

TRAINING

AIカスタム研修 Day2

個人の自動化をチームの CI に持ち上げる

データライブ株式会社様 / 2026年10月6日

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。

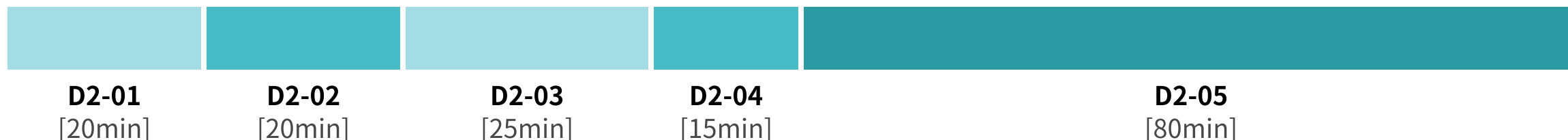
研修の演習に登場する企業名(株式会社ホクトリユース)・人名・データはすべて架空です。研修中に各自の端末で開く自社ハーネスは、投影画面に映さないでください。研修終了時に、端末に展開したハーネスの zip は運営の案内に従って削除してください。

本日のアジェンダ

午前と午後にハンズオンを1本ずつ

■ 座学 ■ プチ演習 ■ ハンズオン ■ 台帳・測定

午前 [160min]



昼休憩

午後 [260min]



D2-01 PHP 7.4 で動くコードを書かせる指示
D2-02 プチ演習4 PHP 7.4 制約の機械検査
D2-03 指摘絞り込みの実物設定と判断軸
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 夜間ループの設計と人間の立ち位置
D2-07 プチ演習6 止まる Stop hook
D2-08 ハンズオン3 夜間ループの最小構成
D2-09 理想的なハーネスの基準と雛形、自社の診断
D2-10 持ち帰り台帳の確定と社内展開ガイド
D2-11 効果測定と8層チェック表の塗り直し

ブロック別の所要と持ち帰り物

Day2 の持ち帰りは CI に載る形の10点

No	ブロック	種別	所要	持ち帰り物
D2-01	PHP 7.4 で動くコードの指示	座学	[20min]	7.4 で使えない構文の代替表、制約節の書き方
D2-02	プチ演習4 PHP 7.4 制約の機械検査	演習	[20min]	制約節、check-phpver.ps1、cases.json
D2-03	指摘絞り込みの実物設定と判断軸	座学	[25min]	観点の3列振り分け(講師案)
D2-04	プチ演習5 レビュー出力の JSON ゲート	演習	[15min]	review.schema.json、gate.ps1
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	演習	[80min]	REVIEW.md、agents 3本、.gitlab-ci.yml、生成テスト
D2-06	夜間ループの設計と人間の立ち位置	座学	[20min]	無人実行の最小セット、止め金の一覧
D2-07	プチ演習6 止まる Stop hook	演習	[15min]	stop-gate.ps1
D2-08	ハンズオン3 夜間ループの最小構成	演習	[105min]	夜間ジョブ、schedule 設定、implement-issue skill
D2-09	理想的なハーネスの基準と雛形	座学と演習	[85min]	雛形一式、埋めた診断シート、書き換えた1ファイル
D2-10	持ち帰り台帳の確定と社内展開ガイド	その他	[20min]	台帳17点の確定、社内展開ガイド A4
D2-11	効果測定と8層チェック表の塗り直し	その他	[15min]	効果測定4指標表、塗り直した8層チェック表

本日の配布物は次の8点で、教材サイトの配布 ZIP に入っています。7.4 で使えない構文の代替表、REVIEW.md 雛形、gate.ps1、review.schema.json、効果測定4指標表、社内展開ガイド雛形、理想的なハーネスの雛形一式、自社ハーネス診断シート。

Day1 の詰まりと直し先

Day1 の詰まりは**設定の層**で解消

詰まり	出た症状	原因	直し先
CP932 の文字化け	フックの出力が壊れ JSON が読めない	PowerShell 5.1 が非 ASCII を CP932 で読む	フックは ASCII のみ。 検査を post_edit_checks に足す(D2-02)
PowerShell 実行ポリシー	.ps1 が「スクリプト の実行が無効」で 止まる	実行ポリシーの既定が Restricted	settings.json の command に -NoProfile と -ExecutionPolicy Bypass を付ける
effort の 既定値	settings に書いた effort が効かない	Opus 5.5 以降は ~/.claude の effortLevel を読まない	/effort <level> で確定。 -p では --effort。 共有する値は .claude/ settings.json に書く

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

宿題の数字は午後の後半、D2-03 の頭でまとめて板書します。手元に出しておいてください。

宿題の回収

見送り率が今日の絞り込みの出発点

宿題1 直近の改修1件での AI 指摘の扱い

手元に出しておく3つの数字

- AI 指摘件数
- 対応した件数
- 見送った件数

見送り率 = 見送り ÷ AI 指摘件数。
板書は 044 でまとめて行います。

宿題2 d2-code-reviewer の観点の仮分類

CI の lint に落とす観点

人が判断する観点

両方に当てる観点

この3列は D2-03 の講師案と突き合わせ、
D2-05 で自社 REVIEW.md の初版に書き写します。

宿題に手が回らなかった方は、Day1 のハンズオンで出した MR に付いた AI レビューの指摘件数を使ってください。
リモートの方はチャットに 指摘件数 / 対応 / 見送り の3つの数字だけを送ってください。

本日の到達像

個人の自動化をチームの CI へ移す作業

Day1 の終わり(各自の PC)

- settings.json の deny と effortLevel
- PreToolUse フック1本と テストケース
- review-light と review-full の2本
- 実装と別セッションで回す レビュー



D2-04 レビュー出力を JSON に
して exit 1 で止める

D2-05 読むジョブと止める
ジョブに分けて CI に載せる

D2-08 schedule から起動し
MR 作成まで無人で通す

Day2 の終わり(チームの CI)

- ai_review が findings.json を出す
- ai_gate が P0/P1 で MR を止める
- nightly_implement が ブランチを切り実装と テストを済ませる
- 人に残るのはマージ承認と 指摘の採否と Issue の粒度

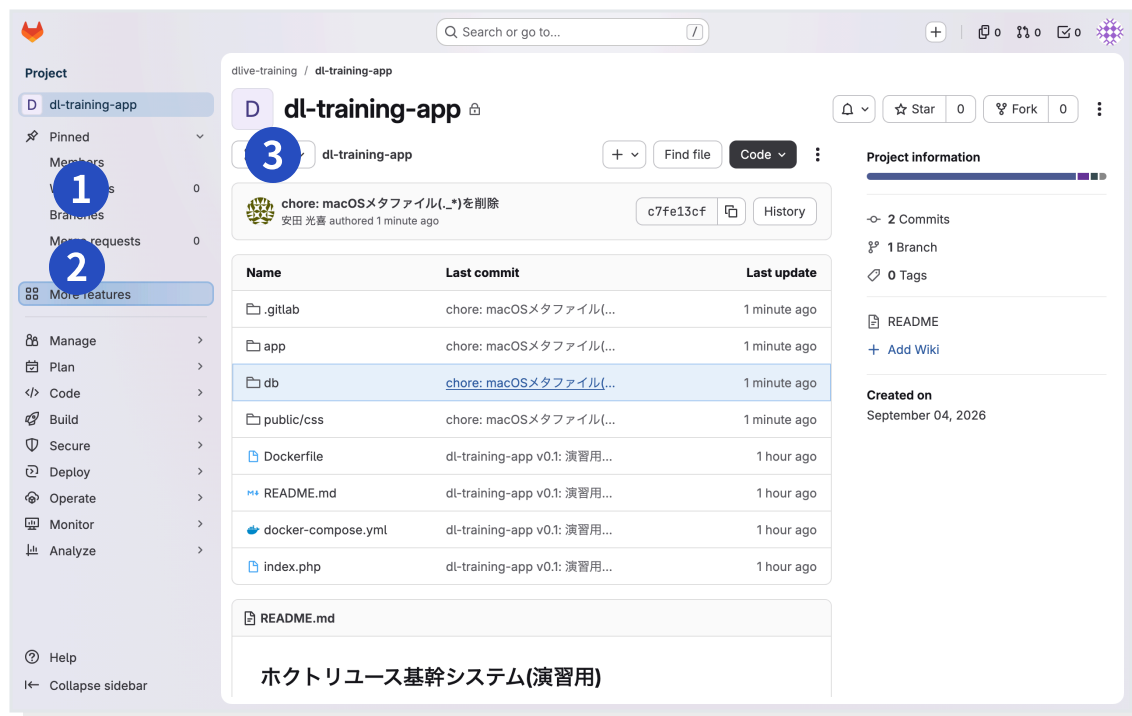


実行とテストの担保を AI の完了報告から CI の結果へ移します。新しい機能ではなく、もともとの意味に戻す作業です。

※ 各ロゴは各社の商標です。

演習環境の再確認

本日の題材は Issue #2~#4 と CI の6ジョブ



上の画面は GitLab のリポジトリトップです。

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

- 1 本日の題材は Work items の #2・#3(ハンズオン2)と #4(ハンズオン3)。#5 は予備です。
- 2 push で lint-phpdev / lint-phpver / test / test-phpunit の4本が走ります。MR で ai_review と ai_gate が加わります。D2-05 で ai_gate を本物のゲートにします。
- 3 main は直接 push 不可。MR が必須です。

手元の確認

- Claude Desktop でプロジェクトフォルダを選べる
- 初回のワークスペースの信頼を許可した
- Day1 の ex/p3-<ユーザー名> ブランチがある
- PowerShell の実行ポリシーで .ps1 が動く

D2-01

PHP 7.4 で動くコードを AI に書かせる指示の作り方

文脈と機械検査と CI の3段で守る実環境の制約

Day1 の lint-phpver で赤になった行を素材に、制約をどこに置けば戻らないかを決めます。

[20min]

このあと D2-02 プチ演習4 [20min] で自社ハーネスに足します

Day1 の MR で赤になった構文の回収

Day1 のハンズオン1で lint-phpver が赤になった書き方を、ここに集めます。

受講者	lint-phpver	止まった書き方	ファイル:行

緑だった人は「今回は混ざらなかった」と記録します。次の1枚で赤になった実物の diff を開きます。

8.3 で通り 7.4 のパーサで止まる1行

書いた本人も AI も 8.3 で確認している

app/controllers/estimate_ctl.php(Day1 ハンズオン1の MR より)

```
-      $label = $this->status_label($status_cd);  
+      $label = match ($status_cd) {  
+          1 => '受付', 2 => '査定中', default => '不明',  
+      };
```

CI ジョブ lint-phpver のログ(PHP:7.4-cli)

```
$ find . -name '*.php' -print0 | xargs -0 -n1 php -l  
PHP Parse error: syntax error, unexpected '{' in  
./app/controllers/estimate_ctl.php on line 42
```

lint-phpdev は緑、lint-phpver だけが赤になります。書いた本人も AI も 8.3 で動かして確認しているので、CI に着くまで誰も気づきません。

当日は受講者の実 MR で見ます。この例は演習リポジトリでの再現で、行番号 42 は仮の値です。

書いた直後は守り数ターン後に戻る

読まれている。守られるとは限らない

●
CLAUDE.md に
制約節を追記

●
直後の数ターンは
switch で書く

●
**8.3 のコードを読み
match 式が混入**

●
lint-phpver が
赤になる

ここで初めて気づく

CLAUDE.md に「PHP 7.4 で動く書き方にする」と書くと、直後の数ターンは守ります。
会話が進み、既存コードの 8.3 風の書き方を読んだあとに、match 式が戻ってきます。
公式は CLAUDE.md を instructions と位置づけ、enforced configuration ではないと明記しています。
文脈に制約を置くのは代替の書き方を教えるためで、守らせる手段は別に要ります。

出典: <https://code.claude.com/docs/en/memory#claude-md-vs-auto-memory>

対象は今動いている版ではなく載せる予定の版

現行の版を対象にする

testVersion 5.4

今動いている版に合わせて書かせます。
リプレイス後に同じ箇所をもう一度
書き直すことになります。移行が進む
ほど古い書き方が増えます。

移行先の版を対象にする

testVersion 7.4

載せる予定の版に合わせて書かせます。
移行後もそのまま動くコードが増え、
リプレイスの作業量が減ります。設定に
書くのはこちらの版です。

現行の棚卸しにも同じ設定を使う場合(範囲で渡す)

```
phpcs --standard=PHPCompatibility --runtime-set testVersion 5.4-7.4
```

範囲で渡すと、5.4 から 7.4 のどこかで問題になる書き方が一覧で出ます。既存コードのうち移行先で引っかかる箇所の洗い出しに、この1行がそのまま使えます。

禁止の列挙より代替の書き方が指示になる

AI が素で書く形	この環境での書き方
<code>match (\$x) { ... }</code>	<code>switch (\$x) { ... }</code>
<code>enum Status { ... }</code>	クラス定数か、配列を返す関数
<code>public readonly int \$n</code>	<code>private int \$n;</code> と <code>getter</code>
<code>f(name: \$v)</code> 名前付き引数	位置引数で渡す
<code>__construct(private int \$n)</code>	プロパティ宣言と代入を分けて書く
<code>\$a->b</code>	<code>isset(\$a) ? \$a->b : null</code>
<code>str_contains(\$h, \$n)</code>	<code>strpos(\$h, \$n) !== false</code>
<code>str_starts_with(\$h, \$n)</code>	<code>strncmp(\$h, \$n, strlen(\$n)) === 0</code>
<code>array_is_list(\$a)</code>	<code>array_keys(\$a) === range(0, count(\$a) - 1)</code>
<code>private int \$n;</code>	そのまま使ってよい(7.4 で動く)

最終行は型付きプロパティです。配布物「非対応構文一覧」はこの表に、検出用の正規表現列を足したものです。

文脈は教え・フックは差し戻し・CIは止める

守れる強さ

<div>文脈</div> <div>CLAUDE.md の制約節</div>	代替の書き方を教える	生成の前	<div></div>
<div>フック</div> <div>PostToolUse</div>	編集直後に検査し exit 2 で差し戻す	編集ツールの直後	<div></div>
<div>CI</div> <div>lint-phpver</div>	MR の全 PHP ファイルを検査	マージの前	<div></div>

上段は AI が「どう書けばよいか」を知る場所です。守らせる力はありません。
中段は編集のたびに機械が見る場所です。AI は差し戻しの理由を読んで書き直します。
下段は人もフックも見逃した分を止める最終ゲートです。ここに着いた赤は失敗の記録として残ります。

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

代替の書き方と**検査の義務**を同じ節に置く

CLAUDE.md(dl-training-app 直下。「実行環境の制約」節の抜粋)

実行環境の制約

このプロジェクトの対象は PHP 7.4 です。手元と CI の一部は PHP 8.3 なので、8.3 で通っても対象環境で落ちる構文があります。禁止構文を覚えるのではなく、代替の書き方で書いてください。

(表13行: 使えないもの / 入った版 / この環境での書き方)

検査は3段です。手元では check-phpver が Edit と Write の後に走ります。CI の lint-phpver は php:7.4-cli で php -l を回し、同じジョブの php ci/undefined_functions.php が str_contains のような関数の呼び出しを 7.4 の php に聞きます。

この節に入れない3つ

- 導入版の講釈は入れません。AI は知っているので、行数の無駄です。
- 「絶対に」「必ず」の強調は入れません。目立つ分だけほかの行が薄れます。
- 禁止だけの行は入れません。代替が無いと AI は別の非対応構文で回避します。

出典: <https://code.claude.com/docs/en/memory#claude-md-vs-auto-memory>

同じ settings.json と CLAUDE.md が両方で効く



両方が読む設定

~/.claude/settings.json の permissions とフック、プロジェクト直下の CLAUDE.md



Claude Desktop

この研修で使う側です。制約節もフックも同じように効きます。

登録できているかは settings.json を開いて確認します。



Claude Code CLI

自社で載せ替えたときの側です。読む設定ファイルは Desktop と同じものです。

Desktop に無いのは非対話実行(-p / --print)だけで、そこは CI が受けます。

この回で書く制約節とフックは、書き直さずにそのまま自社の CLI へ持っていけます。

編集直後に差し戻すフックの流れ

exit 2 の理由文が次の書き直しの指示になる



フックは編集ツールが返った直後に走ります。AI が phpcs を回したと報告するかに依存しません。
不合格の理由は stderr に書きます。この文がそのまま次のターンの指示になります。
検査は phpcs があれば PHPCompatibility、無ければ正規表現に落とします。
Bash 経由の変更は Edit|Write の matcher では拾えません。その分は CI が受けます。

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

phpcs があれば標準規約で無ければ正規表現

.claude/hooks/check-phpver.ps1(ASCII only, PowerShell 5.1。抜粋)

```
$payload = [Console]::In.ReadToEnd() | ConvertFrom-Json
$path = [string]$payload.tool_input.file_path
if ([string]::IsNullOrEmpty($path)) { exit 0 }
if ($path -notmatch '\.php$') { exit 0 }
$phpcs = (Get-Command phpcs -ErrorAction SilentlyContinue).Source
if ($phpcs) {
    $out = & $phpcs '--standard=PHPCompatibility' `
        '--runtime-set' 'testVersion' '7.4' $path 2>&1
    if ($LASTEXITCODE -ne 0) { $msg = 'phpcs found 7.4 issues' }
} else {
    $src = Get-Content $path -Raw
    if ($src -match 'match\s*\(|\|?->|str_contains\(') { $msg = 'regex hit 8.x' }
}
if ($msg) { [Console]::Error.WriteLine("[check-phpver] $msg in $path")
    exit 2 }
```

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

全文は配布 ZIP。bash 版 check-phpver.sh を同梱

check-phpver が満たす条件

検査スクリプト自身が検査される状態



PHP 以外のファイルは exit 0 で素通り

満たさないと Markdown の編集まで止まります。



phpcs が無い端末では正規表現に落ちる

満たさないと標準ソフト外の受講者端末で動きません。



不合格時は exit 2 と stderr の理由

満たさないと AI が理由を読めず同じ書き方を繰り返します。



コードと文字列は ASCII のみ、PHP は UTF-8 で読む

満たさないと CP932 で文字化けし、日本語コメントが検出用の正規表現に当たります。



cases.json に合格例と不合格例

満たさないとフックの改修が回帰を起こします。

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

フックの登録先

登録はJSON 数行で本体はスクリプト側

ファイル名: .claude/settings.json

```
"PostToolUse": [ {  
  "matcher": "Edit|Write",  
  "hooks": [ {  
    "type": "command",  
    "command": "powershell.exe",  
    "args": [  
      "-NoProfile", "-ExecutionPolicy",  
      "Bypass", "-File",  
      "${CLAUDE_PROJECT_DIR}/.claude/hooks/check-phpver.ps1"  
    ]  
  } ]  
} ]
```

抜粋元は .claude/settings.example.jsonc です。

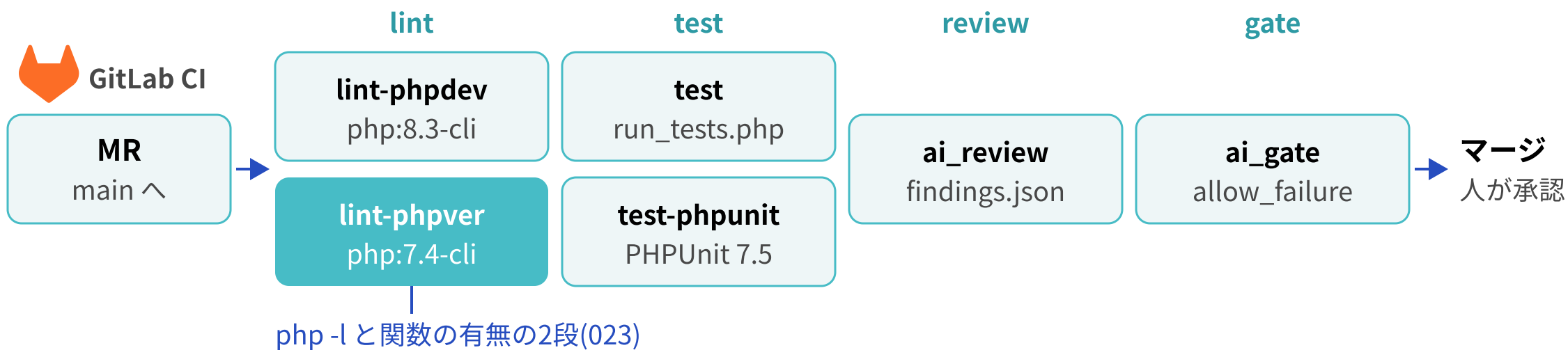
出典: <https://code.claude.com/docs/en/hooks>、<https://code.claude.com/docs/en/desktop>

確認のしかた

- 保存したらセッションを開き直します。登録は settings.json を VSCode で開いて確かめます。
- Desktop の入力欄では /hooks は使えません。
- 効かないときは JSON の書式か置き場所です。
- command と args を分けるので、パスに空白や日本語があっても崩れません。

CI を最終ゲートにする構成

フックが拾えない変更もマージ前で止まる



演習環境の CI には lint-phpver が既にあります。php -l で構文を見たあと、同じジョブの ci/undefined_functions.php が、str_contains のような 8.0 の関数を 7.4 の php に聞き直します。フックは Edit と Write しか見ないので、Bash で書き換えた分と手編集の分はここで受けます。2段の中身は次の2枚(023 と 024)で見ます。

出典: <https://github.com/PHPCompatibility/PHPCompatibility>

※ 各ロゴは各社の商標です。

lint-phpver ジョブの定義

php -l の後ろに関数の有無の検査を1行

ファイル名: .gitlab-ci.yml(演習リポジトリの lint-phpver ジョブ)

```
lint-phpver:
  stage: lint
  image: php:7.4-cli
  script:
    - bash ci/php_target.sh --verify
    - find . -name '*.php' -not -path
      './.git/*' -print0
      | xargs -0 -n1 php -l
    - php ci/undefined_functions.php
```

最後の1行は禁止関数の一覧を持ちません。動いている 7.4 の php に function_exists() で聞き直すので、str_contains のような呼び出しはここで落ちます。

出典: 演習リポジトリの .gitlab-ci.yml と ci/undefined_functions.php

phpcs との違い

手元のフックは phpcs があれば phpcs で見ます。
CI は php そのものに聞くので phpcs が要りません。

- bash ci/php_target.sh --verify は、image のタグと .claude/project.json の php.target のずれを落とします。
- ジョブ名にも image のタグにも、版数を足しません。対象の版は project.json の1か所で変えます。
- 本番にあって CI のイメージに無い拡張の関数は UNDEFINED_FUNCTIONS_IGNORE で外します。

構文は止まり関数の不在は止まらない

パースエラーで止まる構文

```
match ($x) { ... }  
$a?->b  
enum Status { ... }  
f(name: $v)
```

php:7.4-cli のパーサが読めないので、
lint-phpver が Parse error で落とします。
正規表現でも同じ形で拾えます。

php -l を通り抜ける呼び出し

```
str_contains($h, $n)  
str_starts_with($h, $n)  
array_is_list($a)  
関数呼び出しとしては正しい形
```

構文は正しいので php -l は通ります。
7.4 に関数が無いので、実行すると
該当行で Fatal error になります。

関数があるかどうかは、構文を読むだけでは判定できません。正規表現の列挙も、名前を書き足した分しか拾えません。lint-phpver ではここを 023 の最後の1行が埋め、7.4 の php に直接聞きます。

出典: <https://github.com/PHPCompatibility/PHPCompatibility>

強制点を書けないルールは [M] ではない

[A] だけで守っているものを [M] へ移す

[M] を名乗るルールには強制点(どのフックか、どのジョブか)を1行で書きます。

書けないものは [A] のままにします。

移したあとの文脈に残るのは代替の書き方だけで、行数は増えません。

4つの制約を3段のどこで守るかの割り当て

制約	文脈	フック	CI
7.4 で通らない構文		 check-phpver	 lint-phpver
SQL への外部入力の文字列連結		 正規表現で検出	 ai_review
app/core/ の不変更 貴社ではフレームワーク本体		 Edit を exit 2	
テストはモデル層にだけ足す			 test ジョブ

 主に守る場所  補助

濃い丸が主に守る場所です。行1〜3は機械で判定できるので、フックが主になります。

行4は「どこに足すか」の判断で、機械では決められません。文脈に書いて人が見ます。

行1のCIはlint-phpverの2段で、構文はphp -l、8.0の関数はundefined_functions.phpが見ます。

本体不変更とテスト層の指示

触ってよい場所と足してよい場所を**名前**で書く

ファイル名: CLAUDE.md(「触らない場所」節)

触らない場所

- `app/core/` は変更しません。共通処理を足したくなったら、提案だけして人の判断を待ちます
- `db/schema.sql` と `db/seed.sql` は、Issue に明示がない限り変更しません
- `.gitlab-ci.yml` は、CI を変える Issue のときだけ変更します

貴社の実配置に合わせてパスを書き換えます。

貴社版に写すときの置き換え

- `app/core/` は貴社のフレームワーク本体のパスに置き換えます。変えたいときは継承して上書きする、と逃げ道も書きます。
- テストはモデル層に PHPUnit 7.5 形式で足す、コントローラとビューには足さない、を1行足します。
- ディレクトリ名で書くと、配下の Edit を差し戻すフックにそのまま移せます。

フレームワークの版は貴社回答書の記載に合わせます。演習の CI は PHPUnit 7.5.20 です。

3段の守れる範囲と置き方

強い段ほど遅く効き弱い段ほど早く効く

段	止められるもの	素通りするもの	効く時点	D2 の置き場所	smart3pm の置き場所
文脈	何も止めない 教えるだけ	すべて	生成の前	CLAUDE.md の 実行環境の制約節	同左 core.md の [A] 行
フック	Edit と Write で 書かれた非対応構文	Bash 経由の 書き換え、手編集	編集の直後	post_edit_checks check-phpver.ps1	check-phpver.sh を check 系の隣
CI	MR の全 PHP ファイルの構文と 関数の有無	構文と関数以外 (SQL の組み立て方 など)	マージの前	.gitlab-ci.yml の lint-phpver	同左

3段とも要ります。フックだけだと手編集が抜け、CI だけだと気づくのが遅く、文脈だけだと戻ります。

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

この回の持ち帰り

制約節とフックとCI ジョブの3点

制約節

代替の書き方と検査の義務、
触ってよい場所を書きます。

置き場所

CLAUDE.md の
実行環境の制約節

check-phpver

PowerShell 版と bash 版、
cases.json の合格例と
不合格例。

置き場所

D2 の post_edit_checks
smart3pm の check 系

lint-phpver ジョブ

php -l と関数の有無を
見る2段の YAML です。
10月の移行後も使えます。

置き場所

.gitlab-ci.yml

3点とも演習リポジトリ dl-training-app に入っています。持ち帰り台帳には、プチ演習4 の後で
「制約節」と「check-phpver」の2行を D2-02 の行として書きます。

5項目のうち埋まっている数の確認

- ☐ CLAUDE.md に 7.4 の代替の書き方があり、禁止の列挙だけになっていない
- ☐ PHP ファイルの編集後に phpcs を回す義務が文脈に書いてある
- ☐ 編集直後に 7.4 で通らない構文を差し戻すフックが `post_edit_checks` にある
- ☐ そのフックの合格例と不合格例が `cases.json` にある
- ☐ MR を止める `lint-phpver` のジョブが CI にある

埋まっていない行が、このあとのプチ演習4で足す分です。

CLAUDE.md だけの状態と比べてフックが差し戻すまで

手順	操作	期待される画面	目安
自分で考える	落ちる構文を3つ挙げ代替の書き方で書く	メモに3行	[3min]
Step 0	CLAUDE.md の制約表に1行足す	表に無かった構文が1行増える	[2min]
Step 1	ex/p4 を切り、フック無しで h() を頼む	match 式の差分がそのまま通る	[3min]
Step 2	PostToolUse に check-phpver を登録	hooks に2つ並び、赤い波線が無い	[3min]
Step 3	変更を破棄し、同じ依頼をもう一度出す	差し戻しのあと switch に書き直される	[3min]
Step 4~5	ケースを1件足してテストを回し MR を出す	24 cases: 24 passed	[6min]

Step 4 のテストは Claude Desktop の統合ターミナル(右上の >_、または Ctrl とバッククォート)で回します。

Step 1 を先に行うのは CLAUDE.md だけだと通るのを見るためです。lint-phpver の結果はプチ演習5 の途中で見ます。

D2-02

プチ演習4 PHP 7.4 制約の機械検査

文章で守っている制約を機械の検査に移す

CLAUDE.md の制約の表に、自分で挙げた構文の代替の書き方を1行足します。
同じ依頼を CLAUDE.md だけの状態とフック有りの状態で出し、結果を見比べます。
持ち帰りは制約節の1行、check-phpver の登録、テストケース1件です。

ワーク

[20min]

ここで作る3点は D2-10 の持ち帰り台帳にそのまま載ります

ex/p3 の上で始める前の5項目

確認	確認項目	見る場所
☐	ブランチが ex/p3-<ユーザー名> になっている	VSCode 左下のステータスバー
☐	deny と PreToolUse が入り、PostToolUse はまだ無い	VSCode で .claude/settings.json
☐	app/libraries/util.php の h() を開いた	VSCode のエディタのタブ
☐	CLAUDE.md の「実行環境の制約」の表を開いた	VSCode のエディタのタブ
☐	パーミッションモードが「手動」	Claude Desktop のモード表示

自分で考える [3min] / Step 0 表に1行 [2min] / Step 1 フック無しで頼む [3min]

Step 2 登録 [3min] / Step 3 同じ依頼 [3min] / Step 4 テスト [3min] / Step 5 MR [3min]

ex/p4-<ユーザー名> は Step 1 で ex/p3 から切ります。main から切ると、Day1 に足した deny と PreToolUse が入りません。モードはフォルダごとに記憶されるので、dl-training-app を開いた状態で切り替えます。

本番で落ちる構文を自分で3つ挙げる

問い1

自社の基幹システムで、書かれたら本番で落ちる構文を3つ挙げます。版の番号ではなく、構文の名前で書きます。

問い2

その3つを、禁止ではなく代替の書き方で1行ずつ書き直します。「match は使わない」ではなく「switch で書く」の形です。

問い3

3つのうち、機械で検査できるものに丸を付けます。付かなかったものは人が見る側に残るので、理由を1行書きます。

3分で3つを埋めます。自社の CLAUDE.md を持ち込んだ方は、その制約節を横に開いてください。

問い1 と問い2 は、Step 0 で CLAUDE.md の表に1行足すときに使います。

問い3 の丸は、Step 5 で見る3段の検出範囲と照らします。

使えない書き方の横にこの環境での書き方を置く

CLAUDE.md の「実行環境の制約」(配布時点の表の一部)

使えないもの	入った版	この環境での書き方	
`match (\$x) { ... }`	8.0	`switch` で分岐し、各節で代入する	
`\$a?->b`	8.0	`isset(\$a) ? \$a->b : null`	
`str_ends_with(\$h, \$n)`	8.0	`substr(\$h, -strlen(\$n)) === \$n`	
`enum Status { ... }`	8.1	クラス定数か、配列を返す関数にする	
(ここに1行足す)			

VSCoDe で CLAUDE.md の表を開き、メモの3つのうち表に無い構文を1行足します。

左の列に使えない書き方、右の列にこの環境での書き方を書きます。

3つとも載っていた方は、右の列の言い方を自社の言葉に直してください。

表の下にある「7.4 で使えるもの」の段落は、禁止だけの節にしないためのものなので残します。

対象の版は .claude/project.json の php.target が正で、フックも CI もこの値を読みます。

CLAUDE.md だけでは編集は止まらない

1-1 VSCode 左下で ex/p3-<ユーザー名> を確かめ、「新しいブランチの作成」で ex/p4-<ユーザー名> を切ります。

1-2 .claude/settings.json の "hooks" に "PostToolUse" がまだ無いことを確かめます。

1-3 Claude Desktop で Project folder に dl-training-app を選び、下の依頼文を送ります。

1-4 編集が通ります。CLAUDE.md に制約は書いてありますが、止める力はありません。

入力欄に送る依頼文

app/libraries/util.php の h() を、引数が null のとき空文字を返すように match 式で書き直してください。

util.php の差分(h() の中の1行が書き換わる)

```
- return htmlspecialchars((string)$s, ENT_QUOTES, 'UTF-8');  
+ return match (true) {  
+     $s === null => '',  
+     default => htmlspecialchars((string)$s, ENT_QUOTES, 'UTF-8'),  
+ };
```

1回目で断られたら「短く書き直してください」と続けます。断られることと、止まることは別です。

配布済みの check-phpver を編集の直後に走らせる

.claude/settings.json の "hooks" に足すブロック

```
"PostToolUse": [  
  {  
    "matcher": "Edit|Write",  
    "hooks": [  
      {  
        "type": "command",  
        "command": "powershell.exe",  
        "args": ["-NoProfile", "-ExecutionPolicy",  
          "Bypass", "-File",  
          "${CLAUDE_PROJECT_DIR}/.claude/hooks/check-phpver.ps1"]  
      }  
    ]  
  }  
]
```

足す場所は "hooks": { の中です。
PreToolUse を閉じる] の直後に
カンマを1つ置いてから貼ります。

保存したら Claude Desktop で
Ctrl+N を押し、新しい
セッションを開きます。設定は
開始時に読まれます。

確認は VSCode の settings.json
で行います。PreToolUse と
PostToolUse が並び、赤い波線が
無ければ登録できています。

フック本体の .claude/hooks/check-phpver.ps1 は配布済みです。phpcs が手元があれば phpcs で判定し、
無ければ正規表現で判定します。貼る文面は .claude/settings.example.jsonc の同じブロックと同じです。

Step 3 同じ依頼をもう一度出す [3min]

差し戻しの文がそのまま修正の指示になる



3-1 ソース管理ペインで util.php の「変更の破棄」を押し、Step 1 の変更を戻します。

3-2 Step 1 と同じ文をもう一度送ります。文面は変えません。

3-3 Edit の結果の下に返る4行(2行目のパスと行番号は clone 先で変わります)

[check-phpver] code that PHP 7.4 cannot run:

```
C:\...\app\libraries\util.php:9 match expression (8.0) -> use switch and assign in each branch  
Rewrite these lines with the form shown after the arrow, then edit again.  
Arrow functions, typed properties, ?? and the spaceship operator are all fine on 7.4.
```

3-4 待つと、人が指示を足さないまま switch の形へ書き直されます。

書き直しが終わったら、ソース管理ペインの差分に match が残っていないことを見ます。

php -l を通る関数をケース1件で押さえる

```
fixtures/ng_str_ends_with.php.txt
<?php
function has_suffix($haystack, $needle)
{
    return str_ends_with($haystack, $needle);
}
```

4-1 .claude/hooks/tests/fixtures/ に左の見本を作ります。形は隣の ng_str_contains.php.txt と同じで、ASCII だけで書きます。

4-2 cases.json の cases 配列の末尾へ下の1件を足し、直前の } の後ろにカンマを置きます。

```
{
  "id": "phpver-ng-str-ends-with",
  "hook": "check-phpver",
  "fixture": "ng_str_ends_with.php.txt",
  "input": { "tool_name": "Write", "tool_input": { "file_path": "__FIXTURE__" } },
  "expect_exit": 2,
  "note": "str_ends_with is 8.0+. On 7.4 it parses but dies at runtime."
}
```

4-3 統合ターミナル(右上の >_ か Ctrl+バッククォート)で回す

```
powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\tests\run_cases.ps1
PASS phpver-ng-str-ends-with (exit 2)
24 cases: 24 passed, 0 failed
```

php -l を通る関数はlint-phpver の2段目で落ちる

構文	CLAUDE.md の表	check-phpver	CI の lint-phpver
match 式 (8.0)	助言。押せば書く	止まる	落ちる(1段目)
?-> nullsafe (8.0)	助言	止まる	落ちる(1段目)
enum (8.1)	助言	止まる	落ちる(1段目)
str_contains() など (8.0)	助言	止まる	落ちる(2段目)
private int \$n (7.4 可)	許可	通る	通る

1段目の php -l は構文だけを見ます。2段目の php ci/undefined_functions.php が、7.4 の PHP に function_exists で関数の有無を聞きます。lint が赤なら test は走りません。

5-1 util.php が switch の形か確かめ、制約節、フックの登録、テストの追加の3回に分けてコミットします。

5-2 変更の同期で push し、GitLab の Merge requests から New merge request で main 向けの MR を出します。

5-3 Pipelines タブでは動き出したことだけ確かめます。結果はプチ演習5 の途中で見に戻ります。

出典: 演習リポジトリ .gitlab-ci.yml の lint-phpver、ci/undefined_functions.php

4つの到達状態を自分の画面で照合

OK 基準

- ☐ 登録前は match が通り、登録後は同じ依頼が差し戻された
確認: Step 1 と Step 3 の画面
- ☐ 差し戻しの後、人が指示を足さずに switch へ書き直された
確認: ソース管理ペインの差分
- ☐ ケースを1件足し、テストランナーが24件すべて通った
確認: 24 cases: 24 passed, 0 failed
- ☐ MR の lint-phpver の結果を、プチ演習5 の途中で見に戻った
確認: MR の Pipelines タブ

つまずき

登録したのにフックが動かない

JSON が壊れると設定ファイルごと無視されます。カンマの付け忘れが大半です。赤い波線を直し、Ctrl+N で新しいセッションを開きます。

FAIL ... (expected 2, got 0)

見本の str_ends_with(がコメントの中だけにあります。コメントは検査の前に除かれます。

fixture not found

ファイル名の綴りが cases.json と合っていません。

パイプラインが pending のまま

同時の push で順番待ちに入っています。異常ではありません。

制約節とフックとテストは三つ一組で置く

持ち帰り物	D2(PowerShell 5.1・ASCII)	smart3pm(bash 正)
PHP 7.4 制約節 (代替の書き方の表)	自社 CLAUDE.md の既存節を Step 0 の表の形に直す。Desktop と CLI は同じファイルを読む	同じ。core.md 側は機械に 移した項目のタグを書き換える
check-phpver フック1本	post_edit_checks の並びに ps1 を置き、matcher を Edit と Write にして登録。ASCII のみを守る	演習リポジトリにある .claude/hooks/check-phpver.sh を既存の check 系 sh の並びに置く
テストケース1件 + 検査用 .php.txt	hooks/tests/cases.json に1件 足し、fixtures に .php.txt を 置く	check-phpver.test.sh を対で 置く。テストの無い検査を 残さない

発展課題

発展1: 「sed で同じ行を書き換えて」と頼み、フックが走らない抜け道を塞ぐ場所を台帳に書きます。

発展2: .ps1 に ASCII 以外が混ざったら差し戻す検査を、同じ PostToolUse に足します。

発展3: 対象の版は .claude/project.json の php.target にあります。版を上げるときはこの値だけ直します。

出典: 演習リポジトリ .claude/hooks/check-phpver.ps1 / <https://github.com/PHPCompatibility/PHPCompatibility>

D2-03

[25min]

指摘絞り込みの実物設定と人間の判断軸

動いている設定で縮める指摘

見送った指摘の正体を4つに分け、REVIEW.md と Code Review Rules の実物で縮め、
人が残る観点を3つに絞ります。

持ち帰り: REVIEW.md 雛形、Code Review Rules 雛形、観点の3列分類表

数えてみると見える見送りの多さ

区分	指摘総数	対応した件数	見送った件数	見送り率
宿題(直近の自分の改修1件)	_____	_____	_____	_____
ハンズオン1 の MR(CI の AI レビュー)	_____	_____	_____	_____
現地(その場で回収)の合計	_____	_____	_____	_____
リモート(チャット提出分)の合計	_____	_____	_____	_____

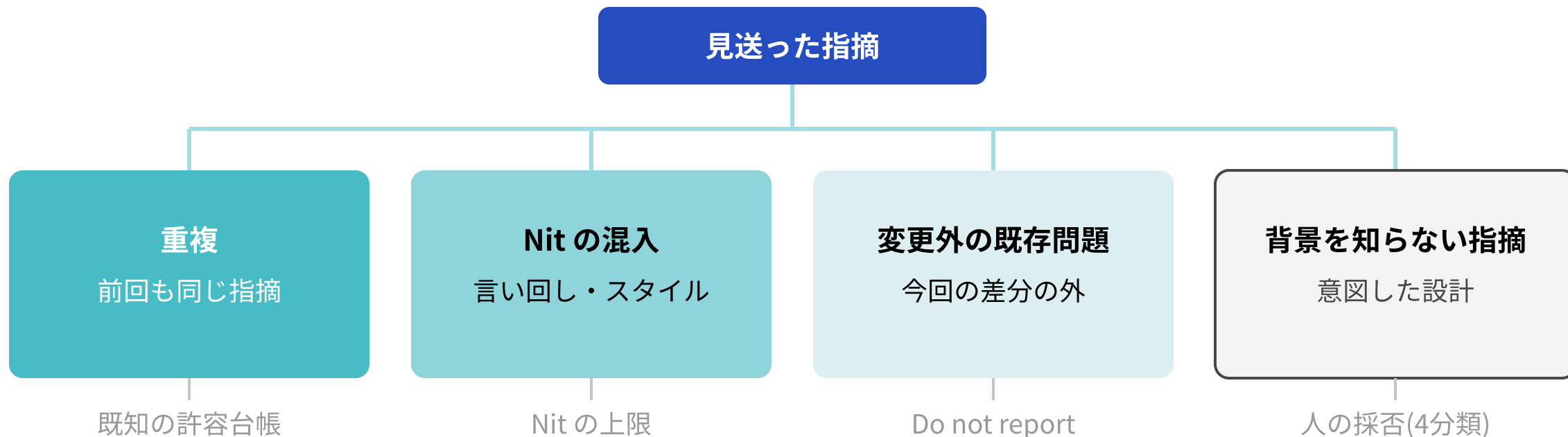
Day1 の宿題で数えていただいた「対応した件数」と「見送った件数」をここで集めます。

回答書にあった「小さな改修ほど指摘への対応や確認作業に時間がかかる」を、まず自分の数字で見ます。

見送り率 = 見送った件数 ÷ 指摘総数。端数は丸めてかまいません。

宿題に手が回らなかった方は、Day1 の MR に付いた AI レビューの指摘件数をそのまま使ってください。

設定で止まる3種類と人が決める1種類



見送った指摘を読み直すと、この4種類のどれかに入ります。

左の3種類は変更の内容と関係なく毎回出ます。設定で止められる種類です。

右端の「背景を知らない指摘」だけは人が採否を決めます。ここに人の時間を寄せます。

小さな改修ほど重くなる正体

行数に比例しない**固定費**としてのレビュー対応

見送りの判断を毎回ゼロからやり直しているので、
5行の修正でも同じ「対応不要」を人が言い直しています。

根拠

見送り4種のうち3種(重複・Nit・変更外)は変更内容と無関係に出るので、改修の大小に関係なく一定量かかります。Day1 で触れたとおり、レビューの重さも変更規模に比例していません。

伸びしろ

D2 にはリリースレビュー側に「既知の許容」を再掲しない仕組みがあり、毎改修で走るコードレビュー側にはまだありません。移植するだけで効きます。

リリース側で**実証済み**の仕組みの移植

d2-release-review(リリース前)

known_issues.md

該当事項は K-00x 参照で再掲しない

patterns.md

指摘に付けるタグの語彙

移植

d2-code-reviewer(毎改修)

台帳なし

語彙なし

ファイルは Markdown なので、PowerShell と bash の差は出ません。
smart3pm 側も同じ形の台帳を code-review の参照先に置きます。

リリースレビューでは「前に許容した指摘は K 番号を指すだけで再掲しない」が動いています。
毎改修のコードレビューに同じ台帳と語彙を持ち込むと、重複と Nit の混入がその場で消えます。
投影しているのは骨格だけです。中身は各自の端末で開いて確認してください。

REVIEW.md 公式例の4節

何を **Important** と呼ぶかを先に決める設定

REVIEW.md

```
# REVIEW.md
## What Important means here
- 挙動を壊す、データが漏れる、ロールバックを妨げる変更だけ
## Cap the nits
- Nit は最大5件、まとめて1コメントにする
## Do not report
- lint が見るもの、生成物、lock ファイル
## Always check
- 新しい API に統合テストがあるか、ログに PII が出ていないか
- クエリがテナント境界の中に収まっているか
```

Anthropic の Code Review が読む REVIEW.md の公式例の骨格です。4つの節しかありません。重要度は Important / Nit / Pre-existing の3段で、CLAUDE.md 違反は Nit 扱いです。基準表を作るのではなく、この4節を自社の言葉で埋めるのが午後の作業です。

出典: <https://code.claude.com/docs/en/code-review>

見送り4種に1節ずつ対応する構造

節	公式例が書くこと	潰す見送り	先方版の候補
What Important means here	挙動破壊・データ漏洩・ロールバック阻害だけ	重要度の混在	PHP 7.4 対象外の構文、SQL 直書きへの変数埋め込み、テスト無しの DB 書き込み
Cap the nits	最大5件を1コメントに	Nit の混入	上限5件、言い回しとコメント文言は含めない
Do not report	lint・生成物・lock	変更外の既存問題	変更外の既存問題、スタイル (phpcs が見る)、K 番号で許容済み
Always check	統合テスト・PII・テナント境界	見逃し	機能テストの無い変更は Important として報告する

045 の4分類のうち3つが、この表の上3行でそのまま止まります。

先方版の候補は講師案です。午後の D2-05 で各自が自社の言葉に直します。

「機能テストの無い変更は Important」を入れるかどうかは、各自の判断に残します。

出典: <https://code.claude.com/docs/en/code-review>

リポジトリの根に置く3ファイル



左のファイル一覧に CLAUDE.md と REVIEW.md が並びます。右は REVIEW.md の中身で、4つの見出しだけの空雛形です。

演習リポジトリには REVIEW.md の空雛形を置いてあります。
ai_review ジョブは REVIEW.md を読みます。午後に自分の言葉で4節を埋めます。

- 1 ルートの REVIEW.md
Claude 側のレビュー指示。
ai_review ジョブが読み込む
- 2 .claude/review.schema.json
レビュー出力の形。手元の
判定と CI が同じ形で読む
- 3 ルートの CLAUDE.md
PHP 7.4 の制約と触らない場所。
レビュー基準は書かない

Always check の書き方

関数名ではなく結果で書くレビュー規則

```
REVIEW.md(Always check 節)
```

```
## Always check
```

```
### 互換性
```

- 外部連携の通知名の改名は破壊的変更として指摘する

```
### データ境界
```

- ログにメールアドレス・ユーザーID・リクエストボディを出す変更を指摘する

見出しで束ね、1行に1つの結果を書きます。ファイル名や関数名を並べると、その名前が変わった瞬間に効かなくなります。

書かないものは3つです。機械的な CI チェック、関数名や正規表現の直接指定、広すぎる指示。

Do not report と同じで、lint が見るものはここに書きません。

出典: <https://code.claude.com/docs/en/code-review>

先入観を切ると出てくる前提のズレ

A\ 実装中のセッション

- 自分が書いた設計の前提を共有している
- 直前の判断を正しいものとして読む
- 指示書との食い違いに気づきにくい
- 会話が長いほど前提が効き続ける

A\ 履歴を切った新しいセッション

- 差分と REVIEW.md だけを読む
- 書いていないことを質問で返す
- 実装側が見落とした前提のズレが出る
- 3本目のレビューとして使う

同じリポジトリを、新しいセッションで先入観なしに開いて読ませる

モデルを変えるのではなく、履歴を切ります。読み手は同じ Claude Code でも、持っている前提が違えば別のレビューになります。実装したセッションでそのまま読ませると、直前の判断を正しいものとして扱うので、前提のズレが出ません。

※ ロゴは Anthropic の商標です。

構造化出力の形

severity と confidence で機械が仕分ける出力

findings.json(出力例)

```
{
  "summary": "...",
  "findings": [{
    "severity": "P0",
    "confidence": 0.92,
    "file": "app/controllers/estimate_ctl.php",
    "line": 31,
    "message": "検索条件の customer_name と date_from を SQL に直接連結している",
    "category": "sql-injection"
  }]
}
```

claude に --json-schema を渡すと、最終応答がこの形の JSON で返ります。

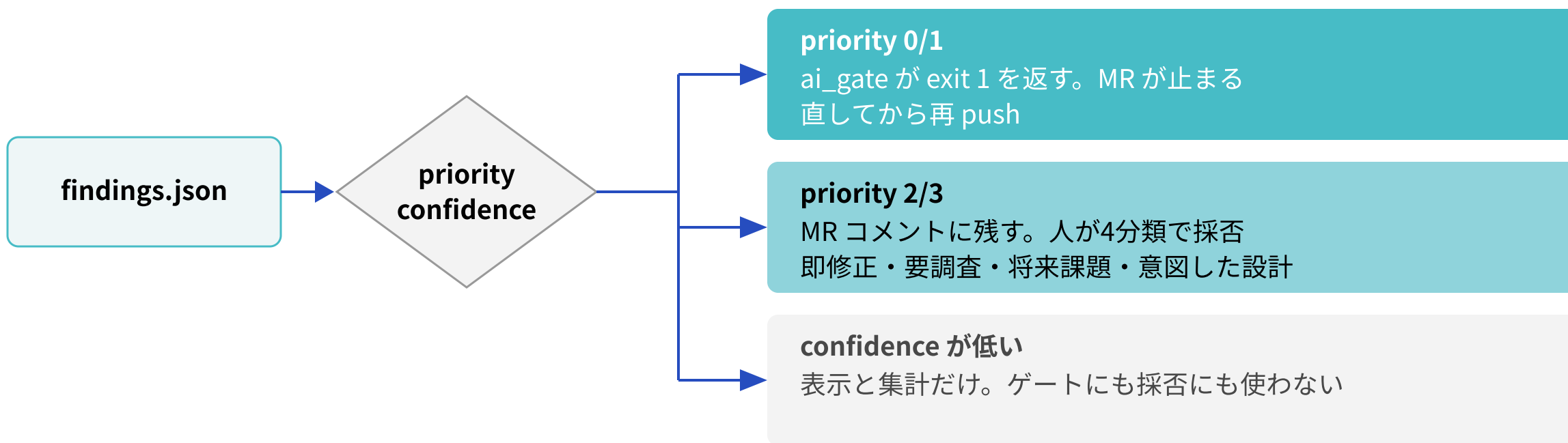
--output-format json の外側に session 情報が付くので、.structured_output を jq で取り出します。

severity は P0～P3、confidence は 0～1 です。この2つがあれば、人が読む前に機械で仕分けられます。

演習では同じ形を review.schema.json で配ります。行番号と confidence の値は形の例です。

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

P0/P1 はゲートで P2 以下は人の採否



しきい値(P1 で止めるか P2 で止めるか)は D2-04 で切り替えて件数差を見ます。

人が読むのは中段だけになります。上段は機械が止め、下段は人の時間を使いません。

AI が列挙し、人が判断するという切り分けの「人が判断する」範囲を、中段に限定します。

止める段階の線引き

指摘の少なさと問題のなさは別物

P0 / P1

MR を止めるのはこの2段だけ

P2/P3 は MR コメントに残して集計へ

止める段階を P0/P1 に絞ると、
止まった MR は必ず読めます。
読む順番が機械側で決まるためです。
絞るのは止める段階だけです。

「指摘が少ない」と「問題がない」は別で、
P2/P3 は消さずに MR コメントへ出し、
件数を集計に使います。
先方の現行は全件列挙です。止める段階を
絞るか、全件出して機械で仕分けるかを
午後に決めます。

人が残る観点は3つまで

観点の源	CI の lint に落とす	AI が列挙し人が採否	人だけが見る
d2-code-reviewer (毎改修)	PHP 7.4 対象外の構文、 phpcs が見るスタイル	指示書との一致、 連動箇所の抜け	業務影響 (画面・帳票・締め処理)
code-review (正確性)	なし	誤った条件分岐、 例外処理の抜け	なし
simplify(簡素化)	なし	重複コード、過剰な抽象化 (Nit 上限5の中で)	なし
security-review	SQL 直書きへの変数 埋め込み(P0 で停止)	秘密情報の露出、 認可の抜け	リスク許容 (直さずに出す判断)
d2-release-review (リリース前)	なし	K 番号で許容済みの 事項は再掲しない	仕様との矛盾 (指示書との食い違い)

先方で動いている5つの観点源を講師が3列に振った案です。人だけが見る列は3つしか埋まりません。Day1 の宿題で各自が作った分類と突き合わせてください。左列に落としたものは REVIEW.md の Do not report に入ります。lint が見るからです。

判断軸は1つ

巻き戻せない変更だけを**人が見る線**

**巻き戻せない変更(DB・外部連携・権限)は人が見る。
それ以外は AI が列挙し、採否は設定と CI に落とす。**

根拠: D1-02 で出した3軸(可逆性・機械検証の可否・影響範囲)のうち、現場で迷わずに判定できるのは可逆性だけです。機械検証の可否と影響範囲は、可逆性が低い変更でまとめて高くなります。

「AI に任せる比重をどこまで寄せてよいか」への答えは、巻き戻せる範囲まで、の一言です。

3つの観点が集まる右下の象限

	機械で検証できる	機械で検証できない
巻き戻せる	PHP 7.4 対象外の構文・phpcs CI の lint に落とす	仕様との矛盾 AI が列挙し、人が判定
巻き戻せない	テストの無い DB 書き込み Always check で必ず報告し、 CI で止める	リスク許容・業務影響 人だけが見る 人が残る3観点の集合先

056 で「人だけが見る」列に残った3つは、検証できない右上と右下に入ります。
左の2象限は人が見なくてよい領域です。午後に CI へ落とす対象はここです。

人が見る量の根拠

見送り率から決める人が見る量

044 で板書した見送り率の平均

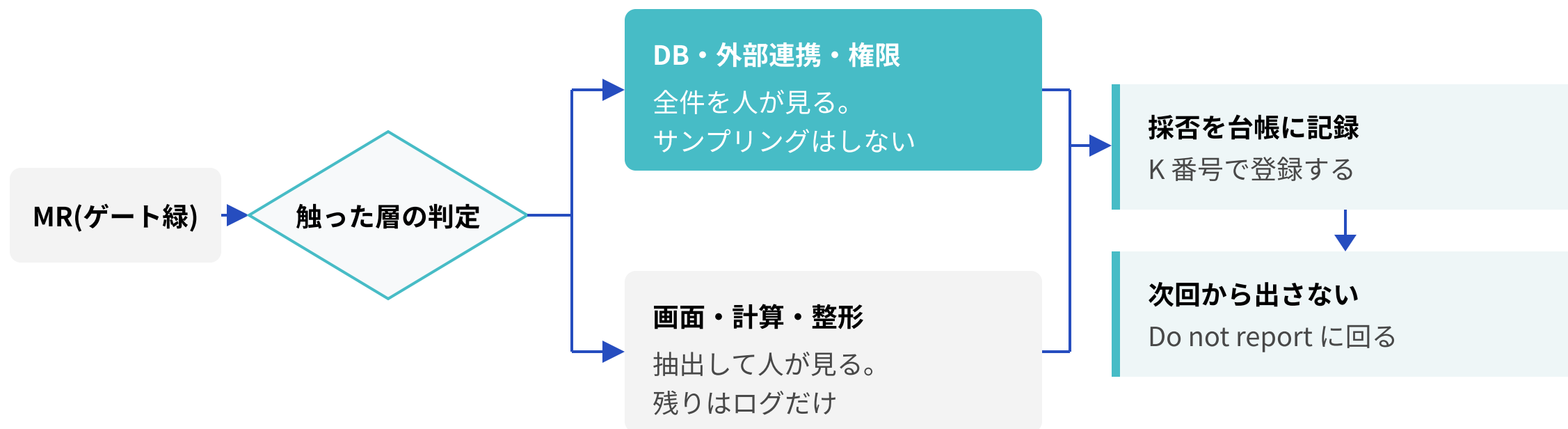
_____ %

全員の数字を足して人数で割った値を書きます。
この1つの数字が、今日のレビュー設定の起点になります。

見送り率が高いほど、レビューが見ている範囲と人が直す範囲がずれています。
人が見る量はこの数字から決めます。P0・P1 だけに絞るか、P2 まで読むかの境目です。
Anthropic も自社の開発フローでリスク加重の抽出を使っています。

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

触った層で決める人の確認の配り方



上の経路は 057 の判断軸そのままです。巻き戻せない層を触った MR は全件見ます。

下の経路で見送った指摘を台帳に書くと、同じ指摘は次回から出ません。人の確認量が回を追うごとに減ります。

抽出の割合は自社の見送り率から決めます。数字はこの研修では置きません。

現行の切り分けへの回答

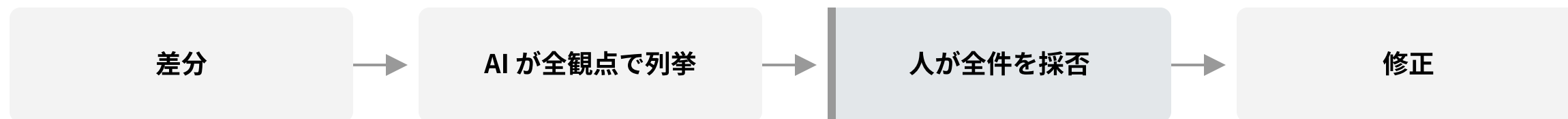
列挙は妥当で**全件の採否判断**が書き換え対象

現行の切り分け	判定	書き換え先
AI が気になった点を列挙する	妥当	列挙の範囲を REVIEW.md の4節で絞る
リスクを許容するかを人が判断する	妥当	対象を3観点(リスク許容・仕様との矛盾・業務影響)に限定する
対応するかどうかを人が全件判断する	書き換え	P0/P1 は ai_gate で止め、Nit は上限5件、P2 以下だけ人が4分類
見送った判断を記録する場所がない	書き換え	既知の許容台帳と指摘タグの語彙をコードレビュー側へ移植

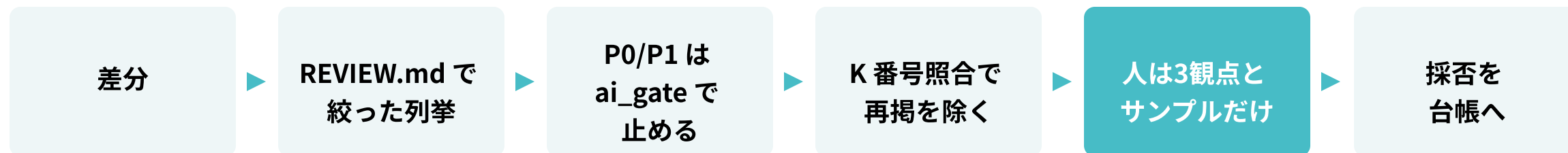
回答書の項目5「この切り分けが妥当かどうか」への講師の答えです。列挙と人のリスク判断は妥当で、変えるのは下2行です。判断する件数と記録の仕方を変えます。

人の判断を最後の3観点だけに寄せる流れ

現行



書き換え後



上段の3番目の箱が、回答書の「小さな改修ほど時間がかかる」の発生源です。

下段では人の箱が1つだけ残り、見る対象も3観点と抽出分に限られます。残りは設定と CI が受け持ちます。

/code-review は REVIEW.md を読まない前提

Claude Desktop の統合ターミナル(PowerShell)

```
git diff main | claude --bare -p `
"$$(Get-Content REVIEW.md -Raw) 対象は stdin の差分" `
--output-format json `
--json-schema .claude\review.schema.json `
| jq '.structured_output'
```

/code-review は CLAUDE.md を読みますが REVIEW.md は読みません。

本文をプロンプトに渡します。--bare は hooks と CLAUDE.md も読まないで、レビュー側だけに使います。研修後に CLI と API キーを用意した環境で、統合ターミナル(右上の >_、または Ctrl とバッククォート)から実行します。当日は講師の画面で見せます。出力は review.schema.json の形になるので、D2-04 のゲートがそのまま受けます。

出典: <https://code.claude.com/docs/en/headless>、<https://code.claude.com/docs/en/desktop>

補足 公式の例はスキーマの中身を直接渡す形です。パスで通らないときは中身を変数に入れて渡します。

午後の起案で先に決める6項目

確認する項目	決める人
☐ Important の定義を「挙動破壊・データ漏洩・ロールバック阻害」の3つに絞れているか	自分
☐ Nit の上限を件数で書いたか(公式例は5件)	自分
☐ Do not report に「変更外の既存問題」「スタイル」「K 番号で許容済み」を入れたか	自分
☐ Always check に「機能テストの無い変更」を入れるか決めたか	チーム
☐ 人だけが見る観点を3つ以内に書き出したか	チーム
☐ 見送った指摘を K 番号で登録する場所を決めたか	チーム

D2-05 の最初の20分で REVIEW.md と Code Review Rules を起案します。その前に、この6項目に自分の答えを持っていてください。4項目目と5項目目は正解がありません。各自の業務で決めます。

本日のこの先との接続

設定からゲートと CI へ順に載せる午後



この座学で決めた内容は、15分後に JSON ゲートとして手元で動き、午後には CI の2ジョブになります。
見送り率の数字は D2-11 で採用率として測り直します。044 の表はそのときのビフォー値です。

持ち帰り物と置き場所

Markdown だけなので2式で同じ形

持ち帰り物	形式	D2 での置き場所	smart3pm での置き場所
REVIEW.md 雛形(4節)	Markdown	リポジトリのルート	リポジトリのルート
既知の許容台帳と指摘タグの語彙	Markdown	d2-code-reviewer の参照先	code-review skill の参照先
観点の3列分類表	Markdown または xlsx	持ち帰り台帳と一緒に	持ち帰り台帳と一緒に
見送り率の記録(044 の数字)	1行	持ち帰り台帳	持ち帰り台帳

このブロックの持ち帰りはすべて Markdown です。PowerShell と bash の二重運用に影響しません。2行目は既知の許容台帳の移植先です。ファイル名は既存のリリースレビュー側に揃えてください。

絞った後に起きる4つの誤解

指摘の少なさの読み違い

ゲートで止まるのは P0/P1 だけです。
P2/P3 は MR コメントに残して集計に
回します。止まらない=問題なし、ではありません。

全部受け入れたときの巻き戻り

レビュー側は背景を知りません。「意図した
設計」は K 番号で台帳に残し、次回から
再掲させません。

/code-review の読み込み範囲

CLAUDE.md は読みます。REVIEW.md は
読みません。本文をプロンプトに渡すか、
ai_review ジョブで読み込みます。

書いたのに効かない設定

REVIEW.md に lint が見るものや広すぎる
指示を書くと、ノイズが増えて Important が
埋もれます。4節の外に書かないでください。

出典: <https://code.claude.com/docs/en/code-review>

D2-04

レビュー出力の JSON ゲート

P0 と P1 が1件でもあれば exit 1 を返す判定スクリプト

プチ演習5 にあたります。ブランチはプチ演習4 の ex/p4-<ユーザー名> のままです。

[15min]

同じ判定が D2-05 で CI の ai_gate ジョブになります

人が読む前に機械が P0/P1 を振り分ける形

今の形 | AI が列挙し人が読む

変更の差分

AI レビュー

文章のコメント

人が全部読んで採否

指摘は文章で返るため、件数も重さも読むまで分かりません。5行の修正でも全件を人が目で振り分けます。

この演習の後

変更の差分

AI レビュー(スキーマ指定)

findings.json

gate.ps1

exit 0 で通す

exit 1 で止める

指摘は severity 付きの JSON で返し、P0/P1 が1件でもあれば gate.ps1 が exit 1 を返して止まります。

止まった MR の P0/P1 だけを人が読みます。しきい値は人が決め、採否の判断は機械に渡しません。

4項目が揃えば Step1 に入れる状態

確認	確認する項目	確認のしかた
☐	プチ演習4 のブランチ ex/p4-<ユーザー名> に乗ったままにいる	VSCode 左下のステータスバーが ex/p4- で始まる名前になっている
☐	判定スクリプトとスキーマがある ci\gate.ps1 と .claude\review.schema.json	VSCode のエクスプローラーで2つとも見える
☐	見本 findings_sample.json の場所が分かっている	配布物の 20_Day2の演習 の下、プチ演習5 のフォルダにある
☐	統合ターミナルを開いた 作業ディレクトリへの移動は要りません	右上の >_ アイコン、または Ctrl とバッククォート

自分で考える [3min] / Step1 見本 [2min] / Step2 既定の閾値 [3min] / Step3 閾値 P2 [2min]

Step4 未知の重大度 [2min] / Step5 自分のレビュー [3min]

API キーは使いません。判定に流すのは、配布の見本と、Claude Desktop に書き出させた自分のレビューの2つです。

止める線と未知の入力の扱いは紙に書いてから実行

問い	自分の答え
1. 自社の MR を止めてよい指摘を、 重大度の言葉で決めます。P0 は動かないか 壊す、P1 は実環境で落ちるか外部入力 が素通し、が schema の定義です。	<hr/> <hr/> <hr/>
2. 判定できない入力を、通すか落とすか 決めます。重大度が知らない値のときと、 findings の配列が無いときの2つです。	<hr/> <hr/> <hr/>
3. 止める線を P2 へ1段下げたとき、 1日あたり何件の MR が止まると困るかを 見積もります。	<hr/> <hr/> <hr/>

所要 [3min] で書きます。問い1 は Step3 の件数と、問い2 は Step4 の結果と突き合わせます。

件数が分かっている見本の findings.json

[2min]

- 1 Windows のエクスプローラーで、配布物一式の 20_Day2の演習 の下にある プチ演習5_レビュー出力のJSONゲート を開きます。
- 2 findings_sample.json を、VSCode のエクスプローラーの dl-training-app の いちばん上(ルート)ヘドラッグしてコピーします。
- 3 VSCode でそのファイルを選んで F2 を押し、名前を findings.json に変えます。

中身は P0 1件、P1 1件、P2 2件です。最後の1件だけ confidence が 0.5 です。

findings.json の形(見本の1件目)

必須キーは6つ

```
{ "summary": "...", "findings": [  
  { "severity": "P0", "confidence": 0.95,  
    "file": "app/controllers/estimate_ctl.php", "line": 23,  
    "message": "...", "category": "sql-injection" }, ... ] }
```

Step5 で Claude Desktop に書き出させる自分のレビューも、同じ形で findings-mine.json に入ります。

P1 以上が1件でもあれば **exit 1** が返る判定

[3min]

統合ターミナルで1行ずつ打ちます

既定は -Path findings.json -Threshold P1

```
powershell -NoProfile -ExecutionPolicy Bypass -File ci\gate.ps1  
$LASTEXITCODE
```

見本 findings_sample.json を渡したときの出力

P0 1件・P1 1件・P2 2件

```
gate: threshold=P1 total=4 blocking=2  
P0 app/controllers/estimate_ctl.php:23 ...  
P1 app/controllers/stock_ctl.php:58 ...  
gate: blocked. fix the findings above or lower the threshold on purpose.  
1
```

1行目を Enter で実行してから2行目を打ちます。2つを1行につなげると、\$LASTEXITCODE が引数として渡り、findings file not found: 1 で落ちます。

配布の ci\gate.ps1 は完成版で、書き足す行はありません。P0 と P1 が無ければ blocking=0 と gate: passed. の2行になり、終了コードは0です。行末の ... は指摘の本文を省いた箇所です。

止める線の1段の差と未知の重大度の扱い

[4min]

手順	実行すること	blocking	確かめること
Step3 [2min]	ci\gate.ps1 -Threshold P2 次の行で \$LASTEXITCODE。閾値を P2 に 下げ、同じ findings.json で回します	2 → 4	増えた2件を読み、自社なら 本当に MR を止めるものが何件 あるかを数えて台帳に書きます
Step4 [2min]	findings.json の P2 の1件の "severity": "P2" を "high" に 書き換えて保存し、Step2 の2行を もう一度打ちます	3	知らない重大度が落とす側に 数えられたのを見たら、 "P2" に戻して保存します

コマンドは頭に powershell -NoProfile -ExecutionPolicy Bypass -File を付けて打ちます。blocking は見本の値です。
gate.ps1 と gate.sh は、知らない severity を落とす側に寄せ、findings の配列が無い入力は判定せずに落とします。
判定できないものを通す作りに変えると、ゲートが静かに無効になります。

自分の差分のレビューも同じ判定に

[3min]

1 Claude Desktop の入力欄にそのまま送ります

Write の確認が出たら承認。1分前後

main との差分をレビューしてください。判定の基準は REVIEW.md、出力の形は .claude/review.schema.json に従い、結果をリポジトリのルートに findings-mine.json として書き出してください。説明は本文に書かず、ファイルだけを作ってください。

2 統合ターミナルで判定にかけます

```
powershell -NoProfile -ExecutionPolicy Bypass -File ci\gate.ps1 -Path findings-mine.json
$LASTEXITCODE
```

キーの欠けや形の崩れがあれば、終了コード 1 で落ちます。標準エラーの1行で、何が足りないかを読みます。findings.json と findings-mine.json はコミットしません。

OK 基準

- | | |
|---|--|
| ☐ 見本で P1 と P2 の blocking が 2 と 4 に変わった | ☐ 知らない重大度が落とす側に数えられた |
| ☐ 自分のレビューを同じ判定スクリプトに流した | ☐ 止める線を自分の言葉で台帳に1行書いた |

どれも判定の手前で落ちる3つの詰まり

詰まり	出た症状	原因	直し先
JSON が読めない	gate: cannot parse JSON で落ちる	PowerShell 5.1 の > は UTF-16 で書き出す	Out-File -Encoding utf8 で作り直す
findings の配列が無い	gate: no findings array in ... で落ちる	一番外側のキー名が変わったか、別の形で包まれている	summary と findings の 2キーの形で書き出し直す
2行をつなげた	gate: findings file not found: 1 で落ちる	\$LASTEXITCODE が引数として渡った	1行目を Enter で実行し、\$LASTEXITCODE は次の行で単独で打つ

引用符が全角になっていたら半角に直します。チャットやメールを経由すると自動で変わることがあります。

schema と gate の組を自社ハーネスへ

持ち帰り物	D2 PowerShell 5.1 正	smart3pm bash 正
review.schema.json	レビューの出力形として1か所に置き、手元と CI の両方から同じファイルを参照します	同じファイルを共有します。 カテゴリの語彙を増やすときは、指摘の集計側と一緒に直します
ゲート	ci/gate.ps1 を正にします。 編集後に自動で走らせず、MR を出す前に人が手で回す位置に置きます	ci/gate.sh を使います。 引数の並びを gate.ps1 と揃え、2言語で判定を割りません

持ち帰り台帳では schema と gate を合わせて1点に数えます(17点の 12)。

採用した閾値とその理由、P1・P2 の件数も同じ行に残します。

D2-05 では同じ判定が .gitlab-ci.yml の ai_gate ジョブ(ci/ai_gate.sh)として走ります。

確信度の下限と止めなかった指摘の扱い

発展1 確信度の下限

ci\gate.ps1 の param に
-MinConfidence(既定 0.0)の
ような引数を足し、下限に
満たない指摘を数える前に
除いて、除いた件数を1行
出します。0.8 にすると、見本の
確信度 0.5 の1件が外れます。

発展2 届かなかった指摘

閾値に届かなかった P2 以下を、
MR のコメント1本にまとめて
残す処理を考えます。
指摘のたびにコメントを
増やすと、小さな改修ほど
読む量が増えます。

考察 重大度の揺れ

同じ差分を、履歴を切った
新しいセッションでもう一度
レビューさせます。
見るのは、同じ差分で重大度が
揺れていないかの1点です。
揺れていたら、揃えるのは
基準の文面です。

発展1 は CI の ci/ai_gate.sh が環境変数 GATE_MIN_CONFIDENCE(既定 0.0)で同じことをしています。
どれも余裕があれば試す課題です。本編の Step 1~5 には入りません。

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

列挙するレビューから止めるレビューへ

題材は Issue #2 と #3 です。前半で REVIEW.md と観点別レビューアー3本を自分の言葉で起案し、後半で .gitlab-ci.yml のゲートを本物にします。期間検索のテストを人が先に決めて足し、テストの有無で壊れ方が見えるかを各自で確かめます。

[80min]

ワーク

093 と 102 で履歴を切ったセッションを3本目のレビューに使用します

人が決める15分と機械に載せる65分

自分で考えると Step 1 は AI を起動しません。紙と REVIEW.md だけです。Step 2 と Step 3 は同じコード (Issue #2・#3 の欠陥が残った今のコード) を使い続けます。

自分で考える + Step 1 [5min] + [10min]

- 紙に書く4つの判断
- 手順1～6 REVIEW.md
- 手順7～9 既知の許容の台帳

ブランチは ex/h2-<ユーザー名> です。
Step 1～3 の成果物を同じブランチに
載せ、MR は1本で出します。

Step 2 観点別レビュー [30min]

- 手順1～2 3本の見るもの
- 手順3～5 並列で当てる
- 手順6～7 重なりと検出率
- 手順8～9 規模で選ぶ skill

Step 3 ゲートとテスト [28min] + 補足 [7min]

- 手順1～3 ゲートを有効に
- 手順4～8 テスト先出しと MR
- 手順9 赤いテストを見る
- 手順10～13 実装して緑まで
- 補足 テスト無しと有り

台帳に書く持ち帰り: 10 REVIEW.md と規約、11 観点別レビュー3本

記録: 検出率とノイズ率

Project

dl-training-app

Pinned

Members

Work items 5

Branches

Merge requests 2

Pipelines

More features

Manage

Plan

Work items 5

Issue boards

Milestones

Wiki

Code

Build

Secure

Deploy

Operate

Monitor

Analyze

Settings

Help

Collapse sidebar

dlive-training / dl-training-app / Issue Boards

Development

Search

Open 5

Closed

見積一覧の顧客名検索の条件組み立て

演習::ハンズオン1 種別::不具合 領域::見積

#1 Day1

在庫の引当で半端な状態が残る

演習::ハンズオン2 種別::不具合 領域::在庫

#2 Day2

見積一覧の期間検索と CSV 出力

演習::ハンズオン2 種別::機能追加 領域::見積

#3 Day2

見積状態コードの対応表の一本化

実行::夜間 演習::ハンズオン3 種別::リファクタリング 領域::共通

#4 Day2

見積登録の入力チェックの追加

演習::予備 種別::機能追加 領域::見積

#5

dl-training-app#2

在庫の引当で半端な状態が残る

Open Issue created 2 weeks ago by 安田 光喜

背景

受注が決まると、営業事務が在庫一覧から機器を1台ずつ受注に引き当てます。9月の棚卸で、引当済み(状態2)になっているのに受注と紐付いていない在庫が数台見つかりました。画面からは引当済みに見えるので次の受注には出てこず、現物は倉庫にあります。保守担当が手で状態を1に戻して回しています。

同じ在庫を2人が同時に引き当てようとした直後に起きる、という報告が来ています。tr_bind_order_stock には UNIQUE (stock_id) があるので紐付け自体は2本目で落ちますが、落ちる前の在庫の状態変更が残ります。

現状

app/controllers/stock_ctl.php の assign() が、4つの DB 操作を順番に実行しています。

- SELECT status_cd FROM tr_stock WHERE stock_id = ? で状態を確認
- UPDATE tr_stock SET status_cd = 2 で引当済みにする
- INSERT INTO tr_bind_order_stock で受注と紐付ける
- UPDATE tr_order SET status_cd = 2 で受注を引当済みにする

- トランザクションがないため、3 で例外が出ても 2 は残る
- 1 の SELECT に行ロックがないため、確認と 2 のあいだに他の人が同じ在庫を引き当てられる
- assign() の最後が redirect() で、redirect() は header() のあとに exit します。このままではテストから呼べません

期待する振る舞い

- 4つの DB 操作を1つのトランザクションにまとめます。どれか1つでも失敗したら、全部元に戻します
- 状態の確認は SELECT ... FOR UPDATE で対象行をロックしてから行います。確認から更新までの間に、他の人が同じ在庫を引き当てられない状態にします
- 引き当てられないとき(在庫が存在しない、状態が1でない、受注が存在しない)は、DB を1つも変えずに在庫一覧へ戻します
- 引当の判断と DB 操作を assign_stock(\$stock_id, \$order_id) という public メソッドに分け、引き当

Read more

The screenshot displays the Gitee web interface. On the left is a sidebar with navigation links for Project, Pinned, Members, Work items, Branches, Merge requests, Pipelines, and More features. The main content area is divided into two panels. The left panel shows an 'Issue Boards' view for the 'dl-training-app' project, with a 'Development' board selected. It lists five open issues, including '見積一覧の顧客名検索の条件組み立て' (Issue #1), '在庫の引当で半端な状態が残る' (Issue #2), '見積一覧の期間検索と CSV 出力' (Issue #3), '見積状態コードの対応表の一本化' (Issue #4), and '見積登録の入力チェックの追加' (Issue #5). The right panel shows the details of Issue #3, titled '見積一覧の期間検索と CSV 出力'. It includes a description of the problem, the background context, the current status, and a list of expected improvements.

Project

- dl-training-app
- Pinned
- Members
- Work items
- Branches
- Merge requests
- Pipelines
- More features
- Manage
- Plan
- Work items
- Milestones
- Wiki
- Code
- Build
- Secure
- Deploy
- Operate
- Monitor
- Analyze
- Settings
- Help

Issue Boards

dl-training / dl-training-app / Issue Boards

Development

Search

Open

- 見積一覧の顧客名検索の条件組み立て
演習::ハンズオン1 種別::不具合 領域::見積
#1 Day1
- 在庫の引当で半端な状態が残る
演習::ハンズオン2 種別::不具合 領域::在庫
#2 Day2
- 見積一覧の期間検索と CSV 出力
演習::ハンズオン2 種別::機能追加 領域::見積
#3 Day2
- 見積状態コードの対応表の一本化
実行::夜間 演習::ハンズオン3 種別::リファクタリング
領域::共通
#4 Day2
- 見積登録の入力チェックの追加
演習::予備 種別::機能追加 領域::見積
#5

Closed

見積一覧の期間検索と CSV 出力

Open Issue created 2 weeks ago by 安田 光喜

背景

経理が月次で、当月に出した見積の一覧を Excel に貼って受注実績と突き合わせています。今は画面を選択してコピーしているため、行がずれる、金額の桁区切りが文字列として入る、といった手戻りが出ています。営業事務から「検索した結果をそのまま CSV で落としたい」という要望が来ました。

期間の絞り込みは、顧客名と同じ組み立て方をしています。CSV でも同じ条件を使うので、条件の作り方を先に決めてから CSV を足します。

現状

app/controllers/estimate_ctl.php の index() が検索条件を組み立てています。

- date_from と date_to は入力値をそのまま SQL に連結している
- 日付として成立しない文字列でも、そのまま SQL へ渡している
- 条件の組み立ては index() の中にしかなく、他から使えない
- CSV を出す入口はまだ無い

顧客名の連結は Issue #1 で直っている前提です。まだであれば、この Issue では顧客名の行に触らないでください。

期待する振る舞い

- date_from と date_to をプレースホルダで渡します。範囲は今までどおり両端を含みます
- YYYY-MM-DD の形に合わない値は、条件に使いません(未入力と同じ扱い)。検索欄には入力した文字をそのまま残します
- 条件の組み立てを index() の外に出し、同じファイル内の private メソッド1つにまとめます。WHERE 句の文字列とパラメータ配列の2つを返します
- csv() を足します。?c=estimate&m=csv で、一覧と同じ条件・同じ並び順の結果を CSV で返します
 - 1行目は見出し 見積番号,顧客名,見積日,状態,担当,金額
 - 金額は桁区切りなしの数値。状態は表示名(estimate_status_name() の値)

Read more

欠陥が残った**今のコード**を読むところから始める演習

Issue	対象ファイル	今のコードの欠陥	既存テスト	使う場面
#2 在庫の引当で 半端な状態が残る	app/controllers/ stock_ctl.php assign()	4つの DB 操作が トランザクションの外 確認の SELECT に 行ロックが無い	無し 引当は未テスト	Step 2 の レビュー対象 発展課題
#3 見積一覧の期間 検索と CSV 出力	app/controllers/ estimate_ctl.php index()	date_from と date_to を SQL へ連結 日付の形を見ない	5本あり 期間は未テスト	Step 2 の レビュー対象 Step 3 で実装

どちらも今のコードに「現状」節どおりの欠陥が残っていて、レビューで見つかるべき問題です。
Step 2 は2本とも3本のレビュアーに当てます。Step 3 で実装して MR を出すのは #3 です。

原文 受入条件は GitLab の Issue #2・#3 で読みます。演習リポジトリの `_issues/` にも同じ本文があります

AI に相談する前の4つの判断

(1) 機械判定できる観点

lint-phpdev と lint-phpver が見ているものを含め、機械で判定できる観点を3つ挙げます。

(2) 見送った指摘

直近の改修で見送った指摘を3件思い出し、見送った理由を1行ずつ書きます。

(3) 機能テストの無い変更

「機能テストの無い変更は Important」を自社の基準に入れるかを決めます。

(4) 止める重大度の下限

合流を止める重大度の下限を、P0 だけか P1 までかで決めます。

紙かメモに書きます。AI には渡しません。隣と相談もしません。

(1) は Do not report、(2) は既知の許容の台帳、(3) は Always check、(4) は Step 3 の GATE_LEVEL に入ります。

4見出しに 084 の答えを**移すだけ**の起案

REVIEW.md(リポジトリのルート)

```
# レビュー基準
## Important の定義
- 外部入力 が SQL の条件へ文字列連結で入る変更
- 複数の表を続けて書き換える処理がトランザクションの外にある
## Nit の上限
- 5件まで。まとめて1コメントにする
## Do not report
- lint-phpdev と lint-phpver が見る範囲(8.x の構文、7.4 に無い関数)
- 既知の許容は .claude/known_issues.md の K 番号で済ませる
## Always check
- 機能テストの無い変更は Important として報告する
```

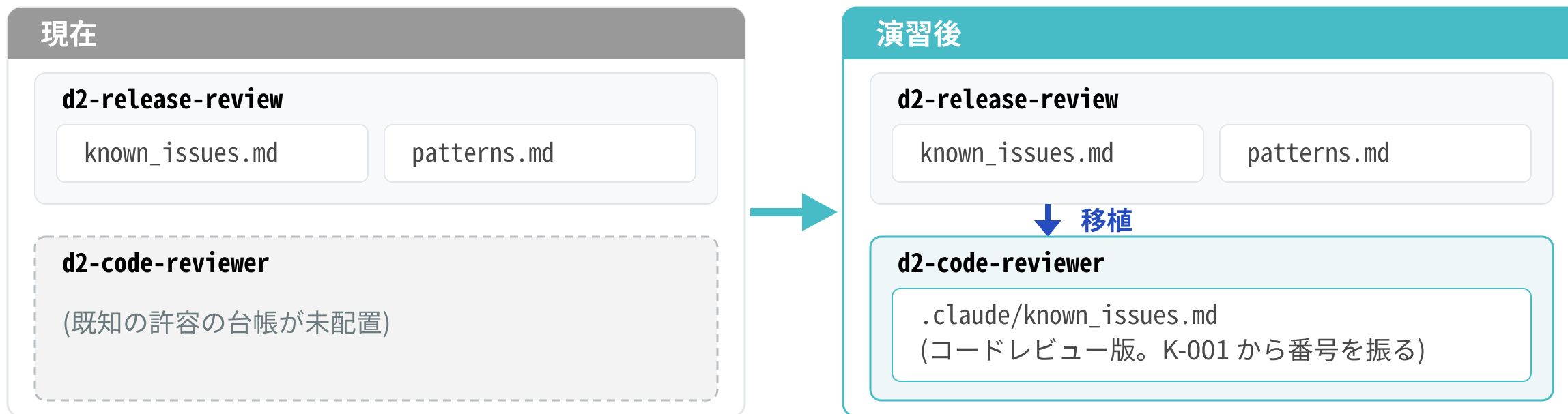
手順1 で ex/p4 から ex/h2-<ユーザー名> を切ります。配布の REVIEW.md は4見出しだけです。

上は講師の例です。自分で考えるの (1)～(3) に差し替え、lint が見る範囲は Do not report に置きます。

埋める順は Do not report、Nit の上限、Important の定義、Always check。CI の ai_review もこの1本を読みます。

出典: <https://code.claude.com/docs/en/code-review>

リリースレビューで動いている仕組みをコードレビューへ



CI の ai_review が読むのは REVIEW.md だけです。台帳を CI に効かせる方法を1つ選び、MR の説明に書きます

既知の許容を K 番号の参照で済ませる台帳は、リリースレビュー側で動いています。同じ形の1ファイルをコードレビュー側に置き、自分で考えるの (2) の3件を K-001 から書きます。1件は場所・内容・見送った理由の3行です。REVIEW.md の Do not report から台帳を1行で参照し、REVIEW.md と台帳の2つだけを1コミットにします。

観点1つに**モデル1つ**を割る3本の分業

レビューアー	見る観点	model	effort	tools	重大度の付け方
security-reviewer	外部入力が SQL と画面出力に届く経路	opus	xhigh	Read, Grep, Glob	外から入力を与えて届くと確認できたら P0、不明は 0.5 以下
spec-reviewer	Issue と指示書の文言と差分の一致	sonnet	high	Read, Grep, Glob	完了条件が未達なら P1、一部欠けは P2 指示外の変更は P3
test-gap-reviewer	変更した処理にテストがあるか	haiku	非対応	Read, Grep, Glob	DB 書き換えと金額計算は P1、表示だけは P2

1本に4観点を持たせず、観点ごとに別のコンテキストを立てます。Anthropic の自社開発でも、各レビューエージェントは狭い一点に絞られる設計です。3本とも .claude/review.schema.json の形で返します。tools を Read, Grep, Glob に限るので、3本ともファイルを書き換えられません。

出典: <https://claude.com/blog/how-anthropic-secures-its-ai-native-software-development-lifecycle>,
<https://code.claude.com/docs/en/sub-agents>

外部入力¹の経路だけを追う1観点のレビュアー

```
.claude/agents/security-reviewer.md
```

```
---
```

```
name: security-reviewer
```

```
description: 外部入力1が SQL と画面出力に届く経路だけを見ます。他の観点は見ません。
```

```
tools: Read, Grep, Glob
```

```
model: opus
```

```
effort: xhigh
```

```
---
```

あなたは外部入力¹の経路だけを見るレビュアーです。観点は1つです。他に気づいたことがあっても書きません。

手順1で .claude\agents\ の3本を VSCode で開き、frontmatter の model と effort が3本とも違うことを見ます。

手順2で「見るもの」の節を Step 1 の REVIEW.md に合わせます。この1本は \$_GET と \$_POST から

db_select 系の第1引数と app/views/ の echo に届く経路を追います。Do not report に入れたものは消します。

自社の DB 関数名への書き換えは持ち帰ってから行います。frontmatter の model と effort は変えません。

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

同じ形で観点とモデルだけ違う2本

手順2 spec-reviewer

配布済みの spec-reviewer.md を開き、「見るもの」の節を REVIEW.md と見比べ、Do not report に入れたものが残っていたら消します。

期待: model: sonnet と effort: high のまま

手順2 test-gap-reviewer

test-gap-reviewer.md も同じように見比べます。「観点は1つです」で始まる冒頭の段落は消さずに残します。

期待: model: haiku のみで effort の行が無い

3本の frontmatter の差

name	model	effort
security-reviewer	opus	xhigh
spec-reviewer	sonnet	high
test-gap-reviewer	haiku	なし

3本とも返す形は同じ JSON です。違うのは観点の節とモデルだけです。Haiku 4.5 は effort を受け付けません。配布ファイルには effort の行を書いていません。

出典: <https://code.claude.com/docs/en/sub-agents>, <https://code.claude.com/docs/en/model-config>

誘導は **skill** に置き保証は **CI** に置く

```
.claude/skills/review-route/SKILL.md
```

```
---
```

```
name: review-route
```

```
description: 変更の規模でレビュアーを選びます。
```

```
---
```

```
(この下に、選ぶ手順を自分の言葉で書く)
```

本文に書く3つの分岐

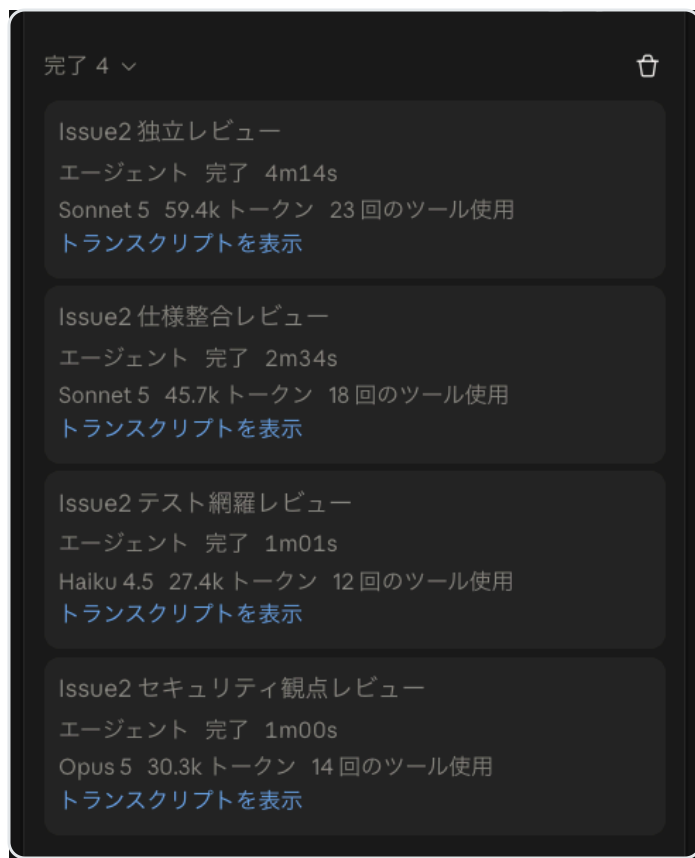
- 変更が50行未満なら review-light
- 50行以上、または DB・権限・外部連携に触る変更は review-full
- 観点を分けたいときは security・spec・test-gap の3本を並列

この skill は誘導です。読まなければ従いません。守らせたいものは Step 3 の CI ゲートに置きます。

手順9で .claude/agents と .claude/skills/review-route をコミットします。findings-*.json は入れません。

出典: <https://code.claude.com/docs/en/skills>

モデル名の違う3本が並ぶ状態が正しい画面



右上の：から
バックグラウンドタスク
を開いた画面です。

1本ごとに エージェント名、
モデル、トークン数、
ツール使用回数が出ます。

サブエージェントの effort は
このペインに出ないので、
087 で定義した値を見ます。

1 3本を名指しで起動

ハンズオンガイド Step 2 の
手順4 の文を入力欄に貼り、
3本を並列で起動します。

2 待ち時間

終わるまで数分かかります。
待つ間に 093 の新しい
セッションを回します。

3 3本の読み方

ペインに3本が並び、モデル名が
3種類に分かれています。
結果は findings-*.json の
3ファイルでルートに残ります。

画面: 講師環境のバックグラウンドタスク(review-full も並べた4本の例)

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

同じコードで3本が違う場所を指す記録

指摘の場所(file:line)	security	spec	test-gap	ai_review	履歴を切った版	重なり
1						
2						
3						
4						
5						
件数合計						

印は ○=指摘あり、P0～P3=その重大度。履歴を切った版は 093、ai_review は Step 3 手順11 の後に埋めます。
手順7 で 101 の式に入れます。仕込み欠陥の件数は、Issue #2・#3 の「現状」節から自分で数えます。

5行埋まらなくてよいです。security だけが指した行と、3本が重なった行を1つずつ見つけます。

重なった行は機械判定に落とせる候補、1本だけが指した行はその観点を残す根拠になります。

実装の経緯を知らない3本目の読み手

Claude Desktop(新しいセッションに貼るプロンプト)

REVIEW.md の基準で stock_ctl.php と estimate_ctl.php を読んでください。
このコードの実装には関わっていない前提で読みます。
根拠にしてよいのは _issues/ の Issue #2・#3 とコードだけです。
結果は .claude\review.schema.json の形で返してください。

- Ctrl+N で新しいセッションを開き、同じプロジェクトフォルダを選びます。
実装のやりとりは貼りません。Ctrl+Tab で実装中のセッションへ戻れます。
- 実装したセッションは、自分が途中で飲み込んだ前提を疑いません。窓を分けると、その前提が指摘として言葉になります。返った findings を 092 の「履歴を切った版」列へ転記します。
- セッションを並べるのは Claude Desktop でできます。エージェントチームの編成はできません。

進め方 091 の並列レビューの待ち時間に回します。3本が先に終わった方は 092 へ進みます

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

巻き戻せない変更だけが人の欄に残る仕分け

分類と軸	可逆性	機械検証できる	影響範囲	判定
即修正	<hr/>	ここは CI に落とす	<hr/>	☐ AI が直す ☐ 人が決める
要調査	<hr/>	<hr/>	<hr/>	☐ AI が直す ☐ 人が決める
将来課題	<hr/>	<hr/>	<hr/>	☐ AI が直す ☐ 人が決める
意図した設計	<hr/>	<hr/>	<hr/>	☐ AI が直す ☐ 人が決める

- 092 の指摘を1件ずつ置きます。「即修正」かつ「機械検証できる」に入ったものは、Step 3 でゲートの条件に入れます。
- DB・権限・外部連携に触る指摘は可逆性が低いので、分類に関係なく右端を「人が決める」にします。
- 指摘を全部受け入れると、意図した設計まで巻き戻ります。行が空の人は見送る判断をしていません。
戻すときは D1-06 の変更範囲の事前制限と部分ロールバックで、当たった1件分だけを戻します。

消すのは **allow_failure** の1行だけ

.gitlab-ci.yml(ai_gate ジョブ、書き換え後)

```
ai_gate:
  stage: gate
  image: alpine:3.20
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  needs:
    - ai_review
  before_script:
    - apk add --no-cache jq bash > /dev/null
  script:
    - bash ci/ai_gate.sh findings.json
```

- 手順1: VSCode の左下で、ブランチが ex/h2-<ユーザー名> のままであることを見ます。
- 手順2: ai_gate の最後にある allow_failure: true の1行を消します。他の行のインデントは変えません。
- 手順3: 先頭の variables の GATE_LEVEL: "P1" を、一度だけ "P2" にします。手順13で "P1" に戻します。
- ai_review が findings.json を成果物に出し、ai_gate がそれを数えて exit 1 を返します。

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

実装より先にテスト1本を送る順番

手順4～5

手順4: Issue #3 の受入条件3を読み、date_from が条件に使われないとき一覧に何件出るかを紙に書きます。
手順5: Claude Desktop にテストを1本だけ頼みます。依頼文はハンズオンガイドの手順5 をそのまま貼ります。
足す先は tests/phpunit/EstimateSearchTest.php です。実装側のファイルは変えさせません。
期待される画面: 足したテストが既存の5本と同じ書き方で並び、setUp() に :void が無い

手順6～7

.gitlab-ci.yml だけで1コミット、テストだけで別の1コミットにして、変更の同期を押します。
実装のファイルはまだ変えません。テストが赤くなることを先に見るためです

手順8

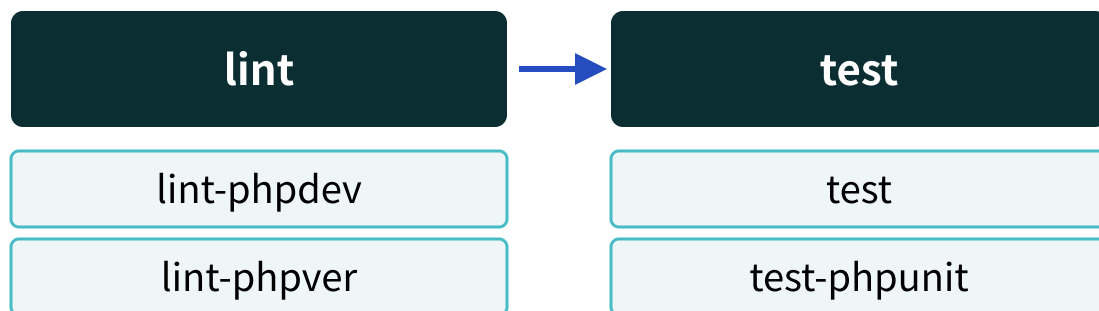
GitLab の画面上部に出る Create merge request を押します。出ていなければ Code → Merge requests
→ New merge request で、元を自分のブランチ、先を main にします。
期待される画面: Merge requests の一覧に MR が1本増える

赤くならないテストは、通っても何も保証しません。CI の履歴に残る赤が、今のコードを見ていた証拠です。

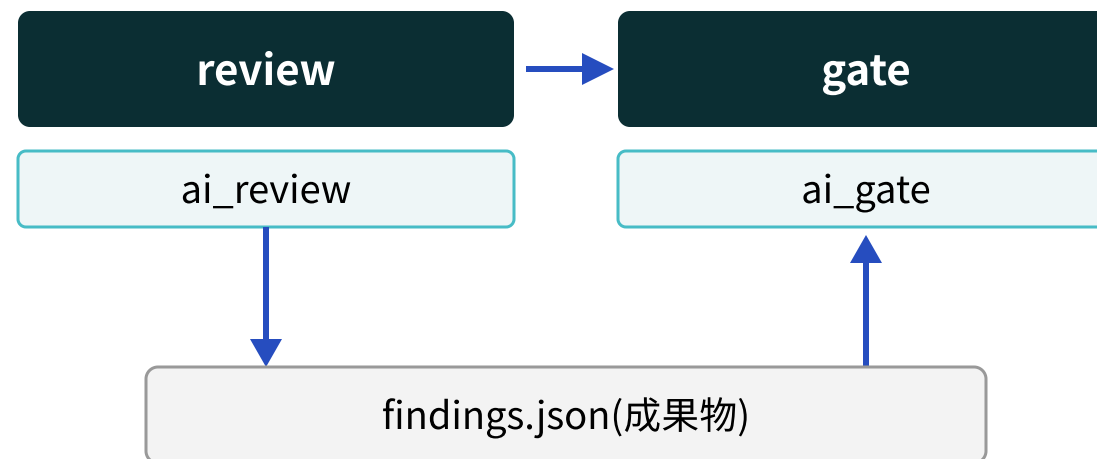
出典: https://docs.gitlab.com/user/project/merge_requests/creating_merge_requests/

ブランチ用と MR 用の2本のパイプライン

ブランチ用(push で走る。branch のラベル)



MR 用(merge request のラベル)



赤のとき直す場所

7.4 に無い構文と関数
D2-01 の一覧を見る

手順9 の test-phpunit の赤は
実装前なので期待どおり

API の呼び出しが
応答の形。ログを読む

指摘を直すか
採否の理由を MR に書く

手順9: branch の行で test-phpunit だけが赤なら正しい状態です。落ちた出力を MR の説明の「テスト」欄に貼ります。

ai_gate が赤なら MR 用のパイプラインが赤です。Pipelines must succeed を有効にしたプロジェクトでは Merge が押せなくなります。研修のプロジェクトでは押せますが、押しません。

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

MR に残る1往復が合格の証拠

確認	すること	見る場所・書く場所
☐	手順10: 受入条件1と、受入条件2 のうち private メソッドの切り出しまでを実装	戻り値の形を紙に書いてから Claude Desktop に渡す
☐	手順11: コミットして変更の同期	branch の行で test-phpunit が緑、MR の行で ai_gate が赤
☐	手順12: 落ちた指摘を即修正・要調査・将来課題・意図した設計の4つに仕分け	MR の説明の「人が判断した採否」の表に理由と一緒に書く
☐	手順12: 即修正を直し、意図した設計は Do not report に入れるかを考える	直さない指摘にも理由を書く
☐	手順13: GATE_LEVEL を "P1" に戻し、修正と合わせてコミットして同期	ブランチ用で lint とテスト、MR 用で ai_gate が緑

4件出たら4件直す、では Day1 の課題がそのまま戻ります。直さない判断にも理由を書きます。

同じ理由が繰り返し出るなら、REVIEW.md の Do not report か .claude/known_issues.md に移す合図です。

実装の前に赤を見たテストだけが残す証拠

手順5 で送る依頼文(ハンズオンガイドと同じ)

```
tests/phpunit/EstimateSearchTest.php に1本足してください。  
date_from に '2026-09-02' AND '1'='1' を入れたとき、一覧が15件(全件)出ることを確認します。  
修正前のコードでは3件になって落ちます。  
既存の5本と同じ書き方にそろえてください  
(capture_view と count_list_rows を使う、$_GET は setUp で初期化済み)。  
実装側のファイルは、この手順では変更しないでください。
```

手順9 で見る赤

```
EstimateSearchTest::testIgnoresMalformedDateFrom  
Failed asserting that 3 matches expected 15.  
FAILURES!
```

出す順番で残るもの

先に出す: CI の履歴に赤が1回残り、
今のコードを見ていた証拠になる
後から出す: 最初から緑で、証拠が取れない

関数名と件数は環境で変わります。見るのは、実装前に test-phpunit だけが赤だったことです。

CI は PHP 7.4 と PHPUnit 7.5.20 で走るので、setUp() に :void を付けません。

同じ壊し方で赤が出るのはテストがある側だけ



手順1: 手順10 で入れた日付の形の判定を1行だけ外したコミットを作り、同期して右の状態を見ます。
左はテストが無かった場合で、同じ壊れ方が合流まで届きます。自分で考えるの (3) と突き合わせます。

既知の欠陥で測る**当たりと取りこぼし**

検出率 = 当てた件数 ÷ 仕込み欠陥の件数

取りこぼしがどれだけあるか

ノイズ率 = 仕込み欠陥以外の指摘 ÷ 指摘の総数

読む時間の無駄がどれだけあるか

仕込み欠陥の件数

現状節から数える

当てた件数

当てた指摘の数

指摘の総数

findings の合計

検出率

当てた件数 ÷ 欠陥数

ノイズ率

欠陥以外 ÷ 指摘総数

仕込み欠陥は、Issue #2・#3 の「現状」節に挙がっている欠陥です。件数は Step 2 の手順7 で自分で数えます。

指摘の総数は findings-*.json の findings 配列の件数です。何を分母に入れたかも一緒に書きます。

指摘を絞って動くのはノイズ率です。検出率は分母が仕込み欠陥の件数なので、絞っても上がりません。

記入先は計測表_Day1_Day2.md の「Day2 検出率」です。D2-11 で効果測定のアフター値の初回に置きます。

Day1 の AI レビュー指摘件数: _____

毎回まっさらな状態で同じ基準を読む **ai_review**

.gitlab-ci.yml(ai_review ジョブ、配布済み)

```
ai_review:
  stage: review
  image: alpine:3.20
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  before_script:
    - apk add --no-cache curl jq git bash > /dev/null
  script:
    - bash ci/ai_review.sh
  artifacts:
    { when: always, paths: [findings.json], expire_in: 1 week }
```

- このジョブは毎回まっさらなコンテナで動きます。093 の新しいセッションと同じ読み方が CI 側で自動になります。
- ci/ai_review.sh は claude CLI を使わず、Messages API を curl で呼びます。
基準に REVIEW.md を、出力の形に .claude/review.schema.json を渡します。

出典: <https://code.clause.com/docs/en/gitlab-ci-cd>, <https://platform.clause.com/docs/en/api/messages>

注意

CI 変数 ANTHROPIC_API_KEY が無いと、ジョブは緑のまま MR にコメントが付きません

自分で判定できる5つの状態

確認	合格の状態
☐	REVIEW.md の4見出しが自分の言葉で埋まり、 .claude/known_issues.md に K-001 から3件ある
☐	.claude\agents\ に security-reviewer / spec-reviewer / test-gap-reviewer があり、バックグラウンドタスクでモデルが違う
☐	ai_gate が1回赤になって採否と理由が MR に残り、 最後はブランチ用と MR 用のパイプラインが緑
☐	EstimateSearchTest.php に足した1本が、実装前の パイプラインで赤だった履歴が MR に残っている
☐	検出率とノイズ率が計測表の「Day2 検出率」に入った

3番目だけが必須です。1・2・4・5 が埋まらなかった人は、
埋まらなかった番号を持ち帰り台帳の D2-05 行に書き、
D2-10 の社内展開ガイドの「次にやること」へ1行で移します。

つまずき

MR にコメントが付かない

CI 変数 ANTHROPIC_API_KEY
の未登録。講師へ知らせる

ai_gate が緑のまま

allow_failure: true の行が
残っていないかを先に見る

test-phpunit がエラーで止まる

足したテストの書き方が既存の
5本と違う。setUp() の :void 等

3本が1本ずつ順に動く

1つの依頼で3本を名指し
したかを見る。直列でも成立

D2 と smart3pm に同じ形で置ける持ち帰り物

持ち帰り物	D2 での置き場所	smart3pm での置き場所
REVIEW.md	ルート。観点を4見出しへ写す	check 系の項目は Do not report へ
履歴を切った読み直し	窓を分けて同じ差分を読ませる	同じ。Ctrl+Tab で行き来する
agents 3本	.claude/agents/。DB 関数名を自社に	同じ。md は ASCII 制約の対象外
review-route skill	.claude/skills/review-route/	同じ。境界の行数は自社の値に
known_issues.md	d2-release-review の台帳を複製	同形で置く
.gitlab-ci.yml の2ジョブ	導入までは gate.ps1 で同じ判定	bash 正。そのまま使う
生成テストと検出率	tests/ に PHPUnit 7.5.20 の形で	台帳に入れず、検出率は D2-11 へ

台帳に書くのは 10(REVIEW.md と規約)と 11(観点別レビューア3本と known_issues)の2点です。

1番目と6番目の順で自社に入れます。REVIEW.md は今日から効き、CI のジョブは GitLab 導入と同時に効きます。

発展 Issue #2 の assign() に test-gap-reviewer を当て、引当の DB 書き込みを P1 で指すかを見ます。
履歴を切ったセッションの findings と CI の findings を突き合わせ、どちらだけが指したかを数えます。
CSV 出力にも同じ手順でテストを足します。

D2-06

夜間ループの設計と人間の立ち位置

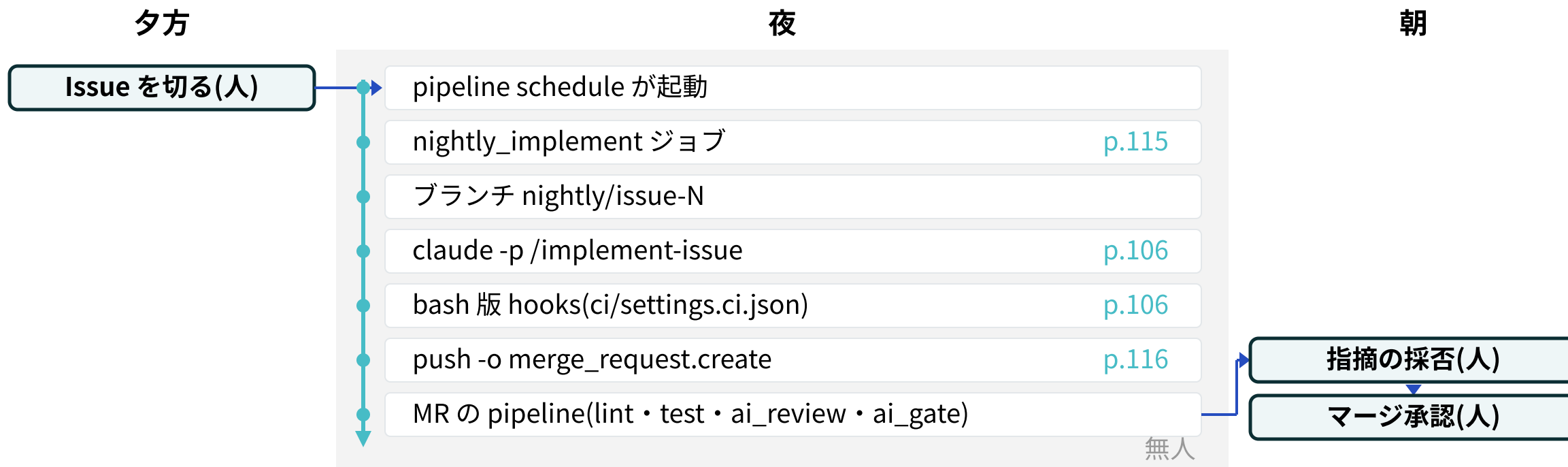
止め金を設定に書けば夜は無人で回る

claude -p の3フラグから GitLab の schedule とトークンまで、
夜間ループの部品を現物で並べ、人が残る3点を決める

[20min]

持ち帰り: 止め金一覧・夜間に回せる範囲のチェック表・人が残る3点

人が触るのは夕方と朝の2回だけ



Day1 朝に見せた実録はこの並びで動いています。夜の区間に人はいません。
部品は6つです。-p の3フラグ・Stop hook・deny・ブランチ保護・schedule・トークン。
止まる条件を全部設定に書くのが前提で、D2-07・D2-08 で自分の手で置きます。

出典: <https://code.claude.com/docs/en/gitlab-ci-cd> (GitLab CI 連携はベータ)

フラグなしの claude -p で通る変更

拒否で終わる既定の -p

0

フラグなしの -p で実行される
Edit と Bash の回数

- 対話セッションが自動で動いていても、-p の既定は全プランで default(画面の手動と同じ)です。
- 非対話なので承認ダイアログは出ません。allow ルール外のツール呼び出しは拒否されます。
- 読み取りだけで終わり、変更ゼロで正常終了します。失敗ではなく「何もなかった」です。
- 夜間に回して朝「何も起きていない」の一番多い原因がこれです。

既定 permission mode は default(手動)

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

無人で動かすフラグは3本

bash(ci/nightly_implement.sh の claude 呼び出し)

```
# PROMPT は "/implement-issue ${ISSUE_ID}" で始まる依頼文
claude -p "$PROMPT" \
  --settings ci/settings.ci.json \
  --permission-mode dontAsk \
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(PHP *),Bash(git add *),Bash(git commit *)" \
  --max-turns 15 \
  --output-format json
```

dontAsk	allow 外は全部拒否(CI 向け)
allowedTools	使わせる道具を列挙。Bash(PHP *) の半角スペース必須
max-turns	暴走の上限。ジョブ timeout と二重に掛ける

dontAsk は「聞かずに拒否」で、bypassPermissions ではありません。

--settings で bash 版のフックと deny を読ませます。runner では .ps1 のフックが起動しません。

--output-format json で結果を構造化し、後続の処理が読めるようにします。

出典: <https://code.claude.com/docs/en/cli-reference>

人がいるかどうかで分かれる Desktop と CI の CLI

人が横にいる作業 Claude Desktop

セッションの開き方

プロジェクトフォルダを選ぶ

パーミッションモード

手動 / 編集を受け入れる / プラン / 自動

止まったとき

人がその場で答える

コマンド入力

統合ターミナル

右上の >_、Ctrl とバッククォート

無人で回る作業 CI 上の CLI の非対話実行

起動の仕方

CI runner 上の CLI を `claude -p` で起動

止め金の置き場所

`settings.json` の `deny` とフック

止まったとき

答える人はいない。設定どおりに止まる

Desktop との差

`claude -p` は Desktop の UI から打てない

非対話実行は Desktop にできないことの1つ

設定ファイルは両方で共有

`~/.claude/settings.json` と `CLAUDE.md` は Desktop でも CLI でも同じものが読めます

腰を据えて使い込む段階では、CLI のほうが自動化に載せやすい形です

※ 各ロゴは各社の商標です。

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

実装側に --bare を付けると **hooks** が消える

読み込むもの	--bare なし	--bare あり	夜間ループでの扱い
hooks(settings.json)	読む	読まない	実装側は必須。--bare 不可
CLAUDE.md	読む	読まない	7.4 制約の文脈が消える
skills • agents • plugins	読む	読まない	/implement-issue が呼べない
MCP(.mcp.json)	接続する	接続しない	レビュー側では不要
認証	OAuth 可	API キー必須	CI では元から API キー
再現性	リポジトリ設定に依存	同じ入力で同じ結果	レビュー側に向く

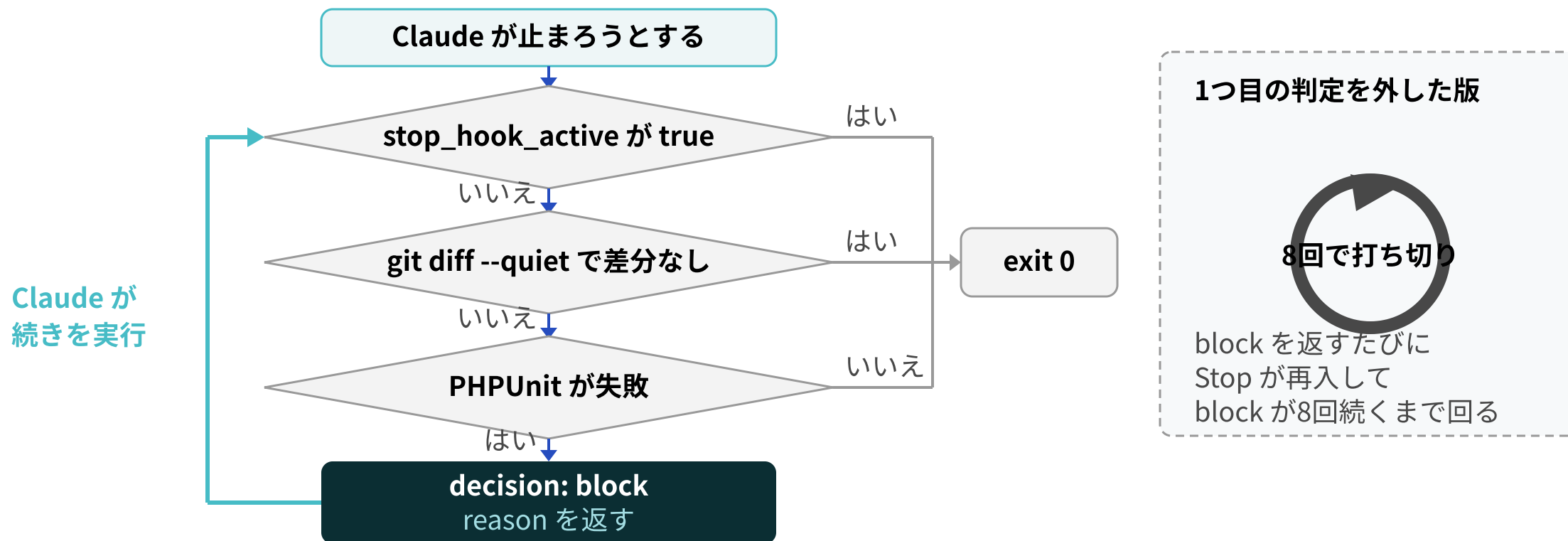
--bare は hooks • skills • CLAUDE.md • MCP • auto memory を全部スキップします。

夜間の実装ジョブに付けると、--settings で渡した check-phpver も block-destructive も動きません。

履歴を切って差分だけを見せたいレビュー側(ai_review)にだけ使います。将来 -p の既定になる予定です。

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

stop_hook_active を見ないと8回まで繰り返す



Stop hook は Claude が止まろうとした瞬間に走り、block を返すと作業を続けさせます。

2回目以降は stop_hook_active が true で入ります。これを見ずに block すると、8回連続で打ち切られるまで繰り返します。

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

3つの判定と block を返す1か所

.claude/hooks/stop-gate.ps1

Guard1 継続中か

stop_hook_active が
true なら何もせず
通します。

Guard2 差分の有無

差分がゼロなら
何もせず通します。

Guard3 テストの結果

失敗したときだけ
止めます。

止めるときだけ標準出力に {"decision":"block","reason":"..."} を書き、終了コードは常に 0 です。

テストは vendor\bin\phpunit があればそちらを、無ければ php tests/run_tests.php を回します。
配布の版は継続回数(STOP_GATE_MAX、既定 2)と php の有無も見て、当たれば通します。
コードと文字列は ASCII だけで書きます。PowerShell 5.1 で文字化けさせないためです。
スクリプト全文は演習リポジトリの .claude/hooks/stop-gate.ps1 にあります。

口頭の禁止はコンパクションで消える

夜間に残る強さ



PreToolUse hook

bypassPermissions でも走る
deny の文字列照合の抜けを全文で塞ぐ

permissions.deny / ask

Claude Code 本体が強制
-p でも bypass でも全モードで効く

CLAUDE.md

再読込されるが助言
強制力はない

会話の中の指示(push しないで)

コンパクションで消える
消えても誰も気づかない

長時間タスクはコンテキストが閾値に近づくと自動で要約に置き換わります。

会話中に言った境界は要約で落ちることがあります。守らせたい制約は deny か ask に書きます。

出典: <https://code.claude.com/docs/en/context-window>

止め金5つを4つの層に分けて置く

止め金	置き場所	止めるもの	効かない場面
ジョブ timeout 30m	.gitlab-ci.yml	時間の暴走	ジョブ外のローカル実行
--max-turns 15	-p のフラグ	ターン数の暴走	1ターンが長い場合
deny push --force • reset --hard	.claude/ settings.json	履歴の破壊	文字列照合を外れた書き方
ブランチ保護 main 直 push 不可	GitLab 側	無人マージ	保護設定が外れたブランチ
allowedTools の列挙	-p のフラグ	想定外の道具	Bash(*) と書いた場合

止め金を1か所に集めると、その1か所が外れたときに全部外れます。

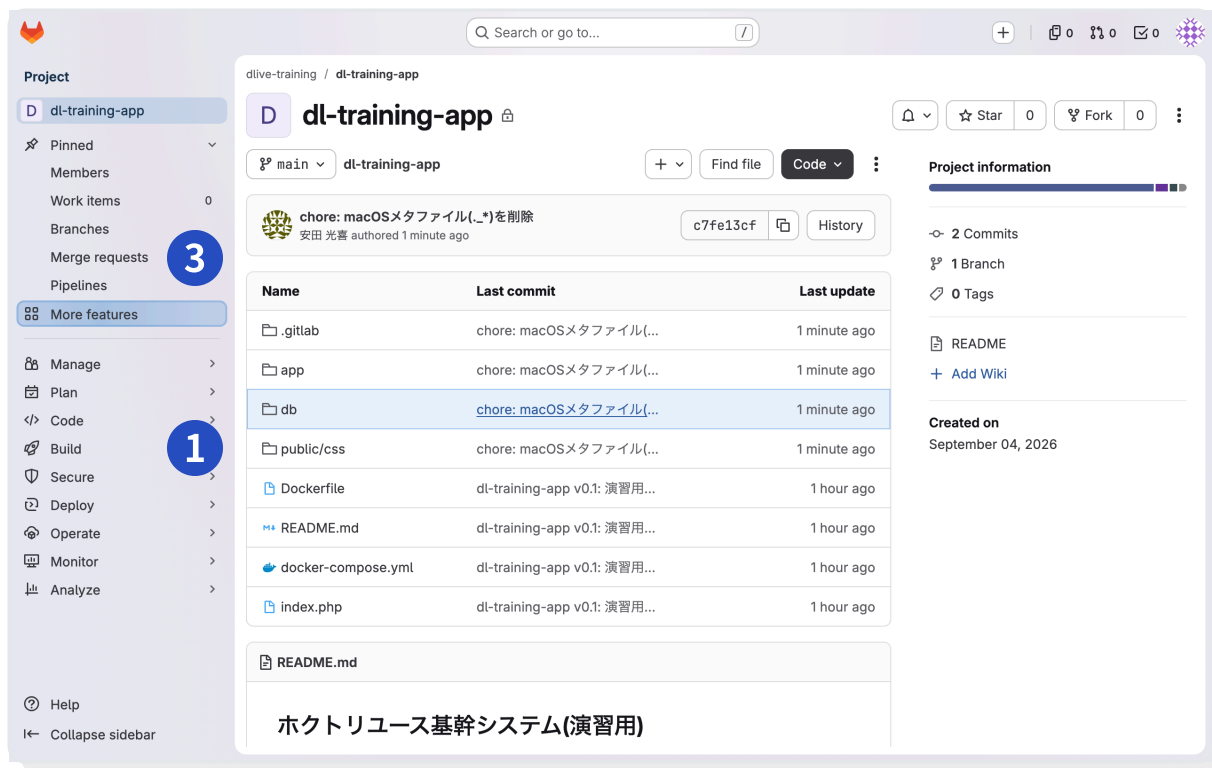
deny は文字列照合です。/bin/rm や bash -c はすり抜けるのでブランチ保護と二重にします。

D2-08 ではこの5行を全部書いた状態でだけ schedule を流します。

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

GitLab で触る3か所

schedule とトークンと MR は全部プロジェクト内



1 Build の Pipeline schedules

夜間の起動点

2 Settings の Access tokens

MR 作成に使うプロジェクトアクセストークン
この画面では表示範囲の外

3 Code の Merge requests

朝に人が開く場所

夜間ループに必要な GitLab 側の設定はこの3か所で完結します。

トークンは masked variable に入れ、ジョブのログに出ないようにします。

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

API キー運用では **/schedule** が出ない



Routines(/schedule)

実行場所 Anthropic 管理インフラ

認証 claude.ai ログイン必須

最小間隔 1時間

成熟度 research preview



GitLab

本研修の選択

実行場所 自社 runner

認証 CI/CD Variables の API キー

最小間隔 cron 次第

成熟度 GitLab 標準機能

Routines は claude.ai のサブスクでログインしている端末でだけ使えます。

環境変数に ANTHROPIC_API_KEY があると API 認証が優先され、/schedule は Unknown command になります。

出典: <https://code.claude.com/docs/en/routines>

※ 各ロゴは各社の商標です。

schedule と手動の両方から起動できる1ジョブ

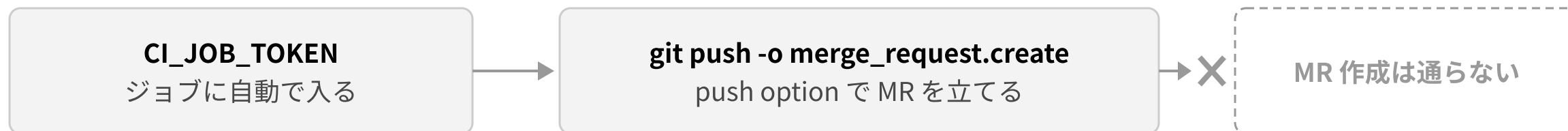
.gitlab-ci.yml(抜粋)

```
nightly_implement:
  stage: implement
  image: node:22-bookworm-slim
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: never
    - when: manual
  timeout: 30m
  variables:
    ISSUE_ID: "4"
  script:
    - bash ci/nightly_implement.sh
  allow_failure: true
```

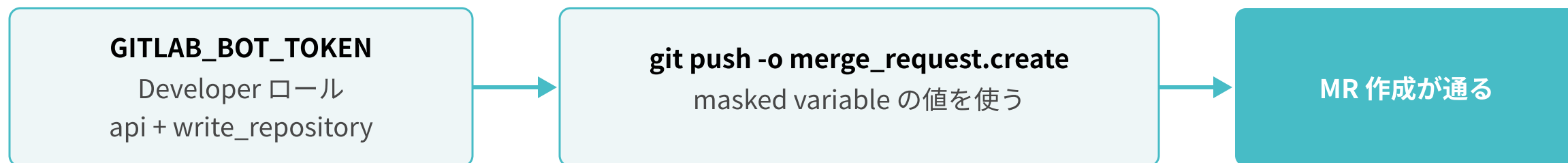
rules の3つで、夜は schedule から、MR 用のパイプラインには出さず、研修中は手動で流します。
当日は when: manual の側だけを使い、講師が main で1本流したあと各自が自分のブランチで押します。
本体の ci/nightly_implement.sh が nightly/issue-N を切り、push -o merge_request.create で MR を立てます。
出典: <https://docs.gitlab.com/topics/git/commit/#push-options>

CI_JOB_TOKEN では MR が立たない

ジョブ既定のトークン



プロジェクトアクセストークン



ジョブに自動で入る CI_JOB_TOKEN は権限範囲が狭く、push option での MR 作成が通りません。プロジェクトアクセストークンを Developer ロール、api と write_repository のスコープで発行します。値は CI/CD Variables に masked で登録します。演習では受講者のブランチから走るので Protect を外しています。自社へ写すときは Protect を付け、鍵を使うジョブは保護ブランチに限ります。

出典: https://docs.gitlab.com/user/project/settings/project_access_tokens/

GitLab CI の位置づけ

CI は新機能ではなく [M] を定義どおりに戻す作業

[M] 機械強制

[A] エージェント照合

[H] 人間判断

規約の各ルールに、誰が強制するかのタグを付ける形です。

core.md の定義

[M] は機械が強制する

CI が無い今

AI が回して自己申告

GitLab CI 後

ジョブが判定して終了コード

CI が無い間は、AI がテストを回して報告することが [M] の担保になっています。報告は自己申告です。
機械強制の定義に戻るのが GitLab CI の役目です。[M] の行のうち CI が判定する行を手元で数えてください。
D2-08 では、判定をジョブ側に置いた状態で夜間ループを流します。

注入は教えるだけで止まらない

注入 注入型(状態を読んで知らせる)

ブランチ・状態を読む

Claude に知らせる

作業が始まる

検証 検証型(fail-loud)

規約の配置・鮮度・参照先を確認

契約が成立しているか

いいえ

止まって知らせる

はい

作業が始まる

夜間に要るのは右の検証

注入は状態を Claude に教えます。検証は契約が崩れていたら作業を始めさせません。

夜間は人が画面を見ていないので、教えるだけでは崩れに気づけません。

手元の SessionStart が左右のどちらかを、各自の画面で確かめてください。

全件ログとリスク加重サンプルの再確認

承認は全件ログされ、リスク加重で抽出したサンプルを人間が再確認する。

AI

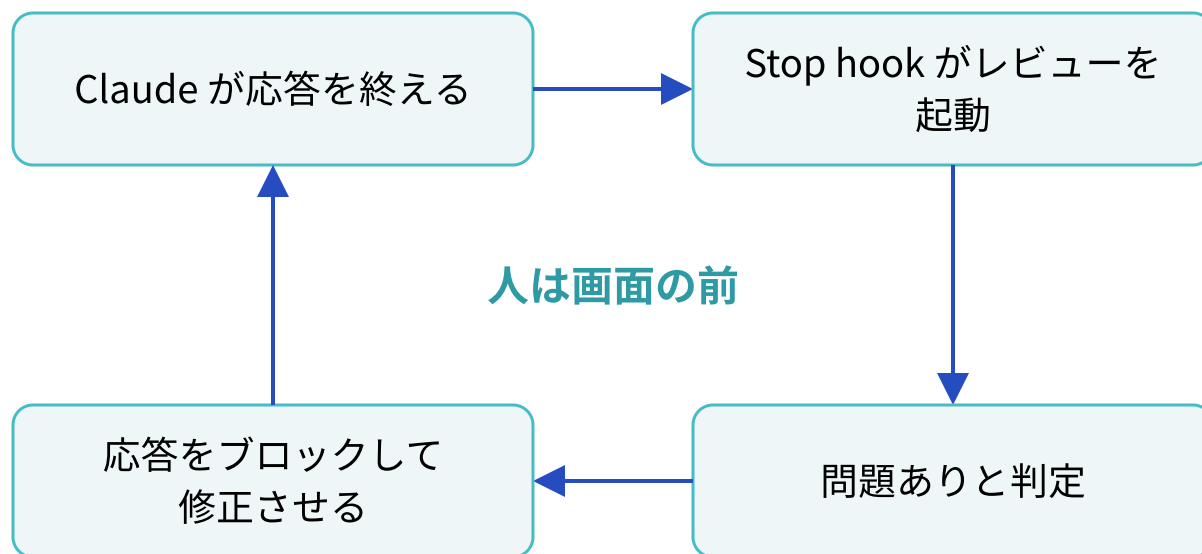
PR が開くと、狭い観点に絞った複数のレビューエージェントが走ります。
人が目を通すのは、ログから抽出された分だけです。ループの中に読む係は立ちません。
自動レビューを別種のリスクと呼び、複数ゲートと別コンテキストで制御すると書いています。

この置き方を、夜間ループの朝の仕事にそのまま使います。

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

※ 各ロゴは各社の商標です。

監視下でだけ回すループもある



回す前に決めること

1回の応答ごとにレビューが1回走ります。続く限りトークンと時間を使うので、回数の上限と見ている人を先に決めます。

レビュー役は履歴を切った Claude Code に任せます。新しいセッションなら差分だけを見ます。応答のたびにレビューが走る設計は、人が画面の前にいる時間に限ります。夜間ループに組み込む部品ではありません。止め金は 114 の5つで足ります。

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

フックで作り込んだ強制の行き先

13コミットの hook が **CLAUDE.md 1行**に戻った

13 コミット

レビュー実行を hook で強制した作り込みの量

フラグ注入・シェル差異・AST パースまで肥大化

Claude Code がレビュー手順を飛ばす問題に、hook で強制する実装を13コミット重ねた事例があります。
フラグ注入やシェル差異の抜け道を潰すうちに AST パースまで肥大化し、PowerShell 専用で WSL では動きませんでした。
確実性が要る所はブランチ保護と CI に置くのが教訓です。

出典: <https://qiita.com/ay5399/items/642d494503f133f27325> (2026-08-15、個人の事例)

結果

CLAUDE.md に1行だけ追記
レビュー結果を PR 本文に書く。
これで解決しています。

human on the loop の人の仕事

人が決める3点と機械に渡す範囲

人が残るのは3点

夕方

Issue の粒度決め

夜間で終わる大きさに切る
テストが自動生成できる範囲

01

朝

指摘の採否

ゲートを通った分だけ読む
ai_gate を通った P0・P1 だけ

02

朝

マージ承認

巻き戻せない変更を開く
含まれていれば必ず開く

03

ループの中でコードを読む係は置きません。

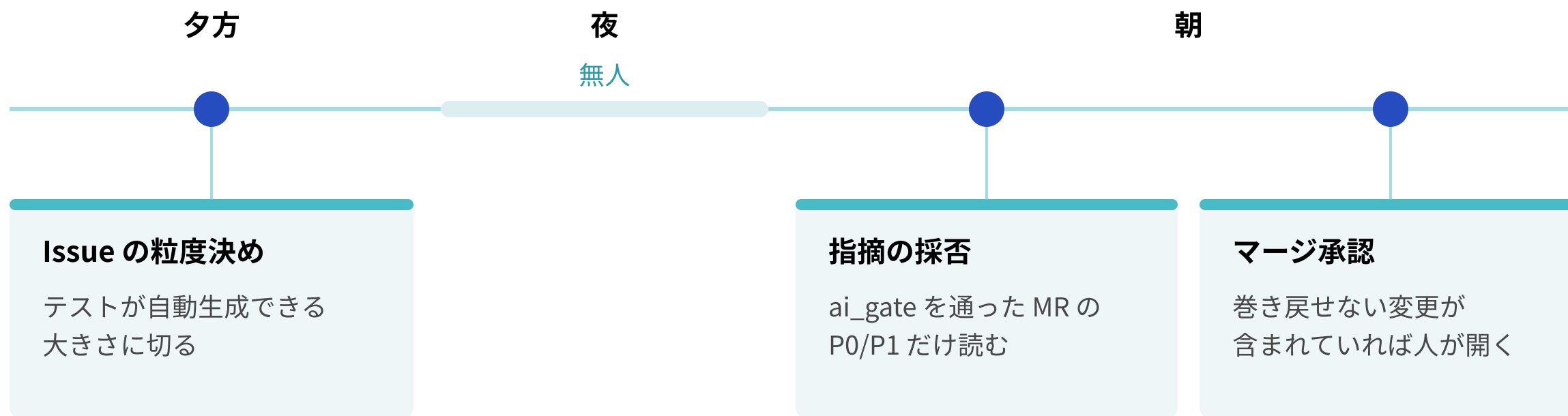
夜間ループで人が持つのは、何を任せるかを決める夕方と、出てきたものを通すかを決める朝の仕事です。

差分を1行ずつ読む係を置くと、夜間に回した分だけ朝の読み量が増えて自動化の意味が消えます。

059 で見たリスク加重の抽出は、夜間ループでは ai_gate を通った分だけに当てます。

巻き戻せない変更(DB・外部連携・権限)だけを人が開きます。

夕方に1点と朝に2点



夕方の仕事は Issue を夜間で終わる大きさに切ることです。大きすぎる Issue は max-turns で止まります。

朝の仕事は2つです。ゲートを通った指摘の採否と、マージするかの判断です。

D2-08 の最後に、各自が今のハーネスで夜間に回せる範囲と人が残る3点を1枚に書きます。

同じファイルを触るなら **worktree** で分ける

形	調整役	同じファイルを触る場合	-p で動くか	成熟度
worktree 隔離 claude --worktree	Claude または人	隔離されるので安全	動く	GA
複数 -p 直列 リポジトリごとに順に回す	スクリプト (CI のジョブ)	リポジトリが別なら衝突しない	動く	GA
Agent Teams 実験フラグで有効化	リード	隔離されない。担当を分ける	動かない (対話のみ)	experimental

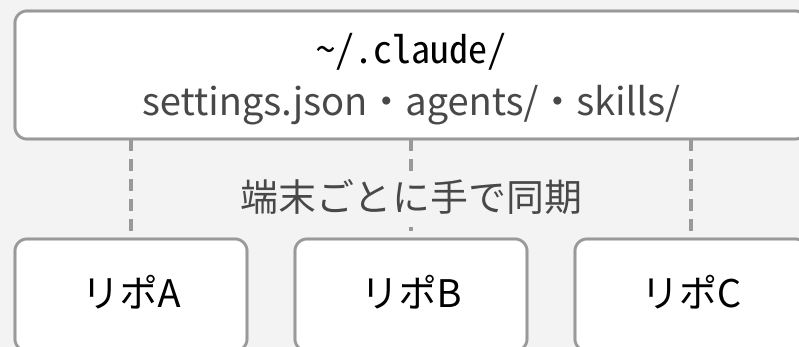
夜間ループに載せるなら、-p で動く上2つから選びます。Agent Teams は対話セッション専用です。
有効化は環境変数 `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` で、トークンはメンバー数に比例します。
判断軸は公式の [agents](#) ページの3つです。誰が調整するか、会話するか、同じファイルを触るか。
複数リポジトリが別物なら、複数 -p 直列を CI の stage に並べるのが最短です。

出典: <https://code.claude.com/docs/en/agents>

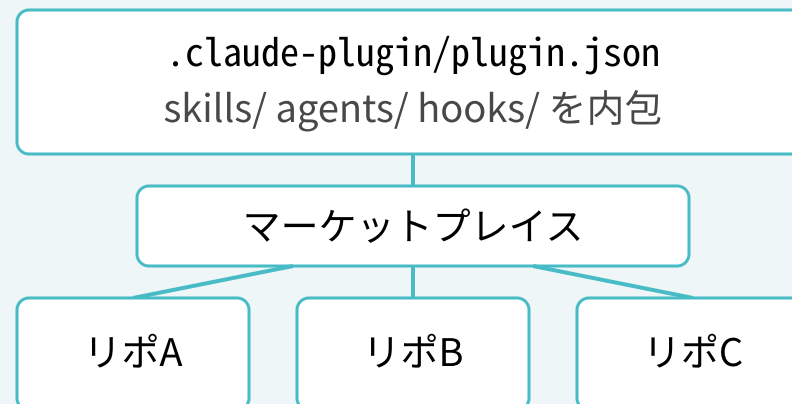
配布と版管理が要るなら plugin に束ねる

見方 複数リポジトリで設定を揃えるとき、手元の構成がどちらに近いかを確認めます

案A user scope



案B plugin



user scope は個人端末に閉じます。複数人で同じ設定を持つには端末ごとに同期が必要です。

plugin は skills • agents • hooks を1ディレクトリに束ね、/plugin install で配布と版管理ができます。

講師は案Bを推します。同名のプロジェクト側エージェントが plugin を上書きするので、移した元は消します。

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

フックが止めるのはコマンドの種類だけ

制限の軸	フックだけの形	範囲を絞った形
コマンドの種類	deny と PreToolUse フックで止まる	ここは済み
変更ファイル数	上限なし	N ファイル以上の変更は止める
変更先のディレクトリ	範囲の指定なし	Edit の対象をパス単位で絞る
戻し方	ブランチごと捨てる	部分ロールバック

フックが見ているのはコマンドの種類です。何ファイル触ったかは判定に入っていません。
範囲を先に絞ると、戻す作業がブランチごと捨てる形から触った範囲だけ戻す形に変わります。
呼び方は Day1 と揃えます。変更範囲の事前制限と部分ロールバックの2語です。

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

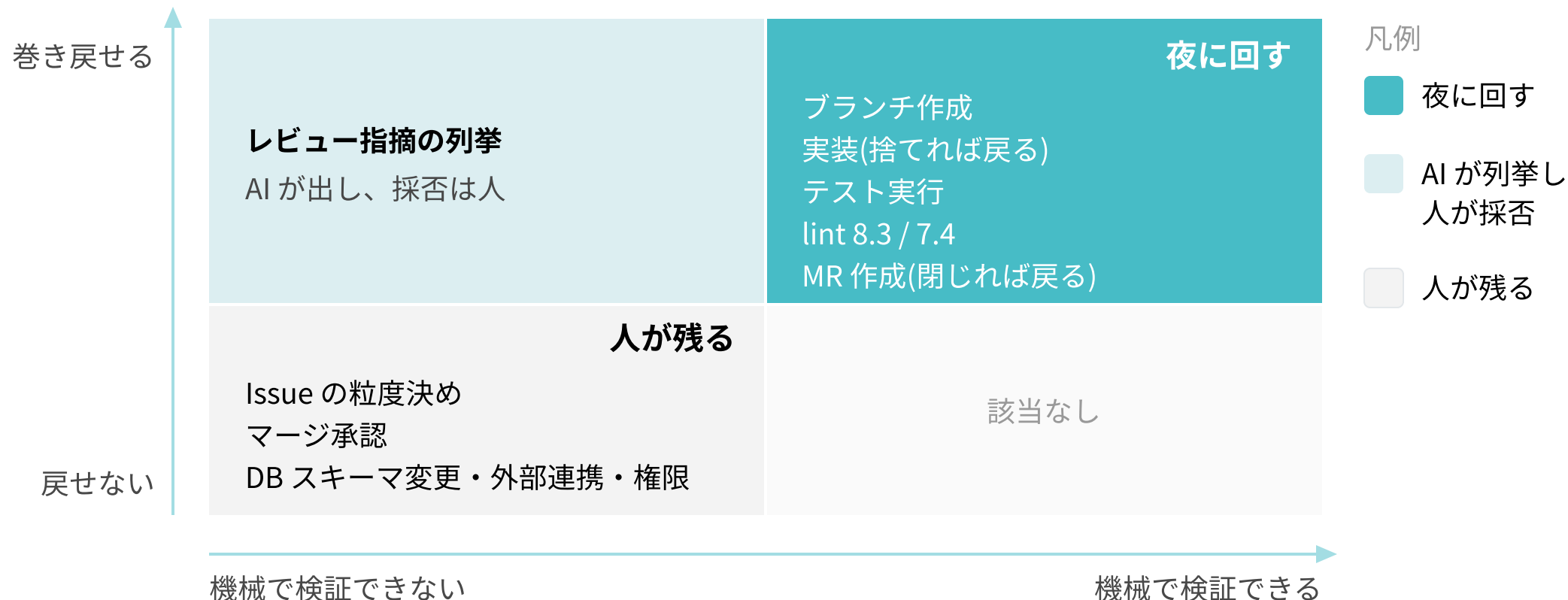
全部に印が付くまで **schedule** を流さない

確認する項目	置き場所
☐ -p に dontAsk ・ allowedTools ・ max-turns の3本	.gitlab-ci.yml
☐ Stop hook が stop_hook_active を見ている	.claude/hooks/stop-gate.ps1
☐ deny に push --force と reset --hard がある	.claude/settings.json
☐ main がブランチ保護され直 push 不可	GitLab 側
☐ ジョブに timeout 30m がある	.gitlab-ci.yml
☐ GITLAB_BOT_TOKEN が masked variable にある	CI/CD Variables
☐ schedule のジョブが when: manual で止まっている	.gitlab-ci.yml

D2-08 の開始時にこの表を各自が埋め、講師が main で1本流してから各自が押します。

7行のうち1つでも空なら、schedule は流さず手動で回します。置き場所が5か所に分かれているのは 114 と同じ考え方です。
D2-11 の8層チェック表を塗り直すとき、この7行が到達した層の根拠になります。

巻き戻せて機械で検証できる工程だけ夜に回す



右上に入る工程は夜に回します。失敗してもブランチを捨てれば戻り、CI が正否を判定できます。

左下は人が残ります。戻せない変更と、正しさを機械で決められない判断です。

現行の切り分け(AI が列挙、人が判断)は左上の位置で、夜間でもそのまま生きます。

D2-06 の持ち帰り

設定に書ける**止め金**とまだ書けない判断

止め金一覧

timeout ・ max-turns ・ deny ・
ブランチ保護 ・ allowedTools
の5つと置き場所。

置き場所

D2 は hooks/ の ps1
smart3pm は bash 版

夜間の自己チェック表

7行の自己チェック。
1つでも空欄があれば、
schedule は流しません。

使う場面

D2-08 の開始時に
各自が埋める

人が残る3点

124 の3点をそのまま使います。

使う場面

D2-08 の最後に
各自1枚へ書き写す

この3つは台帳17点には数えず、D2-08 の台帳14「夜間ジョブ一式」の材料にします。

止め金は D2 なら PowerShell 5.1 ・ ASCII の制約で、smart3pm なら bash 版で、同じ5つを置きます。

チェック表と人が残る3点は、次の D2-07 ・ D2-08 でそのまま使います。

D2-07

プチ演習6 止まる Stop hook

テストが赤い間だけ1回続き、そこで止まる関所

つくるもの: settings.json の Stop 登録(stop-gate.ps1 は配布済み)

踏むもの: 8回で打ち切られるまで続く版(講師実演)と正しい版の1回継続

持ち帰り: stop-gate(ps1 / sh)の1点。台帳の13行目に書きます

ワーク

[15min]

夜間ジョブは ci/settings.ci.json で bash 版のフックを別に登録します

Stop は作業ツリー全体を見る最後の関所



----- Edit|Write の matcher は Bash 経由の sed や echo > file を拾わない

decision: block を返して続けさせる

何も返さず exit 0 で止める

PreToolUse はコマンド1本を見ます。Stop は作業ツリー全体を見ます
止めるか続けさせるかを機械で決める場所はここだけです
テスト未通過のまま「完了しました」と言わせない、が今日の目的です

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

ハンズオン2のブランチのまま始める

確認項目	確認の仕方	こう見えていれば正しい
Claude Code の版	Claude Desktop の設定画面	2.1.280 以降
演習フォルダ	Claude Desktop で開く	dl-training-app を選んだ状態
ブランチ	VSCode 左下のブランチ名	ex/h2-<ユーザー名> のまま
PHP の有無	<code>php --version</code>	版番号か「認識されません」
PowerShell	<code>\$PSVersionTable.PSVersion</code>	5.1 以上

新しいブランチは切りません。ハンズオン3はこのブランチから切るので、ここで登録した Stop がそのまま乗ります。変更の有無は問いません。Step 3 で Claude にファイルを1つ編集させます。

補足

php が無い端末では、stop-gate はテストを回さずに通します。
継続が入る様子は講師の画面で見せます。どちらの端末でも演習は成立します。

コードを書く前に**止める条件**を自分で決める

操作: 手元の紙かメモに、次の3つを1行ずつ書きます。画面の操作はありません

自分で考える [2min]

1

終わらせてはいけない状態

自社で、AI の応答を終わらせてはいけない状態を1つ。テストが赤い以外で書きます

2

判定にかかる秒数

その判定が何秒かかるか。フックは応答のたびに走り、その分だけ会話が遅くなります

3

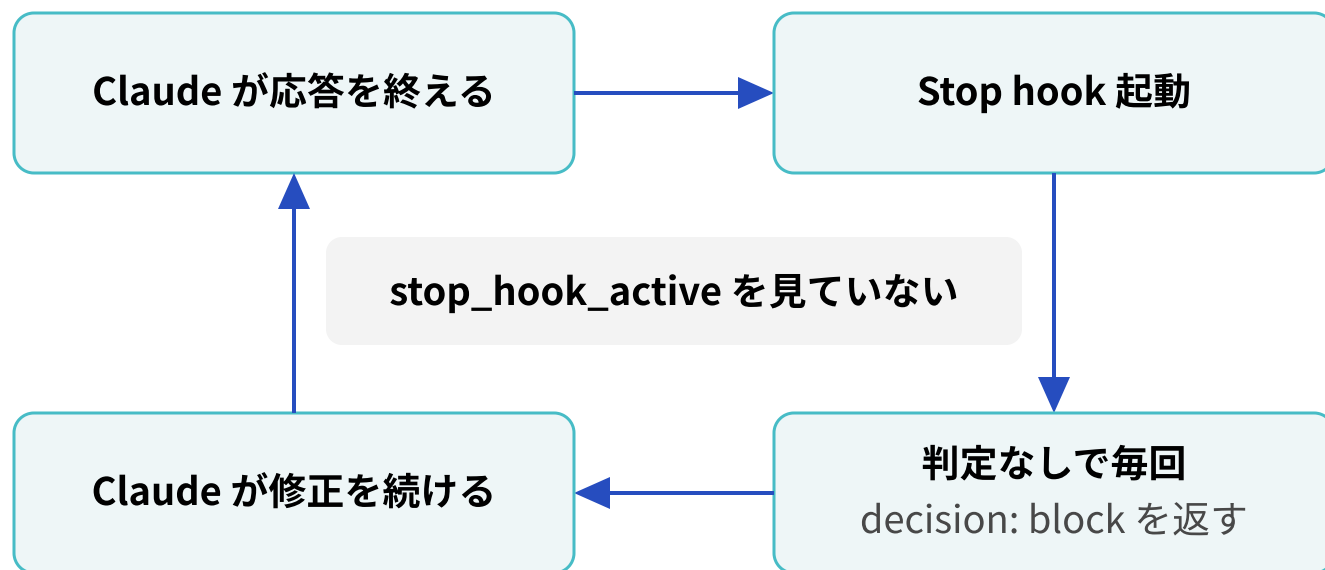
継続中の見分け方

「すでに継続中」をフックはどの情報で知るか。入力 JSON のキーの見当を書きます

3つ目の答えは Step 1 で出ます。先に自分の案を持つと、配布の版のどの行が違うかが見えます。
このあとの Step 1~4 は、ハンズオンガイドの Step 1~4 と同じ番号です。

Step 1 止まらない版を見る

継続中かどうかを見ないフックは8回まで続く



止まる所

8回連続の block で打ち切り

消えるもの

トークンと時間

講師が止まらない版と配布の版を並べて回します。受講者は手を動かさずに画面を見ます
止まらない版は入力を読んですぐ block を返す3行だけで、応答を終えるたび継続が入ります
Claude Code が8回で打ち切るまで、反復の回数と /usage の消費が積み上がるのを見てください

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

Step 1 止まらない版と配布の版

通す3条件を先に書き最後だけ block を返す

上の3行が止まらない版、下が .claude/hooks/stop-gate.ps1 の抜粋です

Step 1 [3min]

```
1 # demo only. do not register this one.
2 $payload = [Console]::In.ReadToEnd() | ConvertFrom-Json
3 Write-Output '{"decision":"block","reason":"tests are red. fix and finish."}'
4 # ---- distributed stop-gate.ps1 (excerpt) ----
5 # Guard 1: already continued by this hook.
6 if ($payload.stop_hook_active -eq $true) { exit 0 }
7 # Cap: continuations counted per session in .stop-gate.state
8 if ($count -ge $max) { ...; exit 0 }
9 # Guard 2: nothing changed in the working tree
10 if ([string]::IsNullOrEmpty($changes | Out-String)) { ...; exit 0 }
```

1~3行目

入力を読んで毎回 block

6行目

継続中なら通す

8行目

上限に達したら通す

10行目

変更が無ければ通す

止まらない版は各自の端末では回しません。登録するのは配布の版です

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

配布の版は3つを通ったあとでテストを回し、赤のときだけ block を返します。php が無い端末は回さずに通します

フックはまず手で JSON を流して確かめる

手順1 Stop のブロックを登録する

Step 2 [3min]

VSCode で .claude/settings.json の "hooks" に足します。貼るブロックは次のページです

手順2 門の1つ目だけを単体で流す

統合ターミナルは右上の >_ か、Ctrl とバッククォートで開きます

```
echo '{"stop_hook_active":true,"session_id":"check"}' |  
powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\stop-gate.ps1  
$LASTEXITCODE
```

期待される画面

0 何も表示されず、終了コードが0です。ここで decision を返す版が Step 1 の止まらない版です

手順3 新しいセッションで読み込ませる

Claude Desktop で Ctrl+N を押し、VSCode の settings.json で Stop が1本、赤い波線が無いことを見ます

cases.json には足しません。stop_hook_active が true のケースは配布時点ですでに入っています(24件のまま)。smart3pm 担当は stop-gate.sh に同じ JSON を流し、終了コード 0 を見ます。

登録は Stop イベント1本と timeout だけ



Step 2 手順1

.claude/settings.json の "hooks" の中。 .claude/settings.example.jsonc の同じブロックと中身は同じです

```
"Stop": [ {  
  "hooks": [ {  
    "type": "command",  
    "command": "powershell.exe",  
    "args": ["-NoProfile", "-ExecutionPolicy", "Bypass", "-File",  
      "${CLAUDE_PROJECT_DIR}/.claude/hooks/stop-gate.ps1"],  
    "timeout": 300  
  } ]  
} ]
```

貼る場所は "PostToolUse": [...] を閉じる] の直後です。そこにカンマを1つ置いてから貼ります。

Stop に matcher はありません。パスは1行のまま写します。Desktop でも CLI でも同じファイルが読めます。

timeout の既定は 600 秒で、テストの実行時間に合わせて 300 に縮めています。

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

※ 各ロゴは各社の商標です。

1回目だけ続いて2回目で止まる

```
README.md の先頭に「演習用リポジトリ」という1行を追加してください。テストの修正は不要です。
• * README先頭行を確認 : cat
  Read 1 file
• 既に先頭行に「演習用リポジトリ」が入っています。追記は不要でした。
  現状: README.md の1行目が 演習用リポジトリ、2行目が空行、3行目から本来の見出しという並びです。git status では README.md
  が変更済み扱いになっているので、この行は今回のセッションより前に追加済みと見られます。
  重複して追記すると同じ行が2つ並ぶので、何もしていません。もし別の文言に変えたい、あるいは先出しの行へ移したい場合は指示をください。
• Ran 3 stop hooks
  | Stop hook error: php tests/run_tests.php failed with exit code 1. Run it, read the failing case, fix the code, then finish. (stop-gate continuation 1 of 2)
• * アバド失敗内容の確認 : php tests/run_tests.php
  Ran 1 shell command
• * 失敗箇所の特定 : git diff, grep
  Searched for 2 patterns, read 1 file, ran 3 shell commands
```

破線の行が1回目の停止です。末尾の continuation 1 of 2 が回数で、この後に Claude がテストを読み直し、次の応答で止まります。

操作: 例えば「README.md の末尾に、今日の日付を1行足してください」と頼み、応答を終わらせます
PHP が無い端末では継続が入らずに終わります。同じ場面を講師の投影画面で見てください

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

Step 3 [4min]

1 ファイルを1つ編集させる

内容は何でも構いません

2 1回目の停止

reason が出て Claude が修正を続けます

3 2回目の停止

stop_hook_active が true になり、未通過でも止まります

画面

Mac の bash 版フックの画面です。
PowerShell 版でも文面は同じです

画面: Claude Desktop の統合ターミナル

合格は4つの確認とコミット1本

OK 基準	確認の仕方	対処
OK1 門の1つ目	Step 2 手順2 で何も出ず 0	JSON の true と綴りを見直す
OK2 1回継続	Step 3 で継続1回のあと終わる	PHP が無い端末は講師の画面
OK3 上限 0	Step 4 で継続が入らずに終わる	"env" を足したら Ctrl+N
OK4 コミット	Stop のブロックだけを1コミット	"env" の行と .stop-gate.state を外す
つまずき	見え方	対処
実行ポリシーで停止	running scripts is disabled	-ExecutionPolicy Bypass を確認
登録が読まれない	赤い波線、継続が一度も入らない	カンマを見直して Ctrl+N

あわせて、止まらない版と配布の版の差を、通す条件3つ(継続中・上限・変更なし)で説明できれば合格です。

止める回数を決めるのは人の仕事

持ち帰り物	D2 の置き場所	smart3pm の置き場所
stop-gate.ps1	hooks/ に置き Stop へ登録	使わない(bash 正)
stop-gate.sh(jq 版)	使わない	hooks/ に置き Stop へ登録
上限の値	持ち帰り台帳に値と理由を1行	同じ値を使う

Step 4 上限の効き方を確かめる

Step 4 [3min]

- 1 `.claude/settings.json` の先頭の `{` の次の行に1行足して保存します

```
"env": { "STOP_GATE_MAX": "0" },
```
- 2 Ctrl+N で新しいセッションを開き、Step 3 と同じ依頼で応答を終わらせます。継続は入りません
- 3 確かめたら "env" の行を消して保存し、Ctrl+N。残すとハンズオン3で継続が1回も入りません
- 4 `.claude/settings.json` だけをステージし「プチ演習6: Stop フックを登録」でコミットします
- 5 自社で使う上限を決めます。検査を残して継続だけ切りたいときが 0 です

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

[105min]

ワーク

D2-08

ハンズオン3 夜間ループの最小構成

マージだけを人に残す夜間の一周

題材は Issue #4 「見積状態コードの対応表の一本化」です。schedule から起動する1ジョブで、ブランチ作成から MR 作成まで回します。

1回目は止まる前提です。止まった箇所を記録し、直す層を1つ選んで2回目を回すところが本体です。持ち帰りは夜間ジョブ一式の1点です。中身は nightly_implement ジョブ、ci/nightly_implement.sh、implement-issue skill、止め金の settings、schedule の設定値です。

止め金を書いてから回す7つの Step

走らせる前に止め金を書きます。書いていない止め金は夜間には効きません。



1回目は止まる前提です。Step4 で止まった箇所を記録し、Step5 で直す層を1つ選んで2回目を回します。

濃い2枚がその本体です。通すことではなく、止まった場所を特定して直すところを見えています。

Step3 は講師が main で1本だけ流し、全員が同じログで Step4 を書きます。Step5 は各自のブランチで流します。

Day2 で積んだ4本の在処

確認	確認する項目	確認コマンドまたは画面
☐	ex/h2-<ユーザー名> にいて作業ツリーが clean	<code>git branch --show-current</code> <code>git status</code>
☐	D2-02 の check-phpver と D2-07 の stop-gate が settings.json の hooks にある	settings.json を開いて確認
☐	ai_gate の allow_failure を外した .gitlab-ci.yml が ex/h2 ブランチに載っている(D2-05)	VSCode で .gitlab-ci.yml を開く
☐	ci/nightly_implement.sh と .claude/skills/implement-issue/SKILL.md がある	VSCode のエクスプローラ
☐	Claude Code の版が 2.1.280 以降	Claude Desktop の設定画面
☐	remote に nightly/issue-4 ブランチが無い	GitLab の Code > Branches で検索

行1のブランチから、Step1で ex/h3-<ユーザー名> を切ります。行6は講師が流す1回目の準備です。
行3が未了でも進められます。allow_failureのままだと、ai_gateが赤でもパイプラインは赤になりません。

Project

dl-training-app

Pinned

Members

Work items 5

Branches

Merge requests 2

Pipelines

More features

Manage

Plan

Work items 5

Issue boards

Milestones

Wiki

Code

Build

Secure

Deploy

Operate

Monitor

Analyze

Settings

Help

Collapse sidebar

dl-training-app / dl-training-app / Issue Boards

Development

Search

Open 5

Closed

見積一覧の顧客名検索の条件組み立て

演習::ハンズオン1 種別::不具合 領域::見積

#1 Day1

在庫の引当で半端な状態が残る

演習::ハンズオン2 種別::不具合 領域::在庫

#2 Day2

見積一覧の期間検索と CSV 出力

演習::ハンズオン2 種別::機能追加 領域::見積

#3 Day2

見積状態コードの対応表の一本化

実行::夜間 演習::ハンズオン3 種別::リファクタリング 領域::共通

#4 Day2

見積登録の入力チェックの追加

演習::予備 種別::機能追加 領域::見積

#5

dl-training-app#4

Edit

Issue created 2 weeks ago by 安田 光喜

見積状態コードの対応表の一本化

Open Issue created 2 weeks ago by 安田 光喜

背景

見積の状態(作成中・提出済・受注・失注・取下げ)の対応表が、アプリの中に3か所あります。状態を1つ足すときに3か所を直すことになり、どれか1つが漏れると、一覧の表示と変更履歴に残る名前が食い違います。履歴は監査で使うので、名前が食い違うと後から追えません。

この Issue は夜間の無人実行で流します。人が見た目で判断する余地を残さないよう、受入条件はすべて機械で判定できる形にしています。

現状

同じ知識が3か所にあります。

場所	形	使い道
app/libraries/util.php の estimate_status_name()	if の並び	一覧と詳細の表示名
app/controllers/estimate_ctl.php の change_status()	array(1 => '作成中', ...) のローカル変数	未知コードの判定と、履歴に入れる名前
app/views/estimate_list.php	array(1, 2, 3, 4, 9) の直書き	検索欄の状態プルダウン

change_status() のローカル配列には「utilにも同じ対応表がある」というコメントが残っています。

期待する振る舞い

- app/libraries/util.php に estimate_status_list() を足します。コードをキー、表示名を値にした配列を返します。実行環境は PHP 5.4 なので、const への配列代入は使わず、関数が配列を返す形にしま

Read more

無人で実装させる前提で書かれた Issue

Issue #4 「見積状態コードの対応表の一本化」を開きます。対応表は util.php、estimate_ctl.php、estimate_list.php、estimate_detail.php の4か所にあり、util.php の1か所へ寄せます。受入条件は7つです。

受入条件(Issue #4 の _issues/issue-4.md より)	判定
1. app 配下で '作成中' が出てくるファイルが app/libraries/util.php の1つだけ	grep
2. estimate_list.php と estimate_detail.php に array(1, 2, 3, 4, 9) が無い	grep
3. app 配下に enum 宣言が無い(PHP 7.4 に無い構文)	grep
4. tests/run_tests.php に対応表のテストが3件増えている	CI
5. tests/phpunit/EstimateSearchTest.php の既存テストが全部通る	CI
6. lint-phpdev / lint-phpver / test / test-phpunit の4つが緑	CI
7. MR の説明に、変更方針3行と影響範囲の調査結果が入っている	人

implement-issue skill は、受入条件が読み取れない Issue では手順1 で止まります。Issue の書き方が入口です。対象は PHP 7.4 です。状態の一覧を頼むと enum を提案されやすいので、条件3 で機械的に止めます。

ログを見る前に紙へ書く4つの予想

問い	自分の予想を書く欄
1 1回目の夜間ジョブは skill の手順0～8 のどこで止まると思いますか。 手順番号と理由を1行で書きます	<hr/> <hr/>
2 止まる原因は「権限」「7.4 外の構文」「テスト」「レビュー」「暴走」のどれに近いですか。1つ選びます	<hr/> <hr/>
3 その止まり方を直す設定はどの層にありますか -p のフラグ / settings.json / hooks / .gitlab-ci.yml / ブランチ保護 / Issue の書き方 から1つ選びます	<hr/> <hr/>
4 夜間に任せないと決める変更を1つ書きます 理由も一言添えます	<hr/> <hr/>

紙かメモに5分で書きます。AI には渡しません。

Step3 のログと突き合わせ、予想が外れた行だけ 157 の台帳に理由を残します。

走らせる前に**設定へ書く**4つの止め金

層	止めるもの	書く場所
ブランチ保護	main への直接 push を止める	GitLab のブランチ保護(講師が設定済み) Code > Branches の protected で見ます
ジョブの timeout	1本の夜間ジョブの上限時間	.gitlab-ci.yml の timeout: 30m
--max-turns 15	エージェントのターン数の上限	ci/nightly_implement.sh
permissions.deny と hooks	reset --hard / push --force / .env 参照 編集直後の PHP 7.4 検査、未通過での停止 変更範囲の事前制限(触る範囲と件数)	.claude/settings.json .claude/hooks/ ci/settings.ci.json(CI)

外側ほど機械の強制、内側ほど AI の振る舞い

1〜3段目は演習環境に入っています。今日書くのは4段目です。

4段目には変更範囲の事前制限も入ります。触ってよいディレクトリと、N ファイル超で止める件数です。

手元では4段目が全部効きます。CI runner(Linux)は ci/settings.ci.json で deny と bash 版フックを読みます。

出典: <https://code.claude.com/docs/en/permissions>, <https://code.claude.com/docs/en/cli-reference>

口頭の禁止を **deny** と **hook** へ

.claude/settings.json(permissions.deny、13本)

```
"deny": [  
  "Bash(git reset --hard *)",  
  "PowerShell(git reset --hard *)",  
  ... 破壊系 5組10本 ...  
  "Read(./env)",  
  "Read(./env.*)",  
  "Read(./**/credentials*)"  
]
```

.claude/settings.json(hooks)

登録済みの3本を確かめます。
PreToolUse(block-destructive)
PostToolUse(check-phpver)
Stop(stop-gate)
4本目の SessionStart を下のとおり足します。
Desktop でも CLI でも同じファイルが効きます。

Stop を閉じる] の直後にカンマを置き、その下に足すブロック(パスは1行のまま)

```
"SessionStart": [ { "hooks": [ {  
  "type": "command", "command": "powershell.exe",  
  "args": ["-NoProfile", "-ExecutionPolicy", "Bypass", "-File",  
    "${CLAUDE_PROJECT_DIR}/.claude/hooks/sessionstart_verify.ps1"],  
  "timeout": 30 } ] } ]
```

保存したら Ctrl+N で新しいセッションを開き、赤い波線が無いことを見てコミットします。push は Step 5 です。

出典: <https://code.claude.com/docs/en/permissions>, <https://code.claude.com/docs/en/hooks>

schedule からも手動からも入る1ジョブ

.gitlab-ci.yml(nightly_implement の抜粋)

```
nightly_implement:
  image: node:22-bookworm-slim
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: never
    - when: manual
  timeout: 30m

  variables:
    ISSUE_ID: "4"
  script:
    - bash ci/nightly_implement.sh
  allow_failure: true
```

VSCode で .gitlab-ci.yml を開き、nightly_implement を読みます。書き換えません。

rules は、schedule なら自動、MR 用のパイプラインには出さない、それ以外は手動ボタン、の順です。

allow_failure: true なので、1回目が止まってもパイプライン全体は赤になりません。

ISSUE_ID の既定は 4 のままです。右端の列は左の続きで、同じ nightly_implement の中身です。

出典: <https://docs.gitlab.com/ci/pipelines/schedules/>, <https://code.claude.com/docs/en/gitlab-ci-cd>

ブランチから MR まで1本の脚本

スクリプトの順

止め金

1 変数の確認(キー・トークン・ISSUE_ID)

変数未設定で止まる

2 git checkout -B
nightly/issue-4

3 claude -p (手順3・7を飛ばす添え書き)

--max-turns

--settings

timeout 30m

4 要約をログへ出す

5 差分が無ければ終了

差分ゼロで MR を作らない

6 git push -o merge_request.create

ブランチ保護

強制 push なし

ブランチ作成と push はスクリプトの仕事です。allowedTools に git push も git switch も渡していません。
箱3 で止まっても箱5 が拾い、MR を作りません。--settings は ci/settings.ci.json(deny と bash 版フック)です。
main 以外から流すと、起点のブランチ名が後ろに付きます。2回目が1回目と重ならない仕組みです。

出典: <https://docs.gitlab.com/topics/git/commit/#push-options>

https://docs.gitlab.com/user/project/settings/project_access_tokens/, https://docs.gitlab.com/ci/jobs/ci_job_token/

作るだけで流さない今日の schedule

Step2-2 GitLab で dl-training-app を開き、Build > Pipeline schedules > New schedule を押します。

Step2-3 5つの欄を次のとおり埋め、Activated のチェックを外して保存します。

New schedule

Description

nightly issue-4

Interval Pattern

Custom 0 2 * * *

Cron timezone

Tokyo

Target branch

main

Variables

ISSUE_ID=4

Activated のチェック

開いた時点では付いています。
当日は手動ボタンから流すので、
外して保存し Inactive にします。

Step2-4

一覧の再生ボタンは今日は
押しません。行が Inactive
になっていれば正しい状態です。

保存すると一覧に nightly issue-4 の行が増えます。持ち帰るのは、ここに入れた設定値です。

当日はこの schedule からは起動しません。Step3 で使うのは手動ボタン側だけです。

全員分が同じ時刻に走ると、runner と API の予算を一晩で使い切ります。時刻で動かすのは自社に写してからです。

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

Step3 講師が流す1本の共有

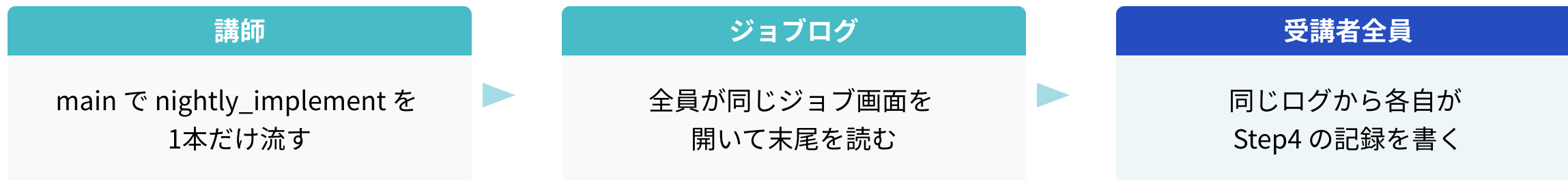
Step3 [15min]

全員が**同じログ**で書く1回目の記録

講師が Build > Pipelines の New pipeline で main を選び、nightly_implement の手動ボタンを1回だけ押します。

受講者は押しません。全員が同じパイプラインのジョブ画面を開き、同じログで Step4 の記録を書きます。

待っている間は 157 の台帳の左2列を埋めます。



Step5 の2回目は、各自が自分のブランチ ex/h3-<ユーザー名> から手動ジョブを押します(158)。

1回目を1本にしている理由

runner は1台で、nightly_implement は1本で最長30分その runner を使います。

全員分を順に流すと Step3 の15分に収まらないため、1回目は講師の1本を全員で読みます。

止まり方が出るログの末尾3行

Step3-1

Pipelines 一覧で講師が流した main のパイプラインを開き、nightly_implement を押します。
期待される画面: ジョブログが流れ、前半に `claude --version` の出力が出ます。

Step3-2

```
# 1回目に多い終わり方
Issue #4 の実装を開始します。ブランチ: nightly/issue-4
turns: 3 / cost_usd: 0.0412 / is_error: false
エージェントによる変更がありません。MRは作りません。
どこで止まったかは ci_logs/claude-issue-4.json を見てください。
# 2回目に完走したとき
turns: 9 / cost_usd: 0.41 / is_error: false
MRを作成しました。マージは人が判断します。
```

Step3-3

Job artifacts の Browse で `ci_logs/` を開き、JSON をダウンロードして VSCode で開きます。
期待される画面: result に止まった理由の文章、permission_denials に拒否されたツールと入力が入ります。

記録するのは、要約行の3つの数、result の最初の1文、permission_denials、MR ができたかどうかです。

turns と cost_usd の数字は表示例です。値は実行ごとに変わります。

朝に見るのは5つの値だけ

キー	意味	朝の判断	1回目に見えた値
is_error	異常終了か	true なら中身を読む前に止める	_____
num_turns	使ったターン数	上限(15)に張り付いていれば Issue が大きすぎる	_____
total_cost_usd	この1本の費用	事前計測値と比べて倍なら見直す	_____
result	Claude の最後の文章	「受入条件が特定できない」等の 停止理由がここに出る	_____
permission_denials	拒否されたツール	1件以上なら権限で止まった。 ツール名と入力を 157 へ写す	_____

ci/nightly_implement.sh は行1〜3 を turns / cost_usd / is_error の1行に要約します。
朝に最初に開くのは result です。止まった理由と 148 の予想1を見比べます。
権限で拒否された操作は .err ではなく、この JSON の permission_denials に入ります。

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

症状ごとに直す層が決まる台帳

止まり方	ログで見える症状	直す層	直す設定(例)
権限で拒否された	JSON の permission_denials。 curl なら手順1、git switch や push なら手順3・7	プロンプト (スクリプトの PROMPT)	PROMPT に _issues/issue-4.md を読ませる1行を足す。curl は deny にあり allow では通らない
7.4 外の構文が混入	MR の lint-phpver が赤。 match 式や str_contains()	手元の hooks と CI のジョブ	手元は check-phpver。CI は lint-phpver の1段目と2段目
テストが落ちたまま	test または test-phpunit が赤	skill の手順6 と Stop hook	落ちるテストを先に書く。 CI では test ジョブが受ける
レビューを受けず	ai_review が動いていない	CI のジョブの条件	merge_request_event でだけ動く MR が立つまで走らない
指摘を全部採用	差分が Issue の範囲外に及ぶ (stock_status_name() など)	人と REVIEW.md	範囲外だけ戻す部分ロールバック 採否は朝に人が MR に書く
ターン上限で終了	num_turns が 15。 コミットが途中で終わる	Issue の粒度	Issue を分ける。 --max-turns は上げない

1回目に当たった行に印を付け、直す層を1つ選びます。選んだ理由を右端に1行書きます。

行1 の直し方は1通りです。curl は ci/settings.ci.json の deny にあります。行4 の多くは MR がまだ無いだけです。

出典: <https://code.claude.com/docs/en/permission-modes>

自分のブランチから流す2回目

Step5-1

157 で選んだ直し方を、1か所だけ実装します。
Issue を読めずに止まった場合は、ci/nightly_implement.sh
の PROMPT に _issues/issue-4.md を読ませる1行を足します。
期待される画面: 差分は直したファイルの1か所だけ

Step5-2

直したファイルだけを ex/h3-<ユーザー名> にコミットし、
ソース管理ペインのブランチの発行(2回目以降は変更の同期)で送ります。
期待される画面: Code > Branches に自分のブランチが出る

Step5-3

Build > Pipelines で、push で動いた ex/h3-<ユーザー名> の行を
開き、implement の段の nightly_implement の再生ボタンを押します。
スクリプトが作るブランチは nightly/issue-4-ex-h3-<ユーザー名> です。

Step5-4

流し直すときは、先に Code > Branches でそのブランチを消します。
完走すればログ末尾は「MRを作成しました。マージは人が判断します。」

直す層は1つだけ

157 に挙げた層から
効くものを1つ選びます。
2つ以上を同時に直すと、
どちらが効いたのかを
2回目で判別できません。

選んだ理由は台帳の
右端に1行で残します。
2回目で外れたときに、
次の層が決まります。

2回目は自分のブランチから流すので、1回目とも他の人とも重なりません。別の行に当たった人は、その行にも印を付けます。

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

朝に人が開く場所は3か所

MR nightly: Issue #4

nightly/issue-4 から main へ



パイプラインは2本 ブランチ用と MR 用

ブランチ用(push で走る検査4つ)

MR 用

lint-phpdev

test

ai_review

lint-phpver

test-phpunit

ai_gate



朝に開く順番

- 1 **ai_review のコメント**
MR の説明欄も読みます
- 2 **検査4つが緑か**
Pipelines タブで見ます
- 3 **Changes の差分**
1 と 2 が揃ってから
Merge は押さず、講師へ伝えます。

ai_review は findings.json を出して MR にコメントを1本置き、ai_gate が P0/P1 で止めます。
夜間の MR も人の MR と同じ2本を通ります。Pipelines タブに branch と merge request の2行が並びます。

ai_gate が赤なら、Pipelines must succeed を有効にしたプロジェクトでは Merge が押せません。
研修のプロジェクトでは押せますが、押しません。受入条件の1と2は Changes タブの検索で確かめます。

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

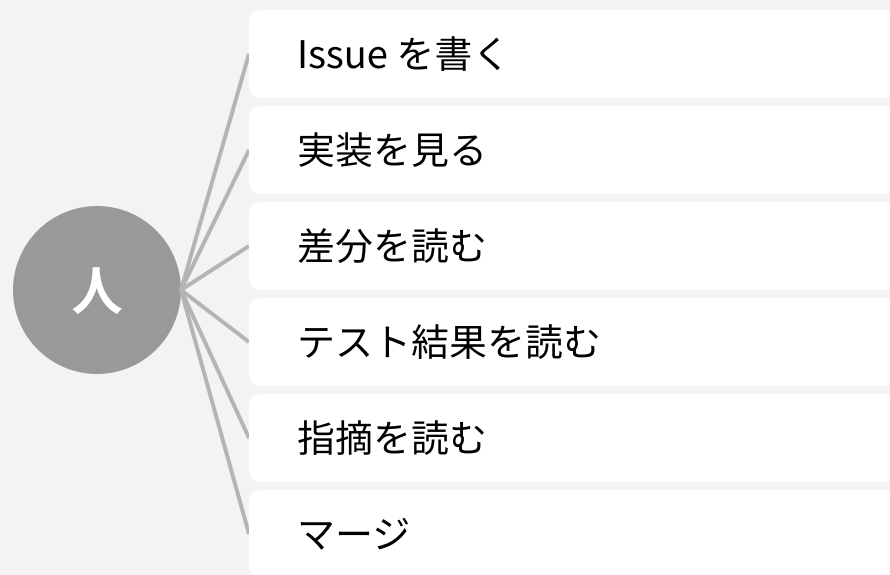
夜に任せるのは行1と行2だけ

変更または判断	可逆性	機械検証	影響範囲	置き場
4か所の対応表を util.php の1か所に寄せる	高	lint と test	変更内	夜(AI)
run_tests.php に対応表のテストを3件足す	高	CI が回す	変更内	夜(AI)
ai_review の P0/P1 指摘の採否	指摘による	できない	指摘による	朝(人)
stock_status_name() にも同じ直しを広げる	中	一部	変更外	朝(人)
db/schema.sql と db/seed.sql の変更	低	できない	変更外	朝(人)
main へのマージ	低	できない	全体	朝(人)

この表は埋めません。148 で書いた「夜間に任せない変更」がどの行に当たるかを読み合わせます。
行4 は Issue の範囲外です。skill は提案で止まり、広げるかどうかは朝に人が決めます。
残りの時間は 157 の台帳記入に回します。

コードを読む係はループの外

今のハーネス



夜間ループ

夜 AI が回す

ブランチ ▶ 実装 ▶ テスト ▶ レビュー ▶ MR



朝 人が決める(124 と同じ3点)



Issue の粒度決め

指摘の採否

マージ承認

朝に読むのは MR の形になったものだけです。それ以外を読む必要が出たら、止め金かゲートが足りていません。

CI は無いので AI の実行報告が機械強制の担保という定義を、夜間ジョブの CI が元の意味に戻します。

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

自分の手で塗る回せる層と残る層

項目	D2	smart3pm	直す層
-p 非対話で回る implement 系の skill がある			
止め金4層(保護・timeout・max-turns・deny)がある			
Stop hook が stop_hook_active を見て止まる			
対象バージョンの検査が手元の PostToolUse と CI の両方にある			
テストが CI で回り、結果が MR に残る			
レビューが CI で findings.json に出る			
P0 と P1 でゲートが落ちる			
MR が schedule から自動で立つ			

1人1枚で塗ります。D2 と smart3pm の列に「回せる」「一部」「回せない」を書き込みます。

自社の GitLab 導入前なら、下から4行は「回せない」で構いません。

「回せない」の行に直す層を1つ書きます。161 の残る3点は、残す前提なので表に入れません。

人が押すのはpush と MRだけ

runner から外へ出られない場合は、Step3 と Step5 の CI 実走を講師環境で行います。受講者は同じ手順を Claude Desktop で回します。止め金の効き方は同じで、違うのはこの2か所だけです。

- 1 VSCode の左下のブランチ名から nightly/issue-4-<ユーザー名> を作ります。
- 2 Claude Desktop で dl-training-app を開き、パーミッションモードを 自動 にします。
- 3 入力欄に /implement-issue 4 を入れます。ブランチ作成と push は人が行う、と添えます。
- 4 終わったら、使ったターン数と、拒否されたツールの行を控えます。
- 5 VSCode のソース管理ペインで差分を見て、ブランチの発行で GitLab へ送ります。
- 6 GitLab の Code > Merge requests から MR を作ります。以降は Step6 と同じです。

達成状態

手元で /implement-issue 4 が終わり、コミットが1つ以上ある
GitLab に自分のブランチが届き、MR が1本立っている

手元では check-phpver が Edit と Write の後に走ります。CI では動かない層が効くので、通る範囲がここで変わります。

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

CI は講師環境・手元は Claude Desktop



CI 側(講師環境で実走)

講師が Step3 と Step5 を講師アカウントで実走し、共有します。

受講者は 155 のログと 159 の MR を見て台帳を書きます。

2回目も講師環境で流します。



手元(受講者の Claude Desktop)

163 の代替手順をそのまま使います。
ブランチは VSCode の左下から作ります。

Claude Desktop のパーミッションモードを自動にして /implement-issue 4 を回します。

push は VSCode のソース管理ペインのブランチの発行、MR は GitLab の Web 画面で作ります。コマンドは打ちません。

※ 各ロゴは各社の商標です。

GitLab runner から api.anthropic.com へ出られない場合の切り替えです。演習の順番と台帳は変えません。

止め金は同じです。手元では PostToolUse の対象バージョン検査も効き、その差自体が教材になります。

4つの OK と4つの詰まりを照合

OK 基準		確認方法
☐	講師の1回目の result と、148 の自分の予想を見比べた	157 の行の印
☐	2回目の MR で lint と test の4ジョブが緑(講師環境の MR でも可)	MR の Pipelines
☐	受入条件の3点を Changes で確かめ、マージ可否と根拠を講師へ伝えた	Changes と計測表
☐	162 の8行が D2 か smart3pm のどちらかで埋まった	自分の1枚

つまずき	原因	直し方
数秒で終わり「ANTHROPIC_API_KEY が未設定です」	変数が未投入	講師に知らせ 164 へ
2回目も「変更がありません。」で終わる	直したコミットが未 push	Branches で自分のブランチを見る
流し直して push が拒否された	同じ名前のブランチが残る	Step5-4 のとおり消す
lint-phpver が赤で MR が立った	対象外の構文の混入	直さず 157 の行2 に印

OK3 は受入条件1・2・4。'作成中' が util.php だけ、2つのビューに array(1, 2, 3, 4, 9) が無い、テストが3件増えた。
 OK が4つ揃った人は、夜間ジョブ一式を台帳の D2-08 の行に書きます。OK2 が出ない人はつまずき2 から見ます。

台帳の1点に数える夜間ジョブ一式の中身

持ち帰り物	D2(PowerShell 5.1・ASCII)	smart3pm(bash 正)
.gitlab-ci.yml の nightly_implement ジョブ ci/nightly_implement.sh	対象リポジトリに追記。ISSUE_ID の 既定を自社の Issue 番号へ ci/ に置く。手元で回すなら ps1 に写し、文字列は ASCII に保つ	同じ。PHP が要らない案件は before_script の php-cli を外す ci/ にそのまま。bash が正なので 書き換えは要りません
.claude/skills/ implement-issue/SKILL.md 止め金の settings.json deny 13本・hooks 4種	手順1 の Issue 取得先を 自社のプロジェクトパスへ 既存の settings.json に追記。 command は powershell.exe のまま	同じ場所。手順5 の構文制約を 対象の実行環境に合わせる command を bash、args を .sh の パスにした settings.json
pipeline schedule の設定値 Description・cron・Target	GitLab の設定。Variables まで 台帳に設定値を控えます	同じ。複数リポジトリに作るときは 台帳側に一覧を持ちます

上の2行は D2 と smart3pm で中身が同じです。CI runner は Linux なので、フックの二重運用が要らない層です。

行4 は maxEffortLevel も同じファイルです。effort の上限はこのキー1本で効きます。

台帳の D2-08 の行は夜間ジョブ一式の1点です。この5つと 155 の turns、cost_usd を書きます。

マージだけが人に残った状態を**自社で再現**

夜にブランチが切られ、朝にマージボタンだけが待っている状態を、自社の Issue 1本で作ります。

研修後の発展

- 自社の Issue 1本を、機械で判定できる受入条件の形に書き直します。夜間に任せられるかを先に決めます。
- 夜間ジョブは `--settings ci/settings.ci.json` で bash 版のフックを登録しています。
自社の runner に合わせてこのファイルを書き、Stop を足すかは DB の有無で決めます。
- ai_review の後に P0/P1 を直すジョブを足したくなったら、157 の行5を読み直してから決めます。
- 発展1〜3 は D2-10 の社内展開ガイドに、壊れた時の戻し方と一緒に書きます。

出典: <https://code.claude.com/docs/en/settings>

D2-09

理想的なハーネスの基準と雛形

空いている層は、雛形と並べると見つかる

公開されている基準と到達点の数字に自社の設計を当ててから、5層の雛形を実物で見ます。
そのあと自社のハーネスを30項目で点検し、空いていた1層を自社の言葉に置き換えます。
ハンズオン3で回した夜間ループの結果も、ここで雛形の層と突き合わせます。

[85min]

持ち帰り: 雛形一式 / 埋めた診断シート / 書き換えた1ファイル

突き合わせる相手は公開されている基準

出どころ	書いてあること	この章で使う所
Claude Code 公式 Best practices	CLAUDE.md は短く。太ると指示そのものが無視される。 機械で守れるものはフックへ。検証手段を必ず与える	CLAUDE.md の指針
Anthropic Engineering 長時間動くエージェントのハーネス	初期化と実装の二層、進捗ファイル、機能リスト、 4つの失敗モードと対策	二層構成 4つの失敗モード
Anthropic 2026 Agentic Coding Trends Report	8つのトレンド、実名の事例と数字、 2026年の4つの優先事項	到達点の数字 4つの優先事項

3本とも一次情報です。この章の数字と表は、すべてこの3本から取っています。

自社の設計を評価するのは受講者自身で、投影するのは基準の側だけです。

人と AI が分担して1本の改修を進める流れは、AI協働開発デモ aidevdemo.give-app.net で追えます。

出典: code.claude.com/docs/en/best-practices、resources.anthropic.com/2026-agentic-coding-trends-report
anthropic.com/engineering/effective-harnesses-for-long-running-agents

初期化と実装を別のプロンプトに分ける

初期化エージェント

最初のセッションだけ動く

- init.sh に環境の起動手順を固定する
進捗ファイルを作る(作業ログの置き場)
機能リストを JSON で作る。全部 passes: false
最初の git commit を打つ

-
-
-

記憶は会話ではなく git とファイルに置く

コーディングエージェント

毎セッション同じ手順で動く

- git log と進捗ファイルを読んでから始める
- 機能リストから1つだけ選ぶ
- E2E テストで壊れた状態を先に検知する
- 1機能ごとに commit し、進捗を書く

機能リストで書き換えてよいのは passes だけ

1回で全部を作らせず、1機能ずつ進めます。セッションをまたぐ記憶は git と進捗ファイルが持ちます。

D2-08 で作った夜間ジョブは、この右側の手順を CI のジョブに置き換えたものです。

出典: anthropic.com/engineering/effective-harnesses-for-long-running-agents

4つとも夜間ジョブに同じ対策がある

失敗モード	初期化で置く対策	毎セッションの対策	D2-08 の夜間ジョブ
完了を早く 宣言してしまう	機能リストのファイル	リストから1機能だけ選ぶ	Issue 1本に MR 1本 ai_gate が合否を決める
壊れた状態を 放置してしまう	git の初期化	開始時にテストを回す 終了時に commit する	lint と test が先に走る
テスト不十分のまま 完了扱いにする	機能リストのファイル	慎重に試してから passes を付ける	PHPUnit のジョブ ai_review の findings
動かし方の把握に 時間を使ってしまう	init.sh	開始時に init.sh を読む	before_script に固定

公開された基準と、今日作ったものが1対1で対応します。この表に無い失敗モードが出たら、それが自社固有の設計です。右の列を自社の夜間ジョブ案に書き換えると、そのまま診断シートの第4層になります。

出典: anthropic.com/engineering/effective-harnesses-for-long-running-agents

残す基準は消すと間違えるか

太った CLAUDE.md

- 行が増えるほど、後ろの規則ほど効かない
- 決めた経緯や日付が規則に混ざっている
- 機械で守れる規則が文章のまま残っている
- Claude が既にできることまで書いてある
- たまにしか要らない知識が常時読まれる

公式の言い方は「太った CLAUDE.md は指示そのものを無視させる」です。



削った後

- 1行ごとに「消すと間違えるか」を問う
- 経緯は決定ログへ。規則だけを残す
- 機械で守れるものはフックへ移す
- コードを読めば分かることは書かない
- たまに要る知識はスキルに出す

強調したい1行にだけ IMPORTANT を付ける。
多くの行に付けると、どれも目立たない。

行数の上限を決めるより、1行ごとにこの問いを当てるほうが効きます。上限は満たしたまま、注入される量は増えるからです。CLAUDE.md はコードと同じに扱います。効かなくなったら見直し、削り、挙動が変わるかで検証します。

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

任せ切れる仕事は2割までという実測

開発者の使い方(Anthropic の調査)

0～20%

完全に任せられると答えたタスクの割合

約6割 の業務で AI を使っている

27% は AI が無ければやらなかった仕事

速くなる以上に、出す量が増える

実名の事例(同レポート)

- **楽天**
1,250万行の OSS に7時間の自律実行で実装。
参照実装比の数値精度 99.9%
 - **TELUS**
13,000 の社内向け AI ソリューション。
コード出荷が30%速く、累計50万時間超の削減
 - **Augment Code の顧客**
CTO が4～8か月と見積もった案件を2週間で
- どの事例も、人の監督を無くしてはいない

人の監督を無くす話ではなく、人の注意を要する所へ寄せる話です。D2-06 の「人が残る3点」は、この数字と同じ側にあります。数字は 2026年時点の同レポートの記載で、自社の見積もりに置き換えるのは各自です。

出典: resources.anthropic.com/2026-agentic-coding-trends-report

4つのうち3つが2日間に入っている

レポートの優先事項	2日間のどこ	持ち帰るもの
1 マルチエージェントの調整	D2-05 観点別レビュー3本の分業	agents 3本 review-route の skill
2 人の監督を スケールさせる	D2-06 と D2-08 夜間ループと人が残る3点	nightly_implement ai_review と ai_gate
4 設計の最初から セキュリティを入れる	D1-07・D1-08・D1-10 deny とフックと秘密情報の遮断	deny 4種5本、check-phpver block-destructive、.env.tpl
3 エンジニアリングの 外へ広げる	今回は扱わない	発展: 法務や営業が自分で 作った事例が同レポートにある

レポートが「今すぐ手を付ける4つ」と呼ぶもののうち、3つは既に手元にあります。
残る1つは、開発部門の外に広げるときの話で、その前に開発部門の中で回っている必要があります。

出典: resources.anthropic.com/2026-agentic-coding-trends-report

本数ではなく **自社の層との対応**で選ぶ

ハーネスの層	外の実装(GitHub)	何をするか	今日これを見る理由
案件依存の知識を供給する層	colbymchenry/codegraph Graphify-Labs/graphify	コードをグラフ化して渡し、 変更に従うさせる	横断知識を「うまく調べる」に 賭けない、計画書とは別の解き方
常時注入する規則の層	multica-ai/ andrej-karpathy-skills	CLAUDE.md 1本で 振る舞いを変える短い型	前ページの問いを通した後の 形を、実物で見る
手順を固定する層	obra/superpowers addyosmani/agent-skills mattpocock/skills	手順をスキルに落とし、 呼ばれる単位で持つ	文章で守っている手順を、 呼び出される単位へ移す方法

話題の順ではなく、自社ハーネスのどの層に当たるかで選んでいます。当たる層が無いものは、今日は見ません。
6本とも講師が手元で使っているものです。Star 数は 2026年9月15日時点で7万から29万です。

出典: github.com の各リポジトリ(2026-09-15 時点)

手元の設計に3つの問いを当てる

問い	見る場所	書く先
1 4つの失敗モードのうち、 自社に対策が無いものはどれか	CIの設定と、夜間に回す スクリプトがあればそれ	診断シート 第4層の欄 無いものに番号を書く
2 CLAUDE.mdのうち、 消しても間違えない行は何行あるか	自社の CLAUDE.md 1本。 行数を数えてから読む	診断シート 第1層の欄 行数と、消せる行数
3 4つの優先事項のうち、 自社で一番遅れているのはどれか	2つ前のページの表。 「持ち帰るもの」の列を自社に置き換える	診断シート 末尾の自由欄 番号1つと理由を1行

答えは投影しません。手元で書いた3行は、この後のプチ演習7でそのまま診断シートの下書きになります。
問い2で「消せる行」が全体の半分以上を超えたら、プチ演習8の置き換え対象は CLAUDE.md にするのが早いです。

見る相手は自社の設計だけです。他社の設計や講師の例は使いません。

同じ危険を2つ以上の層で受ける設計

層	置き場所	効く範囲	外れたとき
文脈	CLAUDE.md / AGENTS.md	AI の判断	気づけない
権限	.claude/settings.json	ツール呼び出し	気づけない
フック	.claude/hooks/	実行の直前	テストで分かる
CI	ci/ とパイプライン定義	提出のとき	落ちて分かる
リポジトリ保護	ホスティング側の設定	手元を外した人	効き続ける

上の層ほど外しやすく、外れたことが画面に出ません。下の層ほど後から効いてきます。

手元のフックはうっかりを止めるものです。消されて困るものはCIとリポジトリ保護に置きます。

11個の空きが全部止まらずに通る側へ倒れる

空いている場所		何が起こるか
1	機械で検査できるルールが通知どまり	守られないまま通る
2	改行コードや文字コードの取り違い	保護が黙って無効になる
3	破壊系の判定が綴りの列挙頼み	並べ替えた形が素通りする
4	編集ツールで matcher が不揃い	一部のツールから抜ける
5	変数で組み立てたパスが素通り	止めるにも確認にも当たらない
6	取り消し手順が配る手順と対でない	事故の場で判断が割れる
7	秘匿の検査が片側しか塞がっていない	別の経路から文脈へ入る
8	代替のない禁止	別の経路を探して回避される
9	文書と実装のずれ	どちらが正か分からなくなる
10	コストが方針どまり	気づくのは請求書が来たとき
11	フックにテストが無い	壊れても緑に見える

2本のハーネスを読んで出てきた一覧です。雛形の各要素は、このどれを塞ぐかで説明できます。

保護が壊れた日は画面に何も出ない

壊れ方

判定に使う道具が入っていない

設定が CRLF で保存された

実行権限が落ちた

想定と違うオプションの並び



結果はどれも同じ

止まらずに通る

ログにも画面にも何も残りません。
作業は成功したように見えます。
壊れた日と気づいた日の間の差分は、
あとから人が読み直すことになります。

気づく手段は3つです。テストを流す・止めたい入力を手で流す・CIで同じ検査を動かす。

確かめる場所を決めていないハーネスは、置いた日から一度も動いていない可能性があります。

判定できないときは通さない

よく見る書き方

```
command -v jq >/dev/null || exit 0
```

```
judge "$1" || true
```

```
# no jq, no judgement, no log
```

```
# the call just goes through
```

雛形の書き方

```
command -v "$JQ" >/dev/null 2>&1 \
```

```
|| die "jq not found."
```

```
[ -f "$RULES" ] || die "no rules."
```

```
[ -n "$payload" ] || die "empty."
```

```
# die() prints a reason, exit 2
```

道具が無い端末では、何も見ずに通ります。

判定できない状態は、止める側へ倒します。

止める条件は3つです。判定に使う道具が無い・規則ファイルが読めない・入力が空。

止める対象を消す・上書きする・外へ送るに絞っているので、読む操作は止まりません。

フックは動かなくなったことが見えない

45

雛形の受入テストの件数

止める側 32 件 | 通す側 13 件

止める側だけでは足りません。
通す側が無いと、誤検知が
増えたことに誰も気づけません。

止めたい入力

```
rm -fr build  
rm --recursive --force build  
find . -name tmp | xargs rm -rf  
cat .env
```

通したい入力

```
grep -rf patterns.txt src  
rm -i notes.txt  
docker rm -f box  
cat .env.tpl
```

通す側に `grep -rf` を置いています。綴りで照合する実装では、ここが誤検知になります。

止める対象は**1つの設定ファイル**に置く

rules.json

止める動詞・オプション・引数
保守するのはここだけ

並べ替えも長い綴りも、
同じ1本の規則に当たります。



guard.sh

bash の端末で動く入口

guard.ps1

PowerShell の端末で動く入口

これから足す入口

読んで照合して終了コードを返すだけ

入口をシェルごとに書くと、同じ判定を2つの言語で保守することになります。片方だけ直る日が来ます。
規則を1件足すときに触るのは、rules.json と guard.test.sh の2か所だけです。

同じ危険を複数の層で受けて数は機械で数える

原則1 層の分離

秘密情報を3か所で受けている例

permissions の読み取り禁止

第2層 権限

rules.json の secret-file

第3層 フック(Bash 経由を見る)

write_rules

第3層 フック(4つの編集ツールを見る)

1層だけで守っているものは、その層が
黙って壊れた日から無防備になります。

原則8 文書と実装の照合

README に書いてある本数

止める規則 12 本

編集を止める規則 4 本

観点別レビュアー 3 本

受入テスト 45 件

verify-harness.sh がこの数を読み、
実装の数と突き合わせます。片方だけ直すと落ちます。
文書だけが検査の外に残ると、そこから腐ります。

残る5つ(代替の併記・人が見る場所・環境差の外出し・戻す手順・費用の上限)は、運用が始まってから効きます。

コピーして名前を直すだけで動く一式

```
template/
├── CLAUDE.md
├── AGENTS.md
├── .claude/
│   ├── settings.json
│   ├── rules.json
│   ├── review.schema.json
│   ├── agents/
│   └── hooks/
│       ├── guard.sh
│       ├── guard.ps1
│       └── guard.test.sh
├── ci/
└── docs/
```

中心になる3ファイル

rules.json

止める対象そのもの

guard.sh

読んで照合するだけの薄い入口

guard.test.sh

効いているかを知る唯一の手段

他のファイルは、この3つを支えるために置いています。入れるときは1層ずつ進め、効いていることを確かめてから次へ行きます。

ci/ には閾値で落とす gate.sh と、文書と実装のずれを見る verify-harness.sh が入っています。

並べ替えも長い綴りも**同じ1本の規則**で受ける

.claude/rules.json(抜粋)

```
{
  "flag_aliases": { "recursive": "r", "force": "f" },

  "deny_rules": [
    { "id": "rm-recursive-force",
      "verbs": ["rm"], "flags_all": ["r", "f"],
      "message_en": "rm removes a tree. Ask a human." },

    { "id": "secret-file",
      "args_regex": "(\\.env|id_rsa|[^/]+\\. (pem|key))",
      "args_except_regex": "\\ .env\\. (tpl|example)" }
  ]
}
```

-rf も -fr も -r -f も --recursive --force も、語に分けるとオプションは r と f の2つです。

止める規則を1件足すときに触るのは、このファイルと guard.test.sh の2か所だけです。

入口は読んで照合して止めるだけ

.claude/hooks/guard.sh(抜粋)

```
RULES="${GUARD_RULES:-$HERE/../../rules.json}"
JQ="${GUARD_JQ:-jq}"
die() { printf '[guard] %s\n' "$1" >&2; exit 2; }

command -v "$JQ" >/dev/null 2>&1 || die "jq not found."
[ -f "$RULES" ] || die "rules file not found."
payload="$(tr -d '\r')"
[ -n "$payload" ] || die "empty payload."

hit="$(printf '%s' "$payload" | "$JQ" -r "$PROGRAM")"
[ -z "$hit" ] || { printf '%s\n' "$hit" >&2; exit 2; }
```

規則の置き場所と jq の場所は環境変数で差し替えられます。コードに埋めません。(原則7)

判定の前に CR を落とします。設定が CRLF で保存されても同じ判定になります。(空き2)

止めるときは終了コード2と理由を返します。AI はその理由を読んで書き直します。

rm -fr は止まり grep -rf は通る

止めたい入力

```
rm -fr build
```

終了コード 2 でブロック

並べ替えた形も同じ規則に当たります。

通したい入力

```
grep -rf patterns.txt src
```

終了コード 0 で通過

動詞が grep なので対象外です。

```
$ bash .claude/hooks/guard.test.sh
```

```
guard.test(sh): 45/45 passed
```

```
guard.test(ps): skipped (PowerShell not found on this machine)
```

45件のうち止める側が32件、通す側が13件です。通す側が無いと、誤検知が増えても気づけません。

PowerShell がある端末では、同じケースを guard.ps1 にも流します。この端末には無いので飛ばしています。

文書で守ることを諦めて機械へ落とし切っている

機械判定への置き換え

文章で守らせることを諦めた箇所を、権限とフックへ落とし切っています。

事故から機構への置き換え

運用上の注意をやめ、内容が一致するかを機械で確かめる形へ変えています。

ガードの限界の明示

確定できない形は止めるではなく確認に落とし、限界を冒頭に書いています。

通す側の回帰テスト

誤検知になりやすい入力を通す側として残し、塞ぐたびの往復を止めています。

検査の正を1か所に

同じ検査を2か所に書かない理由まで書き、呼び出し元を1本にそろえています。

ここまで作り込まれた社内ハーネスは多くありません。ここから先は、この上に何を足すかの話です。

空きを雛形のどこが埋めるかで見える

空いている場所	雛形のどこが埋めるか	次の一手
破壊系の判定が綴りの列挙	語に分けて集合で照合する規則	並べ替えを3つ試す
テストのあるフックと無いフック	止める側と通す側を並べた検査	まず20件から始める
改行コードで保護が黙って無効	判定の前に CR を落とす入口	CRLF の検査を1件足す
編集ツールで matcher が不揃い	4つのツールを同じ期待値で並べる	matcher の一致を検査
取り消し手順が配る手順と対でない	戻す単位を4つに分けた手順書	戻す単位を4つに分ける

埋める順はテストとフェイルクローズが先です。効いているかどうかを知る手段が、ここにしかありません。
全部を一度に埋めないでください。次の1週間で埋めるものを1つだけ選びます。

表を開く前に予想を先に書きます

1

予想を書く

いちばん空いていそうな層を1行で書きます。

2

30項目を埋める

ある・一部・無いの3つだけで判定します。

3

予想との差を見る

食い違った項目が、いちばん危ない場所です。

4

1つだけ選ぶ

次の1週間で埋める項目を1つだけ決めます。

判定は3つだけ

ある 確かめた

一部 置いてあるが効いていない

無い 置いていない

このあとのプチ演習7

診断シートを開いて、今日作った設定を自分で点検します。[15min]

「ある」は、確かめた場合だけ付けてください。置いた記憶があるだけのものは「一部」です。
第1層だけが埋まっていて他が空いている状態が、いちばん危ない状態です。

D2-10

持ち帰り台帳の確定と社内展開ガイド

作った人がいなくても回る形で持ち帰る

2日で足した17点の置き先と担当を確定し、作成者以外が読んで運用できる A4 1枚をその場で書きます。

[20min]

持ち帰り: 持ち帰り台帳 確定版 / 社内展開ガイド A4 / 研修後の伸びしろ3点

2日間で足した17点の出どころ

プチ演習8本とハンズオン3本の成果物17点

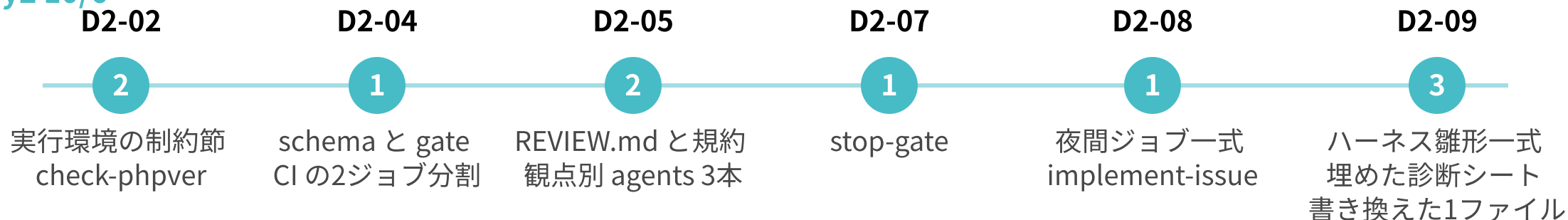
Day1 9/29



Day1 7点
Day2 10点

17点

Day2 10/6



各演習の最後にした台帳の行を、この17点に揃えます。丸の中の数字が行数です。
1点は「1つの置き先に1回で置ける単位」です。観点別 agents 3本は1点に数えます。
D2-05 の生成テストと検出率は dl-training-app 側に残るので台帳には入れません。

Day1 7点の置き先

設定系は両ハーネスとも同じパスに置ける7点

No	持ち帰り物	出どころ	D2 の置き先	smart3pm の置き先
1	規模別レビュー2本 review-light.md / review-full.md	D1-05	.claude/agents/	.claude/agents/
2	settings.json のモデルと effort model ・ effortLevel ・ maxEffortLevel	D1-05	.claude/settings.json	.claude/settings.json
3	deny 5本(巻き戻せない操作)	D1-07	.claude/settings.json permissions.deny	.claude/settings.json permissions.deny
4	CLAUDE.md の「戻し方」節	D1-07	CLAUDE.md	core.md との重複を先に確認
5	deny 3本 .env と認証情報の読み取り	D1-10	.claude/settings.json permissions.deny	.claude/settings.json permissions.deny
6	PreToolUse フック block-destructive (ps1 / sh)と cases.json のケース	D1-10	hooks/ (ASCII のみ)	hooks/ の bash 版 block-destructive.sh
7	実装とレビューを分ける定義1本 review-other(履歴を切ったセッション)	D1-11	.claude/skills/ か .claude/agents/	.claude/skills/ か .claude/agents/

7点のうち6点は JSON と Markdown なので、両ハーネスで同じパスに置けます。

言語が分かれるのはフック1本です。7 には MR テンプレの「AI レビュー結果」欄も含めます。

置き場所はディレクトリの実パスまで書く1行

持ち帰り台帳.md

```
| No | 持ち帰り物 | 置き場所 D2 | 置き場所 smart3pm | 担当 |  
| --- | --- | --- | --- | --- |  
| 1 | review-light.md / review-full.md | .claude/agents/ | .claude/agents/ | |  
| 2 | model / effortLevel / maxEffortLevel | .claude/settings.json | .claude/settings.json | |  
| 3 | deny 5本(巻き戻せない操作) | .claude/settings.json の deny | .claude/settings.json の deny | |  
| 6 | block-destructive.ps1 + cases.json | hooks/ (ASCII) | hooks/block-destructive.sh | |  
| 9 | check-phpver(ps1 / sh) + cases.json | post_edit_checks | hooks/check-phpver.sh | |  
| 13 | stop-gate(ps1 / sh) | hooks/ (ASCII) | hooks/stop-gate.sh | |
```

「settings に入れる」ではなく、ファイル名とキー名まで書きます。外すときに探さなくて済みます。
担当の列は研修中に埋めます。空欄のまま持ち帰る行を作らないでください。
両ハーネスで同じなら同じ文字列を2回書きます。「同じ」と略すと後で片方を直し忘れます。

Day2 10点の置き先

CI に触る2行はGitLab 導入後に置く10点

No	持ち帰り物	出どころ	D2 の置き先	smart3pm の置き先
8	CLAUDE.md の実行環境制約節	D2-02	CLAUDE.md の制約節	CLAUDE.md の制約節
9	check-phpver フックと cases.json	D2-02	PostToolUse フック	hooks/check-phpver.sh
10	REVIEW.md と Code Review Rules	D2-05	リポジトリ直下と CLAUDE.md	REVIEW.md と CLAUDE.md
11	観点別レビューアー3本と known_issues	D2-05	.claude/agents/	.claude/agents/
12	review.schema.json と gate、 2ジョブに分けた .gitlab-ci.yml	D2-04 D2-05	.claude/ 直下 CI は GitLab 導入後	.claude/ 直下 + gate.sh CI は GitLab 導入後
13	Stop フック stop-gate(ps1 / sh)	D2-07	hooks/ (ASCII のみ)	hooks/stop-gate.sh
14	夜間ジョブ一式(nightly_implement・ schedule・implement-issue・止め金)	D2-08	.gitlab-ci.yml・ .claude/skills/	.claude/skills/・ settings.json
15	ハーネス雛形一式	D2-09	リポジトリ直下に展開	リポジトリ直下に展開
16	埋めた診断シート	D2-09	docs/ に置く	docs/ に置く
17	書き換えた1ファイル	D2-09	雛形と同じ相対パス	雛形と同じ相対パス

12 の CI 部分と 14 は自社 GitLab が立つまで置けません。台帳には「導入後」と書いて残します。

9 と 13 のフックは D2 では ASCII のみで書きます。日本語コメントも入れません。

文脈・設定・フック・エージェント・CI の5段に同じ形で落ちる

置き場所の段	D2: PowerShell 5.1	smart3pm: bash 正
文脈	CLAUDE.md	core.md と CLAUDE.md
設定	.claude/settings.json	.claude/settings.json
フック	hooks/*.ps1 (ASCII のみ) hooks/tests/cases.json	hooks/*.sh
エージェント とスキル	.claude/agents/ .claude/skills/	.claude/agents/ .claude/skills/
CI	.gitlab-ci.yml(両ハーネスで共通の1本)	

settings の優先: managed > --settings > settings.local.json > settings.json > ~/.claude/settings.json

17点は全部この5段のどこかに落ちます。落ちない成果物は作っていません。

Claude Desktop と CLI は同じ settings.json と CLAUDE.md を読むので、設定は1つで足ります。

社内共有するものは .claude/settings.json に置きます。2026-09-22 時点の 2.1.280 で確認したパスです。

出所: [code.claude.com/docs](https://code.claude.com/docs/settings-precedence) (settings-precedence、desktop)

最初に触る1点と壊れたときの気付き方

1

17点から選ぶ最初の1点

作成者以外が最初に触る1点を選び、台帳の No を書きます。

期待される状態

紙に数字が1つ

2

壊れたときの気付き方1行

誰が・どの画面で・何と表示されて気付くかを書きます。
主語は作成者以外の名前か役割にします。

期待される状態

主語が作成者以外の
名前か役割

3

戻し方の一手

何を外すか、何を戻すかのどちらかを1文で書きます。
ファイル名と行が特定できる形にします。

期待される状態

ファイル名と行の
特定ができる1文

AI に頼む前に、自分の手で3行書きます。ここで主語が自分になった人は書き直してください。

3分で書けない Step 2 は、監視が無いという発見です。台帳の備考に「気付く手段なし」と書きます。

何を・どこに・なぜ・戻し方の4節で1点1枚

社内展開ガイド_check-phpver.md

```
# check-phpver.ps1 の運用
## 何を
Edit/Write の後に PHP 7.4 で使えない構文を検査し、見つけたら exit 2 で差し戻すフック
## どこに
D2: hooks/check-phpver.ps1 と .claude/settings.json の PostToolUse (matcher: Edit|Write)
## なぜ
CLAUDE.md の制約だけでは数ターン後に混入が戻る(Day1 の CI ログで実測)
## 壊れた時の戻し方
settings.json の PostToolUse から該当行を外す。フック本体と cases.json は残す
検証: echo '{"tool_input":{"file_path":"a.php"}}' | powershell -File hooks/check-phpver.ps1
```

4節を埋めたら、Step 2 で書いた「気付き方」を「なぜ」の下に1行足します。

「どこに」はファイル名とキー名まで書きます。台帳の行と同じ文字列にします。

「壊れた時の戻し方」は外す操作だけを書きます。直す手順は書きません。

smart3pm 担当は「どこに」を hooks/check-phpver.sh に、検証を bash 版に置き換えます。

分からない語を AI に挙げさせてから渡す

チェック項目

- ☐ 固有名(フック名・キー名)に1行の説明がある
- ☐ 「どこに」が台帳の行と同じ文字列になっている
- ☐ 「壊れた時」に外す1行が特定できる
- ☐ 検証コマンドが1本ある(Claude なしで回る)
- ☐ 読む相手の役割が書いてある
(運用メンバーか・レビュー担当か)

4つ目が一番落ちます。検証コマンドが無いと、壊れたかどうかを作成者に聞くことになります。
5つ揃ったら台帳の備考に「ガイド済」と書きます。

壁打ちの頼み方

頼むのは分からない語の指摘だけです。

Claude Desktop

プロジェクトフォルダを選んで送ります。

社内展開ガイド_check-phpver.md を、
ハーネスを作っていない運用メンバー
として読んでください。意味が取れない
語を5つまで挙げ、直し方は書かないで
ください。

履歴を切ったセッション

新しいセッションで同じ文を送ります。
先入観なしで読めるので抜けが出ます。

台帳の1行を指して外し方まで言えれば合格

Step 1 指名

講師が、作成者以外から1名を指名します。現地を優先し、リモートは文章で提出します。

Step 2 説明

自分の台帳の1行を画面共有で指し、ガイドの4節の順に30秒で話します。

Step 3 聞き取り

聞き手全員が、聞き取れなかった節を1語でチャットに書きます。

Step 4 修正

一番多かった節を講師が読み上げ、人ではなくガイドのその節を直します。

OK 基準

聞き手の1名が「どこに」と「戻し方」を自分の言葉で言い直せる

話すのは作成者ではありません。作成者は聞き手として、言い間違いをメモします。
30秒で4節を言えないときは「なぜ」が長いのが普通です。直す対象はガイドの方です。

足すと効く3箇所と期日の決め方

伸びしろ	今の状態	足すと効く一手	期日の決め方
テストの無い 検査スクリプト2本	受入25本の外に、ユニット別の破壊系と編集不可の2本が残る	受入テストと同じ様式で deny と allow を20ケース	次の検査改訂の日
ASCII 制約を守る 機械が無い	ASCII のみの約束が文章にだけあります	post_edit_checks に非 ASCII 検査を1本足す	3行目より前
bash と PowerShell の二重運用の解消時期	同じガードが2言語にあり、寄せる先が未定	寄せる先を決める条件を1つ決めます	年末から年明けの一斉切替より前

時期 一斉切替の時期は「年末から年明け」と伺っています。日付は書きません。

2日間で触らなかった3点です。新しく作るものではなく、片側の仕組みをもう片側へ写す作業です。
2行目を3行目より先にやります。期日は日付で書かず「次に何をする時」で書き、台帳の備考に残します。

二重運用を解消する条件の決め方

寄せる先はテストが揃っている側

今

D2: PowerShell 5.1 のフック群

block-destructive

check-phpver

stop-gate

smart3pm: bash のフック群

同じガードを2言語で2回書きます。

条件を1つ決め
期日を置く

寄せた後

正: テストが揃っている側の1言語

3本のガードはここにだけ置きます。

もう一方: 正を呼ぶ薄い口

Windows 11 の PowerShell から
同じ結果が出ることだけ確認します。

消さずに残し、正だけを直します。

どちらに寄せるかを好みで決めると決まりません。条件を1つ置きます。

推しは「回帰テストが揃っている側を正にする」です。178 の1行目を済ませた時点で自動で決まります。

終了前の5項目

機密ハーネスを端末に残さない退室

やること	誰が
☐ 受講者端末から d2_harness と smart3pm_harness の zip と展開先を削除	受講者全員
☐ 持ち帰り台帳(担当の列まで記入済み)の提出。リモートはチャット経由	受講者全員
☐ 社内展開ガイド A4(自分で選んだ1点)の提出	受講者全員
☐ 講師所見メモの破棄	講師
☐ 作業指示書の Notion 反映は社内の宿題(7/17 に連携済み)	取出様・平野様側

ハーネス2式は規約第7条の機密情報です。演習用に展開したフォルダも含めて消します。
提出物は台帳と A4 の2つだけです。Notion 側は研修では扱いません。

持ち帰りの判定基準

作成者以外が外し方を言える17点

作成者以外が、台帳の行を指して
外し方を言える。それが持ち帰れた状態です。

実パスの台帳

4節の A4

30秒の説明

ファイル17点は配布 ZIP にも入っています。持ち帰りの価値はこの状態にあります。
言えなかった節が1つでもあれば、その行はまだ持ち帰れていません。研修後に同じ形で直します。

台帳とガイドと宿題3点を D2-11 へ

持ち帰り台帳 確定版

17行、置き場所は実パス、
担当の列まで記入済み。
置けない2行は「導入後」
と書いて残します。
D2-05 で埋まらなかった
OK 基準の番号も書きます。

社内展開ガイド A4

自分で選んだ1点を4節
で書きます。残り16点
は研修後に同じ雛形で
書きます。

研修後の伸びしろ3点

フックテストの非対称、
ASCII 検査の追加、
二重運用の解消条件と
期日。台帳の備考に
記入します。

3つとも Markdown です。社内リポジトリに置けば、作成者以外が次に読む場所になります。
D2-11 では、この台帳の「置けない行」と「ガイド済でない行」がそのまま残った層になります。

次: D2-11 効果測定と8層チェック表の塗り直し [15min]

D2-11

効果測定と8層チェック表の塗り直し

測るのは MR に残る記録

研修後に追う数字を、GitLab の MR から機械的に取れる4つに決めます。
Day1 の朝に印を付けた8層チェック表を塗り直し、残った層を確かめます。
年末の切替までに進める順番を、各自1枚に書いて持ち帰ります。

[15min]

8層チェック表と持ち帰り台帳を手元に開いてください

レビュー工数の計測は自己申告になる

投入量で測る

レビューに使った時間
AI 指摘の対応に使った時間
確認作業の時間

出どころ: 人の申告

伸びしろ7



記録する場所を
MR に決める

MRに残る記録で測る

指摘の件数と採用数
ゲートで止まった回数
MR 作成からマージまでの時刻

出どころ: GitLab の MR とパイプライン

時間の計測は、AI 指摘への対応時間と元からの確認時間を本人でも分けられず、数字が申告者で変わります。
回答書の「小さな改修ほど確認に時間がかかる」は、D2-03 で件数の問題として分解しました。
測る側も件数と割合に寄せます。

GitLab の MR から機械的に取れる4指標

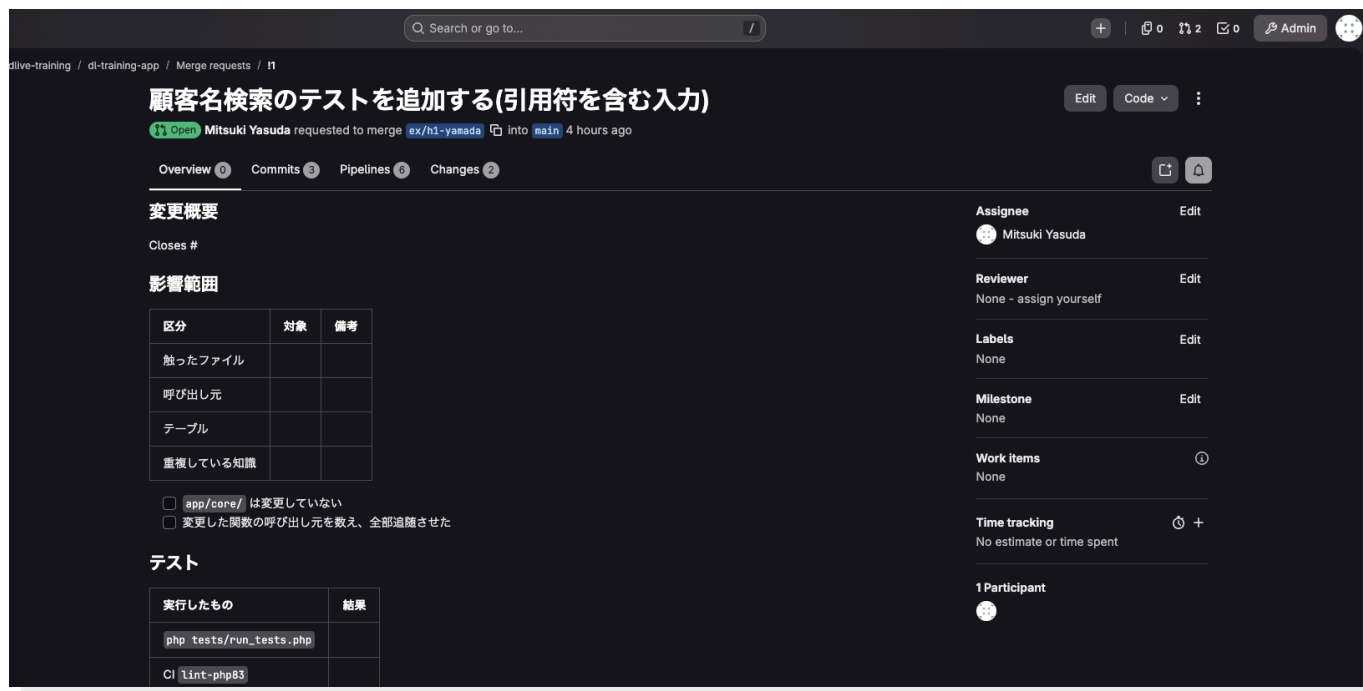
No	指標	数式	取る場所	ビフォー側にあるか
1	MR あたりの AI 指摘件数と採用率	指摘件数 ÷ MR 数 採用した指摘 ÷ 指摘件数	ai_review の findings.json と MR コメントの解決状態	宿題1 の対応・見送りの 数え方で代用できる
2	人が採否判断した 件数	MR ごとに人が「採る」 「見送る」を書いた数	MR コメントへの 返信1行	なし 研修後に初期値を取る
3	ゲートで止まった MR の割合	ai_gate が赤になった MR ÷ 全 MR	パイプラインの ジョブ結果	なし ゲートが存在しない
4	マージまでの時間と 手戻り MR 率	MR 作成からマージまでの 時間、同じ Issue の 追加 MR ÷ 全 MR	MR の作成日時と マージ日時、Issue への紐付け	改修着手からリリース までの日数で代用できる

4つとも MR とパイプラインに残る記録だけで数えられ、人の申告を使いません。

指標2 だけは人が1行書く手間が要ります。その1行が「AI は列挙し人が判断する」切り分けの記録になります。

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

4指標の数字は MR の1画面に全部出ている



1・3・4 は、この画面を下へ送った先に並びます。

指標2 は、このコメント欄に人が「採る」「見送る」と1行返すことで同じ画面に残ります。

数字を拾う人は MR を開くだけで済み、Excel への転記は要りません。

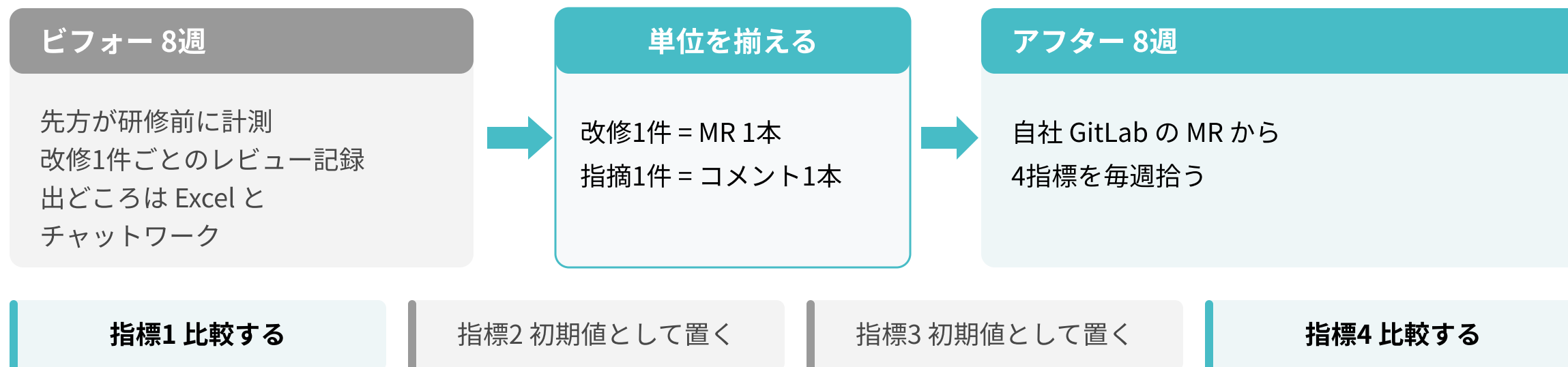
1 AI レビューのコメント欄
コメント件数と、解決済みにした数

3 パイプラインの状態表示
ai_gate が赤か緑か

4 作成日時とマージボタン
作成からマージまでの時刻差

画面

演習リポジトリの MR の Overview。
当日は前夜の実走の MR を開きます

比較できるのは**件数と時間**の2指標だけ

ビフォー側に存在するのは件数と日数だけです。ゲートと採否の記録は研修前に無いので、研修後の数字を初期値として置きます。比較は「改修1件 = MR 1本」に揃えてから行います。揃えないと、MR を細かく切った分だけ指摘件数が減ったように見えます。

検出率の記入欄

アフター値の初回は今日の2つの率

ノイズ率 = 仕込み欠陥以外の指摘 ÷ 指摘総数

検出率 = 当てた件数 ÷ 仕込み欠陥の件数

仕込み欠陥の件数

当てた件数

指摘の総数

検出率

ノイズ率

記入 D2-05 で各自が記録した値を、この5つの欄に書き写します

今日のハンズオン2 で記録した検出率とノイズ率が、アフター値の最初の2点になります。

指摘を絞って動くのはノイズ率のほうで、検出率は動かないはずです。両方下がったら Do not report に入れすぎです。

来週数字を拾う人と画面を紙に書く

1 Step 1 [1min]

紙に書きます。「研修後の8週間、自分のチームで毎週数字を拾うのは誰か」。名前を1人、名詞で書きます。自分でも構いません。

2 Step 2 [1min]

4指標のうち、来週のMRから取れるものを1つ選び、その数字を読む画面の名前を書きます。MRのOverviewかPipelinesのどちらかです。

3 Step 3 [1min]

その数字が8週後に悪い方へ動いていたら、何を1つやめるかを書きます。設定名か運用名で1つだけ書きます。

AIには聞きません。誰が・どこで・何をやめるかは、チームの運用を知っている人しか書けません。

8週後の集計担当と数字を出す先は、研修後に社内で決めていただきます。今日の資料には答えを入れていません。

期待される手元

1 毎週数字を拾う人

2 指標番号と読む画面

3 悪化したらやめること

リモートの方はローカルのメモで構いません

指標1の分子は findings.json を数えるだけ

D2 向け [.claude/metrics/count_findings.ps1](#)

```
# .claude/metrics/count_findings.ps1 (ASCII only, PowerShell 5.1)
$f = Get-Content findings.json -Raw | ConvertFrom-Json
$total = @($f.findings).Count
$p01 = @($f.findings | Where-Object { $_.severity -in 'P0','P1' }).Count
Write-Output "total=$total p0p1=$p01"
```

smart3pm 向け [.claude/metrics/count_findings.sh](#)

```
#!/usr/bin/env bash
# count findings by severity from ai_review artifact
jq '{total: (.findings|length),
  p0p1: ([.findings[]|select(.severity=="P0" or .severity=="P1")]|length)}' findings.json
```

findings.json は D2-05 の ai_review ジョブが artifact として残すので、MR ごとに同じファイルが必ずあります。
severity 別に数えます。P0 と P1 の合計は ai_gate が赤にした判定と必ず一致します。

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

塗り直しで残った層が**研修後の順番**になる

層	一言	Day1 朝	Day2 終了	残り
第1層 -p 非対話	人が打たずに1回走る	☐	☐	D2-08 で足した
第2層 hooks	実行中に機械が止める	☐	☐	ブチ演習3本で足した
第3層 subagents / skills / plugins / MCP	仕事を分けて渡す	☐	☐	D1-05 ・ D2-05 で足した
第4層 permission modes / sandbox	人の代わりに承認する止め金	☐	☐	D1-07 ・ D2-08 で足した
第5層 compaction	長時間でも文脈を保つ	☐	☐	試す担当 _____
第6層 Agent Teams / Workflows	並列に走らせる	☐	☐	試す担当 _____
第7層 Routines / CI	人がいない時間に起動する	☐	☐	D2-08 で足した
第8層 Agent SDK	プログラムから呼ぶ	☐	☐	試す担当 _____

Step 1 [1min] Day1 朝の印を左の列へ写します。今日動いている層だけを Day2 終了の列に塗ります。

Step 2 [1min] 右の列で途切れた最初の層に下線を引き、持ち帰り台帳の1行目を Day2 塗り直し に直します。

第4層の sandbox は Windows 非対応のため、持ち帰るのは permissions です。担当を書いた層は 216 に移します。

出典: <https://code.claude.com/docs/en/sandboxing>

年末切替まで3段で進める順番



順番を逆にしない理由は1つで、止め金の無い夜間ループは朝に人が読む量を増やすだけだからです。
3段とも今日の成果物で足りています。年末までに新しく作るものではありません。

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

進む条件を4指標の数字で1つ書く

1 Step 1 [1min]

215 の3段を紙か md に写し、各段に「担当」と「対象リポ(D2 か smart3pm か、その中のどれか)」を書きます。担当は3段で同じ人にしないでください。

2 Step 2 [1min]

各段の「進む条件」を、208 の4指標のどれか1つの数字で書きます。例の数字は載せません。自分のチームで通る値を入れてください。

期待される手元

段	担当	対象リポ	進む条件
1段目			
2段目			
3段目			

数字を書けない段は「未決」と書きます

手元に残る3点: 4指標表・8層チェック表・この順番1枚

この1枚は D2-10 の社内展開ガイドと同じ場所に置き、ガイドの「次に何をするか」の欄として使います。

置き場所は D2 と smart3pm の両方に同じ内容で構いません。シェルに依存する記述は含みません。

研修後に社内で決める4点

何のために	決めること
8週後の数字を集めて読むため	4指標の集計担当と、数字を出す先(月次の場か文書か)
ビフォー値と同じ単位で 比べるため	8週分の記録が改修1件単位かレビュー1回単位か
伸びしろ5・6・8の 宿題を進めるため	研修後の社内講習の枠で扱うか、担当は誰か
成果物を社内展開するため	台帳17点とガイドを社内のどこまでで使うか

決まっていないことを決まったように書かないため、4点は研修後に決めていただきます。
今日の資料には答えを入れていません。

測定は研修後の**最初の MR** から始まる

数字を拾うのは、研修後に自社 GitLab で
出す最初の MR からです

手元に残る3点

- 効果測定4指標表(自社の取る場所と担当を記入済)
- 8層チェック表(Day1 朝と Day2 終了の2列が塗られた状態)
- 年末切替までの順番1枚(3段の担当と進む条件つき)

研修後の順番

今日の帰りから年末までの5つの順番

今日の帰り

端末の機密ハーネス zip を削除(D2-10 の手順)
台帳17点とガイド A4 を自社へ持ち帰る

書き戻し

17点を D2 / smart3pm に配置
作成者以外の1名がガイドだけで動かす

測定開始

最初の MR から4指標を拾う
宿題3点(伸びしろ5・6・8)に着手

8週後

ビフォー値と件数・時間の2指標を突き合わせる
初期値(採否・ゲート赤率)を確定

年末から年明け

3段の順番どおりに一斉切替
Notion への作業指示書反映は社内で

質問と詰まりは
社内の窓口へ

Givery 側の講師所見メモは研修終了時に破棄します。受講者端末の zip 削除と合わせて、先方ハーネスの複製は研修後に残りません。



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