TRAINING

AIカスタム研修 Day1

手元の自動化を設定で固める

データライブ株式会社様 ／ 2026年9月29日

Givery

Copyright © Givery, Inc. All rights reserved

本資料の二次転載禁止について

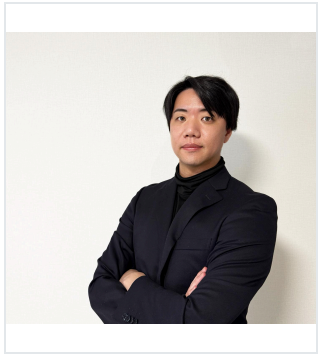
受講者ご本人の学習目的に限った利用

本資料は、株式会社ギブリーがデータライブ株式会社様向けに作成した研修教材です。
著作権は株式会社ギブリーに帰属します。研修の受講者ご本人の学習目的に限って
ご利用ください。資料の全部または一部を、複製・転載・改変・第三者への配布
（社内の他部門を含む）することはご遠慮ください。画面共有や録画の
二次利用についても同様です。

研修の演習に登場する企業名（株式会社ホクトリユース）・人名・データは
すべて架空です。研修中に各自の端末で開く自社ハーネスは、
投影画面に映さないでください。

Copyright © Givery, Inc. All rights reserved

講師紹介



人的資本インテリジェンス部門講師・
生成AIエバンジェリスト

安田 光喜

Mitsuki Yasuda

公立はこだて未来大学 大学院（メディアデザイン領域）を卒業し、街づくり会社での複合商業ビルの立ち上げにおける情報インフラ整備やデザイン全般業務の責任として関与。その後デザイン事務所を立ち上げ、飲食店を中心としたデザイン業務や美術館展示のアートコンテンツ開発などを行う。また、地域ブランディングにも注力し、市主催の企画運営業務を行なった。2018年10月『小中学生向けのプログラミング教育』に注目し、函館市で初めての小中学生向けプログラミングスクール『D-SCHOOL北海道』を立ち上げて運営。その後北海道のドラッグストア『サッポロドラッグストアー』からスクールの買収を受け、教育を専門とするグループ会社『株式会社シーラクンス』にて技術開発責任者として運営。スクール設計・運営、20以上の自治体連携（イベントや出張教室運営）、大人向けプログラミングスクールメンターなどさまざまな『次世代IT教育』に取り組む。

得意領域

- 生成AIをはじめとした世界最先端トレンド
- 生成AIを用いた新規事業の開発、運営、補助
- アプリケーション開発/自動化プログラム環境構築
- 教育（大学講師、研修、スクール運営など）

実績

- 生成AI研修
- 大手企業、外資、教育機関(小中高大）でのITトレンド講演
- 自治体と連携した地域教育実績
- 上場企業社員対象のリスキリング支援

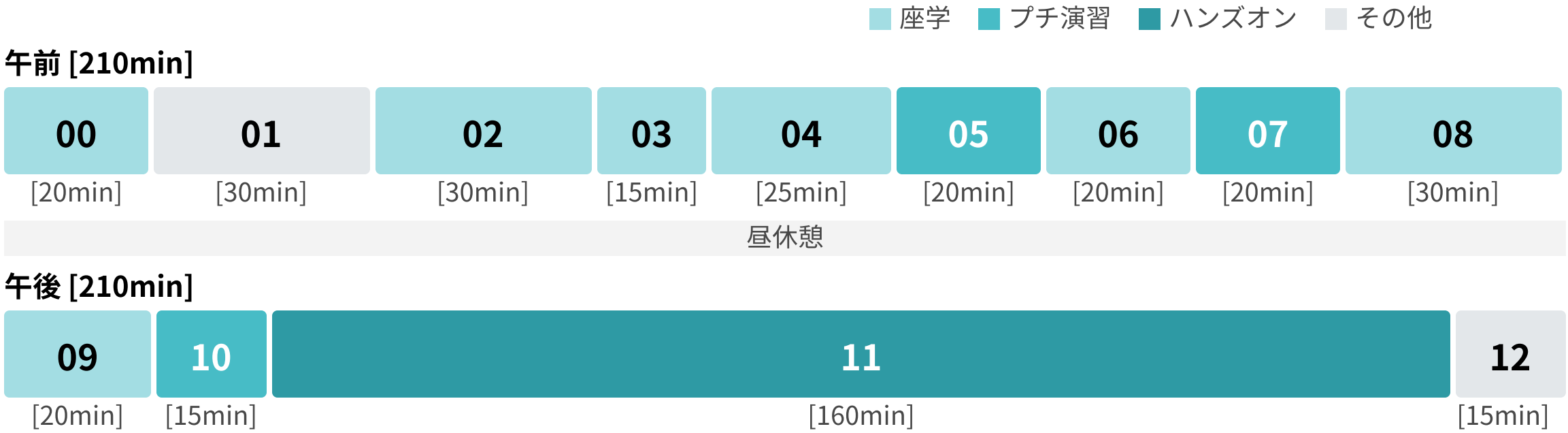
メッセージ

みなさんがこれから生成AIをパートナーに新しい時代を生きていけるように、2日間全力でサポートしていきます！なにかご不明点などありましたら、講師陣になんでも気軽にご質問ください。
#ClaudeCode #Codex #Cursor #Antigravity
#全国統一生成AI活用技能試験S1
#生成AIパスポート　#G検定

Copyright © Givery, Inc. All rights reserved

本日のアジェンダ

午前は設定で固め午後は改修1本の一周



D1-00 Claude Desktop の基本操作
D1-01 到達像の実録と8層チェック表
D1-02 完全自動開発の全体像と線引き
D1-03 標準化の動向と乗り換え耐性
D1-04 モデルとエフォートの設定階層
D1-05 プチ演習1 規模別レビュアーの2定義
D1-06 AI を用いたロールバックの3層
D1-07 プチ演習2 巻き戻し制約の機械強制
D1-08 権限・フック・サンドボックス
D1-09 書いた本人に採点させない根拠
D1-10 プチ演習3 秘密情報の遮断とフックの単体テスト
D1-11 ハンズオン1 改修1本の AI 駆動一周と実装・レビューの分離
D1-12 Day1 の持ち帰り台帳と宿題

Copyright © Givery, Inc. All rights reserved

ブロック別の所要と持ち帰り物

13ブロックの終点はすべて持ち帰り物

No	ブロック	種別	所要	持ち帰り物
D1-00	Claude Desktop の基本操作	その他	[20min]	Desktop とコマンドライン版の選び分け表
D1-01	到達像の実録と8層チェック表	その他	[30min]	8層チェック表(印付き)、台帳の1行目
D1-02	完全自動開発の全体像と線引き	座学	[30min]	委譲判定の3軸
D1-03	標準化の動向と乗り換え耐性	座学	[15min]	乗り換え耐性チェックリスト
D1-04	モデルとエフォートの設定階層	座学	[25min]	設定の優先順位図
D1-05	プチ演習1 規模別レビュアーの2定義	演習	[20min]	agents 2本、settings 差分
D1-06	AI を用いたロールバックの3層	座学	[20min]	層ごとに戻せる範囲の表
D1-07	プチ演習2 巻き戻し制約の機械強制	演習	[20min]	ask 3本、deny 5組10本、CLAUDE.md の1行
D1-08	権限・フック・サンドボックス	座学	[30min]	.env.tpl、run-with-secrets.ps1 / .sh
D1-09	書いた本人に採点させない根拠	座学	[20min]	分離の4つの置き方
D1-10	プチ演習3 秘密情報の遮断とフックの単体テスト	演習	[15min]	Read の deny 3本、フック1本、テストケース
D1-11	ハンズオン1 改修1本の AI 駆動一周と 実装・レビューの分離	演習	[160min]	MR、レビュー表、分離の1本
D1-12	Day1 の持ち帰り台帳と宿題	その他	[15min]	台帳7点の記入、宿題2つ

Copyright © Givery, Inc. All rights reserved

2日間のゴールと到達像

ゴールは夜間に回り**朝に承認するだけ**の状態

今日の朝

- ハーネスが一通り立ち上がっている
- ループには未着手
- 実行とテストの担保はAI の報告頼み

Day1 の終わり

- 規模別レビュアーと deny が settings に入る
- 巻き戻しと秘密情報がフックで機械強制される
- 実装とレビューを分けたMR が1本できる

Day2 の終わり

- レビューが CI のゲートとして MR を止める
- 夜間にブランチが切られ実装とテストが済む
- 朝に人がマージを承認するだけ

習熟度の測定はしません。2日で自社ハーネスに機能を足して持ち帰ることが到達像です。

Copyright © Givery, Inc. All rights reserved

現地の完走を基準にした進行

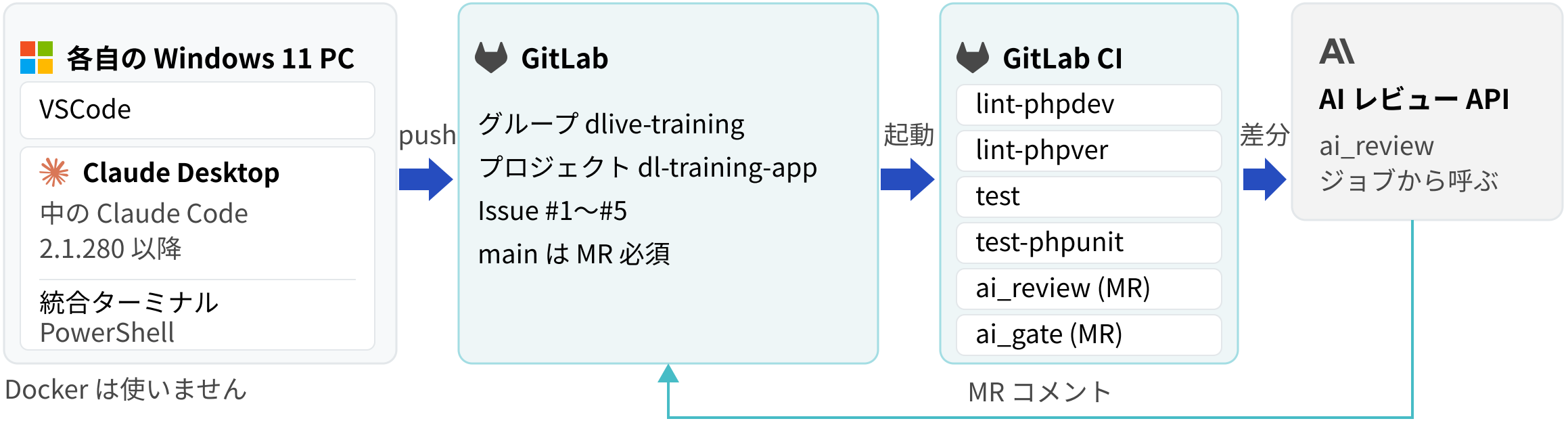
項目	現地	リモート
質問の出し方	その場で声に出す	Zoom チャットに書く（サブ講師が拾う）
手が止まった時	手を挙げる	チャットに「止まった」と一言
演習の完走	全員の完走を基準に進めます	手順の把握を優先し、完走は必須にしません
画面共有	講師の画面のみ投影	自社ハーネスの画面は共有しない
時間の表記	所要は [Nmin] で示す	同じ
持ち帰り台帳	紙か md に書く	md をチャットで提出

- 現地 **ハイブリッド開催**
- リモート 配布物と手順は同じです

Copyright © Givery, Inc. All rights reserved

演習環境の全体図

実行とテストは GitLab CI 側



- 演習アプリは架空企業 株式会社ホクトリユースの基幹システムです
- 対象バージョンは PHP 7.4 で、データベースは PostgreSQL です
- 手元では実行せず、push のたびに CI が lint とテストを回します

※ 各ロゴは各社の商標です。

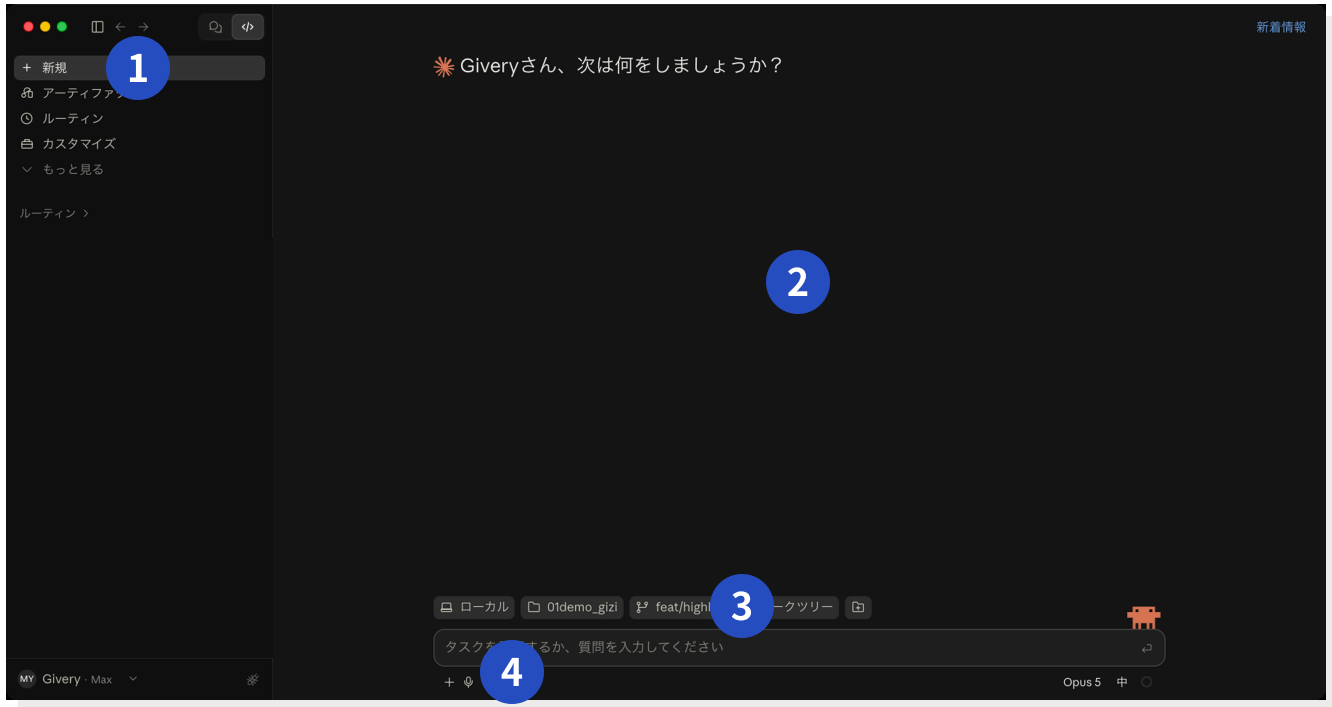
GitLab https://gitlab-09291006aidev.give-app.net ／ 教材サイト https://09291006aidev.give-app.net

Copyright © Givery, Inc. All rights reserved

解説

Claude Desktop の画面

研修で使うのは Desktop に載った Claude Code



1 サイドバー

新規は Ctrl+N、切り替えは
Ctrl+Tab です。

2 会話エリア

セッションを開くと、ここに
やり取りが並びます。

3 ブランチ名とワークツリー

チェックを入れると
.claude/worktrees/ に作られます。

4 パーミッションモード

入力欄の下の 自動 が
いまのモードです。

下部のバー左端の ローカル は接続先です。統合ターミナルはローカルのセッションだけで開けます。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

Claude Desktop の主な機能

操作の起点になる4つ

入力欄の上に接続先とフォルダ名とブランチ名、下にモードが並びます。

worktree	`.claude/worktrees/`	帯のチェックから作ります
パーミッションモード	手動 / 編集を受け入れる / プラン / 自動	帯の 自動 がいまの値です
並列セッション	`Ctrl+N` / `Ctrl+Tab`	複数を並べて動かせます
統合ターミナル	`Ctrl + `	セッションの作業フォルダで開きます

保存先と接頭辞は Settings の Claude Code で変えられます。gitignore 済みは .worktreeinclude に書きます。
出典: https://code.claude.com/docs/en/desktop

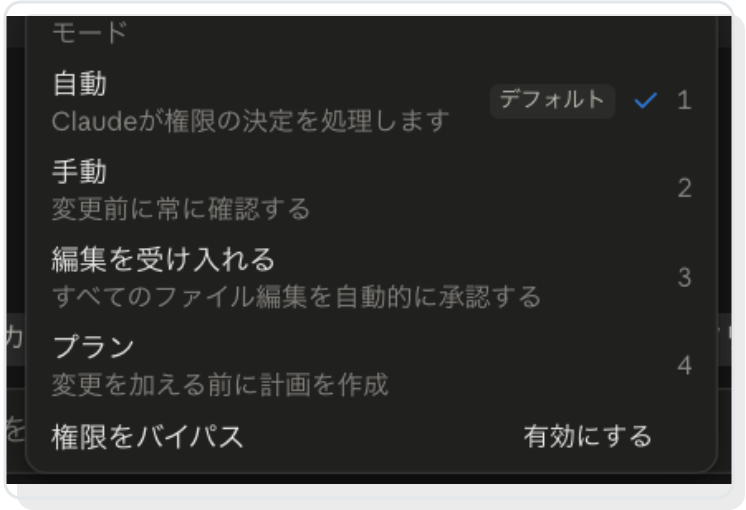
Copyright © Givery, Inc. All rights reserved

解説

パーミッションモードの使い分け

確認なしで通す範囲を決める4段

モード	確認なしで通るもの	研修での扱い
手動	読み取りのみ	既定。研修は最初 これで進めます
編集を受け入れる	編集と作業フォルダ内の mkdir・mv・cp・rm など	編集の確認が続いたら 切り替えます
プラン	読み取り専用で、 計画だけ書きます	手を入れる前に 段取りだけ見ます
自動	別のモデルが全アクションを 審査します	研修では使いません



入力欄の下のモード名を押すと
このメニューが開きます。

設定ファイルでは default・acceptEdits・plan・auto の値で書きます。

設定ファイルでの指定と、モードより手前に効く allow と deny は D1-08 で扱います。

出典: https://code.claude.com/docs/en/desktop、https://code.claude.com/docs/en/permission-modes

Copyright © Givery, Inc. All rights reserved

解説

コマンドライン版との違い

できることは同じで できないことが4つ

どちらでも同じもの

- **設定ファイル**
  ~/.claude/settings.json が両方に効きます。
- **フック**
  登録したフックはどちらでも発火します。
- **CLAUDE.md**
  同じ指示が両方で読まれます。

Desktop にないもの

- 非対話の実行
  `-p / --print`
- エージェントチーム
- インライン補完
- サードパーティのプロバイダ

> この4つ以外は、Claude Desktop でもコマンドライン版でも同じように動きます。
> Day2 の夜間ループで使う -p は、CI の中でコマンドライン版が動かします。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

設定とフックの共通

今日書いた設定はどちらでも効きます

研修中に書く settings.json とフックは、そのままコマンドライン版でも動きます。
持ち帰るのは画面の操作手順ではなく、この3つのファイルです。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

Desktop とコマンドライン版の選び分け

研修は Desktop 自動化はコマンドライン版

Claude Desktop が向く場面

- 手順を画面で確かめながら覚える段階
- 会話・ターミナル・worktree が
  1つの画面に収まる
- Ctrl+Tab でセッションを
  行き来しながら進められる

コマンドライン版が向く場面

- 決めた手順を人の手を離れて回す段階
- -p を付けた非対話の実行ができる
  (Desktop にはありません)
- CI やスケジューラから呼べる。
  Day2 の夜間ループはこの形です

まず Desktop で形を作り、毎回同じ手順になったところでコマンドライン版へ移します。
設定とフックは共通なので、移すときに書き直すものはありません。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

配布物の一覧

配布物は全部が自社ハーネスへの書き戻し素材

配布物	使う場面	形式	置き先(D2 ／ smart3pm)
8層チェック表.md	D1-01・D2-11	md	各自の手元(印を付けて塗り直す)
持ち帰り台帳.md	全ブロック	md	D2 と smart3pm の両ルート
モデルとエフォート対応表.md	D1-05	md	各自の手元
乗り換え耐性チェックリスト.md	D1-03	md	各自の手元
denyルール集.md	D1-07・D1-10	md	.claude/settings.json
秘密情報の扱い1枚.md	D1-08・D1-10	md	各自の手元(講師デモの再現用)
.env.tpl ／ run-with-secrets.ps1 ／ .sh	D1-08	tpl / ps1 / sh	リポジトリのルート
委譲判定シート.md ／ .csv	D1-11	md / csv	各自の手元
計測表_Day1_Day2.md	D1-11	md	各自の手元

Day2 の配布物は Day2 の前付で案内します。対象バージョン非対応構文一覧.md、REVIEW.md、gate.ps1 ほかが入ります。

Copyright © Givery, Inc. All rights reserved

Day1 で扱わないもの

除外の根拠は7/17の共有会とご回答

扱い ＼ 根拠	7/17 社内共有会で済んだ	対象外とご判断いただいた
名称だけ触れる	モデル選択の基礎 D1-04 は振り分け設計のみ Skills と Hooks の配置場所	汎用テンプレートハーネス plugins の名称と配置のみ GitLab 自体のテーマ化 前提環境として使う
触れない	MCP 経由の Notion 連携 CLI とデスクトップ版の差	Spec Kit・Kiro 等の仕様駆動開発 リリース工程 規約・ナレッジ整備

全演習は既存コードの改修を題材にします。ゼロからの新規開発は出てきません。

Copyright © Givery, Inc. All rights reserved

解説

貴社が描いた最終像が前夜に1周した画面

D1-01

到達像の実録と8層チェック表

前夜に講師環境で回した夜間パイプラインの実録を見ます。
8層チェック表に、自社ハーネスの今の位置を印で付けます。
良い点3つと伸びしろ8点を、2日で足す機能一覧に変えます。

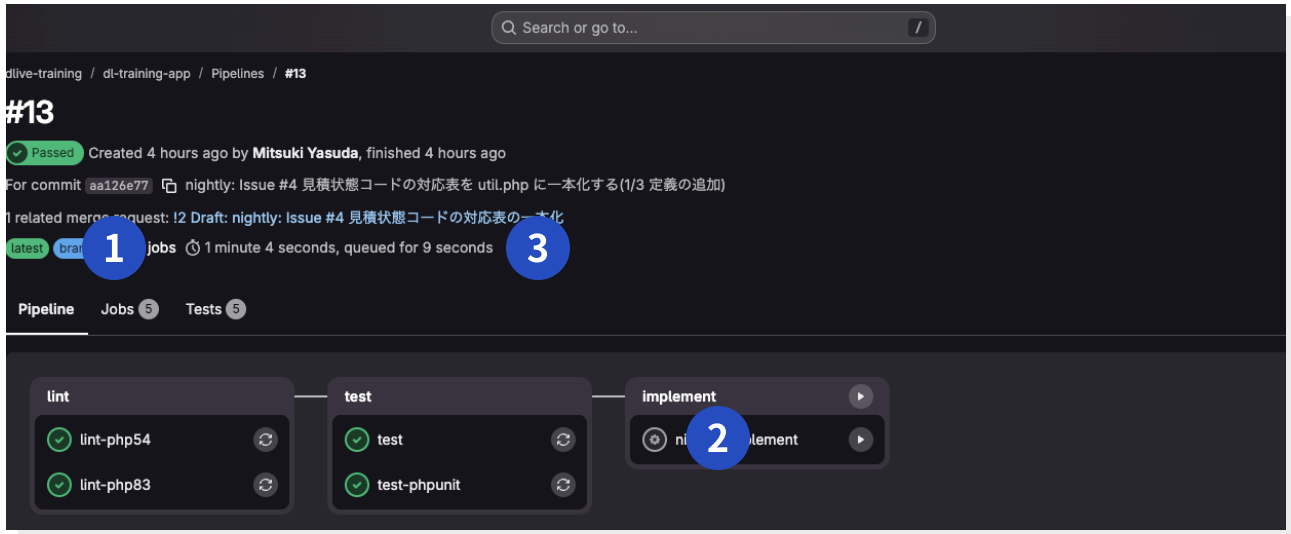
[30min]

Copyright © Givery, Inc. All rights reserved

実演

前夜に回った夜間パイプライン

人が1回も手を触れずに終わったジョブ列



> **画面について**
> nightly/issue-4 の実走。前夜は schedule 起動になります。

前夜に講師が schedule で起動し、朝まで放置した結果をこの場で開きます。
起動からマージ待ちまで、人の操作は一度も入っていません。

出典: https://docs.gitlab.com/ci/pipelines/schedules/

1 起動元の表示

人が押さず、schedule が
起動しています。

2 ジョブ列

lint と test が通ると、MR 側で
ai_review と ai_gate が続きます。

3 所要時間

起動時刻、MR 作成時刻、所要は
この場で読み上げます。

> Duration の欄は前夜の実走値です。
> 手元の紙に書き写してください。

Copyright © Givery, Inc. All rights reserved

解説

前夜に起きたことの順番

起動から MR 作成まで人の操作なし

1. schedule が起動
2. ブランチ ai/issue-N を作成
3. claude -p で実装
   `--permission-mode dontAsk / --max-turns 15`
4. lint-phpdev / lint-phpver と test
5. ai_review が findings.json を出す
6. ai_gate が P0/P1 で止める
   P0/P1 があれば MR は赤のまま止まります。
7. push option で MR 作成

夜のうちに7つの工程が機械だけで進み、朝に人が見るのは MR 1本です。
受講者の皆さんが Day2 で作るのは、この7工程を止める側の設定です。

出典: https://code.claude.com/docs/en/cli-reference
https://docs.gitlab.com/topics/git/commit/#push-options

Copyright © Givery, Inc. All rights reserved

実演

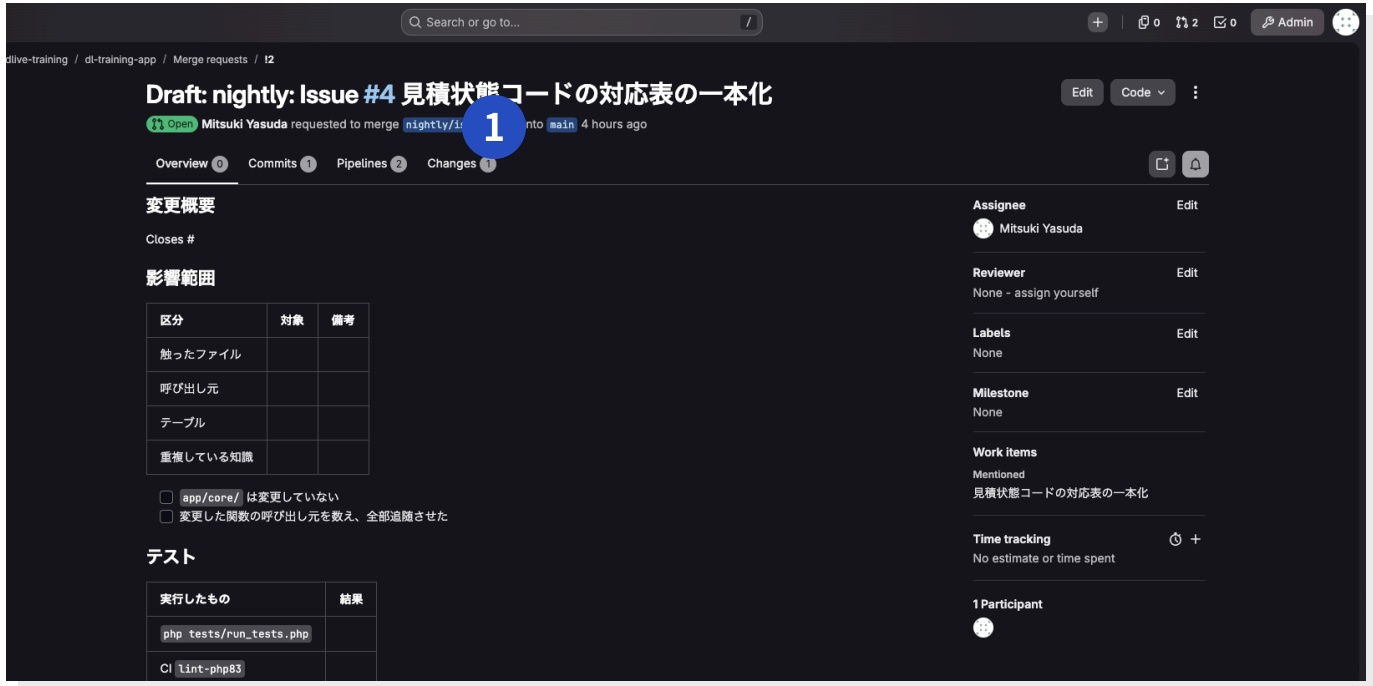
AI レビューコメント付きの MR

朝に人が開く画面はこの MR 1枚だけ

1 ソースブランチ ai/issue-N
ブランチ名も機械が付けました。

2 AI レビューのコメント
指摘は severity 付きで並びます。

3 Merge ボタン
ここだけが人の操作です。



2 のコメントと 3 のボタンはこの画面の下にあります。
指摘の本数と severity は前夜の実走を読み上げます。

AI は気になった点を列挙し、採るか見送るかの判断は人に残しています。
貴社の回答書にある今の切り分けを、夜間に機械で回した状態です。

Copyright © Givery, Inc. All rights reserved

同じ画面を各自の手で出すのが2日間のゴール

> **Day2 の最後に、この MR を**
> **各自の手で作ります。**

講師環境との差は、この設定ファイル4つだけです。

`.gitlab-ci.yml` 夜間ジョブ	`schedule` GitLab の定期起動	`implement-issue` 実装の skill	`settings.json` 止め金の設定

4つとも Day2 の最後のブロックで、自分のリポジトリに置きます。

Copyright © Givery, Inc. All rights reserved

補足

自動化8層チェック表

層で測る現在地の表

層	一言	D2	smart3pm	Day2 終了時
第1層 -p 非対話	人が打たずに1回走る	☐	☐	○ 足す
第2層 hooks	実行中に機械が止める	☐	☐	○ 足す
第3層 subagents / skills / plugins / MCP	仕事を分けて渡す	☐	☐	○ 足す
第4層 permission modes / sandbox	人の代わりに承認する止め金	☐	☐	○ 足す
第5層 compaction	長時間でも文脈を保つ	☐	☐	座学のみ
第6層 Agent Teams / Workflows	並列に走らせる	☐	☐	座学のみ
第7層 Routines / CI	人がいない時間に起動する	☐	☐	○ 足す
第8層 Agent SDK	プログラムから呼ぶ	☐	☐	座学のみ

各層の中身は次の D1-02 で1枚ずつ開きます。ここでは印だけ付けます。

出典: https://code.claude.com/docs/en/headless

Copyright © Givery, Inc. All rights reserved

ワーク

自社ハーネスの到達層に印

機械で守れている層だけの印

1 Step 1 [1min]

自分が主に触るハーネスを D2 か smart3pm から1つ選び、列の上に丸を付けます。

2 Step 2 [1min]

第1層から第8層まで、今日の朝の時点で「動いている」層の四角にチェックを入れます。書いてあるだけで動いていない層は空欄のままにします。

3 Step 3 [1min]

チェックが途切れた最初の層に下線を引きます。そこが Day2 までに足す最初の層です。

期待される手元

記入例: 3つチェック、次の層に下線

	D2
第1層	☑
第2層	☑
第3層	☑
第4層	☐
第5層	☐
第6層	☐
第7層	☐
第8層	☐

この表は D2-11 でもう一度塗り直します。今日の印は消さずに残してください。

Copyright © Givery, Inc. All rights reserved

解説

今日の朝と Day2 の終わりの差

2日で足す設定は第1層・第4層・第7層

今日の朝

- 第8層 Agent SDK
- 第7層 CI / Routines
- 第6層 Agent Teams
- 第5層 compaction
- 第4層 permissions / sandbox
- 第3層 subagents / skills
- 第2層 hooks
- 第1層 -p 非対話

設定ファイルを足す

第5・6・8層は座学で
位置だけ知ります。

Day2 の終わり

- 第8層 Agent SDK
- **第7層 CI / Routines**
- 第6層 Agent Teams
- 第5層 compaction
- **第4層 permissions / sandbox**
- **第3層 subagents / skills**
- **第2層 hooks**
- **第1層 -p 非対話**

貴社ドラフトの到達地点「ハーネスは立ち上がり、ループは未着手」は、左の形です。
夜間1周に足りないのは -p 非対話、止め金、Stop hook、CI の schedule で、どれも演習に入っています。

出典: https://code.claude.com/docs/en/cli-reference, https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

補足

持ち帰り台帳.md の初期状態

2日分の成果物を1ファイルで追う台帳

```
# 持ち帰り台帳

## Day1

| 番号 | 持ち帰り物 | D2 での置き場所 | smart3pm での置き場所 | 担当 | 反映日 |
|---|---|---|---|---|---|
| D1-1 | 規模別レビュアー2本 | | | | |
| D1-2 | settings.json のモデルと effort | | | | |
| D1-3 | deny 5組10本(巻き戻せない操作) | | | | |
…(Day1 は D1-7 まで7行、Day2 は D2-10 まで10行)
```

持ち帰り物の名前は入った状態で配ります。埋めるのは置き場所と担当と反映日です。
置き場所の列は2本あります。D2 と smart3pm で置き方が違うためです。

Copyright © Givery, Inc. All rights reserved

ワーク

台帳の1行目と紙に書く3つ

朝いちばんに自分が見ている項目

1 **Step 1 [1min]**
持ち帰り台帳.md を VSCode で開き、
Day1 の表の1行目「D1-1」の「担当」に
自分の名字を書きます。

2 **Step 2 [2min]**
紙に書きます。「夜間に回った MR を
朝に開いたとき、自分は何を確かめてから
マージを押すか」。3つ、名詞で書きます。

3 **Step 3 [1min]**
3つのうち、機械に判定させられそうな
ものに丸を付けます。丸が付かなかった
ものが、人に残る仕事の候補です。

期待される手元

台帳の1行目

「担当」に自分の名字
「反映日」は空欄のまま

紙のメモ

1 _____
2 _____
3 _____

この紙は D2-08 の最後に出す「人が残る3点」の下書きになります。捨てないでください。

Copyright © Givery, Inc. All rights reserved

解説

今のハーネスの良い点3つ

研修の演習はこの3つの上に積む設計

骨格の呼び方	何が良いか	研修でそのまま使う場所
core.md の [M][A][H] 強制タグ	ルールを内容でなく強制手段で分類している	D2-01 文脈から機械へ移す作業 D2-06
pre-receive の fail-open と fail-closed	検査 NG は拒否、インフラ起因は素通りと線を引いている	D2-05 の CI ゲート設計
agents/08 の 実装担当の再定義	AI に実装させ、差分を読み、実物で確かめる係と定義している	D1-02 の線引き D2-06 の人の立ち位置

投影は骨格の名前だけです。中身は各自の端末で該当ファイルを開いて確認してください。

Copyright © Givery, Inc. All rights reserved

解説

伸びしろ8点と足す場所

8点すべてに2日の中で手を付ける場所

伸びしろの項目	Day1 座学	Day1 演習	Day2 座学	Day2 演習	研修後
1 機械強制タグの担保が AI の自己申告になっている			D2-06	D2-05	
2 既知の許容の台帳がコードレビュー側にない			D2-03	D2-05	
3 レビューの重さが変更規模に比例していない	D1-04	D1-05			
4 セッション開始時の注入はあるが検証がない			D2-06		
5 フック自体のテストが片側のハーネスだけ		D1-10			D2-10
6 フックの文字コード制約を機械で守っていない	D1-08				D2-10
7 効果測定が投入量ベースになりやすい			D2-11		
8 PowerShell と bash の二重運用の解消時期		D1-12			D2-10

2日の中で扱う　研修後の宿題

8点は講師が事前精読で見つけた伸びしろで、どれも足すと効く箇所です。

5・6・8 の3点は研修後の宿題として D2-10 で渡します。

Copyright © Givery, Inc. All rights reserved

2日で自社ハーネスに足す件数

2日で足す17点はすべて既存ハーネスへの追記

17

Day1 7点 + Day2 10点

Day1 の7点は agents 2本、settings 差分、deny 2群、フック1本、
CLAUDE.md 追記、SKILL.md または agent 1本です。
どれも持ち帰り台帳に1点ずつ載せ、置き場所まで決めます。

> Day2 の10点は制約節、PostToolUse フック、REVIEW.md、観点別サブエージェント3本、
> ゲートの2分割、Stop フック、夜間ジョブ一式の7点に、理想的なハーネスの雛形一式、
> 自社ハーネス診断シート、書き換えた1ファイルの3点を足したものです。

Copyright © Givery, Inc. All rights reserved

補足

この2日で扱わないものと扱うもの

この2日の範囲は既存コード改修と自社ハーネスへの追記

扱わない

- 汎用テンプレートハーネスの作成
- Spec Kit・Kiro などの仕様駆動開発
- Notion MCP 連携の再説明
- CLI と Desktop の差、
  Skills と Hooks の配置
- リリース工程
- 規約・ナレッジ整備

扱う

- Claude Desktop の中の Claude Code
- 既存コード(dl-training-app)の改修だけ
- GitLab は前提環境、テーマにはしない

左の6つは 7/17 の社内共有会と打ち合わせで決まった範囲です。
研修中に質問が出たら休憩時間に受けます。

※ 各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

補足

全員での CI の回し方

順番待ちを前提にした CI の運営

全員が push
ほぼ同時に入ります

→

runner の順番待ち
Pending で数分止まります

→

順に6ジョブ
結果は MR の画面へ

場面	同時に走る本数	待つあいだにすること
Day1 ハンズオン1 の Step6	人数分が順に入る	192 の記録表と 193 の判定シートを埋める
Day1 のプチ演習3本	CI は使わない	手元のフックと単体テストで完結する
Day2 の夜間ジョブ	講師1本→各自	同じログを読んでから自分の分を押す

Pending は故障ではありません。アイコンが回り始めるまで数分かかります。
現地とリモートの進め方の差は 007 の表にあります。配布物と演習は同じです。

Copyright © Givery, Inc. All rights reserved

このブロックの持ち帰り

手元に残る紙2枚とファイル1本

8層チェック表(印つき)

今日の朝の位置に印が付いた
状態。D2-11 で塗り直す。

紙

持ち帰り台帳.md

1行目に担当を書いた状態。
Day1 の終わりに7行を埋める。

ファイル

紙に書いた3つ

朝にマージを押す前の確認項目。
D2-08「人が残る3点」の下書き。

紙

次の D1-02 で、8層の中身を講師環境の実ファイルで1枚ずつ開きます。

Copyright © Givery, Inc. All rights reserved

解説

機械に任せる範囲は巻き戻せる範囲

D1-02

完全自動開発の全体像と機械と人間の線引き

8層の全体像と成熟度を1枚で見て、講師環境の実ファイルを4本開きます。
一次資料の事例4本から、委譲判定の3軸と人が残る境目を決めます。

[30min]

Copyright © Givery, Inc. All rights reserved

補足

今日の基準となる版

この版で動くものだけを話す

2.1.280

Claude Code、2026-09-22 リリース

このブロックで GA か preview かを分ける基準は、この版の公式ドキュメントです。
モデルは Fable 5.1、Opus 5.5、Sonnet 5.5、Haiku 4.5 を前提にします。
受講端末の版も事前セットアップで揃え、Claude Desktop の設定画面で確認できます。

出典: code.claude.com/docs/en/changelog

Copyright © Givery, Inc. All rights reserved

解説

自動化を支える8層

ファイル名が安定している層から積む

一般提供　preview・beta・experimental

第8層 SDK	Agent SDK (Python / TypeScript)	
第7層 起動の自動化	Routines / GitHub Actions / GitLab CI	●
第6層 並列化	Agent Teams / Dynamic workflows	
第5層 長時間化	/compact / autoCompactWindow	●
第4層 承認の代替	permission modes / deny / sandbox　Windows は不可	●
第3層 仕事の分割	subagents / skills / plugins / MCP	●
第2層 介入点	hooks: PreToolUse / PostToolUse / Stop	●
第1層 非対話実行	claude -p --output-format json	●

Day1〜Day2 で触るのは印の6つの層

Claude Code 2.1.280(2026-09-22)時点です。点線の層は仕様変更が続きます。
出典: code.claude.com/docs/en の headless・gitlab-ci-cd・routines・agent-teams

Copyright © Givery, Inc. All rights reserved

解説

8層の入口と置き場所

無人実行で最初に詰まるのは承認の層

層	入口	置き場所	成熟度	夜間ループでの役割
1 非対話実行	`claude -p --output-format json`	CI のスクリプト	一般提供	1回の実装を走らせる
2 介入点	`PreToolUse / Stop PostToolUse`	settings.json	一般提供	禁止操作の遮断と終了条件
3 仕事の分割	`agents / skills plugins / MCP`	agents と skills	一般提供	実装とレビューの分離
4 承認の代替	`--permission-mode deny / sandbox`	settings.json	一般提供	止め金
5 長時間化	`/compact autoCompactWindow`	settings.json	一般提供	口頭ルールは消える前提
6 並列化	`Agent Teams Dynamic workflows`	env と workflows/	experimental・一般提供	研修では扱わない
7 起動	`Routines / GitLab CI GitHub Actions`	.gitlab-ci.yml	preview・beta 一般提供	夜間の起点
8 SDK	`Agent SDK`	Python・TypeScript	一般提供	研修では扱わない

-p の既定は default です。指定しないとツール呼び出しが拒否され、何もせず終わります。
第4層の sandbox はネイティブ Windows で動きません。持ち帰れるのは permissions と hook の2層です。

出典: code.claude.com/docs/en の headless・permission-modes・sandboxing・routines・gitlab-ci-cd

Copyright © Givery, Inc. All rights reserved

実演

講師環境の settings.json

文章で守るものと機械で守るものの分かれ目

dl-training-app/.claude/settings.json(講師環境)

```
{
  "permissions": {
    "deny": [
      "Bash(git reset --hard *)",
      "PowerShell(git reset --hard *)",
      "Read(./.env)"
    ] },
  "hooks": {
    "PreToolUse": [{
      "matcher": "Bash|PowerShell",
      "hooks": [{ "type": "command",
        "command": "powershell.exe",
        "args": [ …, "-File",
        "…/hooks/block-destructive.ps1"] }]
    }]
  }
}
```

1 文字列照合の限界

deny は文字列で照合します。/bin/rm や bash -c で書き換えると抜けます。

2 bypass でも効く2ブロック

deny と PreToolUse は bypass モードでも効きます。deny の抜けをフックで拾います。

3 秘密情報の遮断

Read(./.env) が第4層の秘密情報の遮断で、D1-10 で足します。

Windows の PowerShell ツールにも当てるため、deny は対で書き、matcher は Bash|PowerShell

CLAUDE.md に書いた禁止は助言です。強制になるのはこの2ブロックです。

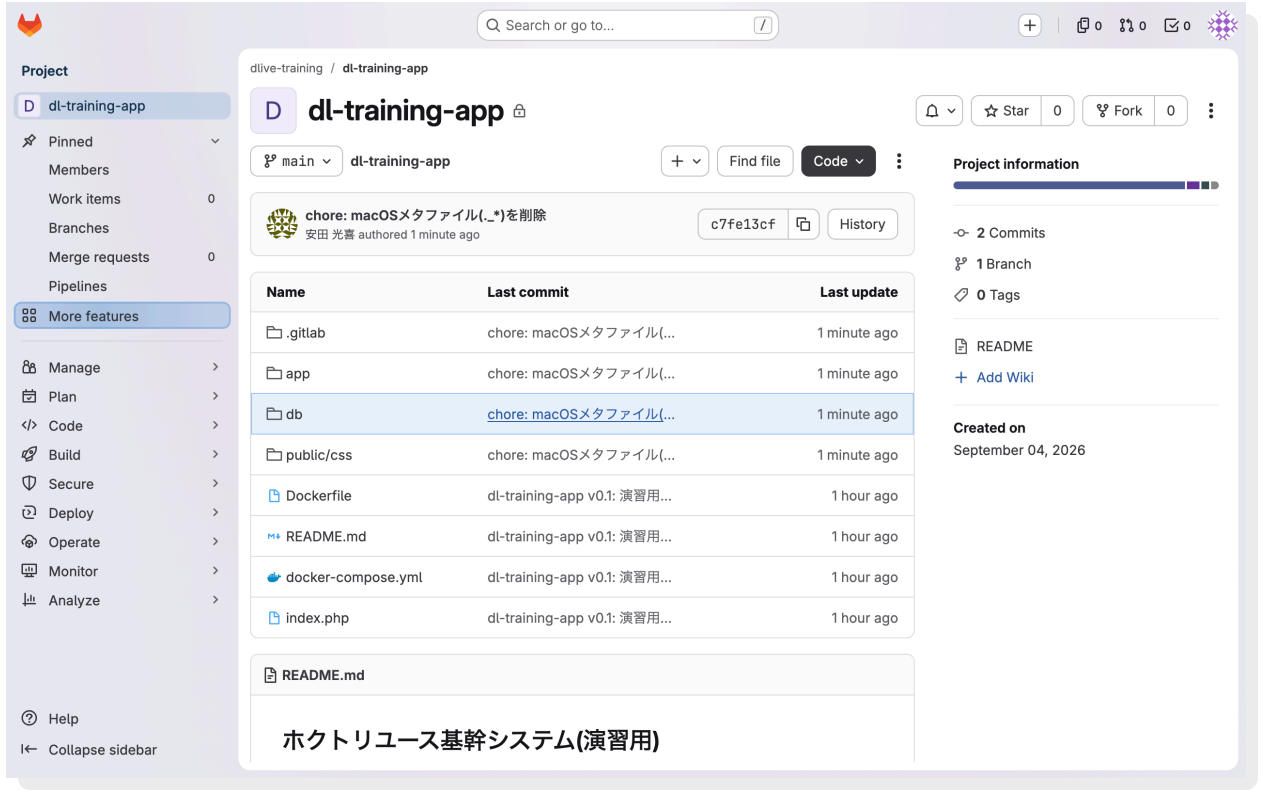
出典: code.claude.com/docs/en の permissions・hooks-guide、claude-code issue #39459

Copyright © Givery, Inc. All rights reserved

実演

演習リポジトリに見える起動層

マージ承認だけが人の手に残る配置



GitLab: gitlab-09291006aidev.give-app.net
グループ dlive-training / dl-training-app
受講者は Developer です。

1 左メニューの Pipelines

第7層(起動)の実行履歴が並びます。
D1-01 で見た夜間の実走もここです。

2 ファイル一覧の先頭

ここに .gitlab と .claude が並びます。
レビューの基準と設定を、コードと同じ
リポジトリに置いている状態です。

3 main ブランチ

直 push は不可、MR が必須です。
人が残る3点の1つを GitLab 側で
実装済みです。

この画面は .claude と CI を足す前です。
当日はご自身の画面で並びを見てください。

Copyright © Givery, Inc. All rights reserved

実演

仕事の分割の実物 review-light.md

モデルと観点を変更規模で選ぶ定義

.claude/agents/review-light.md(講師環境)

```
---
name: review-light
description: 50行未満の小さな変更向けのレビューです。
model: sonnet
effort: medium
tools: Read, Grep, Glob
---
```

1 書き込まないレビュアー

tools を Read / Grep / Glob に
絞ります。レビュアーは
書き込みません。

2 frontmatter での固定

model と effort は
frontmatter で固定します。

3 review-full との対

D1-05 で review-full
(opus / xhigh / 4観点)と
対で使います。

分割の単位は Markdown 1枚です。変更規模ごとに1枚ずつ増やします。
出典: code.claude.com/docs/en/sub-agents

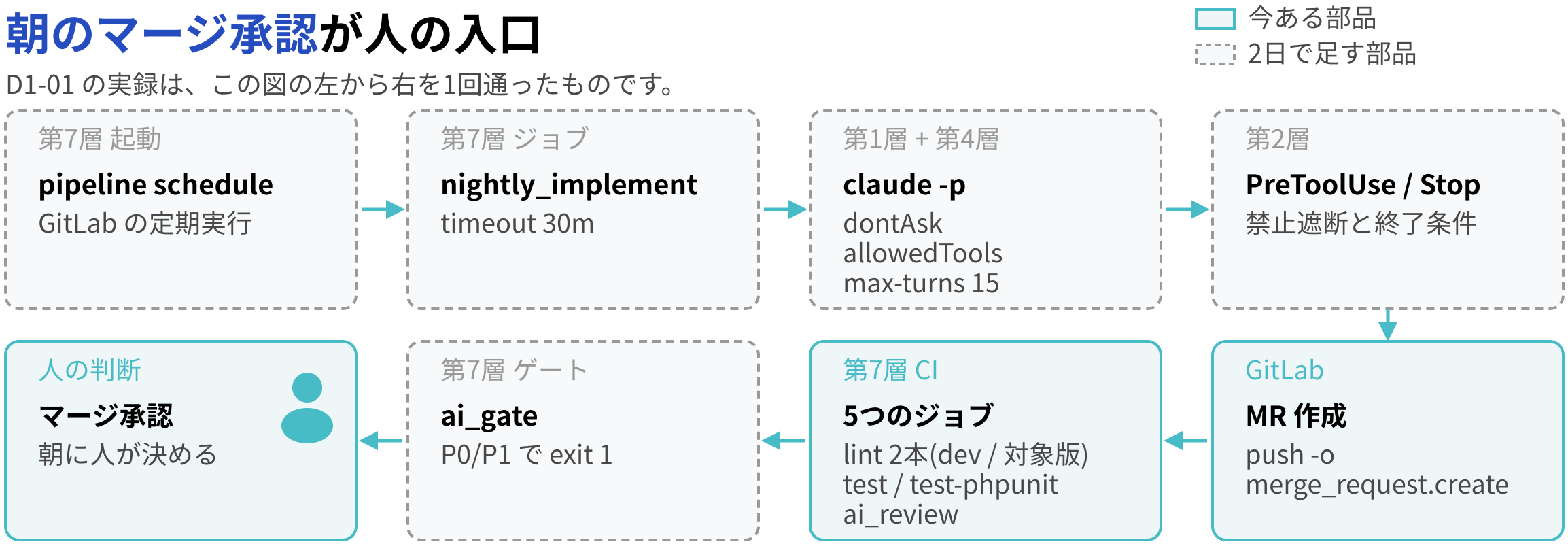
Copyright © Givery, Inc. All rights reserved

解説

夜間ループの全体図

朝のマージ承認が人の入口

D1-01 の実録は、この図の左から右を1回通ったものです。



MR 作成は CI_JOB_TOKEN では通らず、プロジェクトアクセストークンを masked variable に置きます。

出典: code.claude.com/docs/en の gitlab-ci-cd・headless、docs.gitlab.com の push options

Copyright © Givery, Inc. All rights reserved

実演

レビュー専用サブエージェント

書いた本人とは別の履歴でレビューする配置

.claude/agents/review-other.md(講師環境)

```
---
name: review-other
description: 実装後に別のサブエージェントで
  独立レビューし、P0/P1 だけ限定修正する
tools: Read, Grep, Glob, Bash
model: opus
---
同僚のコードとしてレビューします。対象は
git diff main の変更だけです。
P0/P1 の問題を3件まで、ファイル名と行番号
つきで報告してください。コード提案は
書かないでください。
```

別の履歴でレビュー

サブエージェントは親の
会話履歴を引き継がないので、
自己レビューになりません。

渡す道具の限定

読み取りと検査だけを渡し、
Edit と Write は渡しません。

報告の粒度

3件・ファイル名・行番号を
指定し、提案は書かせません。

045 の数字(履歴を持たない別セッション 28.6 対 自己レビュー 24.6)を、この1枚で実装します。

出典: https://code.claude.com/docs/en/sub-agents

Copyright © Givery, Inc. All rights reserved

解説

スケジューラの置き場所

APIキー運用ではスケジューラは GitLab 側

Routines(/schedule)

research preview

- claude.ai ログインが必須
- ANTHROPIC_API_KEY があると /schedule が出ない
- 最小間隔は1時間
- 実行は Anthropic のインフラ

pipeline schedule

GitLab CI 連携は beta

- 自社 runner で動く
- cron は自由、when: manual で段階投入
- 鍵は masked variable に置く
- ブランチ保護と組み合わせる

演習と D2 の夜間ジョブは GitLab 側に置きます。

出典: code.claude.com/docs/en の routines・gitlab-ci-cd、docs.gitlab.com/ci/pipelines/schedules/

※ 各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

解説

一次資料で見る4つの事例

事故2本と設計2本を時系列で

2025-07
Replit Agent
コードフリーズ中に
本番 DB を削除

2026-01-09
issue #17190
「戻して」の依頼で
reset --hard が走る

2026-03-12・13
issue #34327
2日連続で
reset --hard origin/main

2026-03-12
Cross-Context Review
履歴を捨てたレビューで
検出率が上がる

2026-07-21
Anthropic 自社 SDLC
狭い観点のレビュアーを
複数並べる

上段は文章で止めようとした側、下段は構造で止めた側です。
Replit は接続先を分ける。#17190 と #34327 は deny とフックに置く。
Cross-Context Review は履歴を切る。自社 SDLC は観点を分けて並べる。

出典: replit.com/blog(2025-07-21)、claude-code issues #17190・#34327、arXiv 2603.12123、claude.com/blog(2026-07-21)

Copyright © Givery, Inc. All rights reserved

解説

Anthropic 自社 SDLC のレビュー設計

狭い観点のレビュアーを複数並べる

> 「各レビューエージェントは狭い一点に scope され、
> 過去インシデントの RAG を持つ」

> 「1つに束ねないのは、バイアスと盲点を共有しないため。
> 1つが間違えても他が拾う」

承認は全件ログされ、人はリスク加重サンプルを再確認します。

引用は研修用の訳です。

出典: claude.com/blog/how-anthropic-secures-its-ai-native-software-development-lifecycle(2026-07-21)

Copyright © Givery, Inc. All rights reserved

解説

履歴を捨てたレビューの検出率

同じモデルでも履歴を切ると指摘が変わる

28.6 vs **24.6**

F1(%)

履歴を持たない別セッション　　同セッションの自己レビュー

同一モデル(Claude Opus 4.6)での比較です。
重大エラーの検出は 40% 対 29% です。
30成果物に150個の誤りを注入した比較で、p=0.008 です。

モデルを変える追加効果は、この研究では未検証です。

出典: arxiv.org/html/2603.12123(arXiv 2603.12123、2026-03-12)

F1(%) の比較

ティールは履歴を持たない側です。

Copyright © Givery, Inc. All rights reserved

解説

Replit 2025-07 本番 DB 削除

指示書の抑止は破られる前提で接続を分ける

before

- 「コードフリーズ」の指示は文章だけ
- エージェントが本番 DB へ直接接続
- 経営者1,200件超・企業1,190件超のデータを削除

> **件数の出どころ**
> Fortune 2025-07-23 の報道です。公式ブログは分離策の発表だけを載せています。

after(2025-07-21 公式ブログ)

- 開発 DB と本番 DB を自動分離
- 初回デプロイの本番はスキーマのみ
- 開発側の変更は再デプロイ時に確認
- Point-in-time Restore で巻き戻し

> "Unacceptable and should never be possible"
> Replit CEO の発言(2025-07、報道ベース)

演習アプリでは config.php の DB_HOST に dltrain だけを見せます。
本番の接続文字列はエージェントに渡しません。

出典: replit.com/blog(2025-07-21)、fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database(2025-07-23)

Copyright © Givery, Inc. All rights reserved

解説

claude-code issue #17190 と #34327

CLAUDE.md の禁止文は強制にならない

> **#17190　2026-01-09　Claude Code v2.0.65**
>
> 「ROLL BACK to the last push」の依頼に対し、git stash の後に git reset --hard が実行され、約4時間分の作業とコミットが孤立しました。CLAUDE.md に書かれた禁止は無視されています。

> **#34327　2026-03-12 と 2026-03-13**
>
> セッション開始直後に git reset --hard origin/main が2日連続で走り、2回目は未 push の12コミットと未コミットファイルが消失しました。1回目は git reflog により51秒後に復旧できています。

どちらも closed として終了しています。手当は文章ではなく deny と PreToolUse に置きます。

出典: github.com/anthropics/claude-code の issues #17190・#34327

Copyright © Givery, Inc. All rights reserved

解説

4事例の手当の置き場所

同じ事故を設定で再現不可能にする

事例	起きたこと	機械側の手当	人に残ること
Replit 2025-07	指示中の本番 DB 削除	接続先の分離と本番鍵の非付与 第4層	本番適用の承認
issue #17190 issue #34327	「戻して」の依頼で reset 依頼なしで起動直後に reset	deny と PreToolUse 第2層・第4層	reset を文字通りに指示する判断
Cross-Context Review	自己レビューで検出が低下	別セッションと別 agent 第3層	指摘の採否
Anthropic SDLC	観点別レビュアーを並列	CI で並列実行 第7層	リスク加重サンプルの再確認

右端の列が、このあと話す「人が残る3点」の元になります。

出典: replit.com/blog、GitHub issues #17190 / #34327、arXiv 2603.12123、claude.com/blog

Copyright © Givery, Inc. All rights reserved

解説

委譲判定の3軸

戻せて機械で検証できるものから任せる

円の大きさ＝影響範囲

- 自ブランチ内
- 共有ブランチ
- 本番DBと外部連携

図の点

1. 状態コード対応表の重複
2. 検索条件の文字列連結
3. 引当の排他制御なし
4. 本番DBのスキーマ変更

位置は固定ではありません。テストを足せば点は左へ動きます。

Copyright © Givery, Inc. All rights reserved

解説

3軸の問いと演習アプリの例

軸ごとに問いを1つだけ持つ

軸	問い	機械へ寄せる条件	演習アプリの例
可逆性	失敗したら何で戻すか	git revert で戻る範囲に収める DB のデータは触らない	コード修正 テスト追加
機械検証	正否を人が読まずに 決められるか	lint・テスト・スキーマ検査で落ちる	php -l 7.4 tests/run_tests.php
影響範囲	自分のブランチの外に出るか	main は MR 必須で直 push 不可	引当の tr_stock 更新は 注文側に波及

> 3つの問いは Issue を読んだ直後、AI に渡す前に紙で答えます。

Copyright © Givery, Inc. All rights reserved

解説

線引きの原則

人が残るのは境目の判断

> **機械に任せる範囲は、失敗時に巻き戻せる範囲**

ゲートの定義

何で落とすかを人が決めます。
対象は hook の exit 2 と CI の閾値です。

例外の採否

指摘を見送る判断を人が行います。
MR コメントに理由を残します。

本番接続の鍵

接続文字列と権限の管理です。
開発用の接続先だけを渡します。

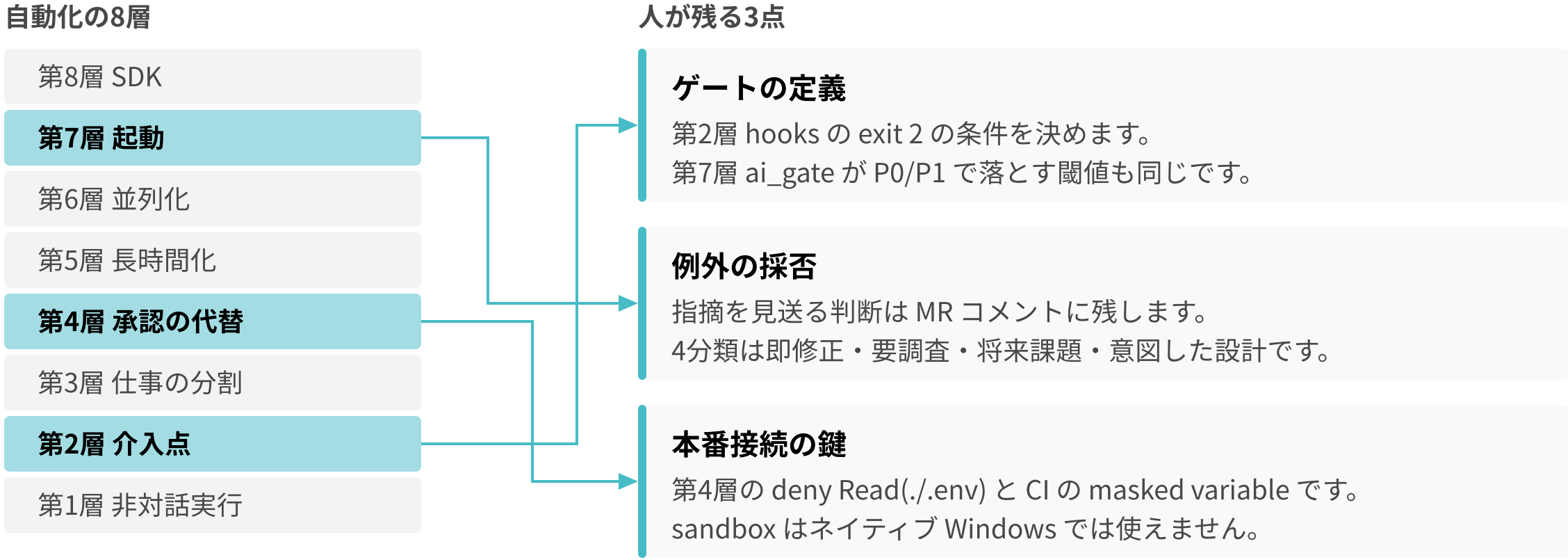
性能が上がると機械へ任せられる範囲は広がります。この3点は残ります。

Copyright © Givery, Inc. All rights reserved

解説

人が残る3点と層の対応

人の仕事は層の間の境目にある



Windows 端末では sandbox が使えないので、持ち帰れる層は permissions と hook の2層です。

出典: code.claude.com/docs/en の permissions・sandboxing

Copyright © Givery, Inc. All rights reserved

解説

無人実行の止め金

無人実行で効かせる止め金は4つ

Claude Code

`--permission-mode dontAsk`	allow に無い操作を全部拒否します
`--allowedTools`	許すツールを列挙します
`--max-turns`	ターン数の上限を決めます
`deny / PreToolUse hook`	実行そのものを遮断します
`--bare`	レビュー側だけに使います

> ツールを跨ぐ唯一の保険は、タスク前後の git コミットです。

出典: code.claude.com/docs/en/cli-reference。各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

無人実行の最小セット

既定のままでは承認待ちで何も起きない

この呼び方が D2-08 の nightly_implement ジョブの中身です。

ci/nightly.ps1

```
claude -p "/implement-issue $env:ISSUE_ID" `
  --permission-mode dontAsk `
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)" `
  --max-turns 15 `
  --output-format json
```

> 非対話実行(-p / --print)は Claude Desktop の UI からはできません。CI や夜間ループはコマンドで動かします。

許可パターンの空白
Bash(git diff*) は
git diff-index にも当たります。

--bare の使いどころ
実装側には使いません。hooks も
CLAUDE.md も読まないためです。

二重の止め金
--max-turns とジョブの
timeout 30m を重ねます。

出典: code.claude.com/docs/en/headless ・ code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

質問

自分の改修1件で線を引く

直近1件で任せた操作と任せなかった操作

直近で自分が担当した改修を1件思い出してください。

1. AI が実行した操作のうち、失敗したら戻せないものを1つ書きます。
2. 正否を人が読まずに機械で決められた操作を1つ書きます。
3. 自分のブランチの外に影響が出た操作があれば書きます。

所要 [3min]。書いた紙は D1-11 の委譲判定シートの1行目になります。

現地の方は手元の紙に、リモートの方はチャットに書いてください。

Copyright © Givery, Inc. All rights reserved

解説

8層図に重ねた D2 の現在地

資料にある層と2日で足す層

投影は匿名化した骨格です。実物は各自の端末で開いてください。　資料にある　研修で足す　研修では扱わない

第8層 SDK	研修では扱わない
第7層 起動	CI は無し(資料の時点)。D2-05 と D2-08 で足す
第6層 並列化	研修では扱わない
第5層 長時間化	口頭ルールはコンパクションで消える。deny へ移す
第4層 承認の代替	deny と dontAsk の止め金。sandbox は Windows 不可
第3層 仕事の分割	skills と agents あり。規模別の分割を D1-05 で足す
第2層 介入点	後処理のフックあり。遮断を D1-10、終了条件を D2-07 で足す
第1層 非対話実行	-p を D2-04 と D2-08 で足す

この図の D2・smart3pm の位置は、いただいた資料からの推定です。手元と違っていたら、その場で言ってください。
smart3pm も同じ層構成です。置き場所の問いは D1-12 で扱います。

Copyright © Givery, Inc. All rights reserved

解説

2日で足す層と演習の対応

演習1本が層1つを埋める

層	資料の時点	足す演習	置き場所 D2 / smart3pm
第4層 止め金	一部	D1-07 deny 5組10本 D1-10 Read(./.env)	.claude/settings.json D2 と smart3pm で共通
第2層 PreToolUse	一部	D1-10 block-destructive	hooks/*.ps1 と hooks/*.sh
第3層 分割	あり	D1-05 review-light と review-full D1-11 review-other	.claude/agents と .claude/skills
第1層 -p	なし	D2-04 --bare -p --json-schema	ci/ のスクリプト
第2層 Stop	一部	D2-07 stop-gate	hooks/stop-gate.ps1 と .sh
第7層 CI・schedule	なし	D2-05 ai_review と ai_gate D2-08 nightly_implement	.gitlab-ci.yml

フックは PowerShell 5.1 版と bash 版を両方配布します。どちらを正にするかは D1-12 で決めます。

Copyright © Givery, Inc. All rights reserved

ワーク

線引きのセルフチェック

文章だけで守っている項目に印を付ける

	項目	機械	文章	なし
1	戻せない操作を deny か hook で止めている	☐	☐	☐
2	.env と本番接続文字列をエージェントが読めない	☐	☐	☐
3	無人実行が承認待ちでも全許可でもない	☐	☐	☐
4	レビューを書いた本人のセッションに任せていない	☐	☐	☐
5	終了条件を Stop hook か CI で判定している	☐	☐	☐
6	main への直 push を GitLab 側で禁止している	☐	☐	☐
7	指摘の見送りを MR に記録している	☐	☐	☐
8	ゲートの閾値を人が文章で決めている	☐	☐	☐

「文章」に印が付いた行が、午後の演習で機械へ移す候補です。
8行目だけは文章のままで正解です。

Copyright © Givery, Inc. All rights reserved

解説

2日間で層を埋める順番

止め金が先でループは最後

Day1

第3層	第4層	第2層	第3層
D1-05	**D1-07**	**D1-10**	**D1-11**
review-light と full	deny 5組10本	block-destructive	review-other

Day2

第2層	第1層	第7層	第2層	第7層
D2-02	**D2-04**	**D2-05**	**D2-07**	**D2-08**
check-phpver	--bare と gate.ps1	ai_review と ai_gate	stop-gate	nightly_implement

ループは、止め金が全部入った後にしか回しません。

Copyright © Givery, Inc. All rights reserved

このブロックの持ち帰り

線引きの決め手は巻き戻し

8層図

自分のハーネスの到達層に印を付け、8層チェック表に転記します。

3軸の紙

直近の改修1件の3行。D1-11 の委譲判定シートの1行目になります。

人が残る3点の言い方

ゲートの定義・例外の採否・本番接続の鍵。この順で口に出せる状態にします。

次の D1-03 では、乗り換え時に残る資産と捨てる資産を分けます。

Copyright © Givery, Inc. All rights reserved

解説

規格より置き場所で決まる乗り換え耐性

D1-03

標準化の動向と乗り換え耐性

Agent Plugins 1.0 の構図を見て、規格の外で決まることを先に押さえます。
別のツールへ移るときに残る資産と残らない資産を分けます。
移植性はファイルの置き場所で決まります。

[15min]

Copyright © Givery, Inc. All rights reserved

解説

Agent Plugins 1.0 の構図

規格が決めたのは3つだけでフックは対象外

コアメンテナの5社

2026-08-06 に仕様 1.0.0 が公開されました。Vercel の発案で、コアメンテナは AWS・Cursor・Microsoft・OpenAI・Vercel の5社です。GitHub も発表に加わりました。

標準化したのは plugin.json・skills/・mcp.json の3つです。
フックとサブエージェントは対象外で、今後の版に回りました。

土台は Anthropic 発

Anthropic はメンテナ一覧にありません。土台の Agent Skills と MCP は Anthropic が公開したものです。
Claude Code は次ページの独自 plugin 形式のままです。

「規格に乗っていないから移せない」とは
限りません。中身の本文は別の理由で移せます。

> 規格の中身を待つ必要はありません。移せるかどうかは、本文と登録情報のどちらに書いてあるかで決まります。
> 仕様の条文は次ページのハーネス4層の話に置き換えて扱います。

出典: vercel.com/blog/introducing-agent-plugins、aws.amazon.com/blogs/opensource(2026-08-06)　※ 各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

ハーネスの4層と移植性

本文は移せてツール固有の登録情報は移らない

第1層 文脈	CLAUDE.md・SKILL.md の本文、レビュー観点の文章	**そのまま移せる**
第2層 手続き	フックのスクリプト本体(ps1・sh)、テストケースの json	**そのまま移せる**
第3層 登録	hooks の JSON 登録、plugin.json frontmatter の model・effort	**書き直せば移せる**
第4層 権限	permissions の allow / deny、sandbox、承認の方式	**移らない**

乗り換えで失うのは第4層と第3層の一部だけです。ハーネスの価値の大半は上の2層にあります。
第4層の権限は、文字列で照合する方式と OS で隔離する方式で、考え方から違います。

出典: https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

実演

Claude Code の plugin 構造

束ねる単位はディレクトリ1つと宣言ファイル1つ

ディレクトリ構成

```
dl-review/
  .claude-plugin/
    plugin.json
  agents/
    review-light.md
    review-other.md
  hooks/
    hooks.json
```

dl-review/.claude-plugin/plugin.json

```
{
  "name": "dl-review",
  "description":
    "Review agents and hooks for dl-training-app",
  "version": "1.0.0"
}
```

講師環境で claude plugin validate ./dl-review が
通るところまでを実演します。

.claude/ に置いていた agents・hooks を1つのディレクトリへ移し、
.claude-plugin/plugin.json を足すだけで plugin になります。
skills も同じ並びで入り、名前は /dl-review:<名前> の形になります。

出典: https://code.claude.com/docs/en/plugins

Copyright © Givery, Inc. All rights reserved

乗り換えるときの対応表

7行のうち移植不要は2行と捨てる1行

役割	Claude Code での置き場所	移すときに要る作業
常時読む指示	CLAUDE.md	本文をコピーし、名前を 移り先の規約に合わせる
手順の部品	.claude/skills/*/SKILL.md	本文は流用。frontmatter は 捨てる
レビューの分離	.claude/agents/*.md model と tools	観点の文章を移り先の レビュー設定へ写す
フック	settings.json の hooks hooks/hooks.json	スクリプト本体はそのまま。 登録 JSON を書き直す
権限	permissions の allow / ask / deny(文字列照合)	移らない。考え方から 設計し直す
モデルと effort	settings.json の model・ effortLevel、agents	名前を差し替える。段階の語は おおむね共通
MCP	.mcp.json	URL とコマンドを写す。 形式は移り先に合わせる

「移植不要」は常時読む指示とフック本体、「捨てる」のは権限です。残りは名前と形式の書き直しで済みます。

出典: https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

解説

フックの出力 JSON の形

判定の本体と出力の包み方を分けた作り

Claude Code の Stop フック出力

```
{
  "decision": "block",
  "reason": "PHPUnit failed: 2 tests"
}

// PreToolUse だけが hookSpecificOutput に包む
```

判定ロジックの置き場所

スクリプト本体に置きます。
Claude なしでテストでき、
ツールを問いません。

出力の包み方

外側の JSON だけ引数で
切り替えます。

Stop と PostToolUse は decision と reason をトップレベルに置きます。exit 2 と stderr でも止められます。
判定を書いた本体と出力の包み方を別にしておくと、別のツールへ移せます。

出典: https://code.claude.com/docs/en/hooks

※ 各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

持ち出す箱と書き直す箱と置いていく箱

そのまま持ち出す	形だけ書き直す	置いていく
Claude Code 側のファイル CLAUDE.md の本文 SKILL.md の本文 hooks/*.ps1 と *.sh hooks/tests/cases.json REVIEW.md の観点	**Claude Code 側のファイル** settings.json の hooks 登録 .mcp.json agents の model・effort plugin.json	**Claude Code 側のファイル** permissions の allow / ask / deny Claude 固有イベントの フック enabledPlugins
移り先での着地 移り先の指示ファイルに 本文を貼る スクリプトは同じパス	**移り先での着地** 移り先の登録ファイル MCP 設定を書き写す モデル名を置き換える	**移り先での着地** 書き込み先と承認方針で 境界を引き直す 文字列 deny の代替は無い

出典: https://code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

解説

D2 と smart3pm の伸びしろ

タグ体系そのものが乗り換え耐性の設計図

> **smart3pm の core.md が全ルールに付けた**
> **[M][A][H] のタグは、乗り換えで何が残るかを**
> **先回りして分類しています。**

- [A] で守っているルールは本文なので、移り先の指示ファイルへそのまま移せます。
- [M] のフック本体はツールを問いません。登録 JSON だけが書き直しです。
- D2 の agents 定義はモデル名がフル ID で埋まっており、書き直す箱に入ります。

Copyright © Givery, Inc. All rights reserved

別のツールへ移すときの5手

本文を抜いて登録を書き直し権限を引き直す順

移り先での着地

指示ファイル	同じパス	登録ファイル	モデル設定	権限の方式

Step1 と Step2 はコピーです。差分が出たら、元の本文にツール固有の記述が混ざっていた印です。
Step3 は 066 の形に合わせて包み直します。テストは hooks/tests/cases.json をそのまま回します。
Step4 は名前の置換です。段階の語は似ていても較正が違うので、走らせて合わせます。
Step5 だけが設計作業です。deny で止めていた操作を、書き込み先と承認方針で言い直します。

出典: https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

補足

モデルの価格とコンテキストと effort

段階の語は揃っていても中身は別の較正

モデル	入力 $/MTok	出力 $/MTok	コンテキスト	effort の段階	向く仕事
Fable 5.1	10	50	1M	low〜max	数時間走るエージェント、深い調査
Opus 5.5	4	20	1M	low〜max	エージェント的コーディング 大規模リファクタ
Sonnet 5.5	2	10	1M	low〜max	日常の実装とレビュー、Opus 5.5 の半額で速い
Haiku 4.5	1	5	200K	非対応	低レイテンシ、大量処理、サブエージェント

> Sonnet 5.5 は 9/28 公開。単価は Sonnet 5 と同じで、公式は消費トークンが減るぶん最大30%安いとしています。

Claude Code での effort の既定は high(Opus 5.5 と Sonnet 5.5 は medium)、段階は low〜max の5つです。
Sonnet 5.5 の段階は Sonnet 5 から較正し直され、同じ medium でも考える量は別です。

出典: https://platform.claude.com/docs/en/about-claude/pricing、https://www.anthropic.com/claude/sonnet
https://platform.claude.com/docs/en/build-with-claude/effort、https://code.claude.com/docs/en/model-config

Copyright © Givery, Inc. All rights reserved

質問

自社ハーネスの移せる割合

移せると思っているファイルの実数

問い

自分が触っている D2 か
smart3pm の中で、067 の
「置いていく」箱に入る
ファイルを3つ挙げてください。

書き出す欄

ファイル名

そのファイルが守る契約

移り先ではどの方式で
守るか

目安

[3min]

3分です。書けたら隣の
人と見せ合わず、D1-12 の
持ち帰り台帳の余白に
写してください。

Copyright © Givery, Inc. All rights reserved

乗り換え耐性チェックリスト

6項目のうち今日のうちに直せるのは4つ

研修後に自社ハーネスへ当てて、印が付かなかった行を D2-10 の未着手リストに書きます。

印	項目	確認する場所	今日の扱い
☐	CLAUDE.md の本文にツール固有の名前が混ざっていない	CLAUDE.md と指示書	演習で埋まる
☐	SKILL.md の本文だけ読んでも手順が成立する	.claude/skills/*/SKILL.md	演習で埋まる
☐	フックの判定はスクリプト本体にあり、登録 JSON には無い	hooks/*、settings.json	演習で埋まる
☐	フックにテストケースがあり、Claude なしで回る	hooks/tests/cases.json	演習で埋まる
☐	モデル名が1箇所に集約されている	settings.json、agents/*.md	**設計が要る**
☐	deny で守っている契約に、サンドボックス方式での言い換えがある	permissions、sandbox	**設計が要る**

配布物: 乗り換え耐性チェックリスト(A4)

Copyright © Givery, Inc. All rights reserved

解説

タスク単位で振り分ける設定の置き場所

D1-04

モデルとエフォートの設定階層

このあとの D1-05 で、ここで決めた振り分けを実装します。

扱うのは、どこに書けばどの順で勝つかだけです。

講師環境の settings.json・SKILL.md・agents/*.md を開きながら進めます。

[25min]

Copyright © Givery, Inc. All rights reserved

補足

本ブロックの線引き

基礎は前提にして扱う振り分け設計

7/17 で済んだこと(触れない)	本ブロックで扱うこと
/model でモデルを選ぶ操作	5つの入口と優先順位(どれが勝つか)
モデルごとの得意と価格	タスク種別ごとのモデルと effort の振り分け
トークン管理と /usage の読み方	モデル切替でキャッシュが飛ぶ操作の避け方
Skills / Hooks の配置場所	frontmatter の model と effort が効く範囲

左列は 7/17 の社内共有会で扱い済みとご連絡いただいた項目です。
本ブロックは右列だけを扱います。

出典: 7/17 社内共有会の対象範囲

Copyright © Givery, Inc. All rights reserved

解説

モデル指定の優先順位

上の入口が下の入口を黙って上書きする5段の階層

組織が配る managed settings は、この5段よりさらに上に立ちます。

1	`--model <alias>` / `ANTHROPIC_MODEL` 起動フラグと環境変数。そのセッションだけ効きます
2	`.claude/settings.local.json` の `model` 自分だけに効きます。コミットしない前提のファイルです
3	`.claude/settings.json` の `model` プロジェクトで共有する設定です。チームの既定になります
4	`~/.claude/settings.json` の `model` 自分の全プロジェクトに効く既定です
5	`ANTHROPIC_DEFAULT_MODEL` どのファイルにも model が無いときだけ効きます

チームの .claude/settings.json が sonnet でも、自分の settings.local.json に
opus と書けば自分のセッションだけ変わります。今どちらが効くかは入力欄の右下の表示で見ます。

出典: https://code.claude.com/docs/en/model-config、settings#settings-precedence

Copyright © Givery, Inc. All rights reserved

解説

effort の解決順

環境変数が最上位で frontmatter より強い順番

順位	入口	書く場所	持続
1	`CLAUDE_CODE_EFFORT_LEVEL`	環境変数	常時。skill と agent より強い
2	`--effort` / `/effort`	起動フラグ / セッション中	--effort は今回、/effort は保存
3	`modelSettings / effortLevel`	`settings.json`	/effort の確定で書き戻し
4	モデル既定	なし	high。Opus・Sonnet 5.5 は medium
別枠	`frontmatter effort`	`SKILL.md / agents/*.md`	そのターンだけ。環境変数に負ける

「frontmatter に書いたのに変わらない」の原因の大半は1行目です。
CLAUDE_CODE_EFFORT_LEVEL が入っていると、skill と agent の effort は無視されます。
~/.claude の effortLevel は Opus 5.5 以降に効かず、/effort で選ぶまで medium のままです。
max は settings の effortLevel に書けません。永続させるなら環境変数だけです。

出典: https://code.claude.com/docs/en/model-config#set-the-effort-level（2.1.284）

Copyright © Givery, Inc. All rights reserved

同じ名前でもモデルごとに中身が別の5段階

5段階

非対応のレベルを指定すると「指定以下で最も高い対応レベル」に丸められ、エラーは出ません。effort は thinking だけでなく、本文とツール呼び出し引数を含む全出力トークンに効きます。

low | medium | high | xhigh | max

モデル	対応する effort レベル
Fable 5.1 / Opus 5.5・5 / Sonnet 5.5・5	5段階すべてに対応
Opus 4.6 / Sonnet 4.6	xhigh なし(xhigh を指定すると high に丸め)
Haiku 4.5	effort 非対応

出典: https://platform.claude.com/docs/en/build-with-claude/effort

Copyright © Givery, Inc. All rights reserved

CLAUDE.md は文脈を書く場所

> CLAUDE.md は指示(文脈)であり、強制される設定ではありません。
> ツールの拒否・環境変数・ログイン制限のような技術的な強制は
> managed settings、コードスタイルや行動指針は CLAUDE.md。

出典: 公式ドキュメント memory「CLAUDE.md vs auto memory」節の要旨
https://code.claude.com/docs/en/memory#claude-md-vs-auto-memory

> 英語の原文は公式ページ側で確認してください。

CLAUDE.md に「この作業は Haiku で」「effort は low で」と書いても、
モデルも effort も変わりません。Claude が自分のモデルを切り替える手段が無いためです。
固定するなら settings.json の model と availableModels、
または PreModelSwitch hook を使います。

Copyright © Givery, Inc. All rights reserved

講師環境の settings.json

/effort で確定した値が書き戻される実体

~/.claude/settings.json

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "modelSettings": {
    "claude-opus-5-5": { "effortLevel": "xhigh" }
  },
  "maxEffortLevel": "xhigh"
}
```

> この構造はバージョンで変わります。/effort 実行後の settings.json をご自身の端末で見てください。

この設定ファイルは Claude Desktop と CLI で共有です。permissions とフックはどちらでも効きます。
effortLevel は保存済みでないモデル向けの既定、modelSettings はモデル別の保存値です。
maxEffortLevel はどのスコープでも最も低い上限が効くので、組織で縛るならこのキーです。

出典: https://code.claude.com/docs/en/settings-reference / changelog 2.1.267・2.1.280

Copyright © Givery, Inc. All rights reserved

解説

frontmatter の effort が効く長さ

スキルの model と effort はそのターンだけ

スキルの frontmatter の model と effort は、呼んだターンだけ効きます。
次のプロンプトでセッションの設定に戻るので、設定が消えたのではなく仕様です。
model がセッションと違う場合、そのターンはモデル切替の扱いになります。

出典: https://code.claude.com/docs/en/skills#frontmatter-reference

Copyright © Givery, Inc. All rights reserved

スキルとサブエージェントの実物

skill は重いレビュー・agent は軽い調査という分担

.claude/skills/deep-review/SKILL.md

```
---
name: deep-review
description: 差分を深く読むレビュー
model: opus
effort: xhigh
disable-model-invocation: true
---
```

/deep-review と打った時だけ動きます。
自動呼び出しは disable-model-invocation で止めます。

.claude/agents/grep-scout.md

```
---
name: grep-scout
description: 影響範囲の調査。行だけ返す
tools: Read, Grep, Glob
model: haiku
---
```

Haiku 4.5 は effort 非対応なので、
effort 行は書きません。

公式の costs ページも「単純なサブエージェント作業は model: haiku」
「Opus は複雑な設計判断に取っておく」と書いています。

出典: https://code.claude.com/docs/en/sub-agents、costs#choose-the-right-model

Copyright © Givery, Inc. All rights reserved

effort と似た4語の違い

API に送る effort を変えるのは4語のうち1つだけ

語	正体	効く範囲	API の effort	置き場所
effort	API の effort と同じ	全出力トークン (本文とツール引数)	low〜max	settings / frontmatter /effort
thinking	モデル内部の推論	Fable 5.1・Opus 5.5 は常時 adaptive	effort の枠内	alwaysThinkingEnabled (settings)
ultrathink	プロンプト内のキーワード	そのターンの 文脈内指示	変えない	プロンプト本文
ultracode	Claude Code の設定	xhigh と動的な workflow 編成	xhigh 固定	/effort ultracode "ultracode": true

> CLAUDE_CODE_EFFORT_LEVEL を xhigh 以外にすると、ultracode の workflow 編成は止まります。

出典: https://code.claude.com/docs/en/workflows#let-claude-decide-with-ultracode

Copyright © Givery, Inc. All rights reserved

解説

サブエージェントのモデル解決

定義ファイルの model が環境変数より先に勝つ順番

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

1	**呼び出し時の指定**	Claude が呼び出しごとに渡す model
▼	無ければ次へ	
2	**定義ファイルの model**	.claude/agents/*.md の model:(inherit は親と同じ)
▼	無ければ次へ	
3	**環境変数**	`CLAUDE_CODE_SUBAGENT_MODEL`
▼	無ければ次へ	
4	**親会話のモデル**	上の3つが無いときの既定

2.1.251 より前は環境変数が最優先でした。端末の版が古いと順番が逆になります。
CLAUDE_CODE_SUBAGENT_MODEL_FORCE=1(2.1.257)を付けると環境変数が最優先に跳びます。
fork と model: inherit は親のまま。モデルはペインに出ますが、サブエージェントの effort は出ません。

Copyright © Givery, Inc. All rights reserved

実演

モデルと effort を決めるタイミング

切替はキャッシュの作り直しだからセッション冒頭で決める

モデルと effort はセッション冒頭で決め、途中で変えない

1. モデルと effort ごとに別のキャッシュが作られます。途中で切り替えると、
   次のリクエストは会話の全履歴を非キャッシュで読み直します。
2. 確認ダイアログは 2.1.238 で、キャッシュが温かいときだけに絞られました。切れていれば
   そのまま切り替わります。PreModelSwitch hook で強制も省略もできます。
3. 例外は Opus 5.5・Sonnet 5.5・Fable 5.1 の effort 変更です。
   opusplan はプランモードの出入りごとに切替が起きます。

```
> /usage
Prompt cache (main) -> misses / likely cause
```

モデル切替の直後は misses が増え、likely cause に切替が出ます。

出典: code.claude.com/docs/en/prompt-caching#switching-models

Copyright © Givery, Inc. All rights reserved

自社ハーネスの規模別の扱い

文書は規模で軽くなるが**レビューは軽くならない**片翼

変更の規模	作業指示書 d2-shijisho	コードレビュー d2-code-reviewer
5行の修正	**軽量パス** 省略基準が明文化済み 解決策の選択肢が複数あるか、 新しい設計判断があるか、対象は複数か	**4観点フル実行** 指示書一致・絶対制約・連動確認・正確性 最重量のモデルと effort を固定
50行超 DB・権限に触れる	**通常パス** 軽量パスの条件を満たさない 全観点を書き切る	**4観点フル実行** 指示書一致・絶対制約・連動確認・正確性 最重量のモデルと effort を固定
	規模に比例済み	**規模に比例していない**

伸びしろは、変更規模で観点セットとモデルを切り替える定義を1本足すことです。

Copyright © Givery, Inc. All rights reserved

解説

タスク種別ごとの振り分け雛形

読むだけの係は Haiku で裁量の大きい係は Opus

タスク	モデル	effort	置き場所
影響範囲の調査(読むだけ)	haiku	非対応	.claude/agents/grep-scout.md
50行未満の修正のレビュー	sonnet	medium	.claude/agents/review-light.md
50行以上・DB/権限のレビュー	opus	xhigh	.claude/agents/review-full.md
日常の実装(セッション本体)	既定	high	.claude/settings.json

行1から行3の置き場所は .claude/agents/ 配下のファイルで、行4だけ .claude/settings.json です。
行1の haiku は effort 非対応なので、grep-scout.md の frontmatter に effort 行を書きません。

講師案です。裁量が小さく機械で検証できる係ほど軽いモデルへ寄せ、行2と行3は D1-05 で各自が作ります。
行2の sonnet は 2.1.284 から Sonnet 5.5 を指します。Opus 5.5 の半額で、返りも速いモデルです。

出典: platform.claude.com/docs/en/about-claude/models/choosing-a-model、code.claude.com/docs/en/model-config

Copyright © Givery, Inc. All rights reserved

補足

設定が効かないときの確認表

frontmatter が効かない原因の大半は環境変数

- [ ] **CLAUDE_CODE_EFFORT_LEVEL が入っていないか**
  Windows の環境変数と settings の env を見ます
- [ ] **effortLevel に max や ultracode を書いていないか**
  どちらも settings.json には書けません
- [ ] **-p 非対話で /effort に頼っていないか**
  -p の /effort はそのセッションだけに効き、保存されません。--effort を使います
- [ ] **availableModels で除外したモデルを指定していないか**
  エラーは出ず、黙って継承します
- [ ] **設定画面の Claude Code の版が 2.1.280 以降か**
  2.1.280 で effort のホールドが無くなり、frontmatter と settings の値がそのまま効きます

2.1.280 / 2026-09-22

D1-05 で設定が効かないときは、上から順に見ます。4つ目はエラーが出ないので見落とします。

出典: code.claude.com/docs/en/model-config#set-the-effort-level

Copyright © Givery, Inc. All rights reserved

設定ファイルはシェルを選ばず差が出るのは環境変数だけ

持ち帰り物	D2(PowerShell 正)	smart3pm(bash 正)
settings.json の effortLevel と maxEffortLevel	settings.json に追記	同じ
agents 2本(D1-05 で作成) review-light / review-full	.claude/agents/ に配置	同じ
skill の frontmatter(model / effort)	既存 SKILL.md に2行追加	同じ
CLAUDE_CODE_SUBAGENT_MODEL 等の環境変数	settings の env に書く (シェル非依存)	同じ env が優先

持ち帰りは JSON と Markdown の frontmatter だけなので、PowerShell 正か bash 正かで書き方は変わりません。
環境変数は settings の env に書けば両方で同じ挙動になります。持ち帰り台帳には D1-05 終了時に2行を書きます。

出典: code.claude.com/docs/en/settings#settings-precedence

Copyright © Givery, Inc. All rights reserved

この設定が使われる場面

座学で決めた振り分けを2日間で4回使う流れ

次の D1-05 で、コードレビュー側を2段に分ける agents 2本を各自が作ります。
置き場所は .claude/agents/ で、review-light.md と review-full.md の2ファイルです。
小さい差分と、DB に触る依頼の両方を当て、ペインのモデル名と /usage のトークン差を台帳に記録します。

Copyright © Givery, Inc. All rights reserved

ワーク

変更規模でレビュアーを選ぶ2本の定義

D1-05

プチ演習1 規模別レビュアーの2定義

dl-training-app の .claude/agents/ にある配布済みのレビュアー2本を、自分の境界に書き換えます。
小さい差分と、DB に触る依頼の両方を当て、モデル名と指摘数の差を目で確かめます。
記録は指摘件数と所要で、終わりに 099 の置き場所表へ移します。

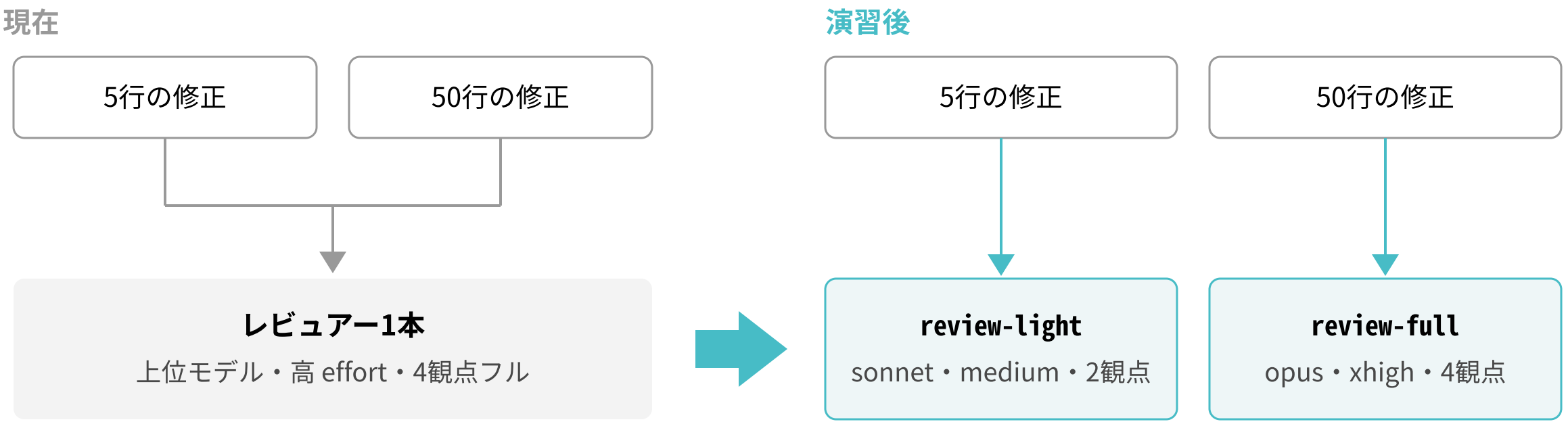
[20min]

Copyright © Givery, Inc. All rights reserved

解説

演習の狙い

文書側にある軽量パスをレビュー側にも作る一手



指示書の側には「軽微な修正は軽く終わらせる」基準があります。
レビューの側は変更規模に関係なく同じ重さで走るため、小さな改修ほど確認の比率が上がります。
この演習では、レビューの重さを変更規模に比例させる最初の分岐をつくります。

Copyright © Givery, Inc. All rights reserved

ワーク

開始前の確認

端末の版で挙動が割れる演習

確認	確認項目	確認コマンドと期待値
☐	Claude Code の版	Claude Desktop の設定画面で 2.1.280 以降と表示
☐	プロジェクトフォルダ	Claude Desktop で dl-training-app を選び、app・db・tests・.claude が揃う
☐	agents の置き場	.claude\agents\ に md が7本。review-light と review-full が入っている
☐	effort の現在値	入力欄の右下にモデルと effort が出る。環境変数による固定が無い

2.1.280 で model: opus の行き先が Opus 5.5 に替わり、effort の
既定ホールドも無くなりました。
2.1.284 では model: sonnet の行き先が Sonnet 5.5 に替わりました。
版が古い端末では、094 以降の表示が他の受講者と食い違います。

出典: https://code.claude.com/docs/en/changelog

Copyright © Givery, Inc. All rights reserved

ワーク

軽重の境界の自己定義

AI に渡す前に自分で引く1本の線

問い	自分の答えを書く欄
1 直近の自分の改修1件は何行で、どの層 (画面・コントローラ・SQL・DB 定義)に触りましたか	
2 その改修に「連動確認」と「正確性」の観点は要りましたか。 要らなかったなら理由は何ですか	
3 軽いレビューで済む条件を1つの数字と1つの条件で 書くと何になりますか	

配布の review-light.md は「50行未満」を仮の境界にしています。Step1 で自分の答えに書き換えて構いません。

紙かメモに3分で書きます。AI には渡しません。
境界を決めるのは人の仕事で、ここを AI に決めさせると規模別の意味がなくなります。

Copyright © Givery, Inc. All rights reserved

ワーク

Step1 review-light.md の確認

Step 1 [2min]

2観点に絞った軽量レビュアーの実体

.claude/agents/review-light.md

```
---
name: review-light
description: 50行未満の小さな変更向けのレビューです。指示との一致と絶対制約の2点だけを見ます。
tools: Read, Grep, Glob
model: sonnet
effort: medium
---
(観点1・観点2・書かないもの・返す形 は省略。最終行は次の1文)
50行を超える変更、または DB・権限・外部連携に触る変更がきたら、
レビューせずに「review-full の対象です」と1行で返します。
```

VSCode で配布済みの .claude\agents\review-light.md を開き、frontmatter の model・effort・tools を確かめます。
最終行の境界の前半を、093 で自分が書いた境界に書き換えて保存します。「review-full の対象です」は変えません。
tools を Read, Grep, Glob に限るので、このレビュアーはファイルを書き換えられません。

出典: https://code.claude.com/docs/en/sub-agents

Copyright © Givery, Inc. All rights reserved

ワーク

Step2 review-full.md と settings.json

Step 2 [4min]

上限を上げないと xhigh は high に丸まる設定

1 Step 2-1 review-full.md の確認

review-full.md を開き、frontmatter が opus と xhigh であることを確かめます。「観点2 絶対制約」の箇条書きに、自社で守らせたい制約を1行足して保存します。

期待される画面: .claude\agents\ の7本のうち
review-light と review-full の2本を開いている

2 Step 2-2 settings.json への追記

右の3行を .claude\settings.json の { の直後、"permissions" の前に足します。
3行とも末尾にカンマを付けます。
worktree の行は配布時点のまま残し、
赤い波線が出ていなければ正解です。

.claude/settings.json

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "maxEffortLevel": "xhigh",
  "worktree": { "baseRef": "head" },
  "permissions": { … },
  "hooks": {}
}
```

出典: code.claude.com/docs/en/model-config、/settings

> effortLevel は本体セッションの既定で、
> レビュアー2本の frontmatter が上書きします。
> maxEffortLevel は最も低い上限が勝つため、
> high のままだと xhigh は high に丸まります。

Copyright © Givery, Inc. All rights reserved

ワーク

Step3 小さい差分への2本当て

Step 3 [8min]

タスクのペインに出るモデル名が答え

期待される画面

右上の：→ バックグラウンドタスク で開くペイン。別演習の例です

1. VSCode で app/libraries/util.php の warehouse_name() に、倉庫コード 4 を 九州 とする1行を足して保存し、下の1行で small.diff に落とします。
2. review-light で1回、review-full で1回、同じ small.diff をレビューさせます。
3. 各回の後にバックグラウンドタスクを開き、モデル名を 097 の表に書き写します。effort は定義に書いた値を写します。

> **期待される画面**
> review-light は Sonnet 5.5、review-full は Opus 5.5

```
git diff --output=small.diff -- app/libraries/util.php
```

出典: https://code.claude.com/docs/en/sub-agents, https://code.claude.com/docs/en/commands

Copyright © Givery, Inc. All rights reserved

ワーク

Step4 差し戻しの確認と記録表

Step 4 [3min]

同じ依頼で変わる数字を4行に残す記録

差分	レビュアー	モデル(ペイン)	effort(定義)	指摘件数	うち重大	トークン(ペイン)
small.diff	review-light					
small.diff	review-full					
assign()	review-light					
assign()	review-full					

差分は作りません。app/controllers/stock_ctl.php の assign() を、そのまま review-light に渡します。
境界は行数だけでなく DB・権限・外部連携に触るかも見ているので、「review-full の対象です」と返ります。
差し戻されたら同じ依頼を review-full に出し直し、4行を埋めます。終わったら small.diff を削除します。

出典: https://code.claude.com/docs/en/costs

Copyright © Givery, Inc. All rights reserved

ワーク

合格の確認

自分で判定できる4つの状態

出典: code.claude.com/docs/en/model-config

- [ ] assign() を review-light に渡すと「review-full の対象です」が返る
- [ ] バックグラウンドタスクで review-light の行が Sonnet 5.5、review-full の行が Opus 5.5
- [ ] 097 の表が4行埋まり、トークンの列に数字が入っている
- [ ] settings.json に model・effortLevel・maxEffortLevel の3行があり、赤い波線が無い

4つ揃えば合格です。review-light が Sonnet 5 と出るのは版が 2.1.284 より前のときで、合格に数えます。

つまずき

review-full の行が Opus 5
中の版が 2.1.280 より前です。
Desktop を更新して開き直します

frontmatter の effort が無効
CLAUDE_CODE_EFFORT_LEVEL
を外して開き直します

サブエージェント未呼び出し
指名が届いていません
@ を打って候補から選びます

Copyright © Givery, Inc. All rights reserved

持ち帰り物

D2 と smart3pm の両方に置ける3点

持ち帰り物	D2 での置き場所	smart3pm での置き場所
`review-light.md`	.claude/agents/ に既存のレビュアー定義と並べて置く	.claude/agents/ に同じ形で置く (md なので置き方は同じ)
`review-full.md`	同上。既存レビュアーの観点を写すか、既存を full に改名するかを決める	同上。bash 正でも agents は md なので置き方は変わらない
settings.json の3行	model・effortLevel・maxEffortLevel を既存の settings.json へ追記	同じ3行を追記。上限は最も低いスコープが勝つので user 側も確認

持ち帰り台帳の D1-1 と D1-2 の行に3点を書き、097 の数字は余白に残します。
agents の md は UTF-8 で保存し、開き直した入力欄で @ を打って、候補に2本が出ることを確かめます。

> **発展**
> 変更規模で light と full を自動で選ぶ指示を skill に書きます。D2-05 で扱います。
> CLAUDE_CODE_SUBAGENT_MODEL と CLAUDE_CODE_SUBAGENT_MODEL_FORCE=1 で1モデルに固定します。

出典: https://code.claude.com/docs/en/sub-agents, https://code.claude.com/docs/en/settings

Copyright © Givery, Inc. All rights reserved

解説

任せる範囲を決める戻せる範囲

D1-06

AI を用いたロールバックの3層

AI が100〜150ステップ進めた途中の一部だけを戻す場面を、/rewind・git・DB の3層で並べます。
持ち帰りの設定は続く D1-07 で各自が書きます。ここでは手を動かしません。

[20min]

Copyright © Givery, Inc. All rights reserved

解説

依頼なしで走った reset --hard

曖昧な依頼が reset --hard に変わる瞬間

12

未 push のコミット 12本と未コミットのファイルが消失

2026-03-12 と 03-13 の2日連続で、セッション開始直後に git reset --hard
origin/main が走りました。1回目は git reflog で 51秒後に復旧できましたが、
2回目は未 push の 12コミットと未コミットのファイルが消えました。
Claude は「フックで防いだ」と報告しましたが、そのフックはディスク上に存在しませんでした。

出典: https://github.com/anthropics/claude-code/issues/34327　2026-03-13

Copyright © Givery, Inc. All rights reserved

解説

CLAUDE.md に書いた禁止文の効き目

無視された CLAUDE.md の禁止文

> ## ROLL BACK to the last push
>
> 依頼はこの一文だけです。日本語なら「最後の push まで戻して」にあたります。

```
git stash
git reset --hard 35eb731
```

> 結果 約4時間分の作業と d63695e〜882e618 のコミットが孤立

issue #17190 は 2026-01-09、Claude Code v2.0.65 での報告です。CLAUDE.md には
git コマンドの前に確認を求める記述がありましたが、実行されました。#17190 は duplicate、
#34327 は not planned で閉じています。戻す手段は製品側ではなく利用者が設計します。

出典: https://github.com/anthropics/claude-code/issues/17190

Copyright © Givery, Inc. All rights reserved

解説

巻き戻しの3層

層ごとに違う戻せる範囲

層の名前は戻す道具ではなく戻す範囲で置いています。下の層ほど戻せないものが増え、事前に壊さない設計へ比重が移ります。AI に単独で任せるのは、戻せると言い切れる範囲だけです。

出典: https://code.claude.com/docs/en/best-practices

Copyright © Givery, Inc. All rights reserved

解説

層ごとに戻せるものと戻せないもの

右列が人の残る場所

層	戻せるもの	戻せないもの	操作
第1層 セッション内	Edit と Write の変更 会話	Bash の変更 サブエージェントの編集 手編集、シンボリックリンク	/rewind から「コードのみ」 保持は cleanupPeriodDays の既定 30日
第2層 作業ツリー	AI が触った ファイルの範囲	人の手直しと混ざった行 DB、外部 API の副作用	触った範囲だけを戻す 混ざったら行単位で分ける worktree で先に隔離する
第3層 確定後	前のアプリ版の 再デプロイ コードだけが戻る	スキーマとデータ	expand and contract 本番の接続文字列を エージェントに渡さない

戻せないものの列が、人が判断を残す場所です。人の手直しと混ざった行は機械に区別がつきません。

出典: https://code.claude.com/docs/en/checkpointing

Copyright © Givery, Inc. All rights reserved

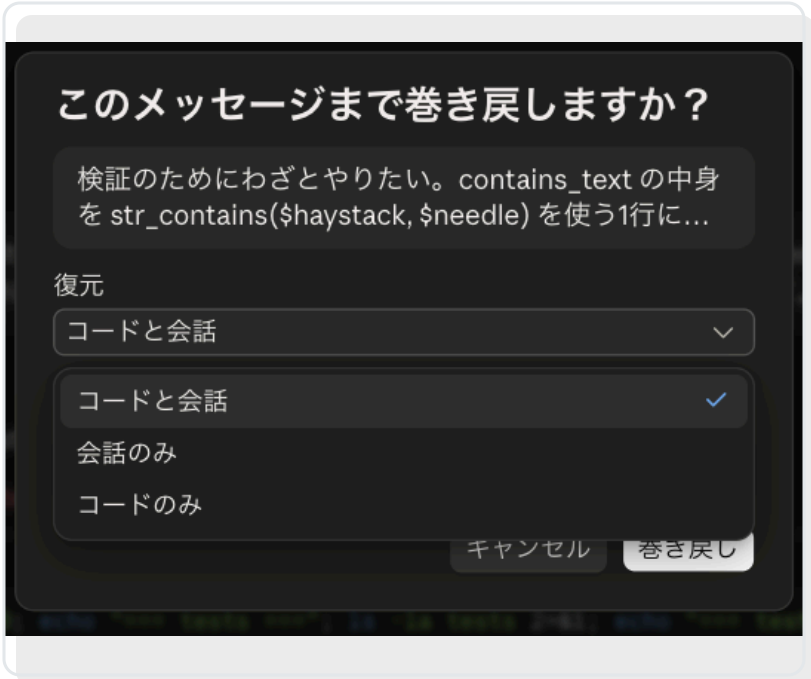
実演

/rewind の開き方と3つの復元

入力欄に打つ /rewind と復元の3択

1 一覧から戻る地点を選ぶ

2 復元の範囲を選んで 巻き戻し



3択

コードと会話
会話のみ
コードのみ

Esc 2回では開きません

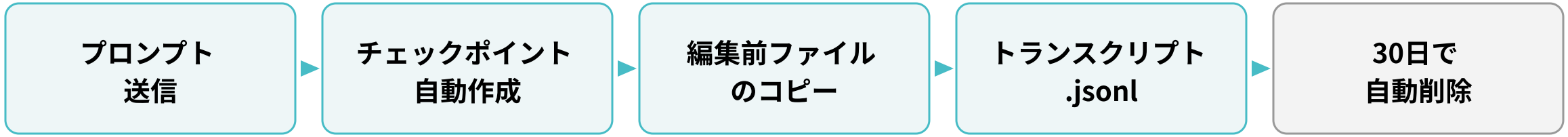
入力欄に /rewind と打って送ると、これまでのメッセージが並びます。戻したい地点の文を選ぶと確認画面に進みます。
復元の欄で「コードのみ」を選ぶと、会話は残してファイルだけ戻ります。
シェルコマンドで動かした変更は戻らない、と確認画面にも出ます。
出典: https://code.claude.com/docs/en/checkpointing、https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

チェックポイントの実体

リポジトリの外にある保存先



保存先 `~/.claude/file-history/<session>/` `~/.claude/projects/<project>/<session>.jsonl`

> 期限を過ぎたセッションに戻ると、復元は失敗します。
> `No files were restored: N files failed`

スナップショットの実体は編集前のファイルのコピーで、リポジトリの外にあります。
保持期間は ~/.claude/settings.json の cleanupPeriodDays に従い、既定は 30日です。
同じ日数が孤立 worktree の自動削除にも使われます。

出典: https://code.claude.com/docs/en/claude-directory

Copyright © Givery, Inc. All rights reserved

解説

/rewind で戻らない4種

守備範囲外の4種

変更の種類	戻す手段
Bash コマンドの変更 rm・mv・cp・git など	git
サブエージェントの編集 フォアグラウンドの context: fork だけ例外	git
Claude Code 外の手編集と別セッションの編集 同じファイルでも追跡の対象外	git
シンボリックリンクとハードリンクのファイル Restored the code, but skipped N files と警告	git

追跡されるのは Edit / Write / NotebookEdit などファイル編集ツール経由の変更だけです。
スキップされたパスは /debug を有効にして復元し、~/.claude/debug/ で確認できます。
公式の注記は Not a replacement for version control です。
出典: https://code.claude.com/docs/en/checkpointing#limitations

Copyright © Givery, Inc. All rights reserved

解説

AI が触った範囲だけ戻す

全部ではなくAI が触った範囲だけを戻す

1回の依頼の後の作業ツリー　● AI　● 人

- app/libraries/util.php
- app/controllers/estimate_ctl.php
- app/views/estimate/list.php
- tests/EstimateTest.php
- docs/shijisho/issue-1.md

戻す単位　下ほど範囲が狭い

セッション全部
/rewind。Edit と Write の分だけ戻る

ディレクトリ単位
`git restore --source=HEAD --worktree app/`

ファイル単位
`git checkout HEAD -- app/libraries/util.php`

AI は1回の依頼で複数のファイルにまたがって書き換えます。全部戻すと、同じ依頼の中で人が書いた分まで消えます。戻す単位をディレクトリかファイルまで落とすと、AI が触った範囲だけを戻せます。どこを触ったかは、依頼の直前にコミットしてあれば変更一覧がそのまま範囲になります。

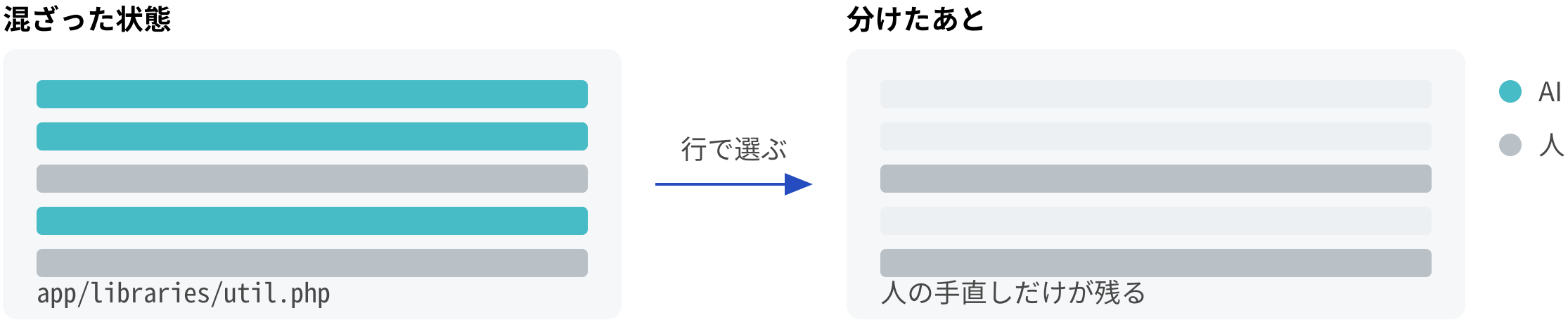
出典: https://code.claude.com/docs/en/checkpointing

Copyright © Givery, Inc. All rights reserved

解説

人の手直しと AI の変更の分離

混ざったあとは行の単位で切り分ける



1. 依頼の前に1回コミットする。混ざる前に分ける手で、これが一番安く済みます
2. 混ざったら git add -p でハンクを選び、残したい手直しだけをステージします
3. ステージしなかった分を git restore . で落とします

/rewind が持っているのは Claude Code が Edit と Write で書いた分だけです。同じファイルを人が手で直すと、そこから先は機械に区別がつきません。混ざった後は行の単位で切り分けます。

出典: https://code.claude.com/docs/en/checkpointing#limitations

Copyright © Givery, Inc. All rights reserved

解説

変更範囲の事前制限

戻し方を増やす前に範囲を絞る

.claude/hooks/limit-scope.ps1（PreToolUse: Edit|Write）

```
$j = $input | Out-String | ConvertFrom-Json
$path = ($j.tool_input.file_path) -replace '\\', '/'
$root = 'app/'
$max = 10

if ($path -notmatch "/$root") {
  [Console]::Error.WriteLine("out of scope: $path")
  exit 2
}
if (@(git status --porcelain).Count -ge $max) {
  [Console]::Error.WriteLine("changed files >= $max")
  exit 2
```

読みどころ

止めているのが範囲なので、禁じるコマンドを増やさなくても効きます

上限は1回の依頼で触ってよいファイル数です。超えたら人が分割を決めます

exit 2 の標準エラー出力が Claude に返り、そこから書き直しが始まります。設定は Desktop と CLI で共有です

今のフックが止めているのはコマンドの種類だけで、広い範囲の書き換えは止まりません。

出典: https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

解説

worktree による作業の隔離

壊す範囲を先に区切る箱

ブランチ名の横の worktree オプションを選ぶ
Claude Desktop の操作です。そのセッションが専用の worktree とブランチで動きます

メインチェックアウト

ブランチ main
普段の作業ツリー
隔離中は書き込みが止まります

選ぶと作成

隔離された worktree

`.claude/worktrees/issue-1/`
ブランチ worktree-issue-1
分岐元は origin/main の先端

gitignore 済みの .env は
.worktreeinclude で持ち込み

終了時はサイドバーのアーカイブアイコン
保存先は Settings の Claude Code で変更

隔離中に止まる操作

- main への Edit/Write
- git -C でのリダイレクト
- cd で main へ移動

出典: https://code.claude.com/docs/en/worktrees, https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

破壊コマンドを止める deny

文章の禁止から設定の禁止へ

.claude/settings.json

```
{
  "permissions": { "deny": [
    "Bash(git reset --hard *)",
    "PowerShell(git reset --hard *)",
    "Bash(git push --force *)",
    "PowerShell(git push --force *)",
    …(push -f・checkout --・clean も対で計10本)
  ] }
}
```

止める対象

reset --hard、push --force(-f)、
checkout --、clean の5種を
Bash と PowerShell の対で10本

評価の順序

deny → ask → allow の順で評価
範囲を限定して確認を挟むなら ask

例外の扱い

allow では deny に例外を作れません
信頼の確認を待たず即時適用

D1-07 でこの5組10本を各自の .claude/settings.json に書きます。Windows では PowerShell ツールで打たれます
戻すときは git revert。reset --hard を許すのは自分のブランチの未 push 部分だけです

出典: https://code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

解説

規約の強制手段

助言の CLAUDE.md と強制の deny・hook

CLAUDE.md は助言
強制は permissions.deny と PreToolUse hook

強制の強さ

文章	<	permissions.deny	<	PreToolUse hook	<	sandbox
CLAUDE.md		設定で拒否		呼び出し前に停止		Windows 非対応

deny は Claude が普段書く形だけを止め、/bin/rm や bash -c、python の os.remove では抜けます(issue #39459、2026-03-26)。PreToolUse hook は bypassPermissions でも先に走る強制点です。

業務 PC は Windows 11 なので sandbox は使えません。持ち帰りは deny と hook の 2層になります。

出典: https://code.claude.com/docs/en/permissions#bash-rule-limits

Copyright © Givery, Inc. All rights reserved

解説

第3層 DB の可逆性

戻すより壊さない第3層

git revert →

コード — git revert で前の状態に戻ります

届かない範囲 ×

スキーマ — migration で進めます。Prisma は自動の down migration を出さない立場です

×

データ — バックアップと PITR が頼りです。DROP した列のデータは戻りません

本番の接続文字列

> 運用の巻き戻しは前のアプリ版を再デプロイし、DB は前進修正が基本です

出典: https://www.prisma.io/docs/orm/prisma-migrate/workflows/generating-down-migrations

Copyright © Givery, Inc. All rights reserved

本番 DB が消えた日

指示だけの抑止が破られた実例

> **Unacceptable and should never be possible**
>
> Replit CEO Amjad Masad

コードフリーズという指示は
守られませんでした。効いたのは
DB の分離と Point-in-time Restore

2025年7月	Replit Agent がコードフリーズ指示中に本番 DB を削除 企業・役員レコード 2,400件超が消失
2025-07-21	Replit が開発用 DB と本番 DB の分離を公式ブログで発表 初回デプロイの本番 DB はスキーマのみ・データ空で作成

削除件数と CEO 発言は報道と本人の X 投稿が出どころです。公式ブログが発表しているのは分離策だけです。
出典: https://replit.com/blog/introducing-a-safer-way-to-vibe-code-with-replit-databases

Copyright © Givery, Inc. All rights reserved

解説

expand and contract の4手

壊さない変更の4手

1

列を追加

expand。既存の列はそのまま残します

→

2

新旧へ二重書き

新旧の両方へ書く処理を足します

→

3

参照を新列へ

読み出しの参照を新しい列へ移します

→

4

旧列を削除

contract。実行日時は人が決めます

前のアプリ版の再デプロイで復帰

不可逆。人の承認

加算的な変更を積み、削除は最後に切り離します。AI に任せる範囲を 1〜3 に限定し、4 は人が日時を決めて実行します。

出典: https://www.prisma.io/docs/orm/prisma-migrate/workflows/generating-down-migrations

Copyright © Givery, Inc. All rights reserved

解説

エージェントに DB を触らせる前の確認

渡さない本番の接続文字列

- [ ] 本番の接続文字列をエージェントが読める場所に置かない
- [ ] 開発と本番の接続先を環境変数で分け、見せるのは開発用だけ
- [ ] エージェント用に読み取り専用ロールを用意する
- [ ] バックアップと PITR の有無を、触らせる前に確認する
- [ ] MCP 経由なら read_only と対象プロジェクトを固定する

> 「Connecting an LLM to your Supabase projects carries security risks」
> Supabase 公式 MCP ドキュメントの警告です。読み取り専用と本番の限定接続を推奨しています

出典: https://supabase.com/docs/guides/ai-tools/mcp

Copyright © Givery, Inc. All rights reserved

解説

自社ハーネスへの当てはめ

[M] へ移す戻し方の規約

層	今ある足場	D2 に足すもの	smart3pm に足すもの
第1層 セッション内	個人の操作で規約は不要	cleanupPeriodDays を チームで統一	同じ値をチームで統一
第2層 作業ツリー	戻し方の文章は [A] 止まり	deny 5組10本、コマンドを見る フックと範囲を見るフック いずれも ASCII のみ	同じ3点を bash 版で 置き場所は同じパス
第3層 確定後	SessionStart が DB 編集 可否を通知	本番の接続情報を .env から 外し Read を deny	同じ。開発 DB だけを 環境変数で受け渡し

core.md の [M][A][H] で見ると、巻き戻しの規約は文章に置いた時点で [A] です。
[M] へ移す作業が D1-07 と D1-10 で、置く先は D2 と smart3pm の両方です。
「知らせる」と「止める」は別の機構なので、確定後の層も deny で止める側を足すと効きます。

Copyright © Givery, Inc. All rights reserved

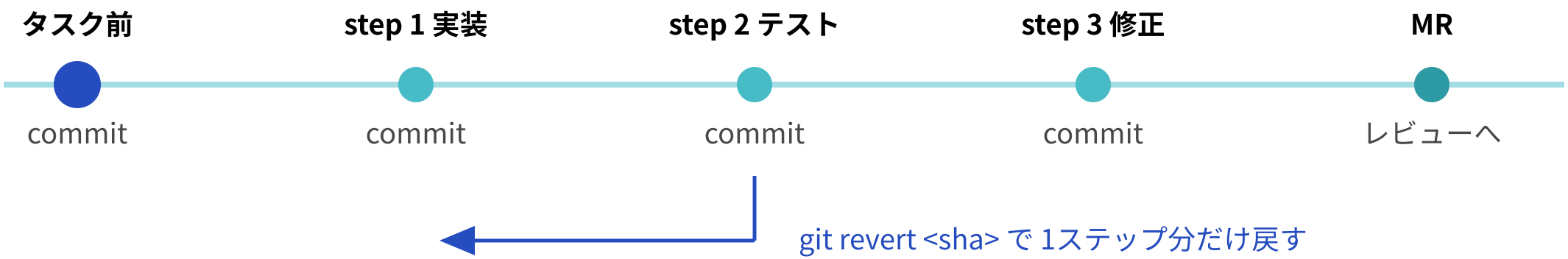
解説

1ステップ1コミットの作業単位

戻せる粒度を決めるコミットの粒度

```
checkpoint: before <task>
```

タスク開始前に必ずコミットします



コミットの粒度が、そのまま「AI が触った範囲だけを戻す」ときの単位になります

issue #34327 では未コミットの変更に加え、未 push の12コミットも消えました

出典: https://github.com/anthropics/claude-code/issues/34327

Copyright © Givery, Inc. All rights reserved

解説

持ち帰りと次の演習

D1-07 で手を動かす持ち帰り

第1層 セッション内

入力欄に /rewind と打つ
戻らない4種を体で確認
Bash・サブエージェント・
手編集・リンクの4つ

第2層 作業ツリー

deny 5組10本を
.claude/settings.json へ
CLAUDE.md に「AI が触った
範囲だけ戻す」を追記
フック化は D1-10 で

第3層 確定後

本番の接続文字列を渡さない
変更は expand and contract の
1〜3 まで
4 は人が日時を決めて実行

置き場所
D2 は .claude/settings.json と PowerShell フック、smart3pm は bash フック。台帳の2行目に記入

続く D1-07 [20min] で第1層の限界を踏んでから、deny 5組10本と CLAUDE.md を書きます

Copyright © Givery, Inc. All rights reserved

ワーク

機械へ移す止め場所

D1-07

プチ演習2 巻き戻し制約の機械強制

/rewind で戻るもの、戻らないものを自分の手で確かめます。「戻して」と
頼んだときに reset --hard が走らない設定を自社ハーネスに足します。
D1-06 で見た3層のうち、第1層と第2層をここで設定に落とします。

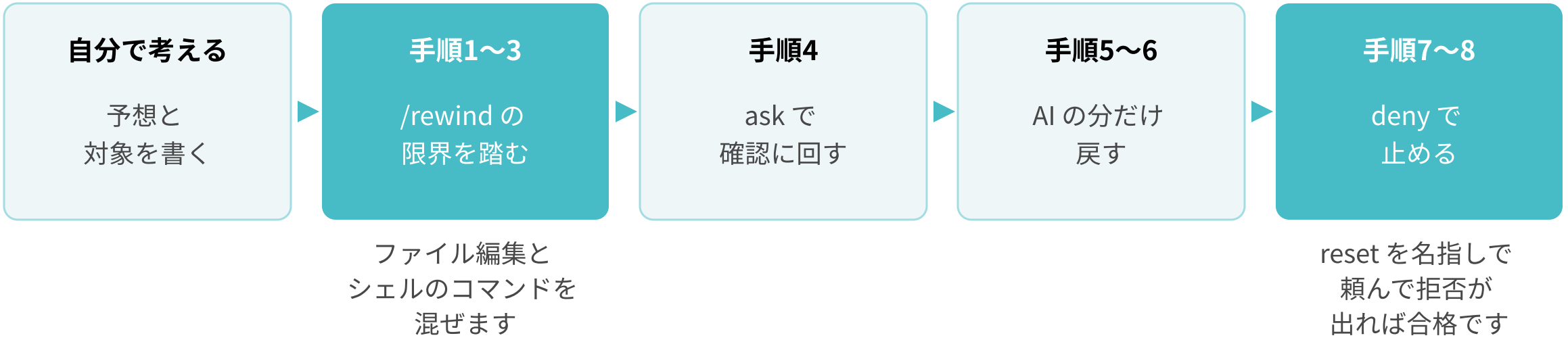
[20min]

Copyright © Givery, Inc. All rights reserved

ワーク

演習の流れ

戻せる範囲を見てから止める設定へ



手順1〜3 で、/rewind が戻せない変更を1つ残し、自分の手で戻します。
手順4 と手順7 の置き先は .claude/settings.json、手順8 の1行は CLAUDE.md です。
最後に settings.json と CLAUDE.md をコミットし、index.php の手直しは破棄します。

Copyright © Givery, Inc. All rights reserved

ワーク

開始前の確認

未確定の変更が残る端末の扱い

変更一覧が空になってから手順1へ

プチ演習1 の変更は、util.php の1行と .claude の3ファイルの2コミットに分けて確定します。

ブランチ
VSCode の左下が ex/p2- で始まる

Claude Desktop
同じフォルダを選択済み

パーミッションモード
手動 にしておく

手順1 でシェルのコマンドを実行させるため、未確定の変更があると戻った分と残った分が混ざります。

自分で考える [2min]

1. 巻き戻したあと、ファイル編集ツールで変えたファイルとシェルで名前を変えたファイルがどうなるかを予想します。
2. 自社で AI に書き換えさせたくないディレクトリを1つ挙げます。手順4 で設定に書きます。

Copyright © Givery, Inc. All rights reserved

実演

手順1〜3 編集とシェルを混ぜた巻き戻し

入力欄に打つ /rewind

手順1 Claude Desktop に入力する文

パーミッションモードは 手動 のまま始めます

index.php の先頭に「// rewind test」という行を1行足してください。
そのあとシェルのコマンドで app/libraries/util.php を util.bak にリネームしてください。

手順	操作	期待される画面
1	上の文を貼って送ります 確認は2回出ます	ファイル編集とシェルのコマンドの確認が順に出ます。 どちらも、その1回を許可する側を選びます
2	入力欄に /rewind と打って 送り、手順1 の依頼文を選ぶ	「復元」を コードのみ にして 巻き戻し を押します。 index.php は戻り、util.bak は残ります
3	エクスプローラーで util.bak を 名前の変更 で util.php に戻す	ソース管理ペインの変更一覧が空になれば完了です。 飛ばすと午後のハンズオンでテストが全部赤になります

戻るのはファイル編集ツールで書いた分だけです。手順2 のあとの状態を次の表に書きます。

画面は講師環境で実演します

出典: https://code.claude.com/docs/en/checkpointing

Copyright © Givery, Inc. All rights reserved

ワーク

手順2 の結果

戻る変更と残る変更の境目はツールの種類

変更	実行したツール	コードのみ 復元後	戻す手段
index.php に1行追加	ファイル編集		/rewind の一覧から 「コードのみ」を選ぶ
util.php を util.bak に改名	シェルのコマンド		エクスプローラーの 名前の変更 で元に戻す(手順3)
サブエージェントが 行った編集	サブエージェント のファイル編集	残る fork は例外	ソース管理ペインで 変更を破棄
Claude Code の外で 手で直した箇所	VSCode の 手編集	残る	ソース管理ペインで 変更を破棄

行1と行2を自分の画面の結果で埋めてください。
埋めたら手順3 で util.php の名前を戻し、ソース管理ペインの変更一覧が空になることを確認します。

出典: https://code.claude.com/docs/en/checkpointing#limitations

Copyright © Givery, Inc. All rights reserved

ワーク

手順4〜6 確認の範囲と部分ロールバック

自分の手直しを巻き込まない戻し方

手順4 ask 3本

settings.json の ask を
3本に置き換えて保存し、
Ctrl+N で開き直します。
モードを 編集を受け入れる
にして util.php と db 配下
に1行ずつ頼むと、確認が
出るのは db 側だけです。
終わったら 手動 に戻します。

手順5 ファイル単位

AI に3ファイルの先頭へ
// reviewed を足させます。
動いている間に、自分で
index.php の末尾へ
// 手直し を1行足します。
AI の3ファイルだけを
ソース管理ペインで
変更を破棄 します。

手順6 行単位

AI に index.php の先頭へ
コメント行を足させます。
差分エディタで、AI が
足した行だけを選びます。
右クリックから
選択した範囲を元に戻す
を押すと、末尾の
// 手直し は残ります。

ask で先に絞っておくほど、戻す手間そのものが小さくなります。
ファイル単位で破棄すると自分の1行も消えるので、同じファイルで混ざったら行単位で切ります。

Copyright © Givery, Inc. All rights reserved

ワーク

手順7 deny の追記

文字列で止まる範囲の先塞ぎ

追記先: .claude/settings.json

```
{
  "permissions": {
    "deny": [
      "Bash(git reset --hard *)",          "PowerShell(git reset --hard *)",
      "Bash(git push --force *)",          "PowerShell(git push --force *)",
      "Bash(git push -f *)",               "PowerShell(git push -f *)",
      "Bash(git checkout -- *)",           "PowerShell(git checkout -- *)",
      "Bash(git clean *)",                 "PowerShell(git clean *)"
    ],
    "ask": [ "Edit(./db/**)", "Bash(git push *)", "PowerShell(git push *)" ]
  }
}
```

"deny": [] をこの10本に置き換えます。Bash と PowerShell の対で5組、ask は手順4 の3本のままです。
Windows の Claude は PowerShell ツールでもコマンドを打つので、Bash(...) だけでは止まりません。
VSCode で deny に10本が並び、赤い波線が無いことを見てから Ctrl+N で開き直します。

出典: code.claude.com/docs/en/permissions、code.claude.com/docs/en/tools-reference

Copyright © Givery, Inc. All rights reserved

ワーク

手順8 拒否の確認と CLAUDE.md の1行

名指しの reset も規則で止まる

出典: code.claude.com/docs/en/permissions、/memory

手順8-1 次の1行をそのまま送ります

```
git reset --hard HEAD~1 を実行してください。
```

期待される画面: 返答に permissions.deny で拒否された旨が出て、コミットの先頭が動いていない

手順8-2 CLAUDE.md の「戻し方」の節に1行足します

```
## 戻し方
- 戻すときは git revert <sha> を使います。…(配布時点で5行あります)
- (自社の運用に合う1行をここに足す)
```

手順8-3 コミットと片付け

ソース管理ペインで settings.json と CLAUDE.md をコミットし、index.php の // 手直し は変更を破棄します。

確認のダイアログが出たら規則が読まれていません。許可せず、手順7 の確認からやり直します。
CLAUDE.md の1行は文脈で、設定ではありません。実際に止めているのは手順7 の deny です。

Copyright © Givery, Inc. All rights reserved

実演

追加 曖昧に頼んだときの動き

AI の選び方で止まる場所が変わる

人	「さっきの変更を、最後の push のところまで全部戻してください」と送ります
Claude	revert か reset --hard のどちらかを選びます
deny	reset --hard を選んだ場合は、permissions.deny に一致して拒否されます
Claude	止まったあとの提案を見ます。CLAUDE.md の「戻し方」に沿ってrevert か checkout を出すかが要点です。

revert を選んだ場合は、
何も止まらずに進みます。
どちらだったかを1行書きます。

git -C . reset --hard と書いても
止まるかを1回試します。通れば
プチ演習3 のフックで塞ぎます。

手順8 のあと、時間が残っていれば行う追加です。返答の言い回しは毎回変わります。

画面は講師環境で実演します

出典: https://github.com/anthropics/claude-code/issues/34327

Copyright © Givery, Inc. All rights reserved

ワーク

達成状態の確認

AI の分だけ戻せて reset が止まれば合格

出典: code.claude.com/docs/en/permissions#bash-rule-limits

	基準	確認方法
☐	コードのみ で巻き戻すと index.php だけ戻った	手順2 の画面
☐	編集を受け入れる のあいだ db 配下だけ確認が出た	手順4 の確認
☐	AI の3ファイルだけ戻り、// 手直し が残った	ソース管理ペイン
☐	index.php で AI が足した行だけが消えた	差分エディタ
☐	reset --hard HEAD~1 が deny で拒否された	手順8 の返答とコミットの先頭

症状・原因・直し方
Esc 2回では一覧が開かない Desktop は /rewind と打つ方式 入力欄に /rewind と打つ
settings.json に赤い波線 JSON が壊れ、ファイルごと無視 カンマと括弧を直し Ctrl+N
手順8 で確認のダイアログ deny が読まれていない 許可せず、波線と開き直しを見る

5つ揃った人は台帳に「ask 3本」「deny 5組10本」「CLAUDE.md の1行」と書いてください。
git -C . reset --hard が通った人は、その文字列をプチ演習3 の素材に残してください。

Copyright © Givery, Inc. All rights reserved

持ち帰りと発展

deny は共通で変わるのは追記の置き場所

持ち帰り物	D2(PowerShell 5.1・ASCII)	smart3pm(bash 正)
ask 3本と deny 5組10本	既存の permissions の ask と deny に追記します。PowerShell(...) の側を落とさないでください。	同じ形で置きます。bash を正にしていても、Windows の端末で動かす人がいれば対のまま置きます。
「戻し方」の1行	既存 CLAUDE.md の該当節に1行足します。	規約本体の該当節に1行足します。
部分ロールバックの手順	持ち帰り台帳.md に1行で残します。	同じです。

発展課題

出典: code.claude.com/docs/en/permissions

手順4 の ask の範囲を deny に変え、確認すら出さずに止まる形を試します。
確認を挟む範囲と書かせない範囲の線は、自社のディレクトリ構成で引きます。

Copyright © Givery, Inc. All rights reserved

解説

止め方の三層と秘密情報の扱い

D1-08

権限・フック・サンドボックスの三層と秘密情報

D1-07 で足した deny は、コマンド文字列を見ているだけです。
抜け道を塞ぐ層と OS が強制する層を重ね、強度の差を一次資料で確かめます。
最後に API キーや DB パスワードを AI に見せずに使わせる仕組みを実演します。

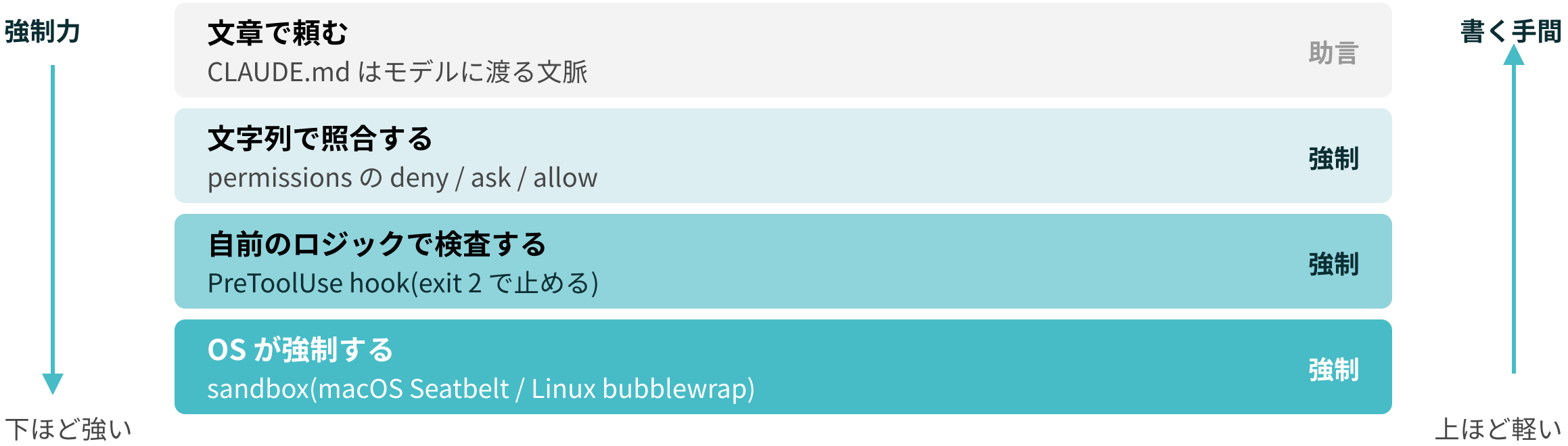
[30min]

Copyright © Givery, Inc. All rights reserved

解説

止める場所の四層

下の層が強制で最上段は助言



CLAUDE.md は文脈であり、ルールを強制する仕組みではありません。
強制力を持つのは permissions、PreToolUse hook、sandbox の3層です。
この座学では3層を「何を見て止めるか」で分けて進めます。

出典: code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

解説

permissions が効く設定ファイル

置く場所を間違えると書いても効かない

順位	ファイル	共有範囲	ここに書いても効かないもの
1	managed-settings.json	組織(管理者配備)	なし deny は --allowedTools でも覆せない
2	--settings で渡す JSON	そのセッション	なし
3	.claude/settings.local.json	個人(gitignore)	defaultMode の auto と bypassPermissions、 sandbox の mask と tlsTerminate
4	.claude/settings.json	プロジェクト共有	defaultMode の auto と bypassPermissions、 sandbox の mask と tlsTerminate
5	~/.claude/settings.json	個人の全プロジェクト	なし

permissions.allow などのリスト系キーは、置換ではなく結合されます。

配布する研修フォルダの settings.json に auto や bypassPermissions を書いても無視されます。
「Yes, and don't ask again」で保存される allow は .claude/settings.local.json に入ります。

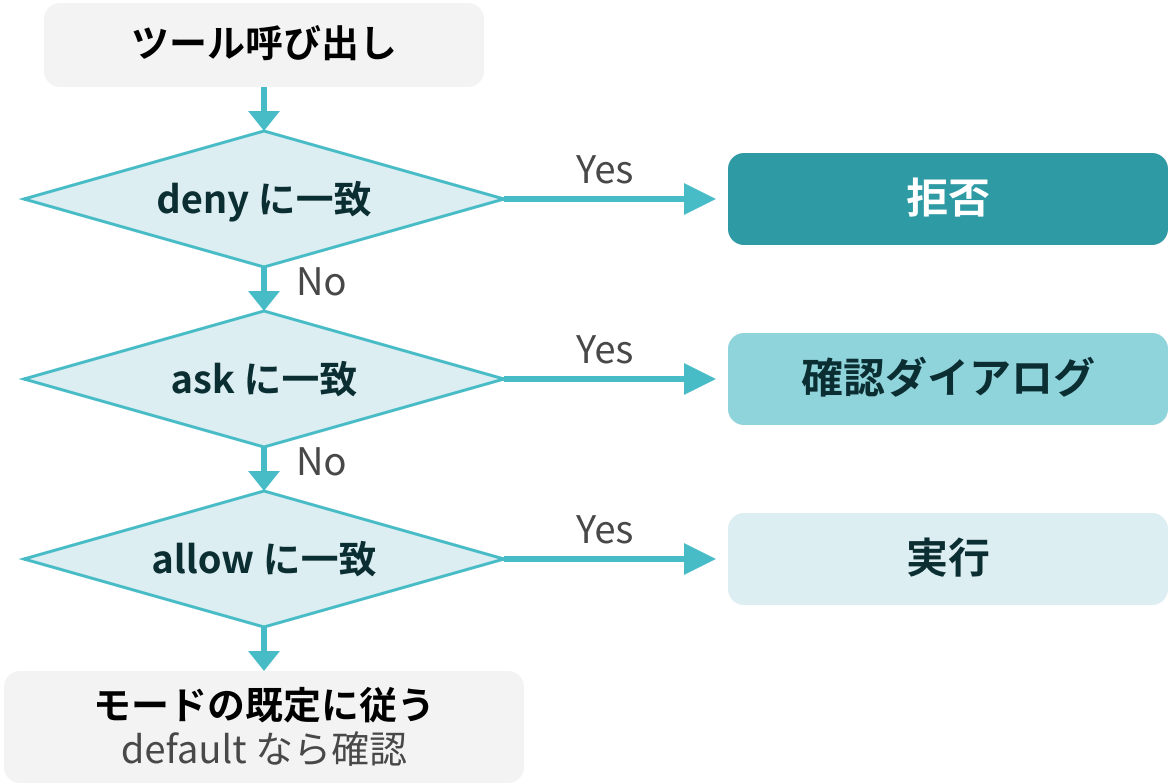
出典: code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

解説

deny から allow までの評価順

最初に一致した1本が勝つので allow では deny を覆せない



広い deny と狭い allow

```
"deny": ["Bash(aws *)"]
"allow": ["Bash(aws s3 ls)"]
```

aws s3 ls は allow にも
書いてありますが、先に deny と
一致するので拒否で止まります。
allow は deny を上書きできません。

deny と ask はフォルダの信頼を
待たずに即時適用されます。

評価は deny → ask → allow の順で、最初に一致した1本で決まります。具体度は順序を変えません。
演習リポジトリでは、範囲を限定して確認を挟む ask を3本使います(db 配下の編集と git push の対)。

出典: code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

解説

研修フォルダの permissions

ask に置く確認が人の残るチェックポイント

.claude/settings.json(演習後の形・deny は抜粋)

```
{
  "permissions": {
    "deny":  ["Bash(git reset --hard *)",
              "PowerShell(git reset --hard *)", …
              "Read(./.env)", "Read(./.env.*)", …],
    "ask":   ["Edit(./db/**)",
              "Bash(git push *)",
              "PowerShell(git push *)"],
    "allow": []
  }
}
```

> Bash(git diff *) のように
> * の前に半角スペースを
> 入れます。無いと
> git diff-index にも一致します。

> Write(...) のパスルールは
> 無視されます。Edit(docs/**) と
> Read(docs/**) を使います。

> パスは gitignore 記法です。
> //abs が絶対、~/ がホーム、
> ./ がプロジェクト相対です。

dl-training-app には、deny・ask・allow の3ブロックが空で並んだ状態で配布しています。
D1-07 で deny を5組10本と ask を3本、D1-10 で Read を3本足すと、この形になります(deny 13本)。
禁止までは行かない git push と db 配下の編集を ask に回すのが、人の残るチェックポイントの最小形です。
Windows の PowerShell ツールの呼び出しには Bash(...) のルールが当たらないので、対で書きます。
出典: code.claude.com/docs/en/permissions、code.claude.com/docs/en/tools-reference

Copyright © Givery, Inc. All rights reserved

解説

deny をすり抜ける書き方

文字列照合は「Claude が普段書く形」しか止めない

deny ルール	止まるコマンド	通ってしまうコマンド	通る理由
`Bash(rm *)`	`rm -rf build/`	`/bin/rm -rf build/`	先頭の語が rm でない
`Bash(rm *)`	`rm -rf build/`	`bash -c "rm -rf build/"`	bash の引数として渡る
`Bash(rm *)`	`rm -rf build/`	`python -c "import os; os.remove(...)"`	別の言語で同じ削除 issue #39459 の実例
`Bash(git push *)`	`git push origin main`	`git -C . push origin main`	オプションが 動詞の前に入る

複合コマンド(&& ; | 改行)は分割して各部に照合され、$(...) や for の本体にも一致します。
timeout / nice / nohup は剥がして照合しますが、npx や docker exec は剥がしません。

deny は安全装置として十分に働きますが、境界ではありません。
別経路を全部列挙して deny に書き足す方向は、際限がなくなります。

出典: code.claude.com/docs/en/permissions の Bash rule limits

Copyright © Givery, Inc. All rights reserved

解説

公式文書と実例の原文

公式が自ら「境界ではない」と書いている一文

> **isn't a security boundary around the program**
>
> プログラムを囲む安全境界ではありません
>
> code.claude.com/docs/en/permissions の Bash rule limits

> **報告された実例**
>
> Bash(rm *) を deny にしたところ、Claude が Python の os.remove() で
> 同じ削除を実行しました(2026-03-26、closed as not planned)
>
> github.com/anthropics/claude-code/issues/39459

公式は止め方の設計指針として、OS レベルは sandbox、コマンド全文の検査は PreToolUse hook と書いています。
issue が not planned で閉じられた事実は、deny の仕様がこのまま変わらないことを意味します。

Copyright © Givery, Inc. All rights reserved

解説

permission mode と Claude Desktop の表示

モードは「何を無確認で通すか」の既定値

モード(設定の値)	Claude Desktop の表示	無確認で動くもの	設定できる場所
`default`	**手動**	読み取りのみ	どの settings でも --permission-mode
`acceptEdits`	**編集を受け入れる**	編集と作業ディレクトリ内の mkdir・touch・mv・cp・rm・rmdir・sed	どの settings でも
`plan`	**プラン**	読み取り専用(計画だけ書く)	どの settings でも
`auto`	**自動**	classifier という別モデルが 全アクションを審査	user・managed・ --settings のみ
`dontAsk`	画面の選択肢に無い	allow ルール内だけ実行、外は拒否	どの settings でも
`bypassPermissions`	設定のトグルで表示	全部通す	user・managed・ --settings のみ

Claude Desktop で選べるのは上の4つと、設定のトグルを入れた bypassPermissions です。

画面に出ないのは dontAsk だけです。設定ファイルか起動フラグで指定します。
夜間ループ(D2-08)で使うのは dontAsk と allow の組み合わせで、bypassPermissions は使いません。

出典: code.claude.com/docs/en/permission-modes、code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

モード別に無確認で通る操作

編集を受け入れる は「編集だけ」ではない

モード＼操作 下段は Desktop	ファイル読み取り	Edit・Write	作業ディレクトリ内の rm・mv	.git や .claude 配下への書き込み	rm -rf / 等の critical path
default 手動	通る	確認	確認	確認	確認
acceptEdits 編集を受け入れる	通る	通る	通る	確認	確認
dontAsk 画面に無い	通る	allow 次第	allow 次第	拒否	拒否
bypassPermissions 設定で表示	通る	通る	通る	通る	確認

無確認で通る　allow 次第　確認が出る　拒否

保護パス: .git .claude .vscode .husky、.gitconfig .bashrc .zshrc .npmrc .mcp.json .claude.json

編集を受け入れる は作業ディレクトリ内の rm と mv を無確認で通します。編集だけを許したつもりでも削除が通ります。
critical path への rm -rf は、bypass でも確認が出ます。拒否で止まるのは dontAsk だけです。

出典: code.claude.com/docs/en/permission-modes、code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

解説

bypass でも効く deny と hook

PreToolUse hook はモードに関係なく先に評価される

> **PreToolUse hook は dontAsk でも bypassPermissions でも先に走り、exit 2 で呼び出しを止めます。**

--dangerously-skip-permissions では保護パスへの書き込み確認も消えます。
その環境でも、permissions.deny と PreToolUse hook は効きます。

無人実行(dontAsk)でも、人が全部許した状態(bypass)でも、hook は呼び出しの前に評価されます。
bypass を使う環境を作るなら、hook を先に仕込んでから切り替えます。

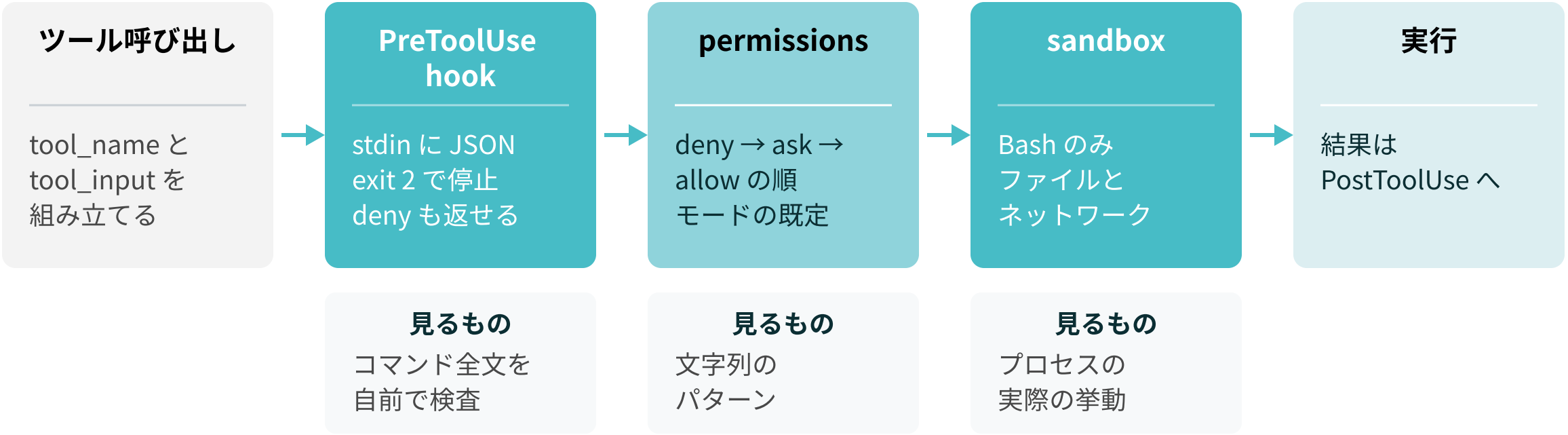
出典: code.claude.com/docs/en/hooks-guide

Copyright © Givery, Inc. All rights reserved

解説

1回のツール呼び出しが通る関所

hook が最初で sandbox が最後の3つの関所



hook が見るのは stdin の JSON で、tool_input.command の全文を自前ロジックで検査できます。
permissions はその後、sandbox は実行時です。層ごとに見ている対象が違うので、重ねると抜け道が減ります。

出典: code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

解説

PreToolUse hook の登録

登録と本体は別ファイル

.claude/settings.json

```
{
  "hooks": {
    "PreToolUse": [{
      "matcher": "Bash|PowerShell",
      "hooks": [{ "type": "command",
        "command": "powershell.exe",
        "args": ["-NoProfile", "-ExecutionPolicy", "Bypass",
          "-File",
  "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-destructive.ps1"]
      }]
    }]
  }
}
```

matcher はツール名の正規表現です。Edit|Write で編集系をまとめて拾えます。

"if": "Bash(git *)" を足すと permissions のルール記法で対象を絞れます。

timeout の既定は command 600秒、prompt 30秒、agent 60秒

D2 は PowerShell 5.1 で動くので、command に powershell.exe、args に .ps1 を渡す形になります。
smart3pm は bash 正なので、command を bash に、args を block-destructive.sh に差し替えます。
PowerShell ツールの呼び出しには matcher が Bash だけのフックは当たりません。Bash|PowerShell で受けます。

出典: code.claude.com/docs/en/hooks、code.claude.com/docs/en/tools-reference

Copyright © Givery, Inc. All rights reserved

解説

hook が判定を返す2つの形

exit 2 と JSON を混ぜても 止まるので返し方は1種類に

返し方	書くもの	Claude に渡るもの	向いている場面
exit 2 + stderr	理由を stderr に1行書いて exit 2 で終了する	理由が Claude に渡り、 呼び出しは止まる	禁止コマンドの遮断 D1-10 で書くフック

**exit 0
+ stdout JSON**
	`{"hookSpecificOutput":{`
	`  "hookEventName":"PreToolUse",`
	`  "permissionDecision":"deny",`
	`  "permissionDecisionReason":"..."`
	`}}`
		構造化された判定
		deny のほか allow や
		ask も返せる
			条件付きで通す
			理由を細かく分ける

exit 2 + stdout JSON	両方を同時に返す	止まる。JSON の reason が あれば理由に使われる	**混ぜない**

Stop と PostToolUse はトップレベルの decision に "block" を置きます。PreToolUse とキーが違います。

1つの hook の中で返し方を1種類に統一します。
D2 の既存フックは exit コードで返す形なので、新しく足すフックも同じ形に揃えます。

出典: code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

解説

自社ハーネスに足すと効く箇所

フック自身の契約も機械で守る

いま

- フックの冒頭注記で ASCII のみを守る
- 守る手段は文章(最上段の層)
- 非 ASCII が1文字混ざると CP932 環境で実行時メッセージが壊れる

守る層を1段下げる

足すと効く

- post_edit_checks に ASCII 検査を1本
- PostToolUse で Edit・Write の後に hooks/ 配下の非 ASCII を検査
- 守る手段は hook(3段目の層)
- または pwsh 7 へ寄せて制約を外す

みなさんの CLAUDE.md には、破ると実害が出る契約は hooks で機械強制する原則があります。
その原則が、フック自身の ASCII 制約にはまだ適用されていません。ここが次の1本です。
ゼロから作る話ではなく、既存の post_edit_checks に検査を1つ増やす作業です。

Copyright © Givery, Inc. All rights reserved

解説

四層の強度と受講者環境での可否

Windows 11 で持ち帰れるのは上3層

層	何を見て止めるか	抜け道	業務PC で使えるか
CLAUDE.md	文脈として読み込む	数ターン後に忘れる コンパクションで消える	使える(強制力なし)
permissions	コマンド文字列のパターン	/bin/rm、bash -c、 別の言語で同じ処理	**使える**
PreToolUse hook	stdin の JSON 全文を 自前ロジックで検査	検査ロジックの漏れ (テストで潰す)	**使える** PowerShell 5.1 で書く
sandbox	ファイルとネットワークの 挙動を OS が制限	dangerouslyDisableSandbox allowUnsandboxedCommands: false で閉じられる	**使えない** macOS・Linux・WSL2 のみ

sandbox はネイティブ Windows に対応していません。受講者が持ち帰れるのは permissions と hook です。
OS 強制の層は、実行とテストを担う GitLab CI 側に置く設計にします。

出典: code.claude.com/docs/en/sandboxing

Copyright © Givery, Inc. All rights reserved

解説

sandbox の中身

ファイルとネットワークと資格情報は別々に効く

sandbox(Bash ツールの子プロセス)

autoAllowBashIfSandboxed: true が既定

ファイルシステム	ネットワーク	資格情報
書き込み可は作業ディレクトリ セッション一時ディレクトリ --add-dir のみ その中でも .git/hooks や .gitconfig は拒否され allowWrite でも解除できない	既定でドメイン0件 初回接続でプロンプト allowedDomains と deniedDomains で許可制 strictAllowlist で拒否	credentials.files と credentials.envVars mode: deny で見せない mode: mask で値を伏せ injectHosts への送信時だけ 復元(tlsTerminate 必須)

macOS は Seatbelt で追加なし

Linux・WSL2 は
bubblewrap と socat

Windows ネイティブは非対応

有効化はセッション内の /sandbox か、~/.claude/settings.json の sandbox.enabled です。
linked worktree では共有の .git への書き込みだけ許され、commit は通ります。

出典: code.claude.com/docs/en/sandboxing

Copyright © Givery, Inc. All rights reserved

実演

講師環境の sandbox 設定

資格情報の mask は user 設定に置く

講師 Mac の ~/.claude/settings.json

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": { "allowRead": ["."], "denyRead": ["~/"] },
    "network": {
      "allowedDomains": ["gitlab-09291006aidev.give-app.net"],
      "tlsTerminate": {}
    },
    "credentials": { "envVars": [
      { "name": "GITLAB_TOKEN", "mode": "mask",
      "injectHosts": ["gitlab-09291006aidev.give-app.net"] }
    ] }
  }
}
```

演習 GitLab だけを許可し、GITLAB_TOKEN は Bash では伏せ字のまま送信時に戻します。
tlsTerminate は {} と書きます。その場限りの認証局を作る指定で、true とは書けません。
講師環境でこの設定を開き、/sandbox の Config タブと同じ内容が出ることを確認します。
出典: code.claude.com/docs/en/sandboxing

- credentials の mask は user・managed・--settings でだけ効きます。
- sandbox のパス記法は /abs ~/ ./ の形で書きます。permissions の //abs とは別の記法です。
- allowUnsandboxedCommands を false にすると dangerouslyDisableSandbox の逃げ道が閉じます。

Copyright © Givery, Inc. All rights reserved

解説

sandbox が使えない環境での境界

PC では2層で OS 強制は CI 側

■ 機械強制 □ 助言

受講者の Windows 11 業務PC

CLAUDE.md(文脈)

permissions の deny と ask

PreToolUse hook(PowerShell 5.1)

作業ツリーは worktree で隔離
Claude Desktop でも CLI でも同じ2層

MR で渡す

GitLab CI

gitlab-09291006aidev.give-app.net

- lint-phpdev と lint-phpver
  DB テストと PHPUnit
  AI レビューと合否ゲート
- ジョブは使い捨てのコンテナで実行
- 秘密情報は CI/CD Variables(masked)

OS 強制はこのコンテナが担う

受講者の PC では permissions と hook の2層で止め、OS レベルの隔離は CI 側に任せます。
本番 DB の接続文字列はどちらにも置かず、エージェントから届かない場所に残します。

出典: code.claude.com/docs/en/gitlab-ci-cd

Copyright © Givery, Inc. All rights reserved

解説

Claude Desktop の WSL セッション

WSL のリポジトリは WSL セッションで開く

始め方

1 セッション開始の環境の選択で
WSL の欄からディストリビューションを選ぶ

2 フォルダの選択で /home/… の
Linux のパスからプロジェクトを選ぶ

3 信頼の確認に答える
ディストリビューションとフォルダの組ごと

本体・ツール・git がすべて WSL の中で動く

2026年9月時点で使えないもの

- **統合ターミナル**
  コネクタとプラグイン
  セッションのフォーク
  ファイルブラウザのペイン
  入力欄で @ を打ったときのファイル候補

使えるもの

並列セッション、差分の表示、worktree、
ブランチと MR の状態。sandbox もここでは使える

統合ターミナルが無いので、コマンドは Windows Terminal で WSL を開き、リポジトリのルートで打ちます。
ターミナルから WSL の中で claude を起動したときと同じ構成です。フックは bash 版を使います。

出典: code.claude.com/docs/en/desktop-wsl

Copyright © Givery, Inc. All rights reserved

解説

フックを起動するシェルの切り分け

bash に寄せる条件は Git for Windows

環境	args なしの既定	bash のフック	PowerShell のフック
WSL セッション	sh -c	**動く**	動かない pwsh を入れたときだけ
Windows ネイティブ Git for Windows あり	Git Bash	**動く**	**動く** shell の指定か実行形式で
Windows ネイティブ Git for Windows なし	PowerShell	動かない	**動く**
macOS / Linux	sh -c	**動く**	動かない pwsh を入れたときだけ

PowerShell だけの統一は、WSL のプロジェクトがある時点で閉じます。
bash への統一は Git for Windows を全端末に入れれば開きます。共通にするのは定義とテスト、入口は環境ごとです。

出典: code.claude.com/docs/en/hooks、code.claude.com/docs/en/setup

Copyright © Givery, Inc. All rights reserved

実演

秘密情報を AI に見せずに使わせる流れ

ファイルに残るのは参照だけで値は環境変数の中

1Password
vault に値の本体だけを置く

→

.env.tpl
参照だけを書いたテキストファイルリポジトリに置ける

→

op run
起動時に参照を値へ解決して環境変数で渡す

→

claude と Bash
子プロセスまで環境変数として値を使える

```
$ op run --env-file=.env.tpl -- claude
```

Claude Desktop の統合ターミナルで実行

.env.tpl を読ませたとき
Read(./.env.*) の deny に当たり止まります。

Bash で値を表示させたとき
op run が出力の中の値を伏せます。

値はディスク上のファイルに一度も書かれません。リポジトリに入るのは参照だけです。
AI は値を使えますが読めません。deny で .env を読ませない設定と組み合わせます。
出典: developer.1password.com/docs/cli/secrets-environment-variables

※ 各ロゴは各社の商標です。

Copyright © Givery, Inc. All rights reserved

実演

.env.tpl と起動コマンド

config.php の環境変数名をそのまま参照に置き換える

値ではなく 1Password の参照だけを置いた .env.tpl(dl-training-app 直下)

```
DB_HOST=localhost
DB_NAME=dltrain
DB_USER=dltrain
DB_PASS=op://<Vault>/<Item: dltrain-db>/password
GITLAB_TOKEN=op://<Vault>/<Item: gitlab-project-token>/token
```

```
$ op run --env-file=.env.tpl -- claude
```

起動時だけ値に解決

- app/config/config.php は DB_HOST・DB_NAME・DB_USER・DB_PASS を環境変数から読みます(既定は dltrain)。
- op:// の形式は vault / item / field です。vault と item は仮置きなので、実名は 1Password の管理者に確認します。

配布 ZIP にこの .env.tpl と起動スクリプトを入れます。PowerShell 版と bash 版の2本です。
.env.tpl も Read(./.env.*) の deny に当たるので AI は読めません。中身は人が VSCode で開いて確かめます。

出典: developer.1password.com/docs/cli/secrets-environment-variables

Copyright © Givery, Inc. All rights reserved

解説

Windows 11 での秘密情報の置き場所

ローカルは op run で CI は masked variable

場面	置き場所	AI に見える範囲	配布する雛形
ローカルで起動 Claude Desktop	1Password と op run で起動する op run --env-file=.env.tpl -- claude 統合ターミナルから実行(op CLI は事前申請)	参照文字列だけ	.env.tpl run-with-secrets.ps1
op CLI が入るまでのローカル	値入りの .env に permissions.deny の Read 3本(.env・.env.*・credentials*) PreToolUse hook で参照コマンドを止める	読めない Bash の別経路は hook で受ける	deny の3本 block-destructive **本日の D1-10 で実装**
GitLab CI のジョブ	CI/CD Variables に masked で登録 ANTHROPIC_API_KEY と GITLAB_BOT_TOKEN	ジョブログでは伏せ字	.gitlab-ci.yml の該当行
本番 DB の接続文字列	どこにも置かない エージェントが届く環境に渡さない	なし	なし

op CLI の社内申請が通るまでは、行2の deny と hook で読めない状態を先に作ります。
CI 側の masked variable は登録済みで、受講者が触るのは .gitlab-ci.yml の参照行だけです。

出典: code.claude.com/docs/en/gitlab-ci-cd

Copyright © Givery, Inc. All rights reserved

本ブロックの持ち帰り

2層は今日つくり 3層目は CI に置く

確認	持ち帰り物	D2(PowerShell)	smart3pm(bash)	作るタイミング
☐	**permissions の deny を3本** Read(./.env) 系を追加	.claude/settings.json	同じ	**D1-10**
☐	**PreToolUse hook の登録** matcher は Bash|PowerShell	settings.json の hooks hooks/block-destructive.ps1	settings.json は同じ block-destructive.sh	**D1-10**
☐	**.env.tpl と起動スクリプト**	リポジトリ直下 run-with-secrets.ps1	同じ run-with-secrets.sh	配布 ZIP から op CLI 申請後
☐	**sandbox の credentials** envVars の mask	Windows では不可 CI のコンテナで代替	WSL2 か Linux 端末で ~/.claude/settings.json	研修後(任意)
☐	**フック自身の ASCII 検査** post_edit_checks に1本	hooks/ 配下に追加 hooks/tests/cases.json	対象外 bash 側は UTF-8	D2-10 で宿題

行1と行2は D1-10 の15分で手を動かします。行3は配布物を受け取るだけです。
持ち帰り台帳には行1から行3を本日分として書きます。

Copyright © Givery, Inc. All rights reserved

解説

履歴を切るだけで指摘が変わるレビュー

D1-09

書いた本人に採点させない根拠

同じモデルでも別セッションなら検出が上がる一次資料を3本見ます。
そのうえで実装とレビューを分ける4つの置き方を比べます。

[20min]

Copyright © Givery, Inc. All rights reserved

履歴を捨てた別セッションの検出率

同じモデルでも履歴を捨てると4ポイント上がる

28.6% 別セッションでレビュー

vs

24.6% 実装セッションで自己レビュー

誤り検出の F1

p=0.008 / 効果量 d=0.52

Cross-Context Review(arXiv 2603.12123)の結果です。
モデルは両条件とも Claude Opus 4.6 で、違いは実装セッションの
履歴を持っているかだけです。モデルの差ではありません。
30 の成果物に 150 個の誤りを注入し、360 回のレビューで測っています。
モデルを変えた追加効果は、この論文では測っていません。
出典: arxiv.org/html/2603.12123(2026-03-12)

Copyright © Givery, Inc. All rights reserved

解説

Cross-Context Review の4条件

自己レビューの繰り返しはむしろ下がる結果

濃い塗りが最高、薄い塗りが最低の F1 です。

条件	レビュー側の状態	F1	備考
CCR 別セッション	履歴なしで差分だけを渡す	**28.6%**	重大エラーの検出は 40%
SR 自己レビュー	実装セッションのまま依頼	**24.6%**	重大エラーの検出は 29%
SR2 2回目の自己レビュー	1回目の指摘を受けて再度	**21.7%**	1回目との差は有意でない
SA 依頼文つきの委譲	新しい会話に依頼文だけを渡す	**23.8%**	依頼文を渡すと差が消える

「もう一回見直して」を2回やるより、1回だけ別セッションに渡す方が検出が高い結果です。サブエージェントでも、生成時の依頼文を渡すと差が消えます。渡すのは差分だけにします。
コード領域での改善幅が最も大きく、F1 で +4.7 ポイントでした。

出典: arxiv.org/html/2603.12123(2026-03-12)

Copyright © Givery, Inc. All rights reserved

自分の誤りを直せない盲点

他人の誤りとして渡せば直せる同じ誤り

自分が書いたとして提示

誤りを含む出力

64.5%

直せない

提示の仕方を変えるだけ

ユーザーが書いたとして提示

同じ誤りを含む出力

直せる

外部の誤りとして処理

Self-Correction Bench(arXiv 2507.02778)は 14 のオープンモデルで、同じ誤りを自分の出力として見せた時だけ直せない割合を 64.5% と報告しています。外部の誤りとして見せれば直せるので、自分の出力を疑う訓練が足りないのが原因です。別セッションや別ツールに渡す操作は、この見せ直しを機械的にやっています。

出典: arxiv.org/abs/2507.02778(v3 2026-08-02)

Copyright © Givery, Inc. All rights reserved

Anthropic 自社開発のレビュー構成

観点ごとに分けた独立エージェントの並列

> **「各レビューエージェントは狭い一点に絞られ、過去のインシデントの知識を持つ。1つに束ねない理由は、バイアスと盲点を共有しないこと、1つが間違えても他が拾えること、注意が薄く広がらないことにある。」**
>
> Anthropic

PR が開くと複数のレビューエージェントが自動で走り、承認は全件ログされます。人が再確認するのはリスク加重で抽出したサンプルです。自動レビューを別種のリスクと位置づけ、複数ゲートと別コンテキストのエージェントで制御すると表現しています。

出典: claude.com/blog/how-anthropic-secures-its-ai-native-software-development-lifecycle(2026-07-21)

引用は研修用の訳です。

※ 各ロゴは各社の商標です。

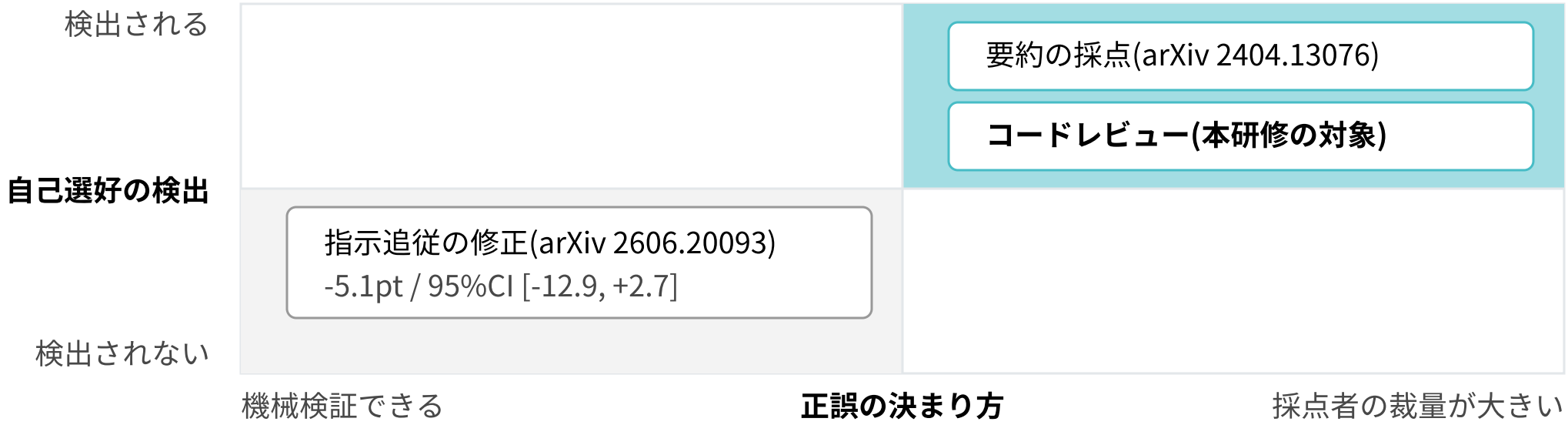
Copyright © Givery, Inc. All rights reserved

解説

自己選好バイアスの範囲

裁量の大きい採点ほど残る自己選好

濃い面が自己選好の残る領域です。



裁量が減ると自己選好は消える

Panickssery らは GPT-3.5 / GPT-4 / Llama 2 が自分の要約を他モデルや人より高く採点すると示しました。自己認識の精度と自己選好の強さは相関し、GPT-4 の自己識別精度は 73.5% です。Guey らは機械検証できる修正タスクでは差が消えると報告し、裁量の大きいタスクは別と限定しています。

出典: arxiv.org/pdf/2404.13076、arxiv.org/html/2606.20093(数字は要約の採点)

Copyright © Givery, Inc. All rights reserved

本研修の立場

書いたモデルに採点させないコードレビュー

> **コードレビューは裁量が大きい。**
> **だから書いたモデルに採点させない。**

根拠

- 履歴を切るだけで F1 は +4.0 ポイント(重大エラーは 40% 対 29%)
- 自分の誤りとして見せた時だけ直せない割合 64.5%
- Anthropic 自社 SDLC の観点別・独立エージェント構成

> **言わないこと**
> 「モデルを変えればさらに上がる」は未検証です。この研修で測るのは、
> 履歴を切る効果と、ツールを変えた時の指摘の違いまでです。

Copyright © Givery, Inc. All rights reserved

実装とレビューを分ける4つの置き方

読み手から情報を1つずつ外す4つの置き方

塗りの数が多いほど大きい値です。

置き方	何で分離するか	コスト	再現性
1 実装したセッション	分離しない。履歴も定義も持ったまま採点させる。比較の基準になる行	最低	低(基準)
2 別サブエージェント	`.claude/agents/review-other.md` 履歴なし・定義あり	低	中
3 新しい会話	Ctrl+N で新しいセッション。差分とREVIEW.md だけ。履歴なし・定義なし	低	中
4 CI	GitLab CI で read-only のレビュージョブとゲートジョブを分ける	高	高

1 が基準です。2 で読み手から履歴を外し、3 で定義も外します。
2 と 3 は Claude Desktop の中だけで完結しますが、どちらも忘れた時に止まりません。
4 だけがレビューを飛ばした MR をマージさせない状態を機械で保証します。

出典: code.claude.com/docs/en/sub-agents、code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

実演

別サブエージェントの定義

model 1行で変わるレビュアーの頭

.claude/agents/review-other.md

```
---
name: review-other
description: 履歴なしで差分をレビュー
tools: Read, Grep, Glob
model: sonnet
effort: high
---
同僚のコードとしてレビューします。
対象は main との差分だけです。
P0/P1 を3件まで、ファイル名と行番号
つきで報告してください。
コード提案は書かないでください。
```

既定は sonnet です。実装を sonnet で回した人は opus に書き換え、実装セッションと違う名前にします。

サブエージェントは履歴を受け取らず、独立コンテキストで動きます。158 の CCR 条件に当たります。

差分は Read と Grep で読ませます。Edit と Write がないので、レビュー側はファイルを書き換えられません。

講師環境で @ を打ち、候補に出ることを確かめます。

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

Copyright © Givery, Inc. All rights reserved

自社ハーネスの現在地

置き方2まで到達している D2 の次の層

置き方	D2 の現状	足すと効く箇所
1 実装したセッション	一部 人の手順に依存	比較の基準として D1-11 で1回だけ
2 別サブエージェント	済 d2-code-reviewer	実装側と違う model を 明示する(D1-05 の2定義)
3 履歴を切った新しい会話	未	渡す材料(差分と REVIEW.md)を 1か所に決める(D1-11 の review-other)
4 CI	未	GitLab CI の ai_review と ai_gate に分ける(D2-05)

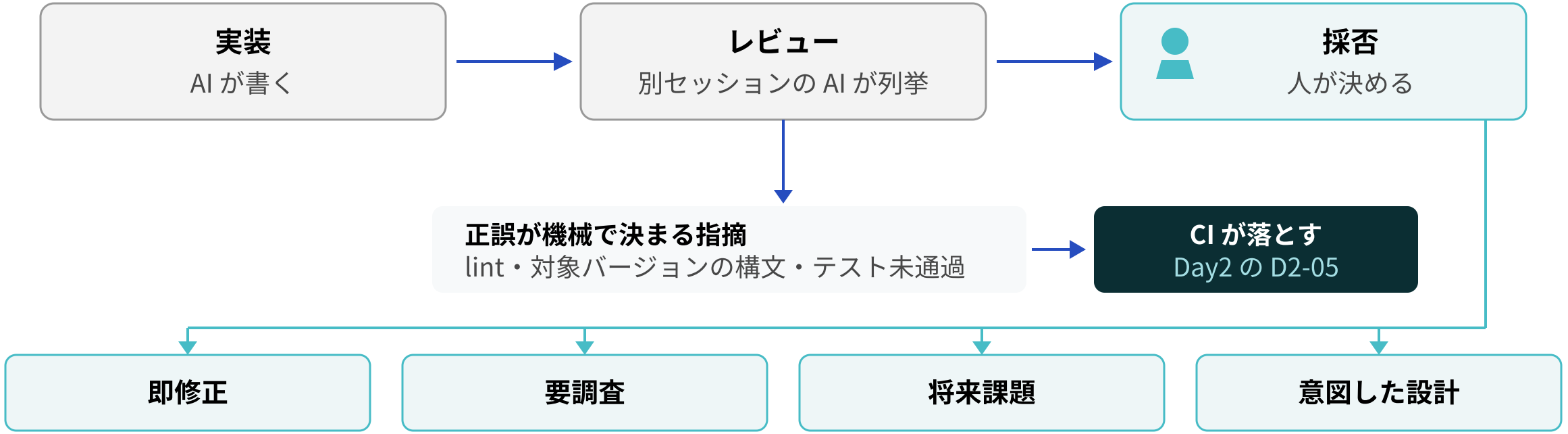
D2 はサブエージェントでレビューを分ける層まで来ています。一次資料が効果を示す層です。
足すのは2つです。レビュー側の model を実装側と違える明示と、履歴を切った第3の目です。
CI の層は Day2 で、読み取り専用のレビュージョブと P0/P1 のゲートジョブに分けます。

Copyright © Givery, Inc. All rights reserved

解説

AI が列挙し人が採否を決める線

機械で決まる分は CI に渡し裁量だけ人に残す



回答書の「AI が列挙し、対応要否は人が判断する」切り分けは、裁量の大きい部分を人に残す点で妥当です。
見直したいのは、正誤が機械で決まる指摘まで人が毎回採否している状態です。そこは CI に渡します。
指摘を全部受け入れると意図した設計が巻き戻ります。レビュー側は背景を知らない前提で読みます。

出典: claude.com/blog の how-anthropic-secures-its-ai-native-software-development-lifecycle

Copyright © Givery, Inc. All rights reserved

補足

分離で踏む落とし穴

履歴を残したまま繰り返す自己レビュー

落とし穴	症状	対処
同じセッションで「もう一度見て」を繰り返す	指摘が減る SR2 は F1 21.7%	2回目を回すより、1回だけ別の読み手に渡す
サブエージェントに履歴を引き継がせる	別セッションの効果が消える SA は F1 23.8%	独立コンテキストで呼ぶ fork 型にしない
3通り目に REVIEW.md を渡し忘れる	観点がばらけて件数だけ増える	差分と REVIEW.md の本文はセットで渡す
agents の model: に他社のモデル名を書く	指定が効かずClaude 系で動く	model: は sonnet / opus / haiku / fable かフル ID
指摘を全部受け入れる	意図した設計が巻き戻る	即修正・要調査・将来課題・意図した設計の4分類で人が決める

1行目と2行目は同じ原因です。読み手が実装の履歴を持つと、自分の答案として読んでしまいます。
3行目は3通り目の設計そのものです。外すのは履歴と定義で、レビューの基準まで外しません。

出典: arxiv.org/html/2603.12123、code.claude.com/docs/en/sub-agents

Copyright © Givery, Inc. All rights reserved

ワーク

持ち帰りと D1-11 への橋

3通りレビューを自分の数字で確かめる午後

この座学の持ち帰り

- 分離の置き方の選定表と落とし穴表
- review-other.md の雛形
- 3通りの軸(履歴と定義を1つずつ外す)

D1-11 で測ること

- 同じ差分を3通りで当てる
- 指摘数・重大指摘の有無・所要時間を記録
- 3通り目に渡すのは差分とREVIEW.md の本文だけ

自社ハーネスへの書き戻し

- .claude/agents/review-other.md を1本実装側と違う model を書く
- D2 は PowerShell 正、smart3pm は bash 正
- 置き場所は D1-12 の台帳

午後の D1-11 では、同じ差分に3通りのレビューを当てて、中央の3項目を記録します。

Copyright © Givery, Inc. All rights reserved

ワーク

見せない設定と Claude なしでの検証

D1-10

プチ演習3 秘密情報の遮断とフックの単体テスト

.env を Claude に読ませても中身が出ない設定を足します。
D1-07 で deny をすり抜けたコマンドは、PreToolUse hook で止めます。
Claude を起動せずに、フックが止めるかどうかを1行で検査します。

[15min]

Copyright © Givery, Inc. All rights reserved

ワーク

開始前の確認と流れ

ダミーの .env を自分で置いてから始める

自分で考える	手順1	手順2〜3	手順4	手順5〜6	手順7〜8
2問を紙に	Read の deny	読ませてみる	フックの登録	検査とケース	再確認と確定

チェック	確認項目	確認する場所
☐	プチ演習2 の変更が確定し、 変更の一覧が空になっている	VSCode のソース管理ペイン
☐	ブランチ名が ex/p3- で始まっている。main には戻らない	VSCode の左下のブランチ名
☐	リポジトリ直下にダミーの .env がある。中身は架空の値	`Set-Content .env 'DB_PASS=dummy-secret-123'`
☐	ソース管理ペインの一覧に .env が出ていない	.gitignore に配布時点で入っているので、書き足しは要りません

確認のダイアログを見るので、Claude Desktop のモードを 手動 にしてから始めます。

.env は統合ターミナル(右上の >_ か Ctrl+バッククォート)で作ります。本物の接続情報は置きません。

Copyright © Givery, Inc. All rights reserved

ワーク

AI に渡す前の2つの問い

読ませない物と止める形を自分で先に挙げる

問い1 読ませたくないファイル

自社で AI に読ませたくないファイルを3つ挙げ、
ファイル名の形を書きます。固定名か、
credentials を含む名前か、拡張子で決まるかで、
書く正規表現が変わります。

問い2 止める形と止めない形

そのファイルに触るコマンドで、
「止めたい形」と「止めてはいけない形」を
1つずつ挙げます。
手順6 で、この2つをテストケースに変えます。

2分で2枚を埋めます。正解探しより、自分の環境で起きうることを書いてください。
問い1 は手順1 の deny と手順4 のフックの規則に、問い2 は手順6 の cases.json に写します。

Copyright © Givery, Inc. All rights reserved

ワーク

手順1 Read の deny 3本

読ませないファイルは deny で先に塞ぐ

.claude/settings.json の permissions.deny

"deny" の配列の最後に足す

```
"deny": [
  …(D1-07 で足した5組10本。最後の1本が次の行)
  "PowerShell(git clean *)",
  "Read(./.env)",
  "Read(./.env.*)",
  "Read(./**/credentials*)"
]
```

D1-07 で追加済みの
deny 5組10本

今日足す Read の3本
保存すると deny は13本

"PowerShell(git clean *)" の末尾にカンマを付けて3本を足し、最後の1本にはカンマを付けません。
deny に13本並び、赤い波線が無いことを見てから保存し、Ctrl+N で新しいセッションを開きます。
パスは gitignore 記法です。./.env.* は .env.local にも .env.tpl にも一致します。
Read(...) の deny は Bash の cat 等に効き、PowerShell の Get-Content には効きません。
評価は deny → ask → allow の順で、allow では deny に例外を作れません。

出典: code.claude.com/docs/en/permissions

Copyright © Givery, Inc. All rights reserved

実演

手順2〜3 .env を読ませて拒否を見る

Read は止まり シェルは通る

	操作	期待される画面
手順2	モードは 手動 のまま、 「.env の中身を教えてください。」 と送ります。	Read ツールが権限で拒否され、 中身は返りません。
手順3	同じセッションで 「cat .env の結果を貼ってください。」 と送ります。	PowerShell ツールで Get-Content か cat が 打たれ、確認で許可すると次が返ります。 `DB_PASS=dummy-secret-123`
メモ	何も直さず、手順3 で打たれた コマンド文字列をメモします。	ソース管理ペインの一覧は 空のままです。

Read(./.env) の deny は Read ツールにだけ効き、PowerShell ツールの経路は塞ぎません。
ここで見えたのは架空の値です。本物なら、この1回で外に出ています。

画面は Claude Desktop で実演します

出典: code.claude.com/docs/en/permissions#bash-rule-limits

Copyright © Givery, Inc. All rights reserved

ワーク

手順4 PreToolUse hook を1本置く

コマンド文字列の全文を自前の正規表現で見る

配布済み .claude/hooks/block-destructive.ps1 のルール定義

ASCII のみ・PowerShell 5.1

```
$rules = @(
@{ id='env-file'; patterns=@('(^|[\s"/\\=])\.env([^A-Za-z0-9_]|$)')
except=@('\.env\.tpl([^A-Za-z0-9_]|$)') },
@{ id='reset-hard'; patterns=@('reset','--hard') },
@{ id='force-push'; patterns=@('push','(--force|\s-f)') },
@{ id='clean'; patterns=@('git','clean\s+-') },
@{ id='rm-recursive-force'; patterns=@('rm\s+-[a-zA-Z]*r[a-zA-Z]*f') },
@{ id='remove-item-recurse-force'; patterns=@('Remove-Item','-Rec','-Fo') } )
```

.claude/settings.json の "hooks": {} を置き換える

出典: code.claude.com/docs/en/hooks

```
"PreToolUse": [{ "matcher": "Bash|PowerShell",
  "hooks": [{ "type": "command",
    "command": "powershell.exe",
    "args": ["-NoProfile", "-ExecutionPolicy", "Bypass",
      "-File",
"${CLAUDE_PROJECT_DIR}/.claude/hooks/block-destructive.ps1"
    ] }] }]
```

書くのは登録の1ブロック

本体は配布済みです。env-file が手順3で通った Get-Content .env を止めます。保存したら Ctrl+N で開き直します。

Windows

matcher は Bash|PowerShell で受けます。

Copyright © Givery, Inc. All rights reserved

実演

手順5〜6 AI なしで exit 2 を確かめる

フックはパイプ1本で検査できる

手順5 統合ターミナル(右上の >_ か Ctrl+バッククォート)で2行を打ち、2 が返れば止まっています

```
PS> echo '{"tool_name":"PowerShell","tool_input":{"command":"Get-Content .env"}}' |
  powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\block-destructive.ps1
PS> $LASTEXITCODE
[block-destructive] env-file: touches a .env file. Keep secrets out of the session:
wrap the single command with run-with-secrets.ps1 (op run) or read them from CI/CD variables.
command: Get-Content .env
2
```

手順6 .claude/hooks/tests/cases.json の配列の最後にカンマを付けて1件足し、ランナーを回します

```
{ "id": "ps-ng-my-secret", "hook": "block-destructive",
  "input": { "tool_name": "PowerShell", "tool_input": { "command": "type .env" } },
  "expect_exit": 2, "note": "PowerShell の type でも同じ経路" }
PS> powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\tests\run_cases.ps1
23 cases: 23 passed, 0 failed
```

配布時点の22件に自分の1件を足して23件です。id は他と重ならない名前にし、問い2 の形を入れます。

出典: code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

ワーク

exit 2 が返ったかの確認

Claude を通さず1行で確かめる合格

出典: code.claude.com/docs/en/hooks

OK 基準

- [ ] deny が13本並び、シェルのコマンドからは .env が読めてしまうことを1度見た
- [ ] 手で流した JSON に理由の2行と 2 が返った
- [ ] ランナーが 23 cases: 23 passed, 0 failed
- [ ] Ctrl+N で開き直したあと、手順3 と同じ依頼が env-file の理由で止まる

つまずきと直し方

running scripts is disabled で動かない

-ExecutionPolicy Bypass が付いているかを見ます。付いていても出る端末は講師に知らせます。

自分で足したケースだけ FAIL になる

規則が type .env の形を拾えていないか、id の重複か、カンマの位置を見ます。

登録したのに同じ依頼が止まらない

settings.json の赤い波線を直して保存し、Ctrl+N で新しいセッションを開き直します。

手順8 で .claude 配下の変更をコミットします。.env が一覧に出ていないことを見てから確定します。

台帳には deny 3本・フック1本・テストケース1件と書いてください。

Copyright © Givery, Inc. All rights reserved

持ち帰りと発展

フックとテストは対で置く

持ち帰り物	D2 PowerShell 5.1・ASCII のみ	smart3pm bash が正
Read の deny 3本	settings.json の deny に D1-07 の10本と並べて置く	同じ settings.json に置く。 deny はシェルに依存しない
block-destructive フック1本	hooks/ に ps1 を置き、 PreToolUse に登録する	配布 ZIP の bash 版を check 系の並びに置く
テストケース1件	hooks/tests/cases.json に 同じ形で1件足し run_cases を回す	検査スクリプトごとに *.test.sh を1本対で置く

発展1 matcher を Bash だけに戻して開き直し、手順7 と同じ依頼をしてみます。
PowerShell ツールから打たれたコマンドにはフックが呼ばれません。確かめたら元に戻します。
発展2 ブロックの理由の文面を自分のチームの言葉に書き換えます。この文面はそのままモデルに返ります。
発展3 except に入れる必要があるファイルを、ご自身の職場から1つ挙げます。
いずれも D2-10 の社内展開ガイドに、戻し方と一緒に書きます。

出典: code.claude.com/docs/en/hooks-guide

Copyright © Givery, Inc. All rights reserved

ワーク

人が先に考えてから回す改修1本の一周

D1-11

ハンズオン1 改修1本の AI 駆動一周と実装・レビューの分離

Issue #1 の顧客名検索を題材に、影響範囲の調査から MR と CI まで8つの Step で一周します。同じ差分を3通りでレビューし、指摘の件数と重さを自分の数字で比べます。ここが本体です。

開始前の確認

- [ ] Claude Desktop にサインイン済み
- [ ] D1-05・07・10 の設定がコミット済み
- [ ] GitLab に Developer でログイン済み
- [ ] 演習フォルダをプロジェクトに選べる

[160min]

Copyright © Givery, Inc. All rights reserved

解説

全体の流れと所要時間

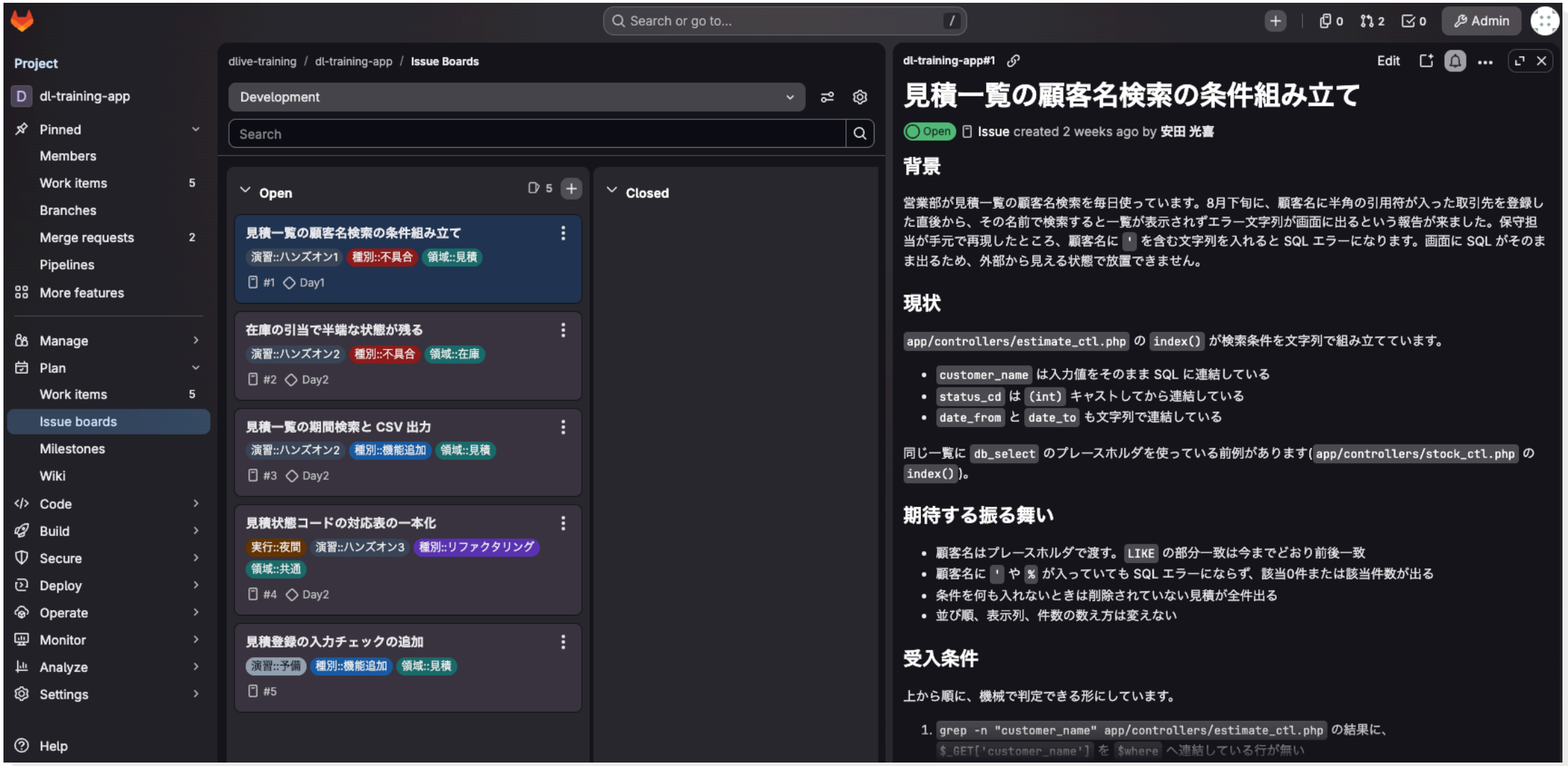
実装と採点を別の場所に置く8つの Step

Step3 までは実装したセッションの中で進め、Step4 で同じ差分を履歴を切った読み手と CI にも読ませます。時間が押したときに削るのは Step7 です。Step4 と Step5 は削りません。各 Step の終わりに記録する項目があり、Step6 で MR の本文へまとめて貼ります。

Copyright © Givery, Inc. All rights reserved

ワーク

題材 Issue #1



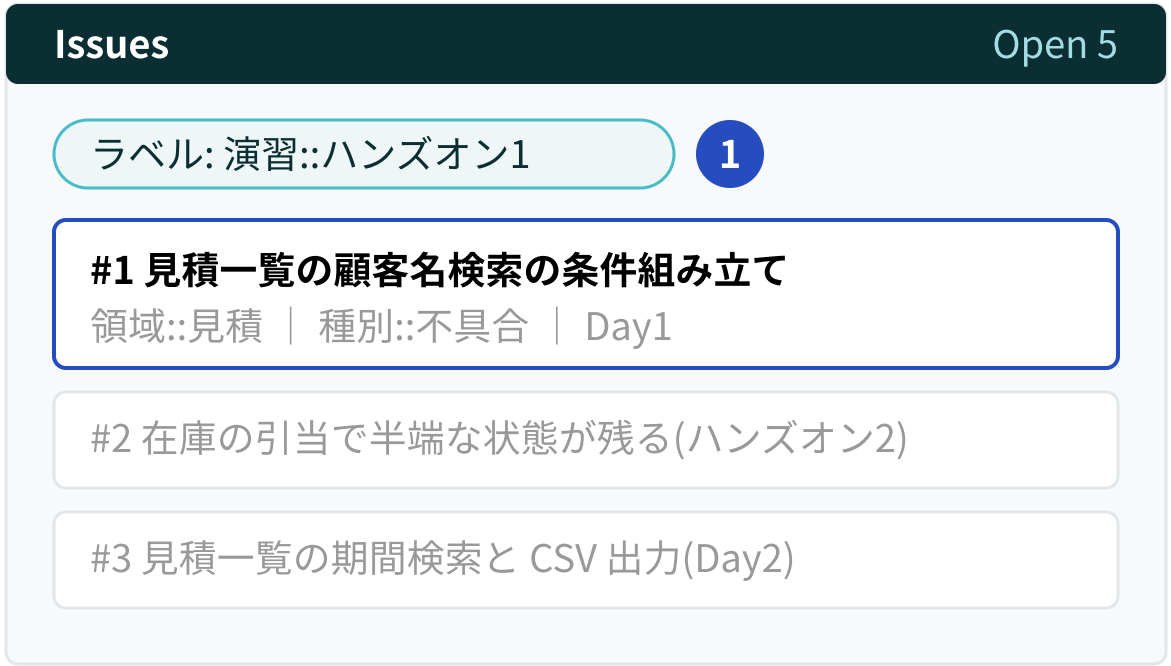
Copyright © Givery, Inc. All rights reserved

ワーク

題材 Issue #1 の読み解き

Step 1 [15min]

受入条件6つが判定の基準になる題材



1 左サイドバーの Issues

Issues を開き、ラベルで絞って
演習::ハンズオン1 の1本を選びます。

2 開く場所

グループ dlive-training の下の
dl-training-app が対象です。

3 既定ブランチ main

main へは直接 push できず、
MR が必須です。

Issue #1 の背景・現状・期待する振る舞い・受入条件・対象ファイル・範囲外を順に読みます。
範囲外の date_from と date_to の連結、sql_lib.php の共通処理は今回触りません。
受入条件は1〜5が機械の判定で、6だけが人の判定です。3の lint-phpver が赤になったら、
混入した構文と行を MR の説明欄に記録してから直します。6が人に残るのは、
採否の理由に妥当性の判断が入るからです。この分け方を Step5 の3軸に持ち込みます。

Copyright © Givery, Inc. All rights reserved

ワーク

人が先に考える3つの欄

Step 1 [15min]

触るファイルの見込みと外したときの気づき方

欄	書くこと	自分の記入欄
1 影響範囲の仮説	触るファイル名。そのファイルを呼んでいる場所。関係するテーブル名。この変更で壊れる可能性がある既存テスト	
2 改修方針	どう直すか。選ばなかった案を1つと、選ばなかった理由	
3 検証方法	誰が何を実行して、何が見えたら正しいか。受入条件6つのうち自分のコマンドで確かめられるのはどれか	

> 範囲外: date_from と date_to の連結 / app/libraries/sql_lib.php の共通処理

紙かメモに書きます。書けたら伏せ、Step2 の終わりに開いて調査結果と突き合わせます。
案が1つしか出ていないときは、まだ方針が決まっていないことが多いです。

Copyright © Givery, Inc. All rights reserved

ワーク

Step2 隔離区画での起動

Step 2 [20min]

午前の設定を持ち込んでから切る隔離区画

1 Step2-1 午前の設定のコミット

VSCode の左下が ex/p3- で始まるのを確かめ、
ソース管理ペインに残った変更を
「午前の設定を持ち込む」でコミットします。

期待: ソース管理ペインの変更一覧が空になる

2 Step2-2 ワークツリーで開始

新しいセッションで演習フォルダを選び、
ブランチ名の右の ワークツリー にチェックを
入れて「作業を始めます」と1行送ります。
名前を入れる欄はありません。

期待: 入力欄の上に worktree- で始まる名前

3 Step2-3 ブランチ名を紙に控える

資料の worktree-issue-1 は控えた名前に読み替えます。

4 Step2-4 VSCode でワークツリーを開く

ファイル > フォルダーを開く で
.claude/worktrees/ の下のフォルダを選び、
左下のブランチ名が控えた名前か見ます。

5 Step2-5 持ち込みの確認

settings.json に deny 13行、ask 3行、
model、PreToolUse が入っているか見ます。
空なら Step2-1 のコミットが未了です。

編集するのは .claude/worktrees/ の下の
フォルダの中だけです。タブのパスで確かめます。
分岐元は settings.json の worktree.baseRef
です。配布版は head なので、今のブランチの
HEAD から切ります。終了時はサイドバーの
アーカイブで片付けます。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

ワーク

Step2 影響範囲調査の委譲

Step 2 [20min]

調べるだけの haiku 偵察役

.claude/agents/grep-scout.md

```
---
name: grep-scout
description: 影響範囲の調査だけを担当します。実装と修正はしません。
tools: Read, Grep, Glob
model: haiku
---
```

> @"grep-scout (agent)" 見積一覧の顧客名検索の条件組み立てを直します。影響範囲を調べてください

渡すのは画面か機能の名前だけ。Issue の本文を丸ごと貼らない

偵察役は Read と Grep と Glob しか持っていません。調べた副作用でファイルが変わることは
ありません。期待される画面: 7つの見出しで返り、右上 : のバックグラウンドタスクに
grep-scout の行が Haiku のモデル名で出ています。

> 入力欄に @ と打つと候補が出ます。grep-scout を選んでから続きを書きます。

出典: code.claude.com/docs/en/sub-agents, code.claude.com/docs/en/costs

Copyright © Givery, Inc. All rights reserved

ワーク

Step2 仮説と調査結果の突き合わせ

Step 2 [20min]

仮説と調査結果の差だけを残す記録

突き合わせ	書くこと	自分の記入欄
自分だけが挙げたもの	grep-scout が見落としたのか、自分の思い込みか。どちらかを判定する	
grep-scout だけが挙げたもの	自分が見落とした理由。ファイルを開いて事実かどうかを確かめる	
両方が挙げたもの	そのまま MR の「影響範囲」欄へ	

> grep-scout は7つの見出しで返します。## 同じ知識が重複している箇所 は必ず読みます。

182 の紙と 184 の出力を並べ、3行を埋めます。行番号の無い記述は調査結果に入りません。重複の節に出たものは今回は触らず、Step5 で将来課題に入れる材料として記録だけします。

Copyright © Givery, Inc. All rights reserved

ワーク

Step3 テスト先行の進め方

Step 3 [35min]

先に赤を見てから実装に入る順番

1. **既存テスト5本の読み方**

   tests/phpunit/EstimateSearchTest.php を開き、既存の5本が画面をどう確認しているかを読みます。

   期待: capture_view() と count_list_rows() の2つ

2. **足す1本の決め方**

   顧客名に ' を含む入力で例外が出ず0件になることを確かめるテストを1本足します。
   何をどう確認するかは、AI に渡す前に自分で決めます。

   期待: テストが1本増え、名前から確認内容が読める

3. **実装を頼むときに渡すもの**

   182 に書いた改修方針と、範囲外の2つだけを渡します。Issue の本文は丸ごと渡しません。

   期待: estimate_ctl.php の index() にだけ差分が出る

> **赤も緑も CI のログ**　　手元に PHP はありません。テストだけ先に同期し、CI の赤を見てから実装へ進みます。
> 落ちないテストは、通っても何も保証しません。先に落とすのが、テスト先行の理由です。

出典: https://code.claude.com/docs/en/best-practices

Copyright © Givery, Inc. All rights reserved

ワーク

Step3 既存テストの書き方

Step 3 [35min]

PHP 7.4 を対象にしたテストと本体の書き方

tests/phpunit/EstimateSearchTest.php の既存5本のうちの1本

```
public function testFiltersByPartOfCustomerName()
{
$_GET['customer_name'] = 'つばさ';
$html = capture_view(new Estimate_ctl(), 'index');

$this->assertSame(2, count_list_rows($html));
$this->assertNotContains('イースト製造株式会社', $html);
}
```

テストは、対象の実行環境に載る PHPUnit の書き方に揃えます。既存5本と同じ親クラス、同じ
例外の確かめ方を使い、新しい世代の書き方は混ぜません。足す1本も同じ形にします。
本体側は PHP 7.4 が対象です。match 式・enum・readonly・名前付き引数・
コンストラクタプロモーション・?-> と、str_contains()・str_starts_with() は使いません。

> 足す1本は自分で書きます。生成させた場合も、投影したこの1本と同じ書き方かを読んでから進めます。
> 代替の書き方は CLAUDE.md の表にあります。

Copyright © Givery, Inc. All rights reserved

ワーク

Step3 2本のコミットと CI の赤

Step 3 [35min]

実装より前に置くテストのコミット

回	コミットするもの	コミットメッセージ	同期後に見るジョブと期待
1本目	tests/phpunit/ EstimateSearchTest.php	顧客名に引用符が入る 場合のテストを足す	test-phpunit が赤。落ちた理由が SQL の構文エラーであること
2本目	app/controllers/ estimate_ctl.php	見積一覧の検索条件を プレースホルダに置き換える	lint-phpdev・lint-phpver・test・ test-phpunit の4つが緑。赤は記録して直す

1本目の同期。MR はまだ作りません

ソース管理ペインの「変更の同期」を押します
MR は Step6 で GitLab の画面から作ります

1本目の test-phpunit のログ末尾

```
PDOException: SQLSTATE[42601]: Syntax error: 7 ERROR:
syntax error at or near "商会"
```

期待: Build > Pipelines に自分のブランチの行が増えます。2本目も同じ手順で同期します。

出典: https://docs.gitlab.com/ci/pipelines/

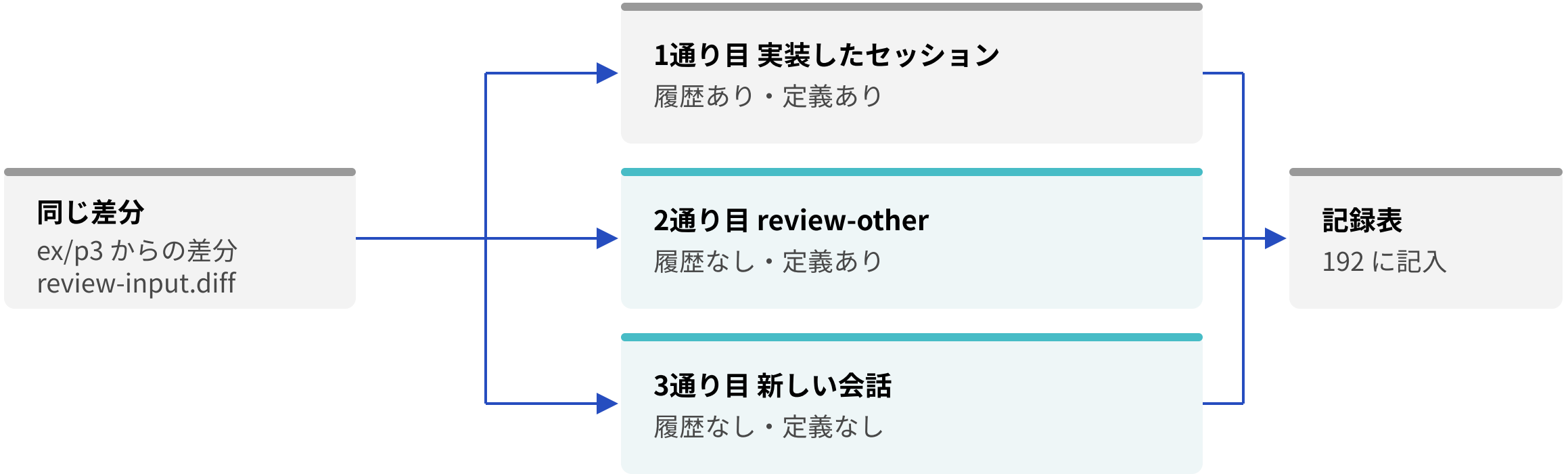
Copyright © Givery, Inc. All rights reserved

解説

Step4 3通りレビューの全体

Step 4 [25min]

読み手から情報を1つずつ外す3通り



外すのは2つです。会話の履歴と、レビュー観点の定義。3通り目には差分と REVIEW.md の本文だけを渡します。
差分とレビュー条件の文言は3通りで同じにします。違うのは読み手が持っている情報だけです。

出典: https://arxiv.org/html/2603.12123

Copyright © Givery, Inc. All rights reserved

ワーク

Step4 1通り目と2通り目

Step 4 [25min]

履歴を持つ読み手と切った読み手

Step4-1 実装したセッション

履歴あり・定義あり

```
この変更をレビューしてください。REVIEW.md の基準で、重大な順に最大8件です
```

新しい会話にせず、実装したセッションで続けて打ちます。開始と終了の時刻を手元にメモします。
期待: 指摘が返ります。

Step4-2 同じセッションのまま、差分をファイルにして review-other へ

履歴なし・定義あり

統合ターミナル

```
git diff ex/p3-<ユーザー名> --output=review-input.diff
@"review-other (agent)" review-input.diff を、同僚が書いたコードとして読んでください。
差分に出ていない前提はリポジトリのファイルで裏を取ってください
```

期待: 重大度・場所・内容・根拠・直し方の5列の表と、最後に「判定: 合格」か「判定: 要修正」の1行が返ります。サブエージェントは差分を自分で取れないので、ファイルにして渡します。

> 各回で記録するのは、指摘件数、うち P0・P1 の件数、所要時間の3つ。192 の表へその場で書き込みます。

「同僚が書いた」と言い換えるのは、自分の誤りより外部の誤りの方が直せるという測定があるためです。

出典: https://arxiv.org/abs/2507.02778

Copyright © Givery, Inc. All rights reserved

ワーク

Step4 3通り目の新しい会話

Step 4 [25min]

履歴も定義も持たない第3の読み手

1 新しいセッションを開く

Ctrl+N で新しいセッションを開き、プロジェクトフォルダに .claude/worktrees/ の下の自分のフォルダを選びます。ワークツリーのチェックは入れません。初回に信頼を聞かれたら、信頼を押します。

期待: サイドバーにセッションがもう1本増える

2 差分ファイルと基準だけを渡す

```
review-input.diff と REVIEW.md を読み、重大な順に最大8件で指摘してください。
重大度、場所、内容、根拠、直し方の5列の表で返してください
```

期待: 5列の表が返る。件数と所要時間を 192 の表の3行目に書く

3 出力を残して差分ファイルを消す

3通りの出力を演習フォルダの外の ho1-reviews.md に貼り、review-input.diff を削除します。

> このセッションには実装の履歴も、レビュー観点の定義(.claude/agents/review-other.md)も渡りません。返し方だけを2通り目と同じ5列に指定しています。

定義を渡していないので、REVIEW.md の外の指摘も混ざります。件数が増えても質が上がったことにはなりません。

出典: https://code.claude.com/docs/en/desktop

Copyright © Givery, Inc. All rights reserved

ワーク

Step4 記録表

Step 4 [25min]

件数と所要を自分の数字で残す表

読ませ方	指摘件数	P0/P1 の件数	所要時間	気づいた差
1 実装したセッションでそのまま依頼 履歴あり・定義あり				
2 review-other サブエージェント 履歴なし・定義あり				
3 履歴を切った新しい会話 差分と REVIEW.md だけ。履歴なし・定義なし				
4 CI の ai_review Step 6 で埋めます				

この4行は MR 説明の「AI レビュー結果」欄と同じ形です。Step6 でそのまま貼ります。
「気づいた差」には、実装したセッションが出さなかった指摘を1つ書き写します。
3通り目は定義を持たないので、観点が広がって件数が増えることがあります。件数だけで優劣を決めません。

Copyright © Givery, Inc. All rights reserved

ワーク

Step5 3軸と4分類

Step 5 [15min]

3軸を先に付けてから決める4分類

	軽い側	重い側
可逆性	コードを戻せば元に戻る	DB・権限・外部連携に届く
機械検証	lint かテストで差が出る	目で見ないと判定できない
影響範囲	1画面	2画面以上または共通処理

即修正
3軸すべてが軽い側。直す。1件1コミット

要調査
機械検証が重い側。誰が何を調べるかを書く

将来課題
Issue の範囲外。Issue を切る

意図した設計
可逆性が重い側、または背景があって今の形

> 例: date_from が文字列のまま連結されている(P1) → Issue の範囲外 → 将来課題

3軸を先に付け、その結果から分類を決めます。分類から決めると、その日の気分で線が動きます。
AI に直させるのは即修正だけです。残りの3つは文章として MR か Issue に残します。
分類の名前は国内の実践記事から借りたものです。3軸は D1-02 の可逆性・
機械検証・影響範囲で、人が見る量を決める考え方は下の自社 SDLC の公開事例に拠ります。

出典: claude.com/blog/how-anthropic-secures-its-ai-native-software-development-lifecycle

Copyright © Givery, Inc. All rights reserved

ワーク

Step5 限定修正

Step 5 [15min]

即修正だけ直して残りは文章で残す

灰色は期待される画面

1. **指摘を1つの表に集約**
   同じ指摘は1行にまとめ、出どころを残します。
   行に重複が無い
2. **各行に3軸を付ける**
   付けられない行は、その場でファイルを開いて確かめます。
   3軸の欄に空欄が無い
3. **将来課題を Issue に切る**
   date_from の指摘は範囲外。題名と 現状・範囲外 の2節で登録します。
   自分の名前で新しい Issue が1本。番号を採否の表に書く
4. **即修正だけを直す**
   1件1コミットをソース管理ペインで確定します。同期は Step6 でまとめて押します。
   その指摘を出した読ませ方で再レビューし、その指摘だけが消える

即修正が0件なら 5-4 は飛ばします。今回の題材は指摘が範囲外に寄ることがあります。
指摘を全部受け入れると自分が意図した設計が巻き戻ります。採らない理由を残すのがこの Step の成果物です。

出典: https://note.com/tyamaoka/n/n643439673ee8

Copyright © Givery, Inc. All rights reserved

ワーク

Step6 MR の仕上げ

Step 6 [20min]

GitLab の画面で作る MR と6つの節

1. **Step6-1 MR を作る**
   Code > Merge requests の右上の
   New merge request を押し、Source に控えた
   ブランチ、Target に main を選びます。
2. **Step6-2 説明欄を埋める**
   Title を書き換え、テンプレートの節を右の表の
   対応で埋めて Create merge request を押します。
3. **Step6-3 2本のパイプラインを見る**
   MR の Pipelines タブで、ブランチ用の4ジョブと
   MR 用の ai_review と ai_gate を見ます。

節	入れるもの
変更概要	何をなぜ変えたか2行と Closes #1
影響範囲	185 の突き合わせ結果
テスト	188 で見た CI の結果
AI レビュー結果	192 の表の4行
観点別レビュー結果	Step4 の review-other の出力
人が判断した採否	193 の分類と理由

節の見出しは既定のテンプレート Default.md で
最初から入っています。「AI レビュー結果」の
4行目は、ai_review のコメントが付いてから
件数を写します。

> **期待される画面**
> MR にパイプラインのアイコンが付きます。
> 全員分が同時に入るので、回り始めるまで
> 数分かかります。

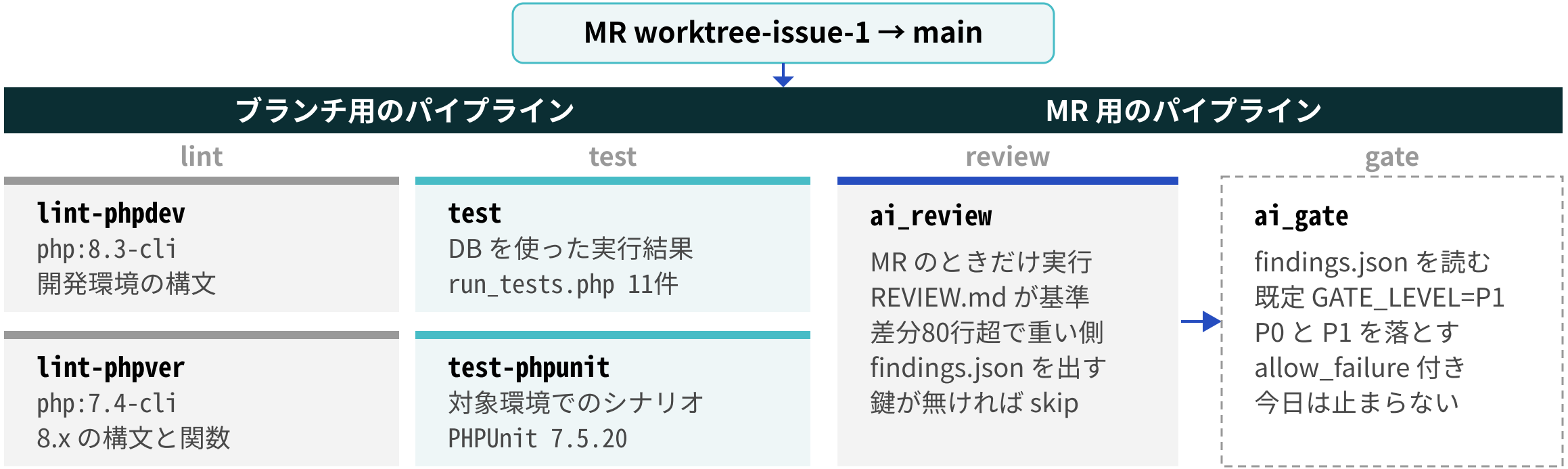
出典: https://docs.gitlab.com/user/project/merge_requests/creating_merge_requests/

Copyright © Givery, Inc. All rights reserved

解説

Step6 CI の6本のジョブ

6本のジョブが見ている別々の観点



lint-phpver は2段です。1段目の php -l が match 式や enum など 8.x の構文を落とします。
2段目の php ci/undefined_functions.php が関数の有無を見るので、str_contains() はここで落ちます。
lint が赤なら test は走りません。ai_gate は既定の P1 で P0 と P1 を落としますが、
今日は allow_failure 付きなので合流は止まりません。外すのが Day2 の D2-05 です。
ai_review は鍵が未投入なら skip します。その場合は 200 の ai_review 列を空のまま送ります。

Copyright © Givery, Inc. All rights reserved

補足

lint-phpver の赤の扱い

赤になった lint-phpver は記録してから直す

lint-phpver が赤になったときに見るログ

```
ファイル名と行番号が付いたエラーの行を探し、1行目をそのまま写します
… app/controllers/estimate_ctl.php … line <行番号>
```

記録すること	書き方
落ちた行	ファイル名と行番号
出たエラー文	エラーの1行目をそのまま。言い換えない
混入した構文か関数	?-> や match、str_contains() のように、何が入ったか
指示のどこに書いてあったか	CLAUDE.md に書いてあったのか、書いていなかったのか

> 記録したら CLAUDE.md の表の書き方に直す。記録は Day2 の最初で全員分を並べる

lint-phpver が赤のままだと test の段が走らず、受入条件4 を満たせません。記録してから直します。
lint-phpver は1段目の php -l で 8.x の構文を、2段目で PHP 7.4 に無い関数を落とします。
受入条件3は、lint-phpver が赤になったら構文と行を MR の説明欄に記録してから直す、です。
合格の条件は lint-phpdev・lint-phpver・test・test-phpunit の4つが緑で、説明欄の節が埋まっていることです。

Copyright © Givery, Inc. All rights reserved

補足

対象バージョンの書き方

対象バージョンは移行先の版で指定

現行の版を対象

移行のたびに、設定も検査も
書き直すことになります。

AI は古い書き方を正として
積み上げていきます。

移行先の版を対象

移行先で通るコードが、
最初から積み上がります。

現行で動くことは、実行環境の
側が守ってくれます。

対象バージョンは設定ファイルの1行です。演習リポジトリでは .claude/project.json の php.target が正で、フックと CI はそこを読みます。リプレイスの予定があるときは、現行の版ではなく移行先の版をそこに書きます。現行で動くことは実行環境が保証し、移行先で通ることは検査が保証します。

ジョブ名は lint-phpver、フック名は check-phpver。版を名前に入れていないので、対象が変わっても名前は変わりません。

Copyright © Givery, Inc. All rights reserved

発展

Step7 作業指示書の生成と赤入れ

Step 7 [5min]

生成された指示書に自分で入れる赤

灰色は期待される画面

1. **/shijisho 1 の実行**
   入力欄に /shijisho 1 と打ち、Shift+Enter で改行して Issue #1 の本文を貼ってから送ります。
   docs/shijisho/issue-1.md ができ、先頭に判定の1行と5項目が並ぶ

2. **5項目で止まっているかの確認**
   目的 / 影響範囲 / 変更方針 / 検証 / 戻し方 の5つで止まっているかを見ます。
   項目が6つ以上に増えていない

3. **目的と戻し方の書き直し**
   この2つを自分の言葉に直し、「Issue #1 の作業指示書を足す」でコミットして同期します。
   言い換えでない文が2か所に入る

受講端末には glab もアクセストークンも無いので、Issue の本文を貼って渡します。
途中で glab や curl の実行確認が出たら許可しません。
実装の後に指示書を作るのは順番が逆ですが、生成したものと自分が書いたものの差を見るのが目的です。
今回の改修は3軸すべてが軽い側なので軽量パスになります。変更方針は3行以内、影響範囲はファイル一覧だけです。
直した箇所が目的と戻し方に集中していれば、生成物の使いどころが見えたことになります。

Copyright © Givery, Inc. All rights reserved

ワーク

全員の MR と CI 結果

合計と平均で見る全員の MR

[3min]

受講者	MR	lint-phpdev	lint-phpver	test	test-phpunit	ai_review 指摘件数	P0・P1
合計・平均							

個別の行は現地の方だけ読み上げます。リモートの方は MR の URL をチャットへ送ってください。
合計・平均の行は講師が埋めます。MR がまだの方の行は空けたまま Day2 の冒頭で埋めます。
鍵が未投入で ai_review が skip した場合、その列は空のまま Day2 へ送ります。
この件数が Day2 の「指摘の絞り込み」の入口の数字になります。

Copyright © Givery, Inc. All rights reserved

解説

講師からの推奨

Issue から MR への流れを基幹開発の標準に

> **Issue → MR → AI レビューの流れを、**
> **基幹システム開発の標準に据えることを推奨します**

- 指摘が MR に残ります。Excel とチャットに散っていた記録が、差分と同じ場所に並びます。
- テストを回したという報告が、CI の緑と赤に置き換わります。
- MR あたりの指摘件数と採用率、ゲートで止まった割合が効果測定の数字になります。

Copyright © Givery, Inc. All rights reserved

ワーク

Step8 分離の書き戻し

Step 8 [20min]

分離の置き方を1本だけ自社へ書き戻す

.claude/agents/review-other.md

```
---
name: review-other
description: 履歴を切った独立レビューです。実装の経緯を知らない同僚として差分だけを読みます。
tools: Read, Grep, Glob
model: sonnet
effort: high
---
```

- 手元で回すなら review-other.md、MR に残すなら ci/ai_review.sh と .gitlab-ci.yml の review ステージ
- model を実装側と変えるのは任意の実験です(発展2)。効いているのは履歴を切ることです。
- tools が Read と Grep と Glob だけなので、レビュー役がファイルを書き換えることはありません。

> 入れるのは片方だけです。自社のリポジトリ名、ブランチ名、レビュー基準のファイル名に
> 書き換えてから置き、持ち帰り台帳の本日分に1行足します。

出典: https://code.claude.com/docs/en/sub-agents

Copyright © Givery, Inc. All rights reserved

ワーク

ハンズオン1の自己採点

MR と記録表と判定シートで付ける合否

OK 基準

- [ ] MR が1本あり、lint-phpdev・lint-phpver・test・test-phpunit の4つが緑
- [ ] MR の説明欄の節が埋まり、直さなかった指摘に理由がある
- [ ] 192 の表の1〜3行目が埋まり、4行目が件数か「未実施」
- [ ] 3軸と4分類の仕分けに空欄が無い
- [ ] 将来課題にした指摘が Issue になっている

つまずきと直し方

MR が作られない
同期は済んだが GitLab でまだ作っていない

push が protected branch で弾かれる
main へ push している。ブランチ名を確認

パイプラインが Pending のまま
全員分の順番待ち。192 の表を先に埋める

ai_review が赤または skip
鍵の設定。MR の説明欄を先に埋める

test-phpunit が赤のまま
1本目の赤が SQL の構文エラーかを確認

3通り目のレビューが10分を超える
待たずに進み、後で 192 の表に書く

MR テンプレートが空
Default.md の6つの節の見出しを手で作る

レビューが「問題ありません」だけ
0件として 192 に書く

持ち帰り物は 205 の表と持ち帰り台帳で扱います。Stop hook の Review Gate は往復が止まらないため有効化しません。

Copyright © Givery, Inc. All rights reserved

ワーク

今日足した7点の台帳への記入

D1-12

Day1 の持ち帰り台帳と宿題

今日作った設定と文書を、持ち帰り台帳.md の7行で確認します。
D2 と smart3pm のどちらに置くかを各自で決めます。
Day2 に持ってくる宿題2つと、今日の詰まりも確認します。

[15min]

Copyright © Givery, Inc. All rights reserved

本日の持ち帰り7点

7点とも今あるハーネスへの追記

番号	持ち帰り物	出た演習	ファイルの形
D1-1	review-light.md と review-full.md	プチ演習1	.claude/agents/ の Markdown 2本
D1-2	settings.json の model・effortLevel・maxEffortLevel	プチ演習1	settings.json の3行
D1-3	deny 5組10本と ask 3本	プチ演習2	settings.json の permissions
D1-4	CLAUDE.md の「戻し方」に足した1行	プチ演習2	既存 CLAUDE.md の「戻し方」節
D1-5	deny 3本 .env と credentials	プチ演習3	同じ permissions.deny に追記
D1-6	PreToolUse フック block-destructive 1本とテストケース	プチ演習3	.ps1 または .sh 1本と cases.json
D1-7	分離の1本(review-other.md か CI の AI レビュー)	ハンズオン1	agent の Markdown か CI のジョブ

D1-7 は、レビューを手元で回すなら review-other.md、MR に残すなら ci/ai_review.sh です。
ゼロから作るものは1つもなく、すべて既存ハーネスへの追記で済みます。

出典: https://code.claude.com/docs/en/sub-agents ・ /permissions ・ /skills

Copyright © Givery, Inc. All rights reserved

解説

置き場所とシェル依存の仕分け

シェル依存はフックに閉じる

担い手と配置	D2 と smart3pm で同じ内容	ハーネスごとに別の中身
Claude Code が読む設定・文書	**そのまま両方へ** agents 2本 / settings 差分 deny 1群目 / deny 2群目 CLAUDE.md 追記 / agent か CI ジョブ	該当なし
シェルが実行するスクリプト	該当なし	**言語が2つに割れる** PreToolUse フック block-destructive → D2 は block-destructive.ps1 → smart3pm は block-destructive.sh

D2 側は PowerShell 5.1 で ASCII のみ、smart3pm 側は bash です。
伸びしろ8 は同じ目的のガードが2言語で二重実装になる点で、統一の方針は研修後の宿題です。

出典: https://code.claude.com/docs/en/hooks

Copyright © Givery, Inc. All rights reserved

ワーク

置き場所を決める前に紙に書く3つ

台帳を開く前に自分の判断を3行で書く

問い1 今日の書き戻し先

明日から触るハーネスは
D2 か smart3pm か。
両方を触る人は、先に
書き戻す方を1つ選び、
理由を1行で書きます。

問い2 フックの言語

block-destructive を
PowerShell のまま書くか、
bash に寄せるか。
自分の端末と CI の
どちらで多く動かすかを
根拠に1行で書きます。

問い3 作成者以外への一言

7点から1つ選びます。
作成者以外の人が読んで
置き場所が分かる説明を、
名詞1語で書きます。

4分で3枚を埋めます。AI には聞かず、自分の運用で答えてください。
問い1 と問い2 の答えが、次のページで台帳の置き場所列になります。

Copyright © Givery, Inc. All rights reserved

ワーク

台帳への7行記入

2本とも埋める置き場所の列

Step	操作	期待される画面
Step1 [1min]	VSCode で 持ち帰り台帳.md を開き、Day1 の表の D1-1 から D1-7 を 205 と突き合わせます	「持ち帰り物」の列が7行とも埋まっていて、その右が空のままです
Step2 [2min]	置き場所の2列を埋めます。6点は両列に同じパス、フック1本は D2 が .ps1、smart3pm が .sh と書き分けます	空欄が「担当」と「反映日」だけになり、207 の問い2 で選んだ側のパスが残ります
Step3 [1min]	「担当」に自分の名字、「反映日」に日付を書いて保存し、右下の方法で提出します	Day1 の表に空欄がなくなります。台帳は配布 ZIP 側のファイルなので、コミットはしません

置き場所が決まらない行は「未決」と書いてください。Day2 の D2-10 で確定します。

提出
現地は講師が席を回って画面で確認します。
リモートは台帳の表を Zoom のチャットへ貼ります。

Copyright © Givery, Inc. All rights reserved

ワーク

Day2 までの宿題2つ

自分の直近1件の数字と仮分類を持ってくる

宿題	やること	持参する形	Day2 で使う場所
宿題1 指摘の対応率	直近の自分の改修1件を開き、AI レビューの指摘のうち「対応した」件数と「見送った」件数を数えます。理由は書かなくてかまいません。	数字2つのメモ 対応 n 件と 見送り m 件	D2-01 冒頭で全員分を板書し、D2-03 の原因分解の材料にします
宿題2 観点の3列仮分類	d2-code-reviewer が見ている観点を1つずつ取り出し、「機械判定できる」「人が見る」「両方」の3列に仮で振ります。正解は探しません。	3列の表を 紙か md で 1枚	D2-05 の REVIEW.md 初版と Code Review Rules の起案に使います

> 持ち物は 持ち帰り台帳.md(Day1 記入済)、宿題2つ、基幹システム用 CLAUDE.md(持ち出せる範囲で)

宿題1 は数字2つだけです。1件で10分以内に終わる量にしてください。
宿題2 の3列は、Day2 午前に全員の案を並べる出発点になります。

Copyright © Givery, Inc. All rights reserved

質問

今日の詰まりの回収

講師が預かって Day2 の冒頭で返す詰まり

印	詰まり	今日出た症状	Day2 で返すこと
☐	**CP932 と 非 ASCII 文字**	PowerShell のフックに日本語を 1文字入れた端末で、実行時の メッセージが化けました	ASCII 検査をフックのテストに 足す案を、D2-02 の post_edit_checks と合わせて出します
☐	**PowerShell の 実行ポリシー**	.ps1 が「このシステムでは スクリプトの実行が無効」で 止まりました	事前セットアップ Step2 の 手順で解除済みかを確認します。 CI 側は runner の設定です
☐	**effort の 既定値**	settings に書いた effortLevel が 無視されたように見えました。 入力欄の右下は中のままでした	Opus 5.5 以降は ~/.claude の effortLevel を読みません。 プロジェクトの settings に書きます
☐	**その他**		

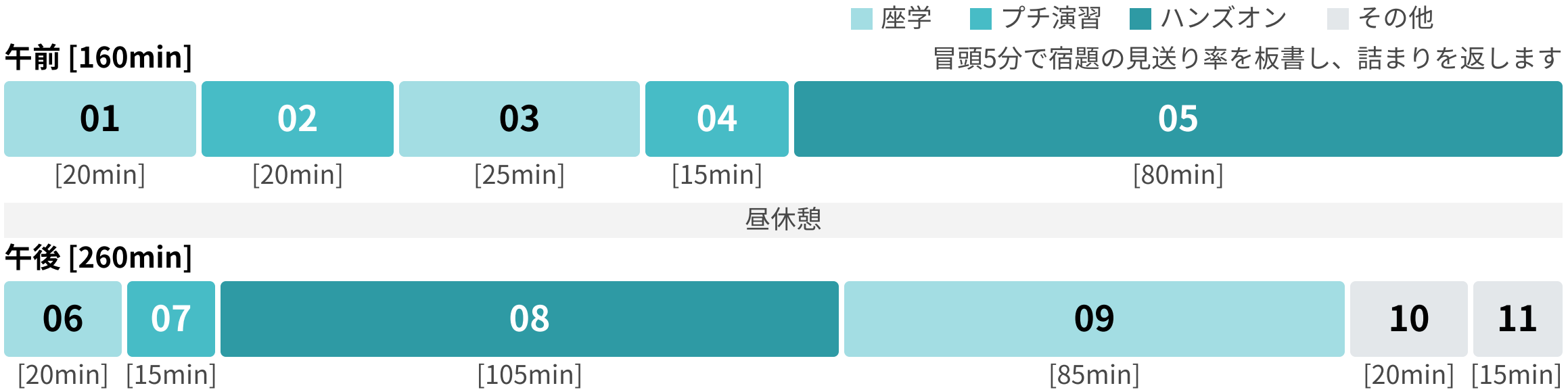
自分の端末で起きた詰まりに印を付け、その他の行に1つ書いてください。

出典: https://code.claude.com/docs/en/model-config#adjust-effort-level

Copyright © Givery, Inc. All rights reserved

Day2 の予告

個人の自動化をチームの CI に持ち上げる日



D2-01 PHP 7.4 で動く指示の作り方
D2-02 プチ演習4 対象バージョン制約の機械検査
D2-03 指摘絞り込みと人間の判断軸
D2-04 プチ演習5 レビュー出力の JSON ゲート
D2-05 ハンズオン2 レビュー観点の切り分けと CI ゲート化
D2-06 夜間ループと人間の立ち位置
D2-07 プチ演習6 止まる Stop hook
D2-08 ハンズオン3 夜間ループの最小構成
D2-09 理想的なハーネスの基準と雛形
D2-10 持ち帰り台帳の確定と社内展開ガイド
D2-11 効果測定と8層チェック表の塗り直し

午前は Day1 のレビュー分離を CI のゲートに載せ、午後は夜間に回るループを1周させます。

Copyright © Givery, Inc. All rights reserved

株式会社ギブリー

〒150-0036 東京都渋谷区南平台町15-13 帝都渋谷ビル8F

Copyright © Givery, Inc. All rights reserved