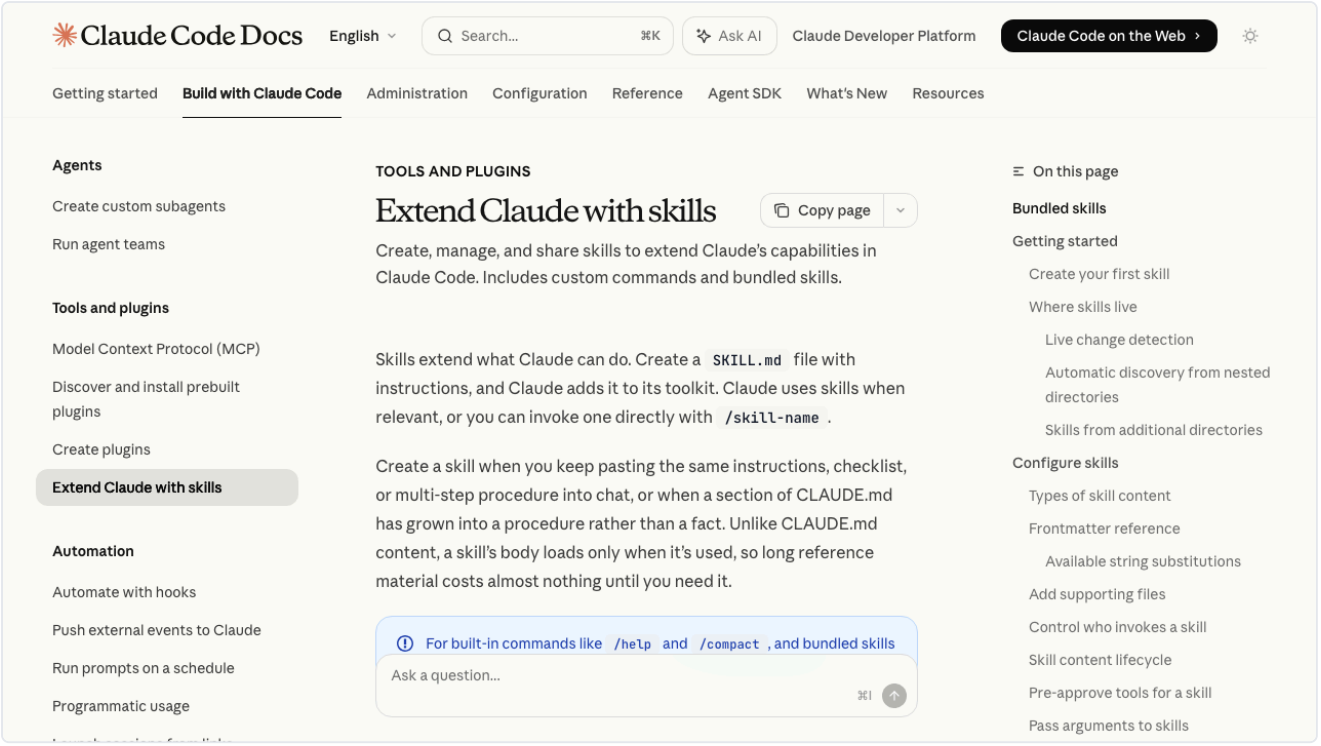
> **早く終わったら**
> PreToolUse の matcher を `"Read"` に変えてみて、`./secrets/` 配下の Read を deny する hook を書き足してみてください。「権限と Hook の重ね方」で防御層を厚くする練習になります。

> **ヒント参照**
> 詰まったら `hints/step02_pretooluse_block_rm_hint.md`、`hints/step02_posttooluse_format_hint.md`、`hints/step02_stop_audit_hint.md` の 3 本。各ファイルに参考プロンプト 3 つと、よくある勘違いが書いてあります。

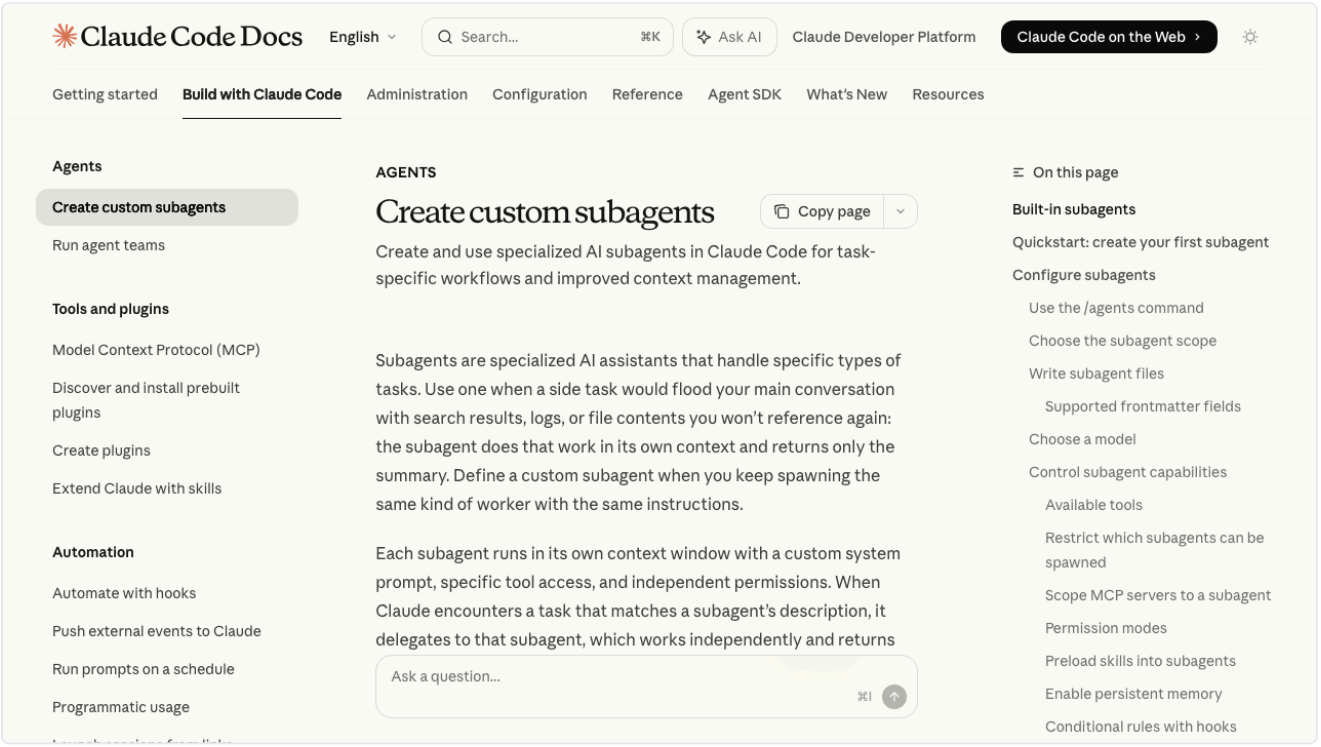
SECTION 06

6. Step 3 — Skill 定義とサブエージェント委譲 [15min]

「ログ全文を読む」のは大量トークンを食う仕事。これをサブエージェント側に切り出し、要約だけ親に戻す。Skill 経由で呼び出します。



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

6.1 考える [3min]

Skill とサブエージェントの分担を決めます。Skill = Claude のターン内で参照される「手順書」、サブエージェント = 別コンテキストで動く「委譲先」。`SKILL.md` の frontmatter に `context: fork` + `agent: log-reader` を入れると、本文の処理が log-reader 側で実行されます。

なぜ分けるのか

サブエージェントは別コンテキスト。読み込んだログ全文は親会話に戻らないので、トークン消費が劇的に減ります。Before デモで 167k → 12k 程度の差を講師が見せた、あのカラクリです。

6.2 書く [8min]

2 ファイルを書きます。

```
# 1) .claude/skills/summarize-log/SKILL.md
---
name: summarize-log
description: アクセスログを読み 5xx 多発の原因を要約する
allowed-tools: Read Grep
context: fork
agent: log-reader
---
1. wc -l でサイズを確認
2. status=5\d{2} を Grep で抽出
3. 上位 10 件と全体傾向を 3 行
4. Markdown で返す
```

```
# 2) .claude/agents/log-reader.md
---
name: log-reader
description: ログ全文を読み、異常行のみ要約して返す。Use proactively when asked to analyze access logs.
tools: Read, Grep, Glob
model: haiku
---
全文は決して呼び出し元に返さない。
最終出力は 200 token 以内の Markdown 要約のみ。
status=5xx を中心に、上位 10 件と日時帯の傾向を抽出する。
```

6.3 実行する [3min]

Claude Code セッションで Skill を起動します。

```
/summarize-log sample-logs/access.log
```

サブエージェント起動の transcript パネルが下に出るはず。完了すると親コンテキストには 3 行程度の要約だけが返ります。`/cost` で現在のトークン消費も見ておいてください。

6.4 比較する [1min]

同じログを「Skill 経由でなく、Claude に直接 `Read sample-logs/access.log` させて要約して」と素朴に頼む。トークン消費が一気に膨らみます。差は Step 4 で数字としてまとめます。

期待される結果

`/summarize-log sample-logs/access.log` を実行すると、サブエージェント起動の transcript パネルが下に表示されます。完了後の親コンテキストには **200 token 程度の Markdown 要約のみ**が返り、ログ本文は親に残りません。`/cost` で確認すると親消費が大幅に圧縮されています。

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してみてください。多くのログ要約タスクで haiku が十分であることが体感できます。

ヒント参照

詰まったら `hints/step03_skill_frontmatter_hint.md`、`hints/step03_subagent_definition_hint.md`、`hints/step03_fork_context_handoff_hint.md`。frontmatter の全フィールドと、context: fork の挙動が書いてあります。

SECTION 07

7. Step 4 — 再実行・ログ読解・Before / After 比較 [10min]

監査ログを集計し、Before との差を数字で確認して締めます。手元に「動くテンプレート」と「数字で語れる差」が残っていれば、このセミナーは成功です。

7.1 考える [2min]

transcript の保存場所を予想してください。Claude Code は `~/.claude/projects/<project>/<sessionId>/` 配下にセッション履歴を持ちます。サブエージェントは `subagents/agent-<id>.jsonl`、Stop Hook の出力は今回 `.claude/audit/` に置いています。どこに何があるかを把握しておけば、後日のチーム説明が楽になります。

7.2 書く [2min]

`hooks/stop-summary.sh` に、フック発火回数を jq で数える 3 行を追記します。audit JSON に `pre_tool_use_count` / `post_tool_use_count` / `subagent_calls` を入れてください。

7.3 実行する [4min]

もう一度別ファイルで再実行します。

```
/summarize-log sample-logs/error.log

# 終了後、別ターミナルで監査ログを集計
cat .claude/audit/summary-*.json | jq
```

7.4 比較する [2min]

Before デモでの transcript（講師が共有しているもの）と、After の audit JSON を並べます。

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 程度に圧縮
危険コマンド対応	permission prompt のみ、判断は人手	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

> **ヒント参照**
> 数字の取り方が分からなければ `hints/step04_read_transcript_hint.md` と `hints/step04_before_after_diff_hint.md`。jq ワンライナーとレポート用プロンプトが置いてあります。

SECTION 08

8. FAQ（よくある質問）

Q. `/agents` や `/skills` に何も出ない	カレントディレクトリが `log-review-agent/` ルートか、`.claude/` が同じ階層にあるかを確認してください。Claude Code は起動した CWD を起点に拡張を読みます。
Q. PreToolUse が止まらない	Hook スクリプトに実行権限がないか、stdin から JSON を読み損ねている可能性が高いです。`chmod +x hooks/*.sh`、jq で `tool_input.command` を取り出してください。
Q. SKILL.md の `context: fork` が無視される	Claude Code 2.1 系以降が必要です。`claude --version` を確認してください。古い場合は `npm install -g @anthropic-ai/claude-code@latest`。
Q. サブエージェントが期待通りに呼ばれない	`.claude/agents/log-reader.md` の description を「Use proactively when asked to analyze access logs.」のように行動指示として書くと、親 Claude が委譲しやすくなります。
Q. 監査 JSON が空	Stop Hook はセッション終了時にしか発火しません。`/exit` でセッションを閉じてからファイルを確認してください。
Q. Copilot ユーザーですが Hook をどう代替しますか	Copilot 単体では同等のフックは標準提供されていません。VS Code 側のイベント（onDidSaveTextDocument 等）でフックする拡張を自作するか、Claude Code に乗り換えるのが手堅い。今回は前者の代替まではカバーしていません。

SECTION 09

9. トラブル即応マニュアル

症状	切り分け	応急処置
Claude Code が起動しない	node -v / claude --version	Node v20 系へ、Claude Code を最新へ
サインインのブラウザが完了しない	localhost コールバックの遮断	個人端末で再実行、社内端末では IT 部門に開放依頼
Hook が発火しない	settings.json の構文エラー	VS Code で settings.json を開き、JSON Schema 警告を確認
Hook の deny が効かない	matcher のミスマッチ	matcher を `"Bash"` または正規表現で指定し直す
SKILL.md が読まれない	frontmatter の YAML エラー	--- を上下に揃え、name と description を先頭で確認
サブエージェントが委譲されない	description が観点ベース	「Use proactively when ...」など行動指示型で書き直す
jq が使えない	パスが通っていない	事前セットアップガイド §2.2 で再インストール
進捗 30% 以下で残り 5 分	諦める判断	`solutions/` をコピーして Step 4 に飛ぶ。後追いで埋める

参考リンク

- Hooks Reference（Claude Code 公式）
- Skills Reference（Claude Code 公式）
- Subagents Reference（Claude Code 公式）
- Settings Reference（Claude Code 公式）
- Permissions Reference（Claude Code 公式）
- Best Practices（Claude Code 公式）
- Commands Reference（Claude Code 公式）
- Equipping agents with Agent Skills（Anthropic）