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

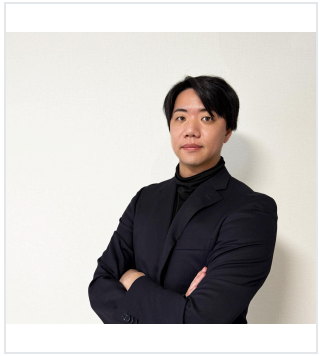
本資料の二次転載禁止について

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。

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

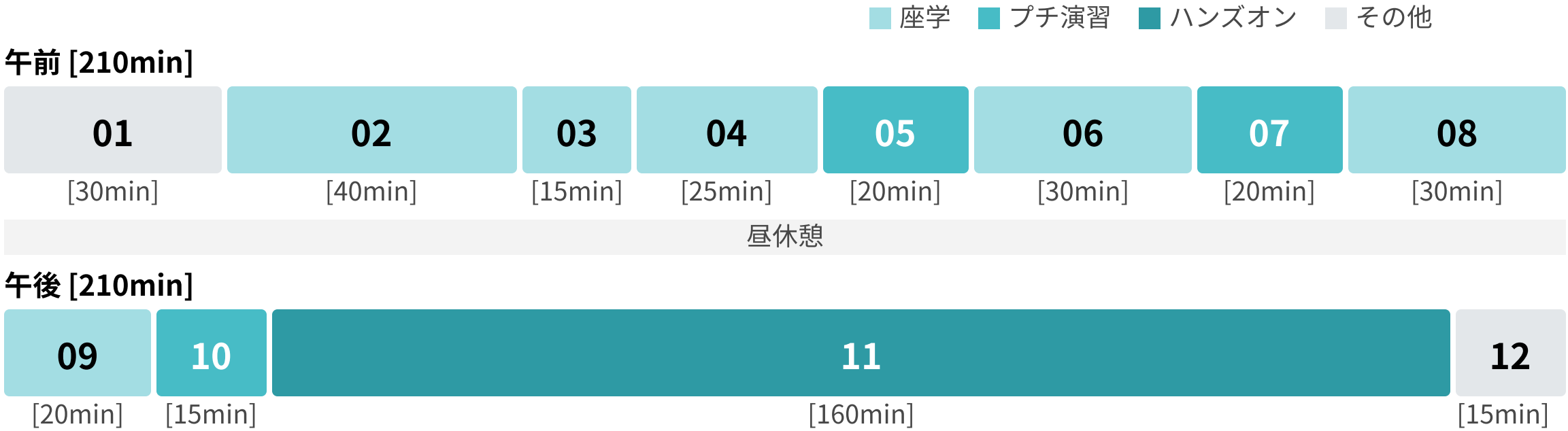
- 生成AI研修
- 大手企業、外資、教育機関(小中高大）でのITトレンド講演
- 自治体と連携した地域教育実績
- 上場企業社員対象のリスキリング支援

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

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



D1-01 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

ブロック別の所要と持ち帰り物

12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
D1-01	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	[40min]	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	[30min]	戻せるもの・戻せないものの表
D1-07	プチ演習2 巻き戻し制約の機械強制	演習	[20min]	deny 4種5本、CLAUDE.md 追記
D1-08	権限・フック・サンドボックス	座学	[30min]	.env.tpl、start-claude.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]	deny ブロック、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、3通りレビュー表、SKILL.md か agent 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

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。

進め方と約束

現地7名の完走を基準にした進行

項目	現地 7名	リモート 11名
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	その場で講師が画面を見て確認します	詰まったら先へ進み、Day2 冒頭で回収します
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 7名

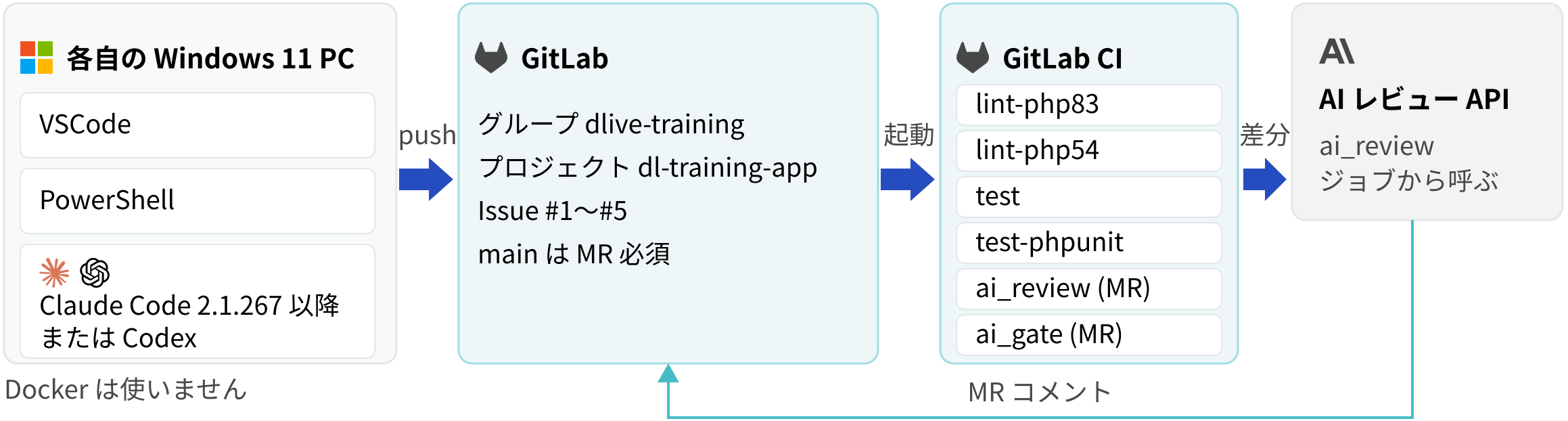
リモート 11名

受講者 18名

配布物と手順は同じです

演習環境の全体図

実行とテストは GitLab CI 側



Docker は使いません

- 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
- PHP のレガシー書法と PostgreSQL で構成しています
- 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

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・D2-10	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 ／ start-claude.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
Codex並行手順.md	全演習	md	Codex のみの受講者が参照

Day2 の配布物(PHP54_非対応構文一覧.md、REVIEW.md、gate.ps1 ほか)は Day2 の前付で案内します。

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。

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

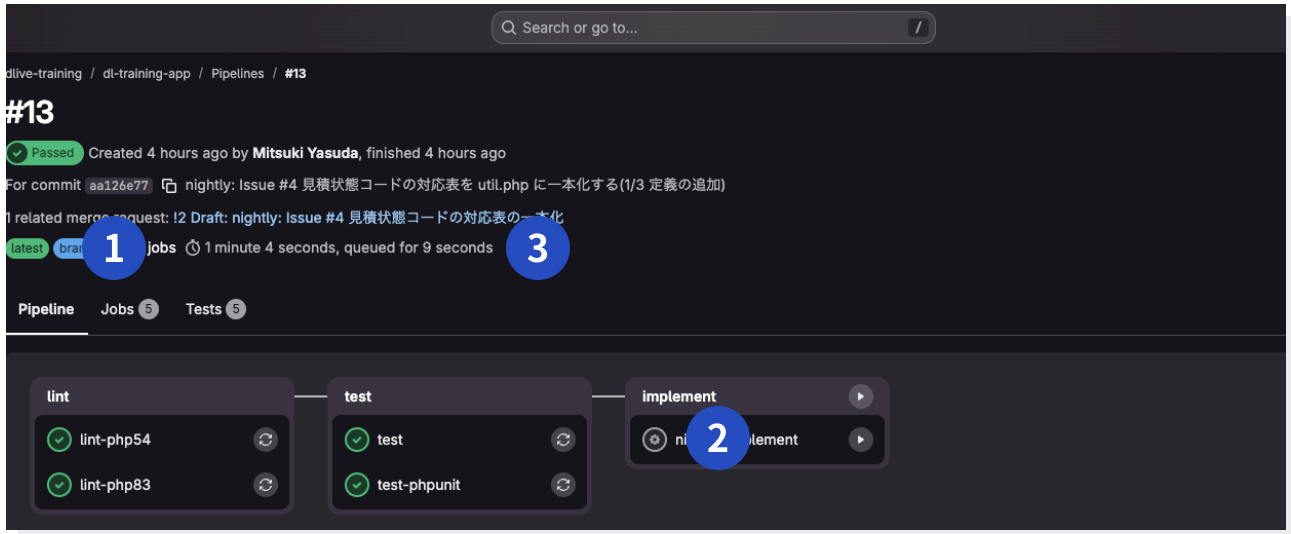
前夜に講師環境で回した夜間パイプラインの実録を見ます。
8層チェック表に、自社ハーネスの今の位置を印で付けます。
良い点3つと伸びしろ8点を、2日で足す機能一覧に変えます。

[30min]

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。

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`
   Codex は codex exec -s workspace-write
4. lint 8.3 / 5.4 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

実演

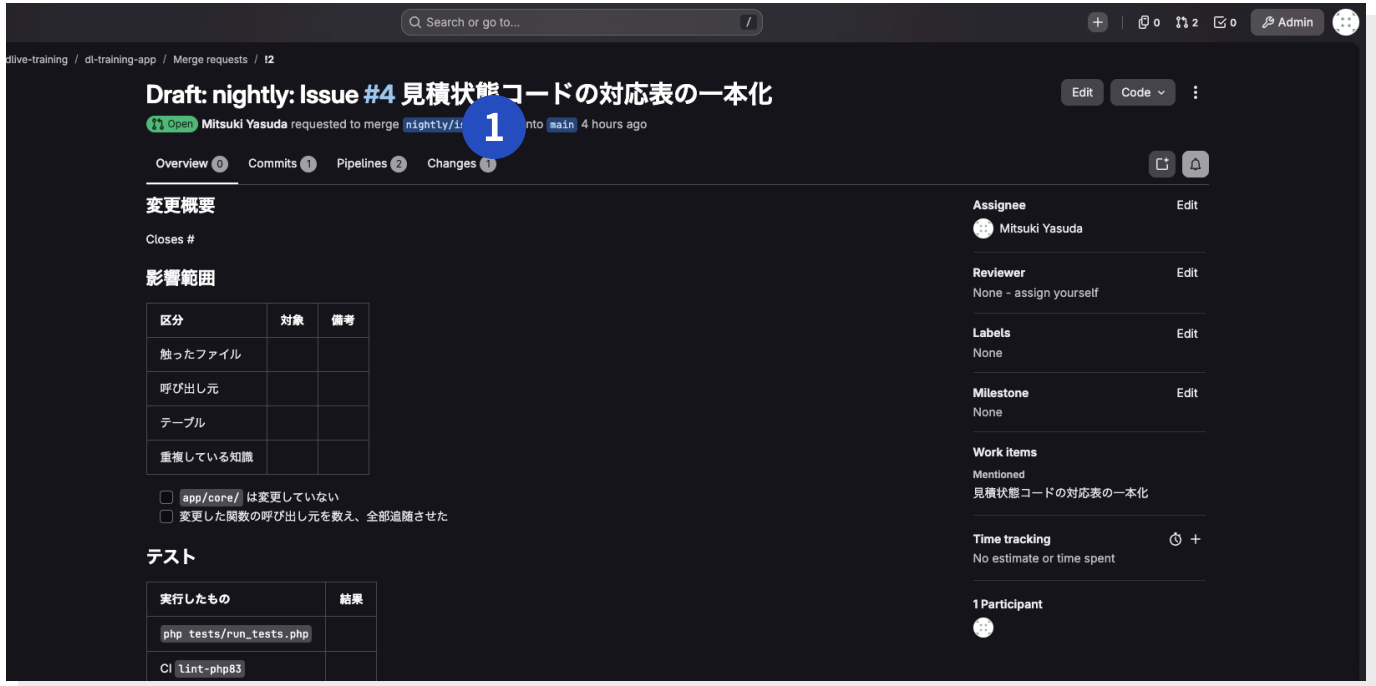
AI レビューコメント付きの MR

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。

Day2 最終ブロックでの再現

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。

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

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 D2-10 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。

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

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本(巻き戻せない操作) | | | | |
…(Day1 は D1-7 まで7行、Day2 の表も7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。

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。

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。

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			D2-09
6 フックの文字コード制約を機械で守っていない	D1-08				D2-09
7 効果測定が投入量ベースになりやすい			D2-10		
8 PowerShell と bash の二重運用の解消時期		D1-12			D2-09

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 D2-09 で渡します。

2日で自社ハーネスに足す件数

2日で足す14点はすべて既存ハーネスへの追記

14

Day1 7点 + Day2 7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7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、夜間ジョブ一式です。台帳の Day2 の表に同じ7行が入っています。

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 Code と Codex の両方の手順

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。

補足

18名での CI の回し方

順番待ちを前提にした CI の運営

18名が 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	18本が順に入る	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	3名ずつ流す	自分の番が来るまで指示書を読み直す

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。

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。D2-10 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。

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。

[40min]

補足

今日の基準となる版

この版で動くものだけを話す

2.1.267

Claude Code、2026-09-09 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、Sonnet 5、Haiku 4.5 を前提にします。
受講端末の claude --version も事前セットアップで同じ版に揃えています。

出典: code.claude.com/docs/en/changelog

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 2.1.267(2026-09-09)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

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 Manual 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

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 *)",
      "Bash(git push --force *)",
      "Read(./.env)"
    ]
  },
  "hooks": {
    "PreToolUse": [{ "matcher": "Bash",
      "hooks": [{ "type": "command",
        "command": "powershell -NoProfile
   -File .claude/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 でも走る強制点

PreToolUse は bypass モードでも先に走る、唯一の強制点です。

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。

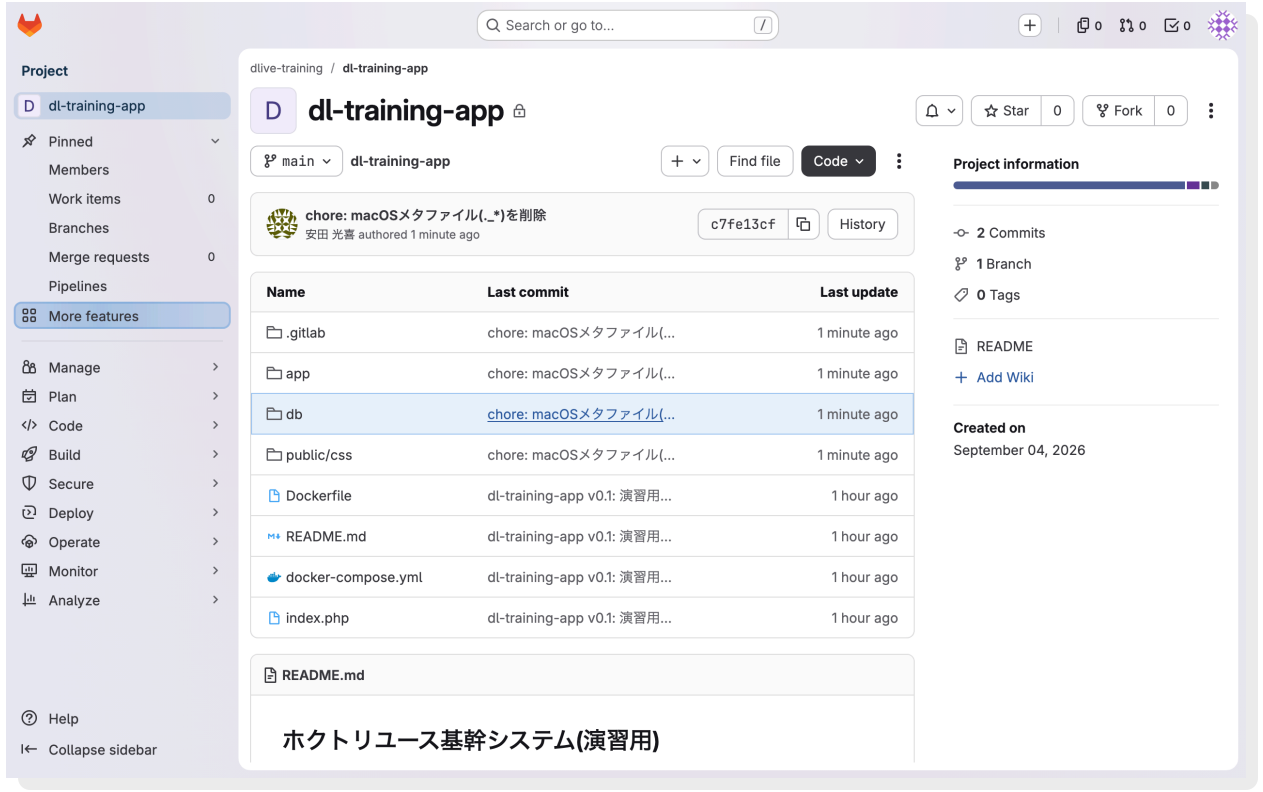
CLAUDE.md に書いた禁止は助言です。強制になるのはこの2ブロックです。

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

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。

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行未満の変更向け。指示書一致と絶対制約だけ見る
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)と対で作ります。

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。

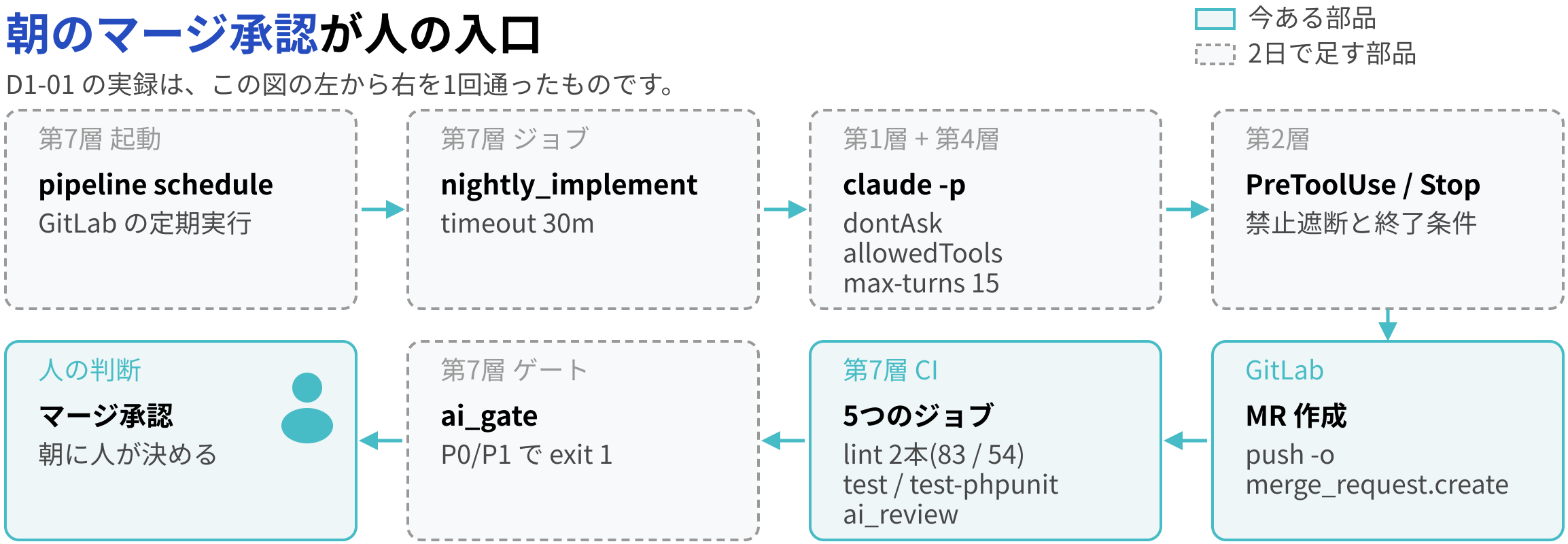
出典: code.claude.com/docs/en/sub-agents

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

実演

別ツールを呼ぶ SKILL.md

人だけが起動できるスキルにレビューを置く

.claude/skills/codex-review/SKILL.md(講師環境)

```
---
name: codex-review
description: 実装後に Codex で独立レビューし、
  P0/P1 だけ限定修正する
disable-model-invocation: true
---
1. codex exec -s read-only \
     --output-schema .claude/review.schema.json \
     -o findings.json "$(cat REVIEW.md) 対象: 差分"
2. findings.json を読み、severity が P0/P1 のものだけ
   「即修正 / 要調査 / 将来課題 / 意図した設計」に仕分ける
3. 「即修正」だけ直す。P2 以下は MR コメントに残す
```

モデル呼び出しの禁止

disable-model-invocation: true で Claude は自走しません。

書き込まない Codex

-s read-only なので Codex はファイルを書き換えません。

採否の4分類

即修正 / 要調査 / 将来課題 / 意図した設計の仕分けは、人が決めます。

Codex 未保有の場合は、次の1行に置き換えます。

```
claude --bare -p --json-schema review.schema.json
```

出典: developers.openai.com/codex/cli/reference、code.claude.com/docs/en の skills・headless

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。

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)

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)

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。

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
- 2,400件超のレコードを削除

> **要確認**
> 削除件数は報道ベースです。公式ブログは
> 分離策の発表のみです。

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/introducing-a-safer-way-to-vibe-code-with-replit-databases(2025-07-21)

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

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」で reset --hard	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

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。

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 5.4 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。

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。

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

自動化の8層

- 第8層 SDK
- **第7層 起動**
- 第6層 並列化
- 第5層 長時間化
- **第4層 承認の代替**
- 第3層 仕事の分割
- **第2層 介入点**
- 第1層 非対話実行

人が残る3点

ゲートの定義

第2層 hooks の exit 2 の条件を決めます。
第7層 ai_gate が P0/P1 で落とす閾値も同じです。

例外の採否

指摘を見送る判断は MR コメントに残します。
4分類は即修正・要調査・将来課題・意図した設計です。

本番接続の鍵

第4層の deny Read(./.env) と CI の masked variable です。
sandbox はネイティブ Windows では使えません。

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

解説

無人実行の止め金 Claude Code と Codex

ツールが違っても止め金は同じ4つ

Claude Code

- --permission-mode dontAsk で allow 外を拒否
- --allowedTools で許すツールを列挙
- --max-turns でターンの上限
- deny と PreToolUse hook で遮断
- --bare はレビュー側だけに使う

Codex

- sandbox_mode は workspace-write
- .git と .codex は読み取り専用
- approval_policy never で承認を挟まない
- network_access は既定オフ
- hooks.json の Stop で decision: block

> 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、learn.chatgpt.com/docs/hooks。各ロゴは各社の商標です。

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_IID" `
  --permission-mode dontAsk `
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)" `
  --max-turns 15 `
  --output-format json
```

ci/nightly-codex.ps1

```
codex exec -s workspace-write -C . `
  "AGENTS.md に従い Issue $env:ISSUE_IID を実装する"
```

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 ・ developers.openai.com/codex/cli/reference

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。

解説

8層図に重ねた D2 の現在地

埋まっている層と夜間に要る層

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。　埋まっている　伸びしろ　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 は未導入。GitLab は10月以降
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 は整備済み。規模別の分割が伸びしろ
第2層 介入点	後処理のフックは整備済み。遮断と終了条件が伸びしろ
第1層 非対話実行	-p は未着手。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。

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	D2 の現在地	足す演習	置き場所 D2 / smart3pm
第4層 止め金	伸びしろ	D1-07 deny 4種5本 D1-10 Read(./.env)	.claude/settings.json D2 と smart3pm で共通
第2層 PreToolUse	伸びしろ	D1-10 block-destructive	hooks/*.ps1 と hooks/*.sh
第3層 分割	整備済み	D1-05 review-light と review-full D1-11 codex-review	.claude/agents と .claude/skills
第1層 -p	未着手	D2-04 --bare -p --json-schema	ci/ のスクリプト
第2層 Stop	伸びしろ	D2-07 stop-gate	hooks/stop-gate.ps1 と .sh
第7層 CI・schedule	未着手	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。

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。

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 4種5本	block-destructive	codex-review

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-php54	--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。

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。

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。
Claude Code から Codex 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]

解説

Agent Plugins 1.0 の構図

規格の外にいる Anthropic と中にいる2社

規格を策定する側

エージェント用プラグインの共通仕様として
2026-08-06 に「Agent Plugins 1.0」の名で
公開されています。策定は AWS と OpenAI 側です。

狙いは、スキルやフック等の配布単位を
ツール間で共有できる形にすることです。

不参加

Anthropic は参加していません。
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。

※ 各ロゴは各社の商標です。

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・AGENTS.md・SKILL.md の本文	**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、Claude Code が文字列照合、Codex が OS のサンドボックスで、考え方から違います。

出典: https://code.claude.com/docs/en/hooks
https://learn.chatgpt.com/docs/agent-approvals-security

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
  skills/
    codex-review/SKILL.md
  agents/
    review-light.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 skills・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。
スキル名は /dl-review:codex-review のように名前空間付きになります。

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

Claude Code と Codex 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	Codex	移すときの作業
常時読む指示	CLAUDE.md	AGENTS.md 変更ファイルに最寄りが適用	本文をコピー。名前を変更
手順の部品	.claude/skills/*/ SKILL.md	置き場所は要確認 (.agents が保護対象)	本文は流用。frontmatter は 捨てる
レビューの分離	.claude/agents/*.md	直接の対応なし。 review_model で別モデル	観点の文章を AGENTS.md の Code Review Rules へ
フック	settings.json の hooks hooks/hooks.json	~/.codex/hooks.json .codex/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)	sandbox_mode と approval_policy	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	model・review_model・ model_reasoning_effort	名前を差し替える。段階の語は 両方とも low〜xhigh
MCP	.mcp.json	[mcp_servers.<name>]	URL とコマンドを写す。 形式は TOML へ

「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 、 https://learn.chatgpt.com/docs/config-file/config-reference

実演

Codex の設定ファイルの構造

モデルと権限と信頼が1ファイルに同居

~/.codex/config.toml

モデル
model ・ review_model ・ model_reasoning_effort

権限
sandbox_mode ・ approval_policy

[sandbox_workspace_write]
network_access

[projects."<path>"]
trust_level

出典: https://learn.chatgpt.com/docs/config-file/config-reference
https://learn.chatgpt.com/docs/agent-approvals-security

Claude Code 側の相当箇所

3つのモデル設定は settings.json の model・effortLevel と agents の frontmatter に当たります。

sandbox_mode と approval_policy が permissions と sandbox を兼ねます。
文字列の deny に当たる項目はありません。

プロジェクト側の .codex/config.toml は projects.<path>.trust_level が trusted のときだけ読まれます。

解説

フックの出力 JSON の形

1行違いの JSON で同じスクリプトが両方に効く

Claude Code の Stop フック出力

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

Codex の Stop フック出力

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

decision と reason の意味は同じで、Codex は継続プロンプトとして reason を使います。
判定ロジックを書いたスクリプト本体は共通にし、出力の包み方だけ引数で切り替えます。

出典: https://code.claude.com/docs/en/hooks 、 https://learn.chatgpt.com/docs/hooks

※ 各ロゴは各社の商標です。

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

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

Codex 側の着地

AGENTS.md に本文を貼る
スクリプトは同じパス

形だけ書き直す

Claude Code 側のファイル

settings.json の hooks 登録
.mcp.json
agents の model・effort
plugin.json

Codex 側の着地

hooks.json
config.toml の [mcp_servers]
model・review_model・
model_reasoning_effort

置いていく

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

Codex 側の着地

sandbox_mode と
approval_policy で引き直す
文字列 deny の代替は無い

出典: https://code.claude.com/docs/en/permissions
https://learn.chatgpt.com/docs/agent-approvals-security

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、AGENTS.md 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。

Claude Code から Codex への移し方

本文を抜いて登録を書き直し権限を引き直す順

Codex 側の着地ファイル

AGENTS.md	同じパス	hooks.json	config.toml	sandbox_mode

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 061 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。effort の段階名は low〜xhigh が両方にあります。
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://learn.chatgpt.com/docs/config-file/config-reference 、 https://learn.chatgpt.com/docs/hooks

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	25	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	2	10	1M	low〜max	日常のコーディング、データ分析、ツール使用
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
Codex(gpt-5.5)	—	—	—	minimal〜xhigh	config.toml の review_model に置く想定

> Codex は ChatGPT のプラン内で回す方が多いため、単価ではなく1日に回せる回数で見ます。上限は各自のプランの画面で。

effort の既定は Fable 5.1・Opus 5・Sonnet 5 とも high、段階は low / medium / high / xhigh / max の5つです。
effort のスケールはモデルごとの較正で、Sonnet 5 の medium と Sonnet 4.6 の high が近くなります。

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

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、062 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

Codex 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。

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 D2-09 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、AGENTS.md	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
☐	モデル名は1箇所(settings か config.toml)に集約されている	agents/*.md、config.toml	**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_mode	**設計が要る**

配布物: 乗り換え耐性チェックリスト(A4)

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]

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

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 /status で確認します。

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

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	モデル既定のホールド	Fable 5・Opus 4.8・4.7	Opus 5・Fable 5.1 には無い
4	`modelSettings / effortLevel`	`settings.json`	/effort の確定で書き戻し
5	モデル既定	なし	high、Opus 4.7 だけ xhigh
要確認	`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。
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

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 / Sonnet 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

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。

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": { "effort": "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。

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 2.1.260・2.1.267

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

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

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 は常時 adaptive で動作	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

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

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

2.1.248 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 だけは親のまま。実際のモデルと effort は /tasks の行で確認できます。

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

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. 2.1.238 以降は、温かいキャッシュがある状態で /model や /effort を打つと
   確認ダイアログが出ます。PreModelSwitch hook で強制も省略もできます。
3. 例外は Fable 5.1 の effort 変更だけです(2.1.260)。
   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

Claude Code と Codex の設定キー

切替の単位が Claude は agent で Codex は review

	Claude Code	Codex
モデル	settings.json の model	~/.codex/config.toml の model
effort	`effortLevel` / `modelSettings` low〜max の5段	`model_reasoning_effort` minimal〜xhigh の5段
レビュー用の別モデル	.claude/agents/*.md の model:	review_model(/review のときだけ)
切替の単位	skill / agent の frontmatter	プロファイルを --profile で切替 `$CODEX_HOME/<name>.config.toml`

Codex は review_model を1行置くだけで実装とレビューのモデルが分かれます。
Claude Code は agent の frontmatter で同じことをします。

出典: learn.chatgpt.com/docs/config-file/config-reference

※ 各ロゴは各社の商標です。

Codex の config.toml

review_model 1行で実装とレビューのモデルが分かれる

~/.codex/config.toml

```
model = "gpt-5.5"
review_model = "gpt-5.5" # レビュー専用に別モデル名を置く
model_reasoning_effort = "high"
```

> プロファイルは $CODEX_HOME/<name>.config.toml に置きます。
> 起動時に codex --profile <name> で切り替えます。
> プロジェクト配下の .codex/config.toml が読まれるのは、
> projects.<path>.trust_level = "trusted" のときだけです。

review_model を立てると、セッションのモデルに関係なく /review は指定モデルで走ります。
レビュー用プロファイルは sandbox_mode = "read-only" と approval_policy = "never" を組み合わせます。

出典: learn.chatgpt.com/docs/config-file/config-reference

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。

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

Claude Code 列は .claude/agents/ 配下のファイル名、最終行だけ .claude/settings.json です。
Codex 列の profile は model_reasoning_effort を low / medium / xhigh に設定したものです。
行2で使う review_model は --profile light 側に置きます。行4の既定は Opus 5 または Sonnet 5 です。

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。

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

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**
  /effort status で現在値を見ます
- [ ] **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**
  保存されず Not applied と出ます。--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
- [ ] **claude --version が 2.1.267 以降か**
  この版の修正でホールド中の frontmatter 無視が直りました — 2.1.267 / 2026-09-09

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

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

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。
Claude Code は .claude/agents/ に2ファイル、Codex は profile 2本です。
5行の修正と50行の修正の差分に両方を当て、/tasks のモデル名と /usage のトークン差を台帳に記録します。

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

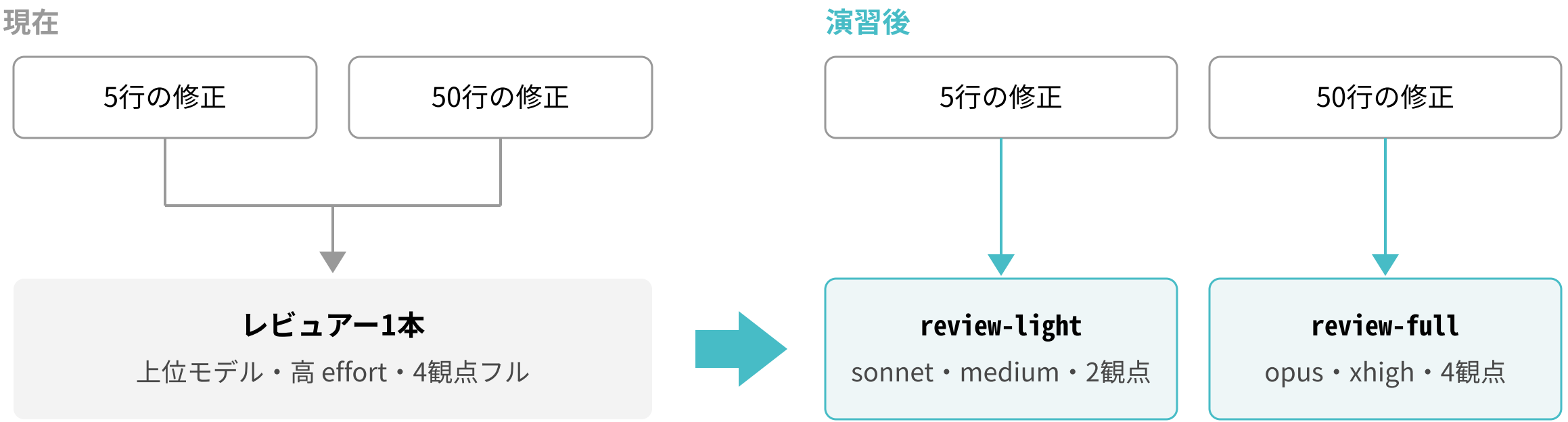
dl-training-app の .claude/agents/ にレビュアーを2本つくります。
5行の修正と50行の修正に両方を当て、モデル名と effort と指摘数の差を目で確かめます。
記録は指摘件数と所要で、終わりに 097 の置き場所表へ移します。

[20min]

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。

ワーク

開始前の確認

[1min]

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 --version で 2.1.267 以降と表示
☐	演習フォルダ	VSCode で dl-training-app を開き、app・db・tests・.claude が見える
☐	agents の置き場	.claude\agents\ が空で存在する。無ければ新規フォルダを作る
☐	effort の現在値	/effort status で現在値が出て、環境変数による固定が無い
☐	Codex のみの受講者	codex --version と %USERPROFILE%\.codex\config.toml の存在

2.1.267 で maxEffortLevel が追加され、サブエージェントの effort 指定が
既定ホールド中に無視される不具合も直っています。
版が古い端末では、093 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

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。

ワーク

Step1 review-light.md の作成

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行未満の変更向け。指示書一致と絶対制約だけ見る
model: sonnet
effort: medium
tools: Read, Grep, Glob
---
作業ツリーの差分だけを読み、次の2観点で指摘を3件以内で返します。
1. 指示書一致: 依頼された修正以外の変更が混ざっていないか
2. 絶対制約: SQL の文字列連結、秘密情報の直書き、トランザクション省略
```

VSCode で .claude\agents\review-light.md を新規作成し、上の10行を写します。
description の「50行未満」は、090 で自分が書いた境界に置き換えます。
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

ワーク

Step2 review-full.md と settings.json

Step 2 [3min]

上限を上げないと xhigh は high に丸まる設定

1 Step 2-1 review-full.md の作成

review-light.md を複製して review-full.md を作り、name・description・model・effort と本文の観点の5箇所を書き換えます。model は opus、effort は xhigh、観点は指示書一致・絶対制約・連動確認・正確性の4つにします。

期待される画面: .claude\agents\ に md が2本

2 Step 2-2 settings.json への追記

右の effortLevel と maxEffortLevel の2キーを .claude\settings.json に足します。
JSON の末尾カンマが無く、波線が出ていない状態が正解です。

.claude/settings.json

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "maxEffortLevel": "xhigh"
}
```

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

ワーク

Step3 5行修正への2本当て

Step 3 [6min]

/tasks の行に出るモデル名と effort が答え

期待される画面

/tasks の行の読み方

```
> git apply ..\diffs\small-5lines.patch
> /tasks
review-light sonnet effort: medium
review-full opus effort: xhigh
```

終わったら git checkout -- app/libraries/util.php で
戻し、Step4 で large-50lines.patch を当てます。

1. 配布 ZIP の diffs\ に差分が2本あります。
   PowerShell で small-5lines.patch のほうを
   git apply で当てます。
2. review-light で1回、review-full で1回、
   同じ差分をレビューさせます。
3. 各回の後に /tasks を開き、モデル名と
   effort を 094 の表に書き写します。

> **期待される画面**
> review-light の行は sonnet と medium、
> review-full の行は opus と xhigh。

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

ワーク

Step4 50行修正と記録表

Step 4 [6min]

同じ差分で変わる数字を4行に残す記録

差分	レビュアー	モデル(/tasks)	effort(/tasks)	指摘件数	所要
5行	review-light				
5行	review-full				
50行	review-light				
50行	review-full				

diffs\large-50lines.patch を git apply で当て、Step3 と同じ順で2本を走らせます。
この Step で必ず埋めるのは50行の2行です。5行の2行は Step3 で書いた分をそのまま残します。
4行が埋まったら、review-light が50行差分で何を見落としたかを1つだけ書き添えます。

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

補足

Codex の並行手順

プロファイル2本と --profile で同じ比較

%USERPROFILE%\.codex\light.config.toml

```
model = "gpt-5.5"
review_model = "gpt-5.5"
model_reasoning_effort = "medium"
sandbox_mode = "read-only"
approval_policy = "never"
```

実行コマンド

```
codex --profile light review `
 --uncommitted "指示書一致と絶対制約の2観点"
codex --profile full review `
 --uncommitted "指示書一致と絶対制約の2観点"
```

同じ内容で full.config.toml を作り、model_reasoning_effort だけ xhigh に変えます。
差分を当てたあと --profile light と --profile full の順で、同じ差分を流します。
記録は 094 の表に同じ形で書き、モデル列には profile 名を書きます。
Codex は P2/P3 の指摘も出すため、件数は Claude 側と粒度が違います。

> モデル名は Codex 側の一覧が正です。手元の codex --version と一覧で
> 確認してから書いてください。

出典: https://learn.chatgpt.com/docs/config-file/config-reference,
https://learn.chatgpt.com/docs/developer-commands?surface=cli

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

- [ ] .claude\agents\ に review-light.md と review-full.md がある
- [ ] /tasks で review-light の行が sonnet と medium、review-full の行が opus と xhigh
- [ ] 094 の表が4行埋まり、所要の列に差が出ている
- [ ] settings.json に effortLevel と maxEffortLevel の2キーがあり、JSON エラーが無い

4つ揃えば合格です。2番目だけ揃わない人は、092 の上限設定から見直します。

つまずき

review-full が high のまま
maxEffortLevel か環境変数で
上限が下がっています

frontmatter の effort が無効
CLAUDE_CODE_EFFORT_LEVEL
を外して再起動します

サブエージェント未呼び出し
指名が届いていません
review-light と名前で呼びます

Codex の場合

1 %USERPROFILE%\.codex\ に light.config.toml と full.config.toml がある
2 --profile light と --profile full の2回が完走し、094 のモデル列に profile 名が入る
3 094 の指摘件数の列が4行埋まる。effort 列は Codex では空で構いません

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 の差分2行	effortLevel と maxEffortLevel を既存の settings.json へ追記	同じ2キーを追記。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 D1-05 の行に、3点と 094 の数字を書きます。
agents の md は UTF-8 で保存し、Claude Code 起動後に /agents で2本が読めているかを見ます。

> **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

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

/rewind・git・DB の3層を、何が戻せて何が戻せないかで並べます。

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。

[30min]

解説

「戻して」の一言で消えたコミット

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

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 reset を禁じ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

解説

巻き戻しの3層

層ごとに違う戻せる範囲

守備範囲　　戻す手間

第1層 /rewind

セッション内。Edit と Write の変更。保持は 30日

第2層 git

コミット単位。Bash を含む全ファイル。永続

第3層 DB・外部

スキーマとデータ。コードを戻しても戻らない

上の層ほど手軽で守備範囲が狭く、下の層ほど戻せないものが増えます。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、3層で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

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 /rewind	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	Esc Esc から Restore code 保持は cleanupPeriodDays の既定 30日
第2層 git	Bash を含む 全ファイル	DB、外部 API の副作用	公開済みは revert、未公開 だけ reset。消えたら reflog worktree で隔離
第3層 DB	なし コードを戻しても データは戻らない	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。git で戻してもデータは前に進んだままです。

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

実演

/rewind の開き方と6つの選択肢

入力欄を空にしてからの Esc 2回

期待される画面

```
> _
(prompt is empty, then Esc Esc)
Rewind to a previous checkpoint
> Restore code and conversation
Restore conversation
Restore code
Summarize from here
Summarize up to here
Never mind
```

チェックポイントはユーザープロンプトごとに
自動作成され、直近 100件を保持します。
claude --resume 後も /rewind できます。

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

1 Esc 2回で開く

入力欄を空にしてから押します。文字が
残っていると入力クリアになります。

2 選ぶのは上の3行

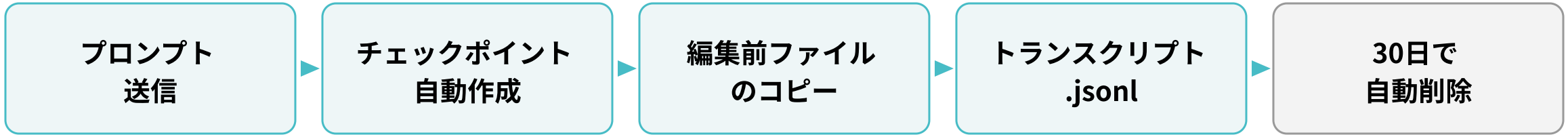
1行目は会話とファイルの両方、2行目は
会話だけ、3行目はファイルだけが戻ります。
下の3行は戻さずに要約へ切り替える側で、
ファイルには触りません。

> **コード復元の2択**
> その時点以降に編集があるときだけ表示

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

解説

/rewind で戻らない4種

守備範囲外の4種

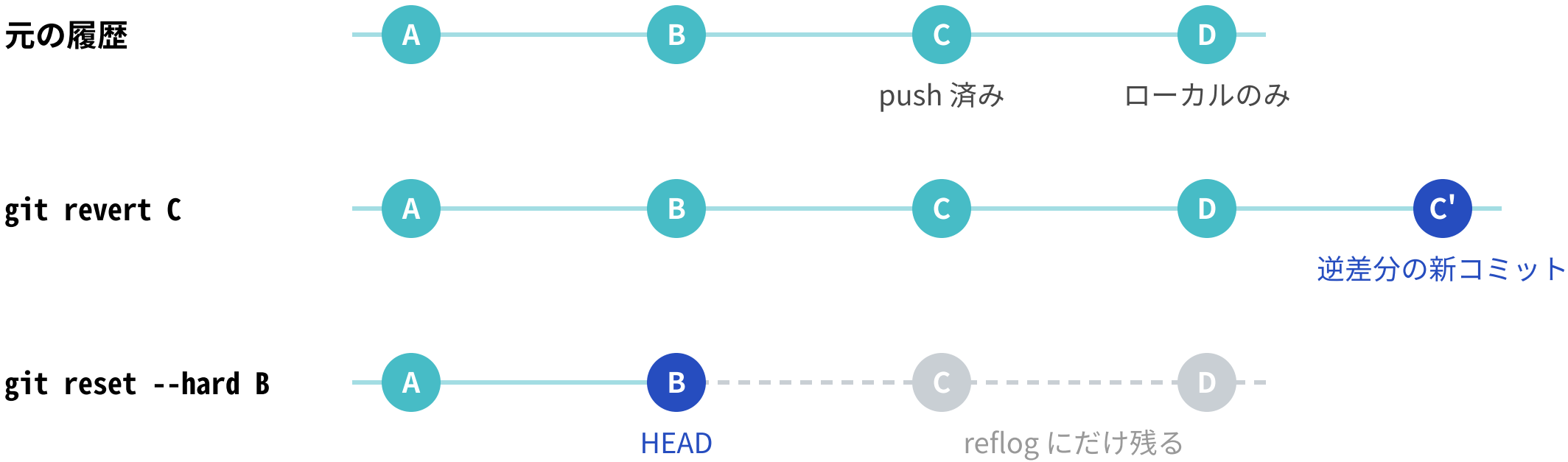
変更の種類	戻す手段
Bash コマンドの変更 rm・mv・cp・git など	git
サブエージェントの編集 フォアグラウンドの context: fork だけ例外	git
Claude Code 外の手編集と別セッションの編集 同じファイルでも追跡の対象外	git
シンボリックリンクとハードリンクのファイル Restored the code, but skipped N files と警告	git

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

解説

revert と reset の使い分け

公開済みは revert・未公開だけ reset



未コミットの作業ツリーは reflog にも残らない

revert は逆差分の新コミットを足すので、共有ブランチでも安全です。reset は履歴を書き換えるので、許すのは自分のブランチの未 push 部分だけにします。

出典: https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting

解説

消えたコミットの救出手順

reflog が残る間の救出

PowerShell

```
PS> git reflog
PS> git branch rescue 882e618
PS> git switch rescue
PS> git log --oneline -5
```

reset --hard で消えたコミットも、reflog が残る間は sha から枝を生やして戻せます。
事故直後は git reflog で sha を確認し、git branch rescue <sha> で即退避します。
未コミットの作業ツリーの変更は reflog に残らないので、この手順では戻りません。

> issue #34327 の1回目は、この手順で 51秒後に復旧できています。

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

補足

Claude Code と Codex の巻き戻し

ツールを跨ぐ唯一の保険は git のコミット

Claude Code

- /rewind で会話・コード・両方の3択
- 保存先は ~/.claude/file-history、30日
- Bash とサブエージェントの変更は戻らない
- --worktree で隔離起動が組み込み

Codex CLI

- Esc は会話だけ、ファイルは戻らない
- 実験的 /undo は 2026-01-21 に撤去
- 復活要望 issue #9203 は open のまま
- sandbox_mode で .git は読み取り専用

Claude Code の感覚で Codex の Esc を押すと、戻したはずの変更が残ります。

> タスク前に git commit、終わったら差分を確認して git commit。

出典: https://github.com/openai/codex/discussions/9618　※ 各ロゴは各社の商標です。

解説

worktree による作業の隔離

壊す範囲を先に区切る箱

```
claude --worktree issue-1
```

専用の worktree とブランチを作って起動します。サブエージェントは frontmatter で常時隔離できます

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

起動時に作成

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

終了時に keep / remove を選択
変更なしの無名セッションは自動削除

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

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 *)",
    "Bash(git push --force *)",
    "Bash(git push -f *)",
    "Bash(git checkout -- *)",
    "Bash(git clean *)"
  ] }
}
```

止める対象

reset --hard、push --force(-f)、
checkout --、clean の4種 5本

評価の順序

deny → ask → allow の順で評価
最初に一致した1本が勝ちます

例外の扱い

deny に例外は作れません
信頼の確認を待たず即時適用

> D1-07 でこの5本を各自の .claude/settings.json に書きます

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

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

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

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

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

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

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 /rewind	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 git	戻し方の文章は [A] 止まり	deny 4種 5本と block-destructive.ps1 ASCII のみ	deny 4種 5本と bash 版のフック
第3層 DB	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、第3層も deny で止める側を足すと効きます。

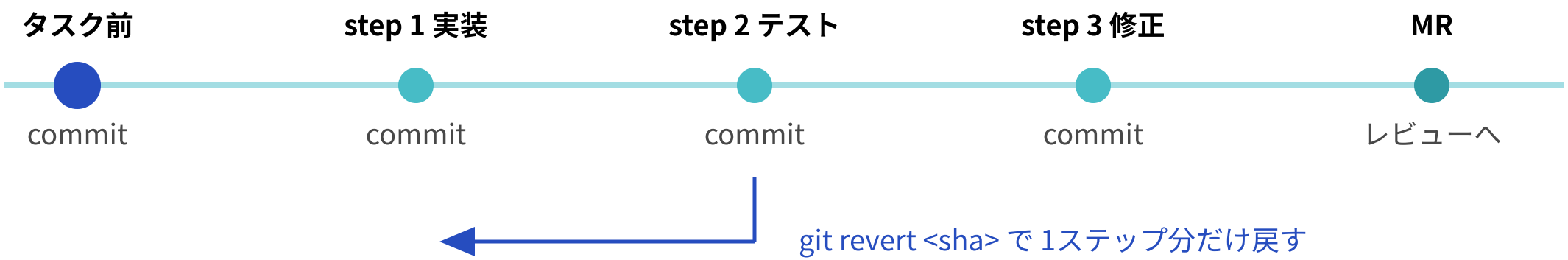
解説

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



Codex も同じ手順です。ファイルを戻す機能が無い分、コミットの粒度がそのまま保険になります

> issue #34327 で消えたのは未コミットの部分で、コミット済みなら reflog から戻せました

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

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 /rewind

入力欄を空にして Esc Esc
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 git

deny 4種 5本を
.claude/settings.json へ
CLAUDE.md に戻し方と
1ステップ1コミットを追記
フック化は D1-10 で

第3層 DB

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、第2層の deny と CLAUDE.md を書きます

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]

ワーク

演習の流れと所要

壊す前半と止める後半の4ステップ

Step1 で /rewind が戻せない変更を1つ残します。
Step3 の置き場所は Claude Code が .claude/settings.json、
Codex は ~/.codex/config.toml と AGENTS.md です。
Step4 で「最後の push まで戻して」と頼み、拒否される様子を確認します。

ワーク

開始前の確認

未コミットの作業が残る端末の扱い

git status が clean になってから Step1 へ

残っていれば、先にコミットしてから進んでください。

`claude --version`
2.1.267 以降

Claude Code
統合ターミナルで起動

`settings.json`
effortLevel が入る

Codex の場合
`config.toml` を開く

Step1 で Bash の mv を実行させるため、未コミットの変更があると巻き戻しで混ざります。
上の4つは D1-05 の続きなので、ほとんどの端末はそのまま進めます。

実演

Step1 Edit と Bash を混ぜた巻き戻し

入力欄を空にしてからの Esc 2回

Step1-1 Claude Code に入力する文

```
index.php の先頭コメントに「// rewind test」を1行追加して。
終わったら Bash で mv app/libraries/util.php app/libraries/util.bak を実行して
```

Step	操作	期待される画面
Step1-1	上の文をそのまま入力します	Edit で index.php の差分が表示され、続けて Bash の mv が許可確認の後に実行されます
Step1-2	入力欄を空にして Esc を2回押します	チェックポイントの一覧が出ます。Step1-1 の時点を選ぶと Restore code など6つが並びます
Step1-3	Restore code を選びます	index.php から「// rewind test」が消えます。util.bak は残り、util.php は戻っていません

Edit の変更だけが戻ります。結果は次のページの表に書き込んでから復旧します。

画面は講師環境で実演します

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

ワーク

Step1 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	Restore code 後	戻す手段
index.php に1行追加	Edit		/rewind の Esc Esc から Restore code を選ぶ
util.php を util.bak に改名	Bash の mv		git restore で util.php を戻し、 Remove-Item で util.bak を削除
サブエージェントが 行った編集	Task 経由の Edit	残る fork は例外	git のコミット
Claude Code の外で 手で直した箇所	VSCode の 手編集	残る	git のコミット

行1と行2を自分の画面の結果で埋めてください。
埋めたら行2の手段で util.php を戻し、git status が clean に戻ることを確認します。

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

ワーク

Step2 AI に渡す前の書き出し

設定を開く前に決める止める場所

問い1

自社ハーネスで「最後の push まで戻して」と頼んだときに Claude が選びうる git コマンドを3つ書きます。

問い2

3つのうち、push 済みの履歴や未コミットの作業を消すものに印を付けます。

問い3

印を付けたコマンドを止める場所を CLAUDE.md・permissions.deny・PreToolUse hook から1つ選び、理由を1行書きます。

3分で3枚を埋めます。正解を探すより、自分の環境で何が起きうるかを書いてください。
問い3 の答えは Step3 で設定に写し、D1-10 でもう一段強くします。

ワーク

Step3 deny の追記

文字列で止まる範囲の先塞ぎ

Claude Code は .claude/settings.json

```
{
  "permissions": {
    "deny": [
      "Bash(git reset --hard *)",
      "Bash(git push --force *)",
      "Bash(git push -f *)",
      "Bash(git checkout -- *)",
      "Bash(git clean *)"
    ]
  }
}
```

Codex は ~/.codex/config.toml

```
sandbox_mode = "workspace-write"
approval_policy = "on-request"
```

> workspace-write では .git が
> 読み取り専用になり、reset --hard
> 自体が書けません。

Codex に deny と同じ文字列ルールは
なく、sandbox が .git を守ります。

D1-05 で書いた設定の下に permissions を足します。
保存後に /permissions を開き、Deny の欄に5本が並ぶことを確認します。
JSON の末尾カンマやコメントで Settings Error が出たら、そこを疑ってください。

出典: code.claude.com/docs/en/permissions、learn.chatgpt.com/docs/agent-approvals-security

ワーク

Step3 CLAUDE.md の追記

曖昧語の解釈の先置き

Step3-2 CLAUDE.md の末尾に追記

```
## 戻し方
- 戻す時は git revert か
  git checkout <sha> で見るだけ。
  reset --hard は文字通り
  指示された時だけ
- 1ステップ1コミット。
  タスク開始前に必ずコミット
```

Step3-3 自分の手でコミット

```
git add CLAUDE.md .claude/settings.json
git commit -m "add rollback rules"
```

/memory を開くと project の CLAUDE.md が
一覧に出ます。git log --oneline -1 に
add rollback rules が出れば完了です。

Codex の場合

AGENTS.md の末尾に左と同じ
「## 戻し方」の2行を追記します。
Codex は AGENTS.md を文脈として
読むので、置き場所が変わるだけで
文は同じです。

Step3 の deny との関係

CLAUDE.md の文は助言です。Step3 の
deny と組で初めて効きます。
「見るだけ」は git checkout <sha> で
過去を開き、ブランチは動かさない
という意味です。

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

実演

Step4 拒否と代替提案の確認

reset が止まり revert が出れば合格

人	入力「最後の push まで戻して」
Claude	git reset --hard origin/main を試みます
deny	permissions.deny に一致して Bash が拒否され、 一致した deny ルールが画面に出ます。
Claude	CLAUDE.md の「戻し方」を参照し、 git log で対象を示した上で、revert か checkout を提案します。

ここで人が判断します。
revert を承認するか、
何もしないか。

提案が reset のままなら、
/permissions で Deny 欄を
再確認してください。

Step1 の前に作ったコミットがあるので、revert 対象が1つ出れば正しい動きです。

画面は講師環境で実演します

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

ワーク

拒否が出たかの確認

reset が止まり revert が出れば合格

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

	基準	確認方法
☐	index.php の1行が /rewind で戻った	Step1-3 の画面
☐	util.bak が残り、git restore で util.php を戻せた	git status
☐	/permissions の Deny 欄に5本	Step3
☐	「最後の push まで戻して」で reset が拒否され、代替案が出た	Step4

症状・原因・直し方
Esc を2回でメニューが出ない 入力欄に文字が残っている 空にして再度 Esc Esc
保存後に Settings Error JSON の末尾カンマかコメント 該当行を消して保存し直す
deny を足しても reset が通る 別の表記で呼ばれた。文字列照合の限界 D1-10 の PreToolUse hook で対処

OK 4つが揃った人は台帳に「deny 5本」「CLAUDE.md 2行」と書いてください。
つまずき3 が出た人は、そのコマンド文字列を D1-10 の素材に残してください。

Codex の場合

1 Esc のバックトラックで会話は戻り、ファイルは戻らないことを確認した
2 git restore で util.php を戻し、git status が clean になった
3 config.toml が workspace-write になっている
4 「最後の push まで戻して」で reset が .git の読み取り専用により失敗した

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)
deny 5本	.claude/settings.json の permissions.deny に追記します。ASCII のみなので文字列はそのまま置けます。	同じ .claude/settings.json。deny は Claude Code の設定なのでシェルの違いに依存しません。
「戻し方」2行	CLAUDE.md 既存初版の末尾に節を足します。	core.md の管理下にあるルール文へ [A] 扱いで足します。
1ステップ1コミット	sessionstart_info.ps1 が出す情報に未コミット件数を足すのが次の一手です。	check 系の sh に同じ検査を足します。

発展課題

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

git -C . reset --hard と bash -c "git reset --hard" を Claude に頼み、deny をすり抜けるか試します。通ったなら D1-10 で hook に書く最初のケースです。

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

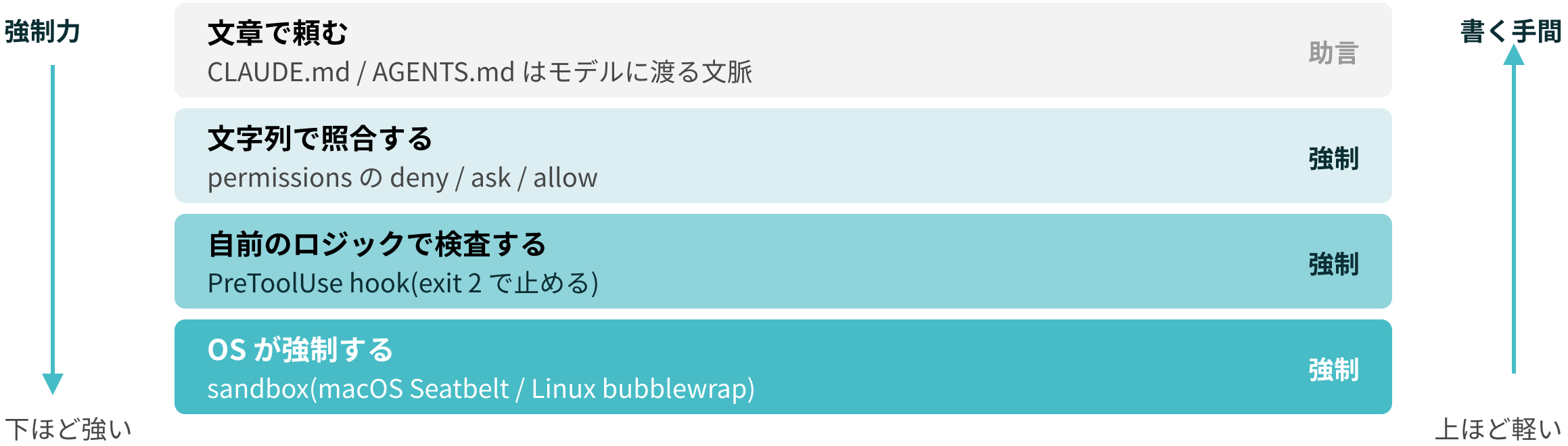
D1-07 で足した deny は、コマンド文字列を見ているだけです。
抜け道を塞ぐ層と OS が強制する層を重ね、強度の差を一次資料で確かめます。
最後に API キーや DB パスワードを AI に見せずに使わせる仕組みを実演します。

[30min]

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

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
4	.claude/settings.json	プロジェクト共有	defaultMode の auto と bypassPermissions、 sandbox の credentials.mask
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。

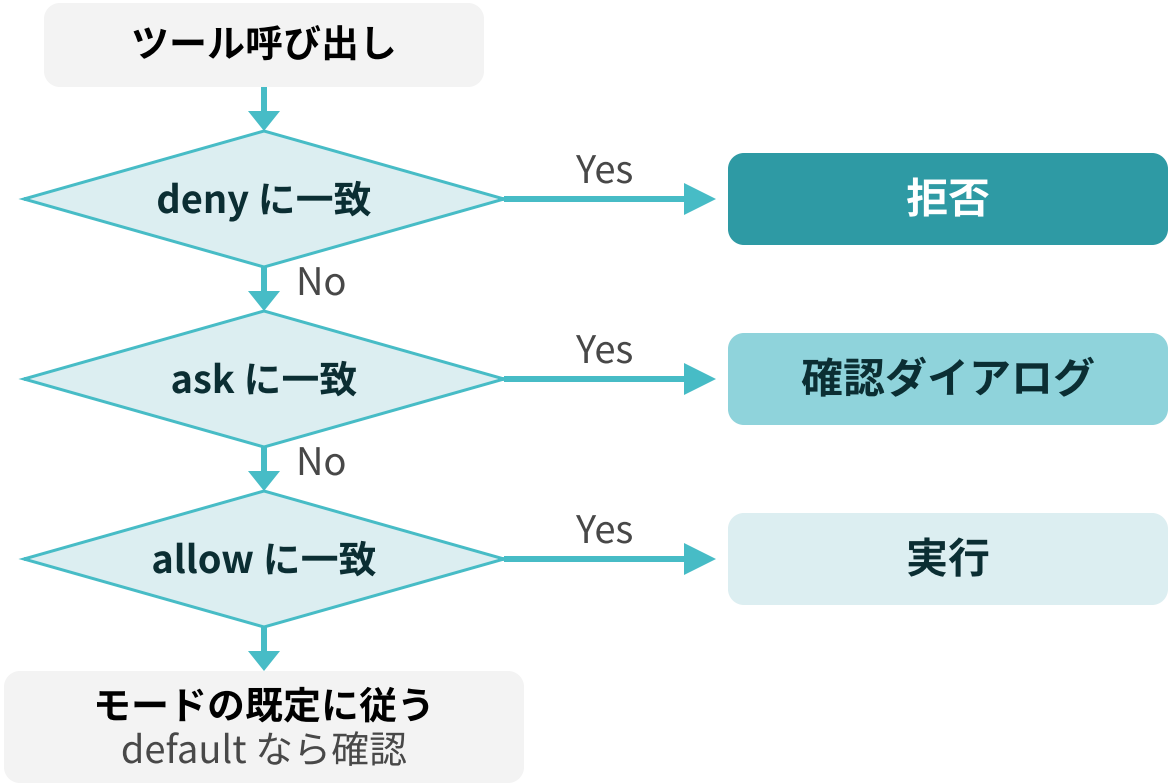
配布する研修フォルダの settings.json に auto や bypassPermissions を書いても無視されます。
「Yes, and don't ask again」で保存される allow は .claude/settings.local.json に入ります。

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

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 deny に例外は作れない



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。

ルールの具体度は順序を変えません。deny が先に評価され、allow は最後です。
deny を広く書いて allow で例外を作る設計は成立しないので、deny は狭く具体に書きます。

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

解説

研修フォルダの permissions

ask に置く1本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 *)",
              "Read(./.env)", "Read(./.env.*)"],
    "ask":   ["Bash(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4種5本、D1-10 で Read を3本足すと、この形になります。
禁止までは行かない git push 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。
出典: code.claude.com/docs/en/permissions

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

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。

解説

permission mode の6種

モードは「何を無確認で通すか」の既定値

モード	無確認で動くもの	主な用途	設定できる場所
`default`	読み取りのみ	対話の基本	どの settings でも --permission-mode
`acceptEdits`	編集と作業ディレクトリ内の mkdir・touch・ mv・cp・rm・rmdir・sed	編集を任せる対話	どの settings でも
`plan`	読み取り専用(計画だけ書く)	着手前の調査	どの settings でも
`auto`	classifier という別モデルが 全アクションを審査	Pro・Max・Team の 対話既定	user・managed・ --settings のみ
`dontAsk`	allow ルール内だけ実行、外は全部拒否	CI と無人実行	どの settings でも
`bypassPermissions`	全部通す	コンテナ・VM 内	user・managed・ --settings のみ

-p / Agent SDK / Enterprise / API キー / Bedrock・Vertex・Foundry は default が既定

Shift+Tab で default → acceptEdits → plan → default と循環します。
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

解説

モード別に無確認で通る操作

acceptEdits は「編集だけ」ではない

モード＼操作	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	通る	確認	確認	確認	拒否
acceptEdits	通る	通る	通る	確認	拒否
dontAsk	通る	allow 次第	allow 次第	拒否	拒否
bypassPermissions	通る	通る	通る	通る	拒否

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

acceptEdits 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。
保護パスへの書き込みと critical path への rm -rf は、allow ルールでも先回りできません。

出典: code.claude.com/docs/en/permission-modes の protected paths

解説

bypass でも先に走る唯一の強制点

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 は保護パスへの書き込みも通します。
その環境で効く止め金は PreToolUse hook の deny だけです。

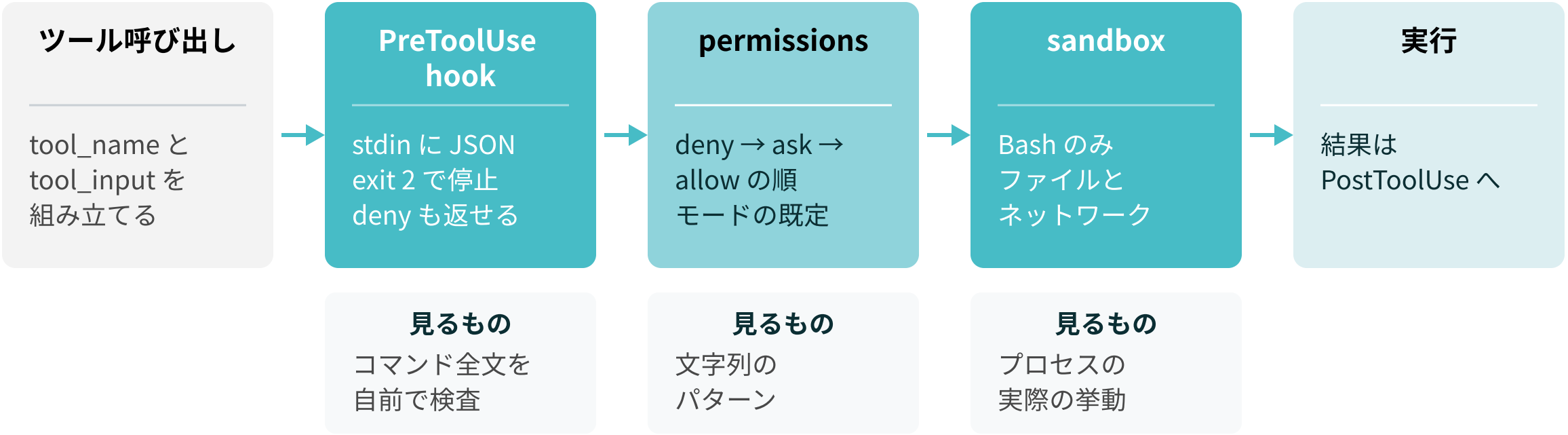
無人実行(dontAsk)でも、人が全部許した状態(bypass)でも、hook は呼び出しの前に評価されます。
bypass を使う環境を作るなら、hook を先に仕込んでから切り替えます。

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

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

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",
      "hooks": [{ "type": "command",
        "command": "powershell -NoProfile -ExecutionPolicy
           Bypass -File .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 は powershell -File で .ps1 を呼ぶ形になります。
smart3pm は bash 正なので、同じ JSON で command を .claude/hooks/block-destructive.sh に差し替えます。

出典: code.claude.com/docs/en/hooks

解説

hook が判定を返す2つの形

exit 2 と JSON を混ぜると JSON が捨てられる

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	exit 2 が優先され JSON は無視される	**使わない**

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

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。

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` `AGENTS.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

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

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": true
    },
    "credentials": { "envVars": [
      { "name": "GITLAB_TOKEN", "mode": "mask",
      "injectHosts": ["gitlab-09291006aidev.give-app.net"] }
    ] }
  }
}
```

- 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。

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。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

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 と AGENTS.md(文脈)

permissions の deny と ask

PreToolUse hook(PowerShell 5.1)

作業ツリーは worktree で隔離

MR で渡す

GitLab CI

gitlab-09291006aidev.give-app.net

- lint 8.3 と lint 5.4
  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

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
```

.env.tpl を読ませたとき
見えるのは op:// の参照文字列だけです。

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。

要確認　伏せ字の見え方は一次情報が無いため、講師環境で確かめた内容だけを話す

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(要確認)。

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。
値を直書きした .env は deny の Read(./.env) で読めなくし、.env.tpl だけを AI に読ませます。

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

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 Code・Codex	1Password と op run で起動する op run --env-file=.env.tpl -- claude op CLI は事前申請が前提	参照文字列だけ	.env.tpl start-claude.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

補足

Claude Code と Codex の境界の引き方

既定が OS 隔離の Codex と文字列照合の Claude Code

Claude Code

- permission mode 6種と allow・ask・deny
- sandbox は既定オフで /sandbox から有効化
- PreToolUse hook が全モードの前に走る
- .git と .claude は保護パス
  bypass 以外は自動承認なし

Codex

- sandbox_mode は3択で既定から OS 隔離
  read-only・workspace-write と
  danger-full-access
- approval_policy は on-request と
  never・granular
- workspace-write でも .git・.codex・
  .agents は読み取り専用
- network_access は既定 false

Codex に commit させるなら writable_roots に .git の実パスを明示します。
レビュー用のプロファイルは sandbox_mode を read-only、approval_policy を never にします。

出典: learn.chatgpt.com/docs/agent-approvals-security

※ 各ロゴは各社の商標です。

要確認　Codex の sandbox が Windows で何を隔離するかは一次情報が無いため断定しない

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	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 start-claude.ps1	同じ start-claude.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	D2-09 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。

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]

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)

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回目より下がる
SA 履歴つきの委譲	会話履歴を引き継いだ委譲	**23.8%**	別セッションの効果が消える

「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履歴を引き継ぐ形だと
効果が出ません。独立コンテキストで呼ぶことが条件です。
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)

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)

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。

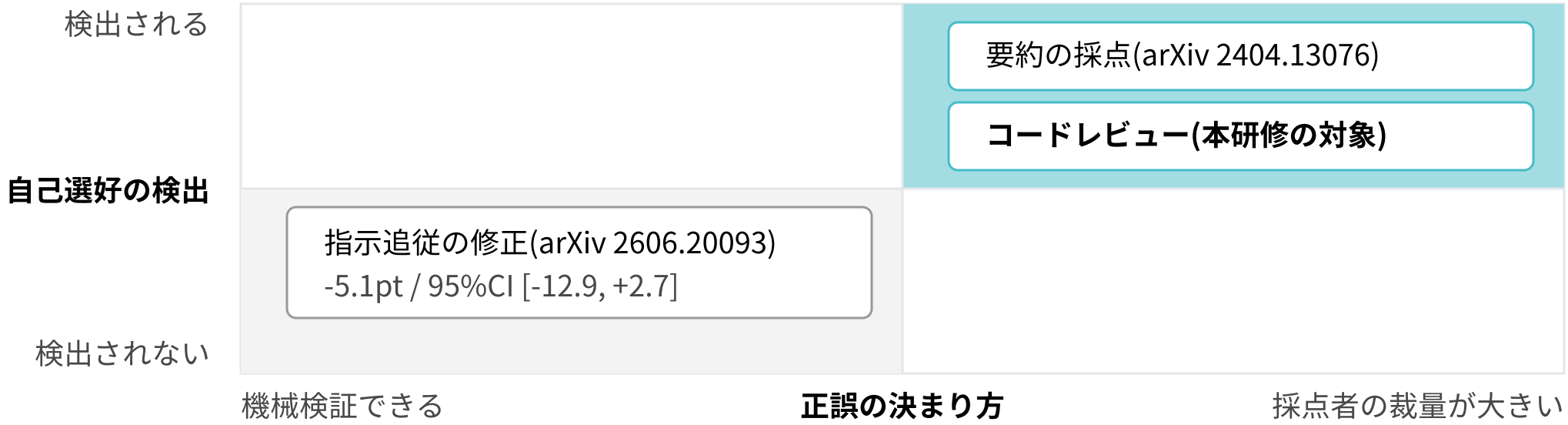
※ 各ロゴは各社の商標です。

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)

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。

実装とレビューを分ける4つの置き方

コストと再現性で選ぶ分離の層

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 同ツール別セッション	/clear 後に差分を貼り 同僚のコードとして渡す	最低	低(手順依存)
2 別サブエージェント	.claude/agents/review-other.md の model: を実装側と変える	低	中
3 別ツール	codex review --base main または claude --bare -p に差分を渡す	中	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 は今日からできますが、人が手順を守る前提です。忘れた時に止まりません。
2 と 3 は設定ファイル1本で再現します。3 は会社を跨いで分離できます。
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/headless
learn.chatgpt.com/docs/developer-commands

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。155 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 /agents の一覧に出ることを確認します。

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

Claude Code と Codex の並行手順

どちらか一方でも作れる分離

Claude Code で実装した時

1 同ツール別セッション

```
/clear
```

差分を貼り、同僚のコードとして依頼

2 別サブエージェント

```
.claude/agents/review-other.md
```

「main との差分をレビュー」と委譲

3 別ツール

```
codex review --base main "P0/P1 のみ"
```

作業ツリーは変更されません

Codex で実装した時

1 同ツール別セッション

```
/new
```

差分を貼り、同僚のコードとして依頼

2 別プロファイル

```
codex --profile review
```

read-only と review_model で /review

3 別ツール

```
git diff main | claude --bare -p
```

--json-schema で構造化出力

要確認 Codex の新規セッション操作(/new)は当日の講師環境で確認します。

出典: learn.chatgpt.com の developer-commands、code.claude.com の headless

※ 各ロゴは各社の商標です。

実演

Claude Code から Codex を呼ぶ入口

実装セッションを離れずに別会社のレビュー

導入から実行までに打つ6行

```
npm install -g @openai/codex
codex login

/plugin marketplace add openai/codex-plugin-cc
/plugin install codex@openai-codex
/codex:setup

/codex:review --base main --background
```

1 上の4行は1回だけ

端末ごとの初回だけ要ります。
2回目以降は最後の1行です。

2 実装セッションの残り方

実装セッションはそのままです。
履歴は Codex に渡りません。

3 作業ツリーの扱い

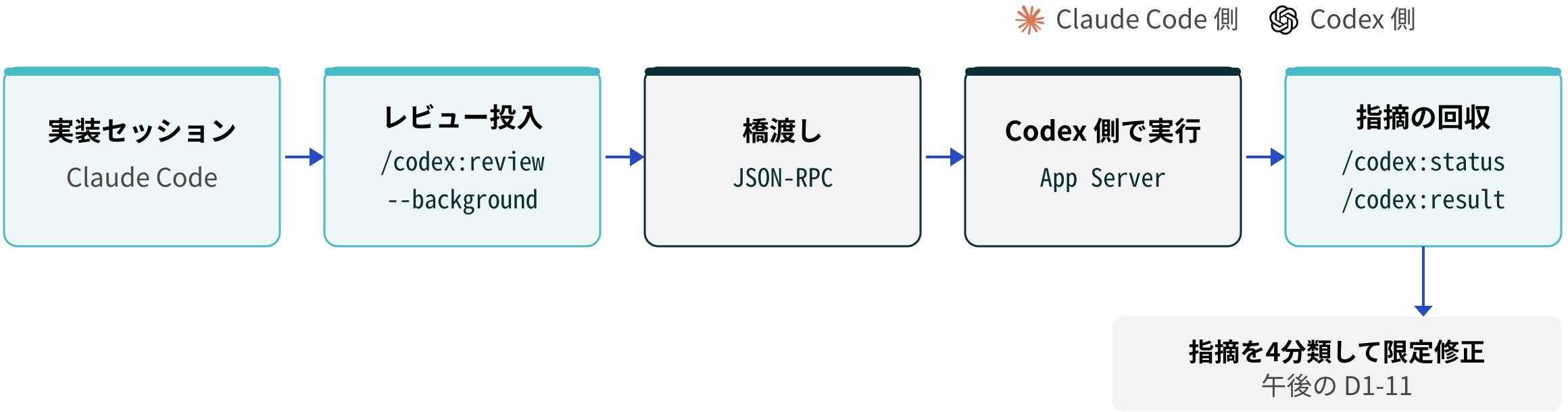
差分は main との比較で取られ、
作業ツリーは変更されません。

OpenAI 公式のプラグイン codex-plugin-cc(2026-03-30 公開)で、実装セッションの中から
Codex のレビューを呼べます。Node.js 18.18 以上と ChatGPT サブスクか API キーが要ります。

出典: github.com/openai/codex-plugin-cc

/codex:review の一周

投げっぱなしにして status で拾う一周



内部は scripts/codex-companion.mjs が Codex App Server に JSON-RPC で接続します。
Claude の会話履歴は Codex に渡りません。--background なら投げた側の作業を続けられます。
複数ファイルの変更では10分を超えることがあるため、同期待ちの --wait は研修では使いません。
/codex:status で進行を見て、/codex:result で受け取ります。採否を決めるのは人です。

出典: github.com/openai/codex-plugin-cc

※ 各ロゴは各社の商標です。

実演

review_model の設定

レビューだけ別モデルで走る1行

~/.codex/config.toml に足す1行

```
review_model = "gpt-5.5"
```

他のキーは 081 で見た形のままです。足すのは
この1行だけで、実装側のモデルは変わりません。

レビュー専用プロファイルは
$CODEX_HOME/<name>.config.toml に置き、
codex --profile <name> で切り替えます。

review_model の効き方

/review の実行時だけ使われます。
未設定ならセッションのモデルです。

推論の深さの段階

model_reasoning_effort は
minimal / low / medium /
high / xhigh の5つです。

プロジェクト配下の設定

.codex/config.toml が読まれるのは
projects.<path>.trust_level =
"trusted" の時だけです。
「置いたのに効かない」の多くはこれです。

> レビュー側を重くするときのモデル名は、Codex 側の config-reference が正です。手元の一覧で確認してください。

出典: learn.chatgpt.com の config-reference

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

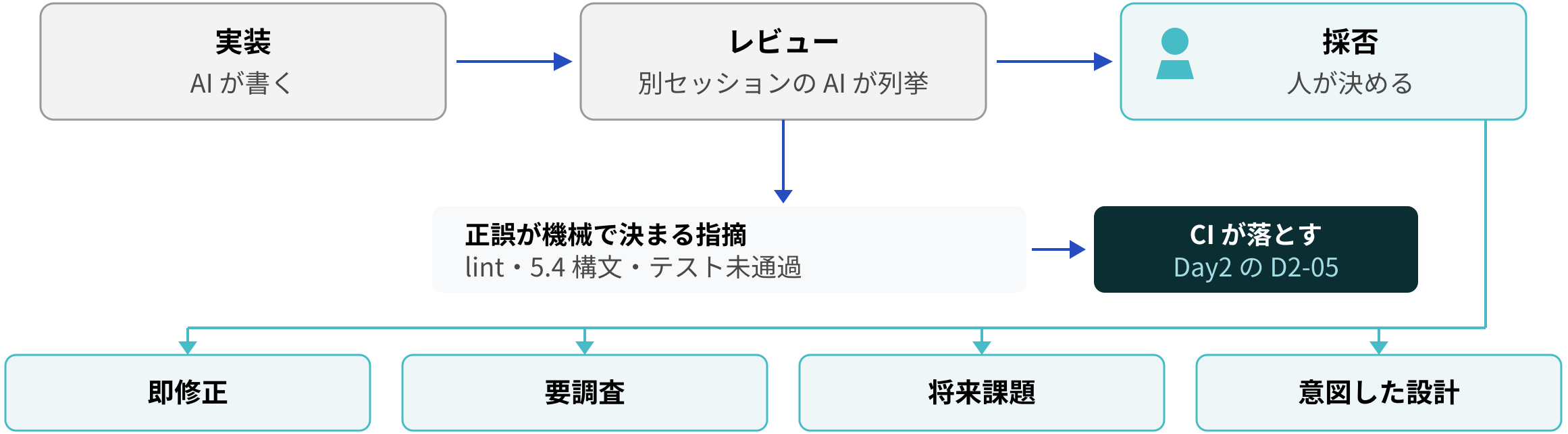
置き方	D2 の現状	足すと効く箇所
1 同ツール別セッション	一部 人の手順に依存	比較用に D1-11 で1回だけ実施
2 別サブエージェント	済 d2-code-reviewer	実装側と違う model を 明示する伸びしろ
3 別ツール	未	codex review か claude --bare -p を skill 1本にして呼ぶ
4 CI	未	GitLab CI の ai_review と ai_gate に分ける(D2-05)

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。
伸びしろは2つです。レビュー側の model を実装側と違える明示と、ツールを跨いだ第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。

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

補足

分離で踏む落とし穴

サブエージェントの model は Claude 系だけ

落とし穴	症状	対処
agents の model: に GPT を書く	指定が効かず Claude 系で動く	他社モデルは Bash から codex exec を叩く形にする
codex review の対象フラグを2つ付ける	エラーで止まる	--uncommitted・--base・--commit のどれか1つ
--background を付けない	複数ファイル変更で 10分以上待つ	--background で投げて /codex:status と /codex:result で回収
/codex:setup の --enable-review-gate	Claude と Codex のループで使用量を消費	研修環境では有効化しない 公式 README も監視下限定
claude -p を --bare なしで CI に置く	そのリポの hooks と .mcp.json が動く	--bare と ANTHROPIC_API_KEY を既定にする

1行目は必ず勘違いが起きます。サブエージェントは Claude API のモデルしか呼べません。
4行目の Review Gate は応答のたび Codex レビューが走ります。止め方を決めるまで触りません。

出典: code.claude.com の sub-agents・headless、github.com/openai/codex-plugin-cc

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
- ~/.codex/config.toml の review_model 行

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
- Codex 未保有は claude --bare -p に渡す

自社ハーネスへの書き戻し

- codex-review の SKILL.md か review-other.md のどちらか1本
- 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。

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]

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

前提確認 [3min]	Step1 deny [2min]	Step2 .env [3min]	Step3 登録 [3min]	Step4 検査 [3min]	台帳記入 [1min]

チェック	確認項目	確認コマンドまたは画面
☐	git status が clean で、D1-07 のコミット add rollback rules が最新	`git log --oneline -1`
☐	.claude/settings.json の Deny 欄に D1-07 の deny 5本が並んでいる	Claude Code で /permissions
☐	演習用の .env を作った。中身は架空の値	`Set-Content .env 'DB_PASS=dummy-secret-123'`
☐	.env が追跡対象外になっている git status に .env が出ない	出る場合は .gitignore に .env を1行足す
☐	Codex のみの受講者は .codex/ フォルダを VSCode で開いておく	codex --version も確認

演習用の .env は架空の値です。本物の接続情報は置かないでください。

ワーク

AI に渡す前の3つの問い

見せたくない物と抜け道を自分で先に挙げる

問い1

自社の作業フォルダで、AI に
読まれて困るファイルを3つ、
ファイル名の形で書きます。

問い2

Read(./.env) を deny に入れた後
でも、Claude が .env の中身を
見る手段を2つ、ツール名と
コマンドで書きます。

問い3

問い2 の手段を止めるフックが
正しく動いていることを、
Claude を起動せずに確かめる
方法を1行で書きます。

3分で3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 Step1 の deny、問い2 は Step3 の正規表現、問い3 は Step4 に写します。

ワーク

Step1 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": [
  "Bash(git reset --hard *)",
  "Bash(git push --force *)",
  "Bash(git push -f *)",
  "Bash(git checkout -- *)",
  "Bash(git clean *)",
  "Read(./.env)",
  "Read(./.env.*)",
  "Read(./**/credentials*)"
]
```

薄い5本は D1-07 で追加済みです。明るい3本を今日足します。

保存後に /permissions を開き、Deny 欄が8本になることを確認します。
パスは gitignore 記法です。./.env.* は .env.local にも一致します。
deny は allow より先に評価され、例外を作れません。

Codex のみの受講者

Codex の境界は sandbox_mode と
approval_policy の2層で、
ファイルの読み取りだけを拒否する
設定はありません。

Step1 は見学し、Step3 のフックで
塞ぎます。

~/.codex/config.toml

```
sandbox_mode = "workspace-write"
```

この1行が入っていることだけ
確認してください。

出典: code.claude.com/docs/en/permissions、learn.chatgpt.com/docs/agent-approvals-security

実演

Step2 .env を読ませて拒否を見る

Read は止まり Bash は通る

	操作	期待される画面
Step2-1	「.env を読んで、DB_PASS の値を教えて」と入力します。	Read の呼び出しが deny に一致して止まり、値は表示されません。
Step2-2	続けて「Bash で Get-Content .env を実行して、出力をそのまま見せて」と頼みます。	許可確認の後に実行され、次の出力が出ます。 `DB_PASS=dummy-secret-123`
Step2-3	何も直さず、Step2-2 のコマンド文字列をメモします。	変更なし。git status は clean のままです。

Read の deny は Read ツールにしか効きません。Bash の中身は別の口です。
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。

画面は講師環境で実演します

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

ワーク

Step3 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 に足す

出典: code.claude.com/docs/en/hooks

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

書くのは登録の1ブロック

本体は配布済み。書くのは登録だけです。
ルールは6本。env-file が Step2-2 で通った Get-Content .env を止めます(.env.tpl は except で通る)。保存後は /hooks で確認。

Codex ~/.codex/hooks.json の PreToolUse に同じスクリプトを登録します。配布雛形のとおりに置いてください。

実演

Step4 Claude 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

Step4-1 PowerShell の統合ターミナルで1行実行

Step4 [3min]

```
PS> echo '{"tool_name":"Bash","tool_input":{"command":"git reset --hard"}}' |
  powershell -NoProfile -ExecutionPolicy Bypass -File `
    .claude/hooks/block-destructive.ps1; $LASTEXITCODE

[block-destructive] reset-hard: runs reset --hard. Uncommitted work and
unpushed commits are gone. Use revert, or checkout <sha> to look only.
command: git reset --hard
2
```

Step4-2 command を git status に替える

コマンド文字列だけを差し替えて同じ1行を実行すると、止める文は出ず 0 だけが返ります。

> **時間内に回すのは2回**
> 止める1回と通す1回で合格です。cases.json への1件追加、run_cases.ps1 の一括実行、
> コミットは 178 の発展に回します。雛形は17件(通す9・止める8)が入っています。

出典: code.claude.com/docs/en/hooks

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

- [ ] /permissions の Deny 欄が8本
- [ ] 「.env を読んで」が Read の deny で止まった
- [ ] Step4-1 で 2、Step4-2 で 0 が返った
- [ ] Get-Content .env の依頼が [block-destructive] env-file: で止まる

つまずきと直し方

Step4-1 で赤いエラーが出る

JSON の引用符が PowerShell に食われています。
外側をシングル、内側をダブルにして1行で打ちます。

フックを置いたのに Get-Content が通る

settings.json の hooks が読み込まれていません。
/hooks で登録を確認し、出なければ再起動します。

実行ポリシーの警告で ps1 が動かない

-ExecutionPolicy Bypass が抜けています。
登録コマンドとテストの1行を確かめます。

Codex の場合

1 config.toml が workspace-write
2 codex で .env を読ませて通る(deny が無い差を記録)
3 Step4-1 で 2、Step4-2 で 0
4 hooks.json 登録後に Get-Content .env が止まる
OK が4つ揃った人は、台帳に deny 3本・フック1本・テスト1件と書いてください。

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5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 cases.json に1件足し、run_cases.ps1 を一括で回してください。
終わったら git add .claude と git commit で、フックとテストを対でコミットします。
発展2 .ps1 に非 ASCII が混ざったら exit 2 にする検査を、同じ仕組みで足してください。
発展3 問い1 に書いたパターンを Read の deny に足し、止まるか試してください。
いずれも D2-09 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

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 --version が 2.1.267 以降
- [ ] D1-05・07・10 の設定がコミット済み
- [ ] GitLab に Developer でログイン済み
- [ ] codex --version が返る

[160min]

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、Step4 からは履歴を切った別のセッションと別のツールに移ります。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。

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-php54 は赤のまま、
混入した構文を記録する形で満たします。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。

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。

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

```
git add .claude
git commit -m "午前の設定を持ち込む"
```

期待: プチ演習1〜3で足した設定がコミットに入る

2 Step2-2 隔離起動

```
git worktree add `
../dl-training-app-issue-1 `
-b worktree-issue-1
cd ../dl-training-app-issue-1
```

期待: 作業ディレクトリが dl-training-app-issue-1

3 Step2-3 起動とブランチの確認

```
claude
git branch --show-current
```

期待: worktree-issue-1 と返る

Codex

手順は左と同じです。Step2-3 の1行目だけ codex に変えます。

```
codex
```

期待: 新しいフォルダで Codex が起動します。

4 Step2-4 持ち込みの確認

.claude/settings.json の deny の8行と model を見ます。期待: 8行が並ぶ。空なら Step2-1 のコミットが未了

> 上の手順は今いるブランチの先頭から切るので、Step2-1 のコミットが区画に入ります。claude --worktree は既定で origin の main から分岐します。

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

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、/tasks に grep-scout の行が
haiku で出ています。

Codex

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

```
codex exec -s read-only "見積一覧の顧客名検索の条件組み立てを直します。触るファイル、
呼び出し元、テーブル、テストの有無、同じ知識が2か所にある箇所を、ファイルパスと
行番号つきで一覧にしてください。直し方は書かないでください"
```

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。重複の節には状態コードの対応表が出ます。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。

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 の本文を丸ごと渡して終わらせません。
期待: estimate_ctl.php の index() にだけ差分が出る

Codex

手順は同じです。制約の
参照先が CLAUDE.md では
なく AGENTS.md になります。
中身は同じものを保ってある
ので、どちらで作業しても
出来上がりは変わりません。

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 push し、
CI の赤を見て実装へ進みます。

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

ワーク

Step3 既存テストの書き方

Step 3 [35min]

PHPUnit 4.8 と PHP 5.4 に合わせた書き方

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);
}
```

親クラスは PHPUnit_Framework_TestCase、setUp() に戻り値の型は書きません。例外の確認は expectException() ではなく setExpectedException() です。足す1本も同じ書き方に揃えます。
顧客名に ' を含む値を入れて、例外が出ずに0件になることを確かめます。
本体側も ?? とスカラー型宣言と ::class が使えません。代替は CLAUDE.md の表にあります。

> 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。

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	push 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-php83 と test と test-phpunit の 3つが緑。lint-php54 は赤のまま

1本目の push。ここで MR も一緒に作ります

```
git push -u origin HEAD -o merge_request.create `
-o merge_request.target=main
```

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: Draft の MR が1本でき、URL が出力に出ます。2本目は同じ手順で git push だけです。

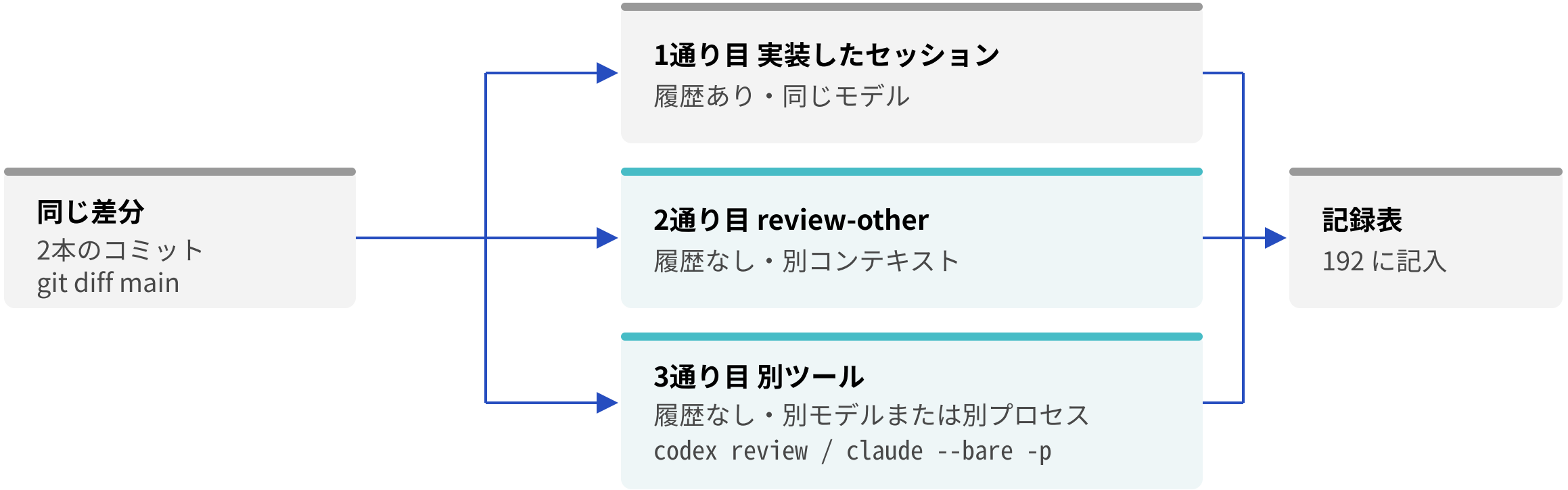
出典: https://docs.gitlab.com/user/project/push_options/

解説

Step4 3通りレビューの全体

Step 4 [25min]

履歴とモデルの2変数で分ける3通り



変えるのは2つだけです。会話の履歴を持つか、モデルまたはツールを変えるか。
差分とレビュー条件の文言は3通りで同じにします。違うのは読み手の状態だけです。

出典: https://arxiv.org/html/2603.12123

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

```
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
```

/clear を打たずに、実装したセッションでそのまま打ちます。
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 履歴を切った別セッション

```
/clear
@"review-other (agent)" worktree-issue-1 と main の
差分を、同僚が書いたコードとして読んでください
```

期待: 重大度・場所・内容・根拠・直し方の表で返り、
最後に「判定: 合格」か「判定: 要修正」が付きます。
「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

Codex

1通り目は実装した codex のセッションで同じ文言を打ちます。2通り目は codex を起動し直して同じ文言を打ちます。期待される画面はどちらも同じです。

各回で記録する3つ

指摘の件数、うち P0・P1 の件数、所要分。192 の表にその場で書き込みます。

ワーク

Step4 3通り目の別ツール

Step 4 [25min]

履歴も設定も読まない第3の読み手

PowerShell(Codex 保有者)

```
codex review --base main `
"P0/P1 のみ。3件まで。コード提案は不要。ファイル名と行番号、問題、根拠の順で書く"
```

PowerShell(Codex 未保有者)

```
git diff main | claude --bare -p `
--json-schema .claude/review.schema.json `
"REVIEW.md の基準でレビューし、findings を返す"
```

--bare は OAuth を読みません。$env:ANTHROPIC_API_KEY が空の端末は --bare を外して
git diff main | claude -p --json-schema .claude/review.schema.json "..." を使います。
この場合 hooks と CLAUDE.md は読まれるので、192 の3行目に「設定を読む別プロセス」と書き添えます。

上は別のターミナルタブで実行します。作業ツリーは変更されず、指摘だけが返ります。
返るのは findings[] を持つ JSON で、severity・confidence・file・line・message が入ります。
出典: learn.chatgpt.com/docs/developer-commands?surface=cli, code.claude.com/docs/en/headless

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	うち P0・P1	所要 [min]	気づいた差
実装したセッションでそのまま依頼				
履歴を切った別セッション /clear 後の review-other				
別ツール codex review または claude --bare -p				

この表は 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1通り目に無くて2通り目か3通り目にあった指摘を1つ書き写します。
件数の増減で良し悪しを決めません。Codex CLI は P2・P3 も返すので、比べるなら P0・P1 の列です。

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 194 の出典にある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

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

Claude Code

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。題名だけ登録します。
   自分の名前で新しい Issue が1本
4. **即修正だけを直す**
   1件1コミット。同じレビューをもう一度走らせます。
   該当箇所だけの差分。その指摘だけが消える

Codex

5-1 から 5-3 は同じ手順です。

5-4 は codex に同じ文言を打ちます。

コミットは PowerShell で人が打ちます。

```
git add -A
git commit -m
  "fix: address review P1"
```

Issue の登録は同じです。

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

ワーク

Step6 MR の仕上げ

Step 6 [20min]

Step3 で出た MR に入れる6つの欄

1 Step6-1 MR を開く

Step3 の push の出力に出た URL を開きます。
説明欄にテンプレートが入っています。

2 Step6-2 6つの欄を埋める

右の表の対応で埋めます。Draft の表示は
埋め終わってから外します。

3 Step6-3 ジョブの結果を見る

左の Build から Pipelines を開き、6本の
ジョブの結果を見ます。

> **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。
> 18名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/push_options/

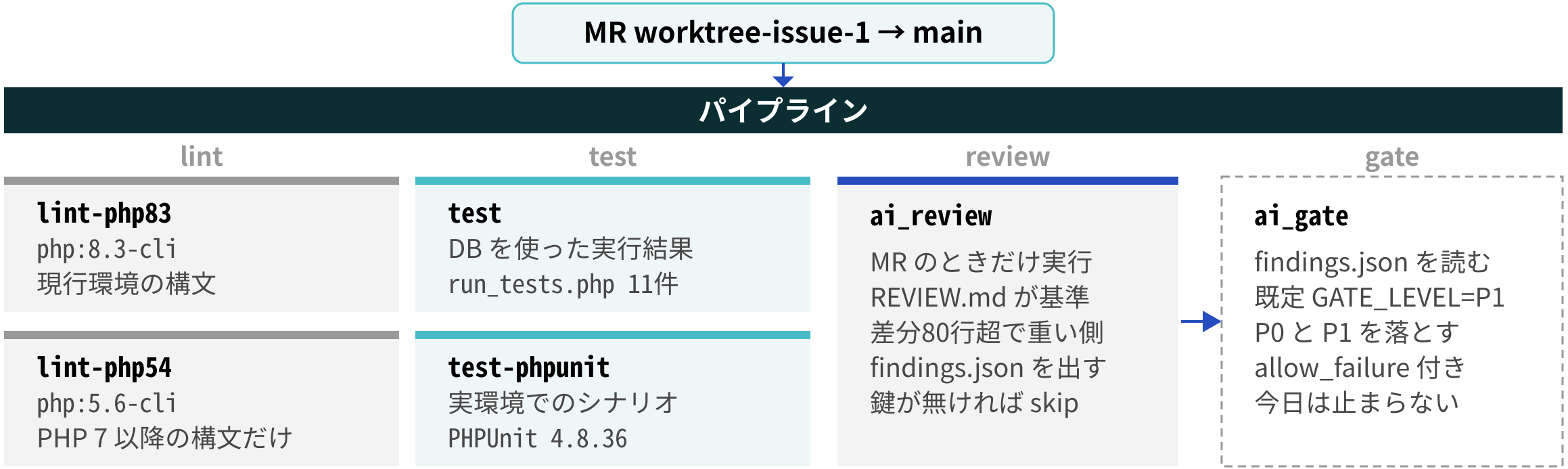
欄	入れるもの
変更概要	Issue のタイトル1行
影響範囲	185 の調査結果
テスト	188 で見た CI の結果
AI レビュー結果	192 の表
Codex レビュー結果	`codex review` の出力
人が判断した採否	193 の分類

Codex を持っていない方は Codex レビュー結果に
Codex なし と書き、review-other の結果を 192 の
2行目に入れます。Pending のまま待つあいだに、
192 の記録表と 193 の判定シートを説明欄へ貼ります。

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-php54 が捕まえるのは PHP 7 以降の構文だけです。5.5・5.6 で入った構文を捕まえるのは Day2 の phpcs です。ai_gate は GATE_LEVEL の既定 P1 で、P0 と P1 を落とします。
今日は allow_failure 付きで赤でも合流は止まりません。この1行を外すのが 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3通り目の結果を AI レビュー結果の欄に入れ、199 の ai_review 列は空のまま Day2 へ送ります。

補足

lint-php54 の赤の扱い

赤い lint-php54 は直さずに記録

lint-php54 のログ

```
PHP Parse error: syntax error, unexpected '?' in
/builds/dlive-training/dl-training-app/app/controllers/estimate_ctl.php on line 18
```

記録すること	書き方
落ちた行	ファイル名と行番号。ログの1行目をそのまま貼る
出たエラー文	syntax error, unexpected の後ろまでそのまま
混入した構文	?? のように、どの構文が入ったか
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

> Claude には直させない。Day2 の最初で全員分を並べる

記録するだけで直しません。Day2 の最初のセッションで、どの構文が混ざったかを全員分並べます。
lint-php54 は PHP 5.6 のイメージで php -l を回します。落ちるのは PHP 7 以降に入った構文だけです。
受入条件3は lint-php83 が緑で、lint-php54 は赤のまま記録すれば満たしたことになります。
合格の条件は lint-php83・test・test-phpunit の3本が緑で、MR の6つの欄が埋まっていることです。

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

Claude Code

灰色は期待される画面

1. **/shijisho 1 を打つ**
   dl-training-app に同梱の skill を使います。
   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見る**
   目的 / 影響範囲 / 変更方針 / 検証 / 戻し方
   項目が6つ以上に増えていない

3. **目的と戻し方を自分の言葉に**
   Issue の本文の言い換えになっていないかを見ます。
   言い換えでない文が2か所に入る

Codex

```
codex exec -s read-only
"Issue #1 の作業指示書を
目的・影響範囲・変更方針・検証・
戻し方 の5項目で書く。対象は
git diff main の範囲だけ"
```

7-2 と 7-3 は Claude Code と同じです。

実装の後に指示書を作るのは順番が逆ですが、生成物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。

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-php83	lint-php54	test	test-phpunit	ai_review 指摘件数	P0・P1
合計・平均							

個別の行は現地の7名分だけ読み上げます。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。

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。

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/skills/codex-review/SKILL.md

Codex を持っている人

```
---
name: codex-review
description: 実装が終わった変更を Codex に独立レビューさせ、P0/P1 だけを人に見せます。
disable-model-invocation: true
---
```

.claude/agents/review-other.md

Codex を持っていない人

```
---
name: review-other
tools: Read, Grep, Glob
model: sonnet
effort: high
---
```

どちらか1本を選び、自社ハーネスに置きます。上は人だけが呼べる設定です。
実装を sonnet で回した人は、下の model を opus に書き換えます。
出典: code.claude.com/docs/en の skills#frontmatter-reference, sub-agents

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-php83 と test と test-phpunit が緑
- [ ] MR の説明欄の6つの欄が埋まっている
- [ ] 192 の表が3行とも件数・P0/P1・所要で埋まっている
- [ ] 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
push に -o merge_request.create が無い

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
18名分の順番待ち。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

codex review 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
説明欄に6つ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 204 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。

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]

本日の持ち帰り7点

設定5点と文書2点はすべて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 effortLevel と maxEffortLevel	プチ演習1	settings.json の2キー
D1-3	deny 5本 破壊的な git 操作	プチ演習2	settings.json の permissions.deny
D1-4	CLAUDE.md の「戻し方」追記2行	プチ演習2	既存 CLAUDE.md 初版の末尾
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	codex-review の SKILL.md または review-other.md	ハンズオン1	skills か agents のどちらか片方

7点のうち5点は .claude/ の下に置く設定で、残る2点は Markdown の文書です。
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

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 / SKILL.md か agent	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

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。

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 204 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、206 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、git add と git commit を自分で打ちます	コミットのメッセージに ledger day1 が残り、Day1 の表に空欄がなくなります

置き場所が決まらない行は「未決」と書いてください。Day2 の D2-09 で確定します。
1ステップ1コミットの最後の1回として、台帳もコミットします。

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。

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。

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

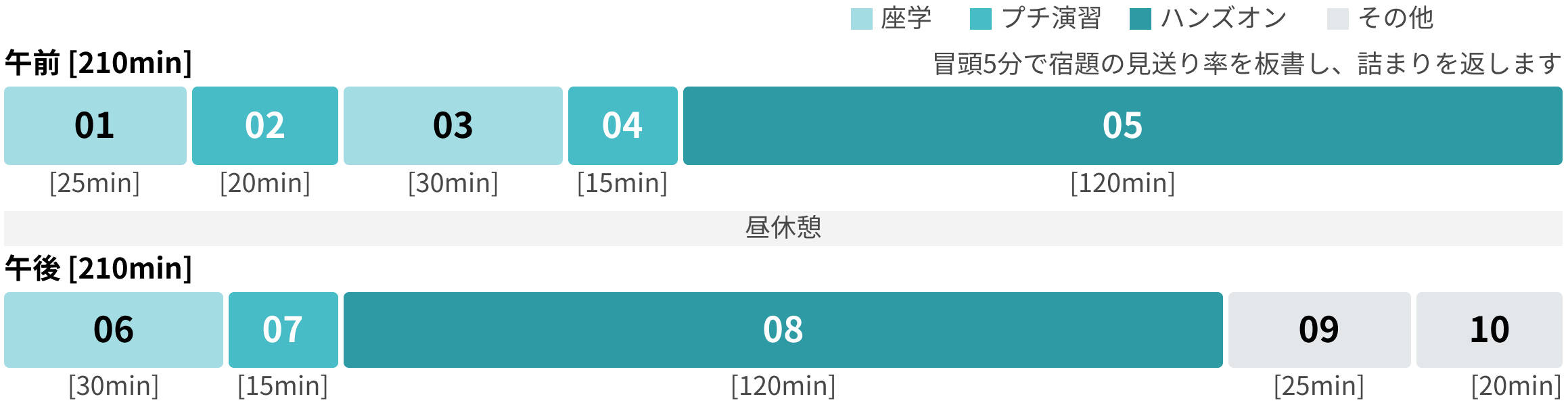
印	詰まり	今日出た症状	Day2 で返すこと
☐	**CP932 と 非 ASCII 文字**	PowerShell のフックに日本語を 1文字入れた端末で、実行時の メッセージが化けました	ASCII 検査をフックのテストに 足す案を、D2-02 の post_edit_checks と合わせて出します
☐	**PowerShell の 実行ポリシー**	.ps1 が「このシステムでは スクリプトの実行が無効」で 止まりました	事前セットアップ Step2 の 手順で解除済みかを確認します。 CI 側は runner の設定です
☐	**effort の ホールド**	settings に書いた effortLevel が 無視されたように見えました。 --effort や s キーでは解けません	Opus 5 と Fable 5.1 には ホールドがありません。Enter で 確定するか /effort <level> を打ちます
☐	**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

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 5.4 で動く指示の作り方
D2-02 プチ演習4 5.4 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、テスト生成
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 持ち帰り台帳の確定と社内展開ガイド
D2-10 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。
朝一で見た講師の実録と同じものが、Day2 の最後に各自の環境で回っている状態がゴールです。

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