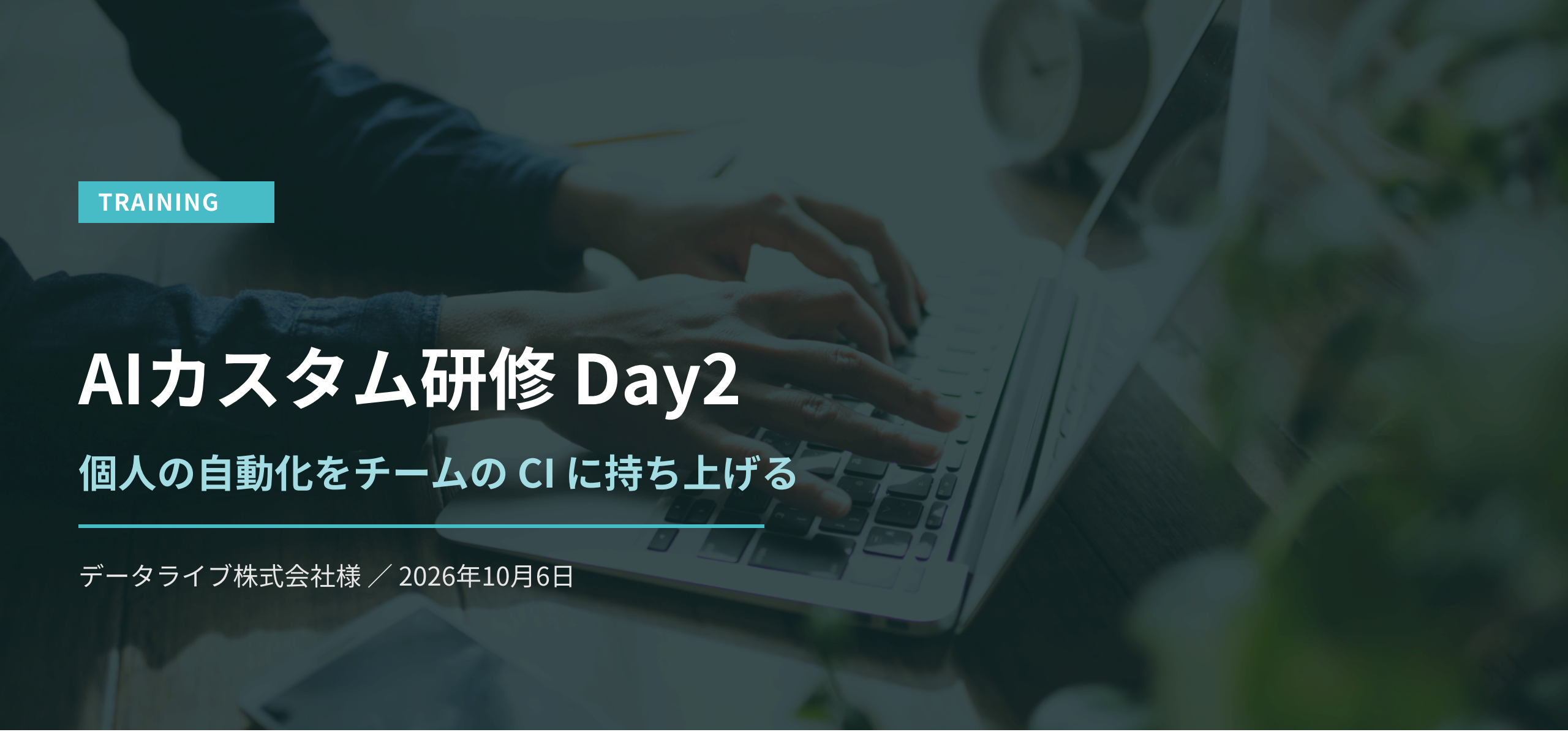


TRAINING

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

Givery

本資料の二次転載禁止について

受講者ご本人の学習目的に限った利用

本資料は、株式会社ギブリーがデータライブ株式会社様向けに作成した研修教材です。
著作権は株式会社ギブリーに帰属します。研修の受講者ご本人の学習目的に限って
ご利用ください。資料の全部または一部を、複製・転載・改変・第三者への配布
(社内の他部門を含む)することはご遠慮ください。画面共有や録画の二次利用
についても同様です。

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

D2-01 PHP 5.4 で動くコードを書かせる指示
D2-02 プチ演習4 5.4 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 持ち帰り台帳の確定と社内展開ガイド
D2-10 効果測定と8層チェック表の塗り直し

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7点

No	ブロック	種別	所要	持ち帰り物
D2-01	PHP 5.4 で動くコードの指示	座学	[25min]	5.4 非対応構文一覧、制約節の書き方
D2-02	プチ演習4 5.4 制約の機械検査	演習	[20min]	制約節、check-php54.ps1、cases.json
D2-03	指摘絞り込みの実物設定と判断軸	座学	[30min]	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	[120min]	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	[30min]	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	[120min]	夜間ジョブ、schedule 設定、implement-issue skill
D2-09	持ち帰り台帳の確定と社内展開ガイド	その他	[25min]	台帳14点の確定、社内展開ガイド A4
D2-10	効果測定と8層チェック表の塗り直し	その他	[20min]	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 5.4 非対応構文一覧、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の6点です。教材サイトの配布 ZIP に入っています。

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 のホールド	settings に書いた effort が効かない	Opus 4.7/4.8/Fable 5 がモデル既定を優先する	/effort <level> で確定。-p では --effort。Opus 5 と Fable 5.1 にはホールドなし

この3行は Day1 に実際に出た詰まりで入れ替えます。

Day1 の記録で差し替え

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

宿題の数字は午前の後半、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 042 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)

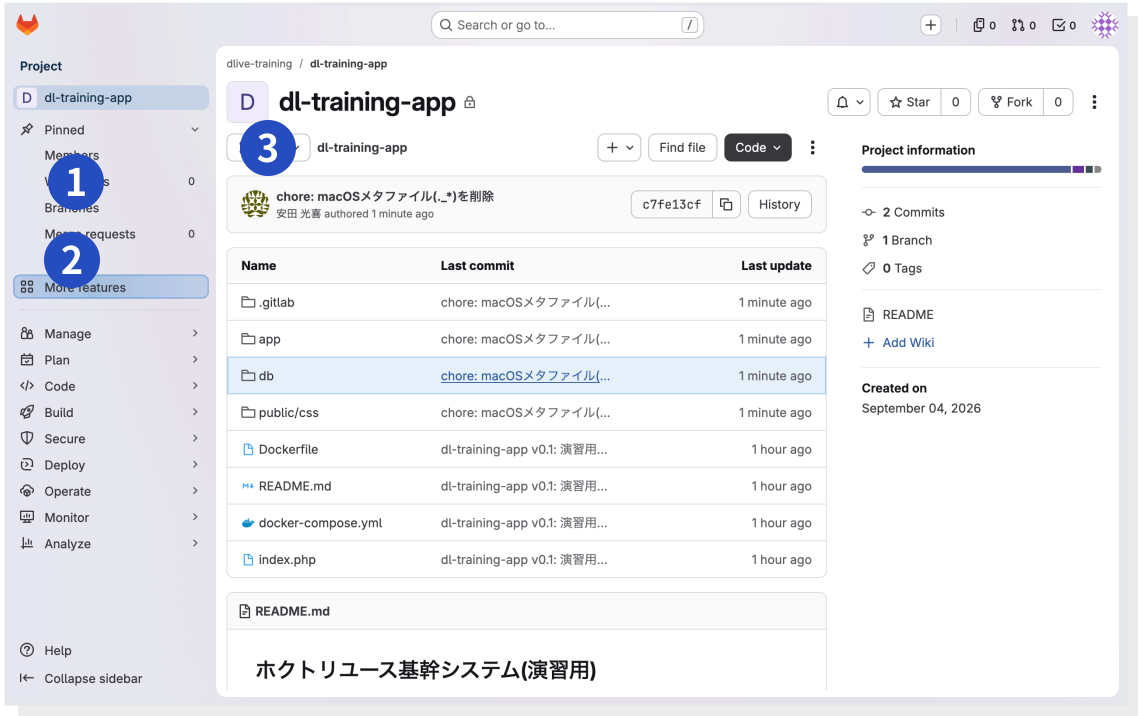
- ai_review が findings.json を出す
- ai_gate が P0/P1 で MR を止める
- nightly_implement がブランチを切り実装とテストを済ませる
- 人に残るのはマージ承認と指摘の採否と Issue の粒度

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4ジョブ



上の画面は GitLab のリポジトリトップです。

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. Pipelines は今朝の時点で4ジョブ。lint-php83 / lint-php54 / test / ai-review です。D2-05 で2つ増えます。
3. main ブランチは直接 push 不可。MR が必須です。

手元の確認

- claude --version が 2.1.267 以降
- jq が PowerShell から呼べる
- PowerShell の実行ポリシーで .ps1 が動く
- Codex は任意。未保有なら claude --bare -p の並行手順

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

D2-01

PHP 5.4 で動くコードを AI に書かせる指示

文脈と機械検査と CI の3段で守る実環境の制約

Day1 の lint-php54 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。

[25min]

このあと D2-02 プチ演習4 [20min] で自社ハーネスに足します

質問

Day1 の lint-php54 で赤になった行

Day1 の MR で赤になった構文の回収

Day1 のハンズオン1で lint-php54 が赤になった書き方を、ここに集めます。

受講者	lint-php54	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。

実演

8.3 で通り 5.6 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)

```
-       $status_cd = (int)$_GET['status_cd'];
+       $status_cd = (int)($_GET['status_cd'] ?? 0);
+       $date_from = $_GET['date_from'] ?? '';
```

CI ジョブ lint-php54 のログ(php:5.6-cli)

```
21  03:18:11  $ find . -name '*.php' -not -path './.git/*' -print0 | xargs -0 -n1 php -l
22  03:18:11  No syntax errors detected in ./tests/run_tests.php
23  03:18:11  No syntax errors detected in ./tests/phpunit/EstimateSearchTest.php
24  03:18:11  No syntax errors detected in ./tests/phpunit/bootstrap.php
25  03:18:11  No syntax errors detected in ./index.php
26  03:18:11  Parse error: syntax error, unexpected '?' in ./app/controllers/estimate_ctl.php on line 18
27  03:18:11  Errors parsing ./app/controllers/estimate_ctl.php
28  03:18:11  xargs: php: exited with status 255; aborting
29  03:18:12  Cleaning up project directory and file based variables   00:01
30  03:18:13  ERROR: Job failed: exit code 124
```

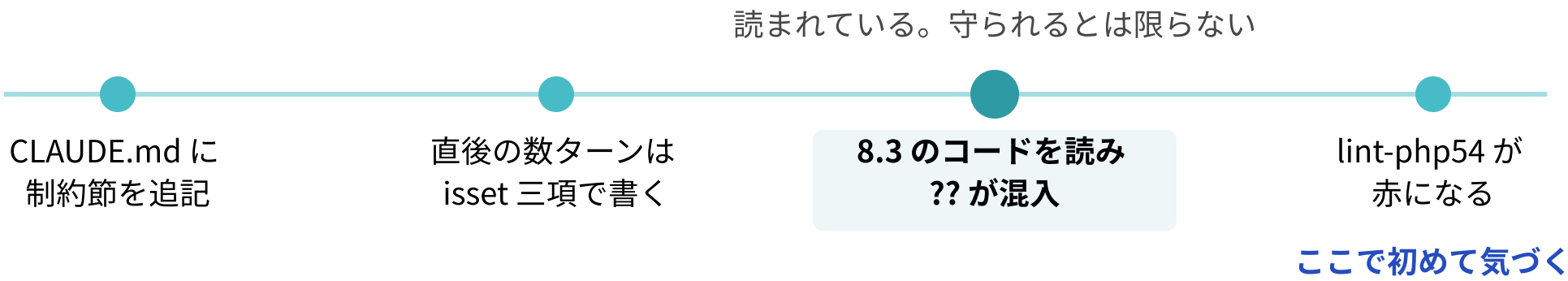
lint-php83 は緑、lint-php54 だけが赤になります。書いた本人も AI も 8.3 で動かして
確認しているので、CI に着くまで誰も気づきません。

当日は受講者の実 MR で見ます。画面は演習リポジトリでの再現例で、行番号 18 で止まっています。

実演

文脈だけで守る制約の寿命

書いた直後は守り数ターン後に戻る



CLAUDE.md に「PHP 5.4 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、null 合体 ?? 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

解説

5.4 で通らない書き方と代替

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
`$a ?? $b`	`isset($a) ? $a : $b`
`function f(int $n): string`	型宣言を書かず、phpdoc の @param で示す
`function f(...$args)`	`$args = func_get_args();` 固定引数は `array_slice` で切る
`$a <=> $b`	`($a == $b) ? 0 : (($a < $b) ? -1 : 1)`
`fn($x) => $x * 2`	`function ($x) { return $x * 2; }`
`yield / yield from $gen`	配列を組んで返し foreach で回す
`Foo::class`	'Foo' の文字列を直接書く
`const STATUS = array(...)`	静的メソッドで配列を返す
`try { } finally { }`	catch で後処理して投げ直し、try の外にも同じ後処理
`$x = [1, 2];`	**そのまま使ってよい(5.4 で動く)**

try の中で return する形はやめます。配布物「5.4 非対応構文一覧」はこの表に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
文脈 CLAUDE.md / AGENTS.md	代替の書き方を教える	生成の前	
フック PostToolUse	編集直後に検査し exit 2 で差し戻す	編集ツールの直後	
CI lint-php54	MR の全 PHP ファイルを検査	マージの前	

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

CLAUDE.md の制約節の書き方

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

実行環境は PHP 5.4.45 です。手元と CI の一部は
PHP 8.3 なので、8.3 で通っても本番で落ちる構文が
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10行: AI が素で書く形 / この環境での書き方)

検査は2段です。CI の lint-php54 は php -l だけで、
構文エラーになる書き方しか止まりません。これらを
捕まえるのは phpcs --standard=PHPCompatibility
--runtime-set testVersion 5.4 です。手元では
.claude/hooks/check-php54.ps1 が Edit と Write の
後に走ります。
```

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

補足

Codex 側は AGENTS.md に同じ節

ファイル名だけ変えて本文はそのまま

Claude Code

CLAUDE.md の制約節

015 の節をそのまま置きます。
読み込みは自動です。

phpcs の実行は次の PostToolUse
フックで機械化します。

本文コピー

Codex

AGENTS.md の同じ節

015 の節を AGENTS.md に置きます。
見出しも本文も変えません。

変更ファイルに最も近いものが
適用されます。

AGENTS.md の Code Review Rules(互換性の節)

```
### 互換性
- PHP 5.4.45 で落ちる構文が入った変更を指摘します
```

出典: https://learn.chatgpt.com/docs/third-party/github

※ 各ロゴは各社の商標です。

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

実演

check-php54.ps1 の本体

phpcs があれば標準規約で無ければ正規表現

.claude/hooks/check-php54.ps1(ASCII only, PowerShell 5.1。抜粋)

```
$payload = [Console]::In.ReadToEnd() | ConvertFrom-Json
$path = [string]$payload.tool_input.file_path
if ([string]::IsNullOrWhiteSpace($path)) { exit 0 }
if ($path -notmatch '\.php$') { exit 0 }
$phpcs = (Get-Command phpcs -ErrorAction SilentlyContinue).Source
if ($phpcs) {
    $out = & $phpcs '--standard=PHPCompatibility' `
        '--runtime-set' 'testVersion' '5.4' $path 2>&1
    if ($LASTEXITCODE -ne 0) { $msg = 'phpcs found 5.4 issues' }
} else {
    $src = Get-Content $path -Raw
    if ($src -match '\?\?|::class|\.\.\.\$') { $msg = 'regex hit 5.5+' }
}
if ($msg) { [Console]::Error.WriteLine("[check-php54] $msg in $path")
            exit 2 }
```

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

全文は配布 ZIP。bash 版 check-php54.sh を同梱

check-php54 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 ?? に化けて誤検知します。
- [ ] **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 -NoProfile
          -ExecutionPolicy Bypass -File
          .claude/hooks/check-php54.ps1" }
    ]
  }
]
```

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

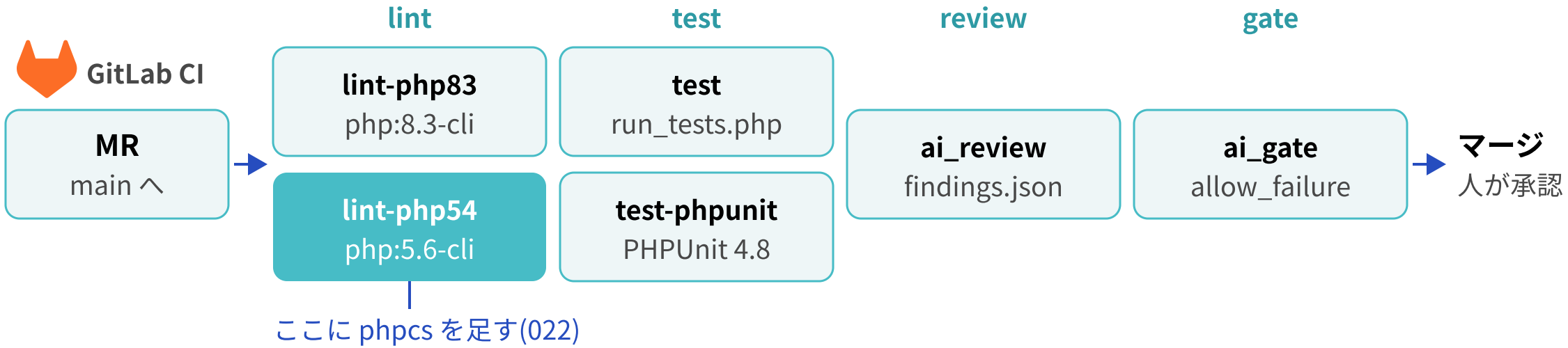
Codex 側の置き方

- Codex は ~/.codex/hooks.json に同じ PostToolUse を持ちます。
- 本体の check-php54.ps1 は共通です。登録の JSON だけ書き直します。
- リポジトリ直下の .codex/ に置くと対話セッションで発火しない報告があるので、ユーザー側に置きます。

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

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-php54 が既にあります。php -l だけなので、構文エラーになる書き方しか止まりません。::class や ... は php:5.6-cli のパーサを通るので、PHPCompatibility を testVersion 5.4 で足します。フックは Edit と Write しか見ません。Bash で書き換えた分と手編集の分は、この最終ゲートで受けます。5.4 の実機は CI にありません。通るのは「5.4 で通らない書き方が無い」ことだけです。

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

※ 各ロゴは各社の商標です。

lint-php54 ジョブの定義

php -l の後ろにphpcs を1行足す

ファイル名: .gitlab-ci.yml(現状の lint-php54 ジョブ)

```
lint-php54:
  stage: lint
  image: php:5.6-cli
  script:
    - find . -name '*.php' -not -path
      './.git/*' -print0
      | xargs -0 -n1 php -l
```

足す1行(案。script の末尾に追加)

```
    - phpcs -p --standard=PHPCompatibility
      --runtime-set testVersion 5.4
      --extensions=php app index.php
```

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

要確認

phpcs の取得経路は講師環境で
確定します。

- フックと CI が同じ PHPCompatibility を使うので、手元で通ったものが CI で落ちる差が出ません。
- php -l の行は残します。phpcs が入らなかったときの保険です。
- phpcs は standard の登録が要ります。installed_paths に PHPCompatibility を設定しないと見つかりません。
  入らない場合、lint-php54 は 7.0 以降の構文だけを落とし、5.5/5.6 はフックで守ります。

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。

解説

制約ごとの置き場所

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
5.4 非対応構文	○	● check-php54	○ lint-php54
SQL への外部入力の文字列連結	○	● 正規表現で検出	○ ai_review
app/core/ の不改変 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-php54 の php -l です。5.5/5.6 構文は phpcs を足すまで通るので補助です。

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 4.8 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。

CodeIgniter 2.0.3 と PHPUnit 4.8.27 は貴社回答書の記載です。

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 と AGENTS.md 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-php54.ps1	check-php54.sh を check 系の隣
CI	MR の全 PHP ファイルの構文 (php 5.6 の範囲)	5.5/5.6 の構文 (::class、...) 5.4 実機の挙動差	マージの前	.gitlab-ci.yml の lint-php54	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 と
AGENTS.md

check-php54

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-php54 ジョブ

php -l だけの現状の YAML と、
phpcs を足す1行の案。10月の
移行後も持ち込めます。

置き場所

.gitlab-ci.yml

3点とも配布 ZIP の d2-01/ に入っています。持ち帰り台帳の D2-01 行に書き足してください。

質問

自社ハーネスの現在地

5項目のうち埋まっている数の確認

- [ ] CLAUDE.md に 5.4 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 5.4 非対応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 5.4 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。

ワーク

プチ演習4 の進め方

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
紙に書く	自社 CLAUDE.md で「禁止」とだけ書いてある項目を3つ書き出す		[2min]
Step1	制約節を代替の書き方で直す	代替が並び、末尾に phpcs 行がある	[3min]
Step2	CLAUDE.md だけで書き直しを頼む	?? の差分がそのまま保存される	[2min]
Step3	check-php54.ps1 を置いて登録する	/hooks に PostToolUse が出る	[3min]
Step4	同じ依頼をもう一度出す	isset 三項に書き直される	[3min]
Step5〜6	Claude なしで exit 2 と 0 を確認し MR を出す	lint-php54 が queued で出る	[6min]
Codex	Step1 は AGENTS.md に。Step3 は .codex/hooks.json に PostToolUse 相当を登録(135 と同じ形) 発火しない版のときは節だけで Step2 と Step4 を比べ、OK 基準は1つ目と4つ目で判定する		

Step2 を先に行うのは CLAUDE.md だけだと通るのを見るためです。lint-php54 の緑は D2-03 の冒頭で確認します。

D2-02

プチ演習4 5.4 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

自社 CLAUDE.md の 5.4 制約節を、D2-01 の書き方で直します。
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、check-php54 フック1本、テストケース1件です。

ワーク　[20min]

ここで作る3点は D2-09 の持ち帰り台帳にそのまま載ります

ワーク

開始前の確認と流れ

作業ツリーを空にし対象ファイルを開いてから始める

確認	確認項目	確認コマンドまたは画面
☐	main にいて作業ツリーが clean	`git branch --show-current` `git status`
☐	Day1 の deny 8本と PreToolUse フックが残っている	`/permissions` と `/hooks`
☐	app/libraries/util.php の h() を開いた	VSCode のエディタタブ
☐	phpcs の有無を確かめた(無くても進める)	`Get-Command phpcs`
☐	自社の CLAUDE.md(Codex は AGENTS.md)を開いた	制約に関する節を表示

前提確認と紙に書く [3min] / Step1 制約節を直す [3min] / Step2 CLAUDE.md だけで頼む [3min]
Step3 フックを置く [3min] / Step4 同じ依頼で差し戻し [3min] / Step5 Claude なしで検査 [2min]
Step6 MR と CI [2min] / OK 基準と台帳 [1min]

phpcs が無い人は、Step3 のフックが正規表現側で動きます。結果は同じです。

ワーク

AI に渡す前に紙に書く

文章でしか守っていない制約を自分で3つ挙げる

問い1

制約節の中で、代替の書き方が
無い項目を3つ書き出します。
項目名だけで足ります。

問い2

D2-01 で見た 5.4 非対応構文
のうち、自分のコードに AI が
混入させそうな2つを、構文
そのままの形で書きます。

問い3

CLAUDE.md にしか書いていない
制約が破られたとき、自分は
いつ、どの画面で気づくかを
1行で書きます。

3分で3枚を埋めます。正解探しより、自分の環境で起きうることを書いてください。
問い1 は Step1 の書き直しに、問い2 は Step3 の正規表現に使います。
問い3 は Step6 の比較に使います。

ワーク

Step1 制約節を代替の書き方で直す

禁止の列挙をやめ代替と検査の義務を書く

CLAUDE.md(dl-training-app 直下)

既存の節の下に追記します。

```
## 実行環境の制約 (PHP 5.4)
- 本番は PHP 5.4 で動きます。5.5 以降は使いません
- ?? は使わず isset($x) ? $x : $default と書く
- ::class は使わず文字列 'ClassName' と書く
- 引数の型宣言(int/string/bool/float)は使わない
- 可変長引数 ... も使わない。配列の [] は可
- .php を編集したら phpcs --standard=
    PHPCompatibility --runtime-set
    testVersion 5.4 を実行し、NG が 0 に
    なってから完了と報告します
```

Codex のみの受講者

> AGENTS.md に左と同じ内容を
> 追記します。置き場所は
> dl-training-app 直下です。
> Codex は CLAUDE.md を読まない
> ので、制約節は AGENTS.md 側が
> 正です。両方使う人は2ファイル
> に同じ文を置きます。

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

「使うな」の行には、必ず同じ行に「代わりにこう書く」を添えます。
最後の1行が、D2-01 で言った phpcs 実行の義務化です。
保存後に Claude Code で /memory を開き、CLAUDE.md が読み込まれていることを見ます。

実演

Step2 CLAUDE.md だけで頼む

制約節があっても差分は止まらない

入力する依頼文

```
app/libraries/util.php の h() を、引数が null のとき空文字を返すように null 合体演算子で書き直して
```

Step	操作	期待される画面
2-1	上の依頼文をそのまま 入力します	Edit の差分に $s = $s ?? ''; が 入ります。isset 三項で書いた場合は 「今回は ?? を使って」と押します
2-2	Esc を2回押し Restore code で この Edit を戻します	git status が clean に戻ります
2-3	依頼文をメモします	変更なし。Step4 で同じ文を使います

制約節を読んでも、差分が保存されるまでに止める仕組みは一つも働いていません。
Step2-2 で戻すのは、Step4 で同じ依頼をもう一度出すためです。

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

ワーク

Step3 PostToolUse hook を1本置く

編集直後にphpcs か正規表現で検査する

.claude/hooks/check-php54.ps1(ASCII のみ)

```
$in = [Console]::In.ReadToEnd() |
      ConvertFrom-Json
$f = $in.tool_input.file_path
if (!$f -or $f -notmatch '\.php') { exit 0 }
if (Get-Command phpcs -EA SilentlyContinue) {
  phpcs -q --standard=PHPCompatibility `
    --runtime-set testVersion 5.4 $f
  if ($LASTEXITCODE -eq 0) { exit 0 }
  $m = "phpcs NG"
} else {
  $bad = '\?\?|::class|\.\.\.\$|<=>|\bfn\('
  $t = Get-Content $f -Raw
  if ($t -notmatch $bad) { exit 0 }
  $m = $Matches[0]
}
[Console]::Error.WriteLine("php54: $m in $f")
exit 2
```

Codex は ~/.codex/hooks.json に同じ登録をします。

.claude/settings.json

```
"hooks": {
  "PreToolUse": [ ... ],
  "PostToolUse": [ {
    "matcher": "Edit|Write",
    "hooks": [ {
      "type": "command",
      "command":
      "powershell -NoProfile
      -ExecutionPolicy Bypass
      -File
      .claude/hooks/check-php54.ps1"
    } ] } ]
}
```

phpcs があれば phpcs の結果で、
無ければ正規表現で exit 2 です。
PostToolUse は編集の後に走り、
差し戻します。/hooks で登録を確認します。

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

実演

Step4 同じ依頼で差し戻しを見る

差し戻しの文がそのまま修正の指示になる

Edit 1回目の差分

```
$s = $s ?? '';
```

フックが返す stderr

```
php54: ?? in
app/libraries/util.php
```

Edit 2回目の差分

```
$s = isset($s) ? $s : '';
```

Step2 と違うのは設定1箇所だけです。依頼文も CLAUDE.md も同じです。
2回目の書き直しは人が頼んでいません。stderr の1行が修正の指示になっています。
再検査まで来たら git diff で isset 三項だけが残っていることを見ます。

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

実演

Step5 Claude なしで exit 2 を確かめる

検査用ファイル1本とパイプ1本で足りる

1. **検査用ファイルを作ります**

```
Set-Content .claude/hooks/tests/fixtures/bad54.php.txt '<?php $a = $b ?? 1;'
```

拡張子は .php.txt です。CI の lint-php54 は .php だけを見るので拾われません。

2. **フックに Claude が渡す形の JSON をパイプで流します**

```
$p = ".claude/hooks/tests/fixtures/bad54.php.txt"
'{"tool_name":"Edit","tool_input":{"file_path":"' + $p + '"}}' |
  powershell -NoProfile -ExecutionPolicy Bypass -File .claude/hooks/check-php54.ps1
$LASTEXITCODE
```

```
php54: ?? in .claude/hooks/tests/fixtures/bad54.php.txt
2
```

file_path を Step4 で直した
util.php に替えると 0 だけが出ます。

3. **cases.json に1件足し run_cases.ps1 を回すと 19 passed, 0 failed**

```
{"id": "ex4-null-coalesce", "hook": "check-php54",
 "fixture": "bad54.php.txt",
 "input": {"tool_name": "Edit", "tool_input": {"file_path": "__FIXTURE__"}},
 "expect_exit": 2}
```

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

ワーク

Step6 MR を出し三段の検出範囲を見比べる

フックが拾いCI がすり抜ける構文が残る

構文	CLAUDE.md の制約節	check-php54 フック	CI lint-php54 (php:5.6 の php -l)
?? null 合体 (7.0)	助言。押せば書く	止まる	落ちる
::class 定数 (5.5)	助言	止まる	通る(php -l は 5.6 を受理)
... 可変長引数 (5.6)	助言	phpcs 版・正規表現版とも 止まる	通る(php -l は 5.6 を受理)
int $x 型宣言 (7.0)	助言	phpcs 版は止まる 正規表現版は見ない	落ちる
[] 短縮配列 (5.4 可)	許可	通る	通る

要確認
CI の列は php -l だけの現状です。phpcs を足すと ::class と ... も落ちます(手順は 022)。

Step6 は ex4-php54 ブランチを作り、push して MR を出し、Pipelines を見ます。
3列目と4列目がずれる行があります。ジョブ名は 5.4 でも、中身は php 5.6 の構文チェックです。
出典: https://gitlab-09291006aidev.give-app.net / https://github.com/PHPCompatibility/PHPCompatibility

ワーク

OK 基準とつまずき

4つの OK と3つの詰まりを自分の画面で照合

OK 基準

- [ ] CLAUDE.md(または AGENTS.md)の制約節に
代替の書き方と phpcs の義務が入った

- [ ] Step2 で ?? の差分が出て、Step4 で
同じ依頼が isset 三項で終わった
Codex: ログに ?? が残るかで判定

- [ ] Step5-2 で 2 が返り、util.php に
替えると 0 が返った
Codex: hooks が発火しない版は対象外

- [ ] run_cases.ps1 が 19 passed, 0 failed
で終わった

つまずき

Step4 でフックが走らない
settings.json の PostToolUse が未読込
/hooks を見て、無ければ再起動

Step5-2 で赤いエラーが出る
JSON の引用符が PowerShell に食われた
外側を単引用符、内側を二重引用符に

正しい isset 三項にも exit 2
PHPCompatibility が phpcs に未登録
phpcs -i で確認し、無ければ正規表現側へ

OK が4つ揃った人は、台帳に「制約節」「check-php54 フック1本」「テスト1件」と書きます。
2つ目の OK が出ない人は、右のつまずきを上から順に見てください。Codex は1つ目と4つ目で判定します。

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

持ち帰りと発展

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)
5.4 制約節 (代替 + phpcs 義務)	自社 CLAUDE.md の既存節を Step1 の形に書き直す。Codex 併用なら AGENTS.md にも同文	同じ。core.md 側は機械に移した項目のタグを書き換える
check-php54 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	配布 ZIP の hooks/bash/ check-php54.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-php54.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: 冒頭の問い1 で挙げた「禁止だけ」の3項目を、代替の書き方つきで自社の制約節に書き直します。
発展2: 正規表現版で拾えない引数の型宣言を止める正規表現を1本足し、cases.json に1件足します。
どちらも D2-09 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://github.com/PHPCompatibility/PHPCompatibility

D2-03

[30min]

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

質問

宿題の回収と Day1 の指摘件数

数えてみると見える見送りの多さ

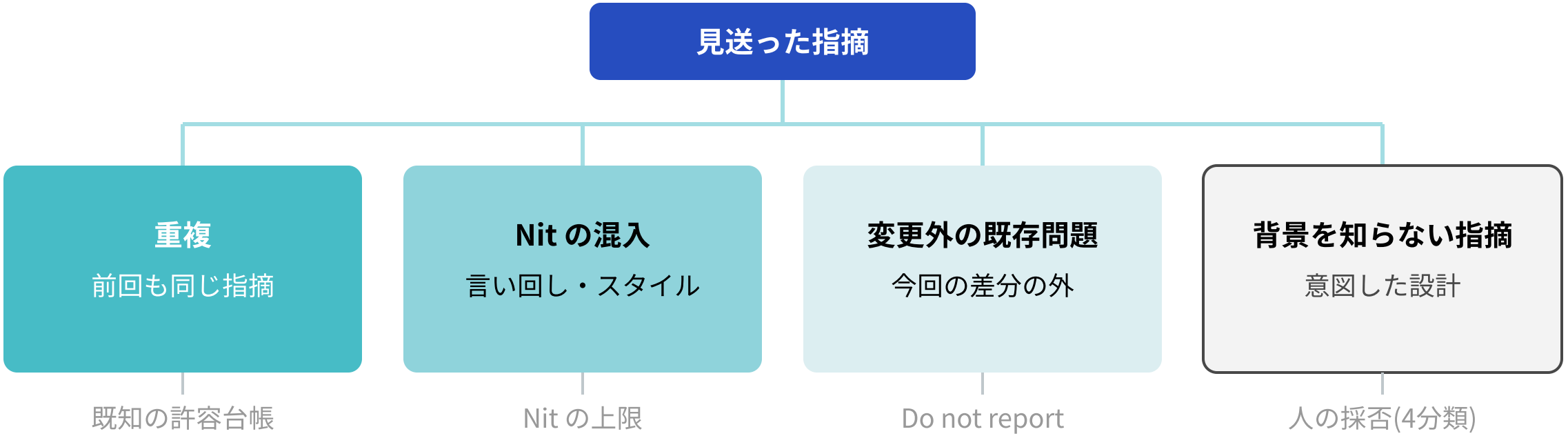
区分	指摘総数	対応した件数	見送った件数	見送り率
宿題(直近の自分の改修1件)				
ハンズオン1 の MR(CI の AI レビュー)				
現地7名の合計				
リモート(チャット提出分)の合計				

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。

解説

見送った指摘の4分類

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。

解説

既知の許容台帳の置き場所

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

解説

4節が潰す見送りと先方版の書き方

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	5.4 非対応構文、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

043 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

実演

設定ファイルの置き場所

リポジトリの根に置く3ファイル

左のファイル一覧に AGENTS.md、CLAUDE.md、REVIEW.md の3つが並びます。右は REVIEW.md の中身で、4つの見出しだけの空雛形です。

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. **ルートの AGENTS.md**
   Codex 側の Code Review Rules を書く場所

3. **ルートの CLAUDE.md**
   5.4 制約と触らない場所。
   レビュー基準は書かない

演習リポジトリには REVIEW.md と AGENTS.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。

画面　演習リポジトリで REVIEW.md を開いた状態。研修中に4節を埋めると、ここに中身が入ります。

Codex の Code Review Rules

結果で書くレビュー規則

AGENTS.md

```
## Code Review Rules
### 互換性
- 外部連携の通知名(rawResponseItem/*)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
```

Codex はレビュー方針を AGENTS.md の「## Code Review Rules」見出しで読み、
変更ファイルに一番近い AGENTS.md を適用します。
公式ブログが挙げる注意点は、機械的な CI チェックを書かないこと、結果の形で書くこと、
広すぎる指示を避けることです。
REVIEW.md の 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://learn.chatgpt.com/docs/third-party/github
https://developers.openai.com/blog/custom-code-review-rules-for-codex

Claude と Codex のレビュー指示の対応

置き場所は違っても節の役割は同じ

Claude Code

- REVIEW.md をルートに置く
- 重要度は Important・Nit・Pre-existing の3段
- CLAUDE.md 違反は Nit 扱い
- /code-review は REVIEW.md を読まない

Codex

- AGENTS.md の「## Code Review Rules」
- 変更ファイルに一番近い AGENTS.md が適用
- GitHub のクラウドレビューは P0/P1 だけ投稿
- CLI の codex review は P2/P3 も出す

書かないもの: 機械的な CI チェック(lint が見る)、広すぎる指示、変更外の既存問題

書く場所と重要度の呼び方は違いますが、何を Important(P0/P1)と呼ぶか、
何を報告しないかを決める点は同じです。一方しか持っていなくても、午後の起案は同じ内容で済みます。

出典: https://code.claude.com/docs/en/code-review
https://learn.chatgpt.com/docs/third-party/github

※ 各ロゴは各社の商標です。

codex exec の構造化出力

priority と confidence で機械が仕分ける出力

> **要確認**　行番号と confidence の値は、講師環境で Issue #1 の差分を実走した結果に差し替えます。

findings.json(出力例)

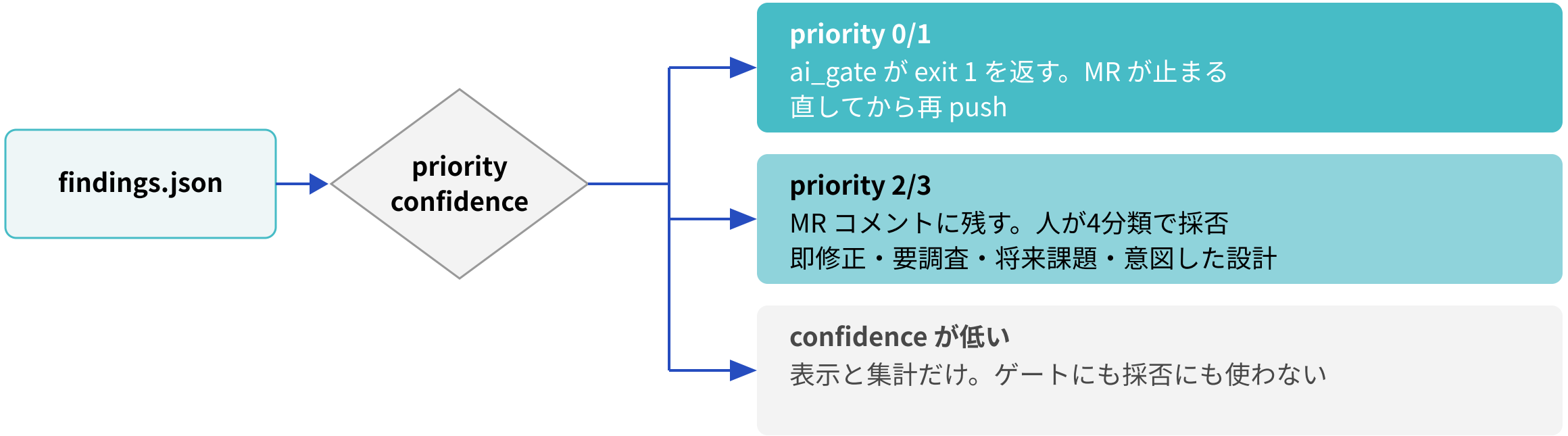
```
{
  "findings": [{
    "title": "検索条件の文字列連結による SQL インジェクション",
    "body": "customer_name と date_from/date_to を SQL に直接連結しています。",
    "confidence_score": 0.92,
    "priority": 0,
    "code_location": {"absolute_file_path": "app/controllers/estimate_ctl.php",
                      "line_range": {"start": 31, "end": 38}}
  }],
  "overall_correctness": "patch is incorrect"
```

codex exec に --output-schema を渡すと、最終応答がこの形の JSON で返ります。
priority は 0〜3、confidence_score 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(severity / confidence / file / line / message)で配ります。
出典: https://developers.openai.com/cookbook/examples/codex/build_code_review_with_codex_sdk
https://developers.openai.com/codex/cli/reference

解説

findings から3つの行き先

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。

指摘の少なさと問題のなさは別物

P0 / P1

GitHub に投稿されるのはこの2段だけ

Codex の GitHub コードレビュー(クラウド)

Codex の GitHub コードレビューは、
P0/P1 だけを PR コメントに投稿する設計です。
絞る側の設計として参考になりますが、
「指摘が少ない」と「問題がない」は別です。
P2/P3 は CLI で出して集計に使います。
先方の現行は全件列挙です。投稿する段階を
絞るか、全件出して機械で仕分けるかを午後に決めます。

出典: https://learn.chatgpt.com/docs/third-party/github

解説

5つの観点源を3列に振る講師案

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)	5.4 非対応構文、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。

解説

人に残す観点の位置

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	**5.4 構文・phpcs のスタイル** 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

054 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。

見送り率から決める人が見る量

042 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。

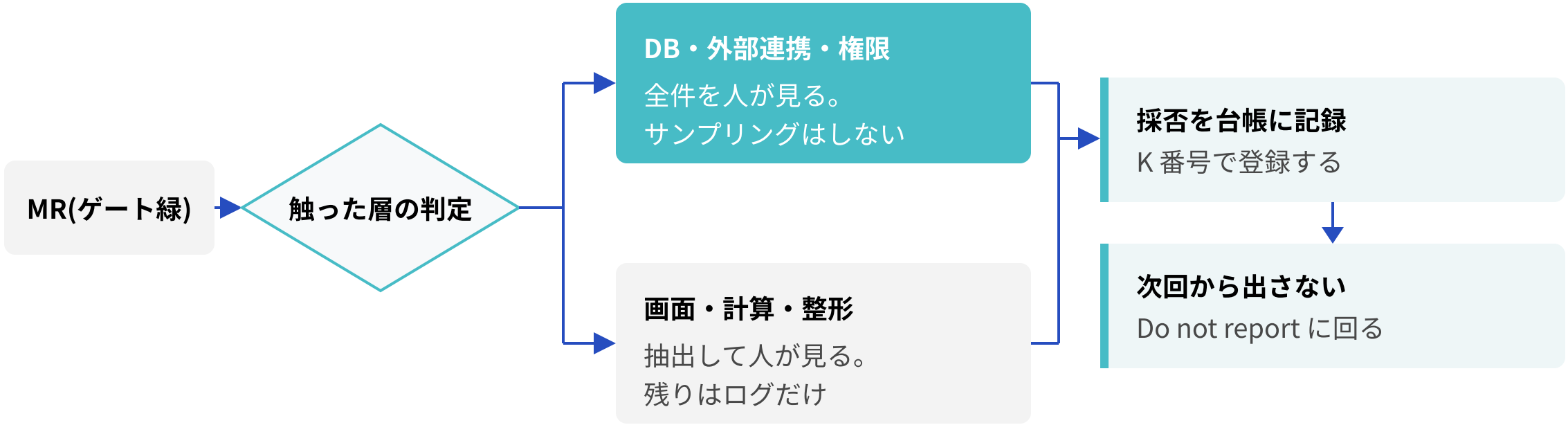
見送り率が高いほど、レビューが見ている範囲と人が直す範囲がずれています。
人が見る量はこの数字から決めます。P0・P1 だけに絞るか、P2 まで読むかの境目です。
Anthropic も自社の開発フローでリスク加重の抽出を使っています。

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

解説

人が見る量の決め方

触った層で決める人の確認の配り方



上の経路は 055 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。

レビュー1本の流れの書き換え

人の判断を最後の3観点だけに寄せる流れ

現行

ここに時間がかかる

差分 → AI が全観点で列挙 → 人が全件を採否 → 修正

書き換え後

差分 ▶ REVIEW.md で絞った列挙 ▶ P0/P1 は ai_gate で止める ▶ K 番号照合で再掲を除く ▶ 人は3観点とサンプルだけ ▶ 採否を台帳へ

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。

実演

REVIEW.md を渡すコマンド

/code-review は REVIEW.md を読まない前提

PowerShell(Claude Code)

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

PowerShell(Codex)

```
codex exec -s read-only `
--output-schema review.schema.json `
-o findings.json `
"$(Get-Content REVIEW.md -Raw)
対象: git diff main"
```

Claude 側の /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 も読まないので、レビュー側だけに使います。
Codex 側は -s read-only で作業ツリーを触らせず、--output-schema で出力の形を固定します。
どちらも出力は review.schema.json の形になるので、D2-04 のゲートは1本で両方を受けます。
出典: https://code.claude.com/docs/en/headless 、 https://developers.openai.com/codex/cli/reference

要確認 --json-schema にファイルパスを渡せるかは公式例がインライン JSON のため、講師端末で実走確認します。
通らない場合は 070 も含め -Raw で読んだ内容を1行に畳んで渡す形に差し替えます。

ワーク

自社 REVIEW.md 初版の起案前チェック

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 D2-10 で採用率として測り直します。042 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
Code Review Rules 雛形	Markdown	AGENTS.md に追記	AGENTS.md に追記
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(042 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。
3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。

補足

絞り込みで踏みやすい落とし穴

絞った後に起きる4つの誤解

指摘の少なさの読み違い

Codex の GitHub レビューは P0/P1 だけを PR コメントに投稿します。P2/P3 は CLI で出して集計に回します。

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 、 https://learn.chatgpt.com/docs/third-party/github

D2-04

レビュー出力の JSON ゲート

P0/P1 が1件でもあれば exit 1 を返す gate.ps1 を完成させる

題材は Day1 で自分が出した issue-1 ブランチの差分です。プチ演習5 にあたります。

[15min]

ここで作る gate は D2-05 でそのまま CI の ai_gate ジョブになります

ワーク

演習の狙い

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 --json-schema

↓

findings.json

↓

gate.ps1

↓

exit 0 で通す ／ **exit 1 で止める**

指摘は severity 付きの JSON で返り、
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。

ワーク

前提確認

5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
☐	Day1 の issue-1 ブランチで VSCode を開いている	`git branch --show-current` が `issue-1` と表示される
☐	配布の2ファイルが .claude の下にある review.schema.json と gate.ps1 の2本	VSCode のエクスプローラで見えている
☐	jq が PowerShell から呼べる	`jq --version` が版番号を返す
☐	ANTHROPIC_API_KEY が環境変数にある キーの配布方法は当日朝に講師が案内します	`$env:ANTHROPIC_API_KEY.Length` が `0` 以外になる
☐	Codex を使う人は起動を確認 codex --version	Codex が無い人は Claude の手順だけで完走できます

--bare は OAuth やキーチェーンを読まないため、サブスクでログインしていても API キーが無いと認証エラーになります。キーも Codex も無い方は 071 から合流し、findings.json は配布 ZIP の d2-04/findings.sample.json を使ってください。

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

ワーク

AI に渡す前に決めること

しきい値と止め方は紙に書いてから実行

問い	自分の答え
1. 自分のチームで止めたい severity は どこまでか。P0 だけか、P1 も含めるか。 理由を1行で書きます。	
2. Day1 の MR に付いた AI レビュー指摘の うち、止めてほしかったのは何件か。 見送った指摘はどの severity に当たるか。	
3. severity の定義を AI に任せるか、 自分で書くか。自分で書くなら P0 と P1 の 境目を何で引くか。データ消失・ セキュリティ・仕様違反のどれかで決めます。	

所要 [2min] で書きます。書いたしきい値を Step3 で切り替えて、件数の差と突き合わせます。

ワーク

Step1 レビュー結果の JSON 受け取り

同じ schema なら Claude でも Codex でも同じ findings.json

[4min]

Claude Code の手順

```
git diff main | claude --bare -p "Review the
  diff from stdin. Report only defects.
  severity: P0 data loss or security,
  P1 wrong behavior, P2 maintainability,
  P3 nit." --output-format json
  --json-schema .claude\review.schema.json
  | jq ".structured_output" | Out-File
  -Encoding utf8 findings.json
```

Codex の手順

```
git diff main |
  Out-File -Encoding utf8 diff.patch
codex exec -s read-only -o findings.json
  --output-schema
  .claude\review.schema.json
  "Review diff.patch in this folder.
  Report only defects. severity: P0 data
  loss or security, P1 wrong behavior,
  P2 maintainability, P3 nit."
```

期待される findings.json

schema は severity・confidence・file・line・message の5項目だけ

```
{ "findings": [ { "severity": "P1", "confidence": 0.8, "line": 31,
  "file": "app/controllers/estimate_ctl.php",
  "message": "date_from is concatenated into SQL without binding" } ] }
```

--json-schema の形は 061 と同じファイルパス渡しです。Claude 側は .structured_output を jq で取り出します。
出典 https://code.claude.com/docs/en/headless https://developers.openai.com/codex/cli/reference

ワーク

Step2 gate.ps1 の完成

未完成の1行を埋めて exit 1 が返る状態

[4min]

.claude\gate.ps1

配布版・PowerShell 5.1・ASCII のみ

```
1  param([string]$Findings = "findings.json", [string]$Threshold = "P1")
2  $levels = @("P0", "P1", "P2", "P3")
3  $max = $levels.IndexOf($Threshold)
4  $json = Get-Content $Findings -Raw | ConvertFrom-Json
5  # TODO: put into $hit only the findings whose severity is $Threshold or worse
6  $hit = @()
7  $hit | ForEach-Object { Write-Output ("{0} {1}:{2} {3}" -f
     $_.severity, $_.file, $_.line, $_.message) }
8  if ($hit.Count -gt 0) { Write-Output ("GATE FAIL: {0} finding(s)" -f
     $hit.Count); exit 1 }
9  Write-Output "GATE PASS"
10 exit 0
```

操作 6行目を書き換えて保存し、.\.claude\gate.ps1 -Findings findings.json を実行します
期待される画面 P1 以上が1件でもあれば GATE FAIL: 1 finding(s) と 1、無ければ GATE PASS と 0
答えは配りません。exit 0 は、P0 と P1 を消した no_findings.json を -Findings に渡して出します。
P1 以上が0件だった方は、配布 ZIP の d2-04\findings.fixture.json を -Findings に渡します。

ワーク

Step3 しきい値の切り替えと件数差

P1 と P2 の差が人の読む量の差

[2min]

閾値	実行コマンド	止まった件数	exit code	069 の答えと一致
P1	`.\.claude\gate.ps1 -Threshold P1; $LASTEXITCODE`			はい / いいえ
P2	`.\.claude\gate.ps1 -Threshold P2; $LASTEXITCODE`			はい / いいえ
差	P2 で増えた分が、止めなくても MR コメントで足りる指摘です			

件数は持ち帰り台帳の D2-04 行に書きます。Codex の findings は P2/P3 が多めに出るため、両方を持つ人は2通りの差も記録してください。

出典 https://developers.openai.com/codex/cli/reference

ワーク

OK 基準

4つが自分の端末で再現できれば完了

確認	できていること	確認のしかた
☐	findings.json が schema の5項目だけで返っている	`jq ".findings[0] | keys" findings.json` が5つのキーを返す
☐	gate.ps1 が P1 以上で exit 1、無ければ exit 0 を返す	`$LASTEXITCODE` が 1 と 0 の両方を一度ずつ出た
☐	しきい値を P1 と P2 で切り替えて件数差を台帳に書いた	持ち帰り台帳の D2-04 行に2つの数字が入っている
☐	069 で書いたしきい値と実測を見比べて採用する閾値を決めた	台帳の備考欄に P0 / P1 / P2 のどれかと理由が1行ある

4つ目が一番大事です。決めた閾値と理由は台帳に残し、D2-05 の REVIEW.md 起案でそのまま使います。
exit 0 を一度も見ていない人は、P0 と P1 の要素を消した no_findings.json を作り、
.\.claude\gate.ps1 -Findings no_findings.json で 0 を確認します。
P0 か P1 が1件でも残る限り、-Threshold を P2・P3 に緩めても 1 のままです。

補足

つまずきと直し先

3件とも PowerShell 5.1 と --bare の仕様

詰まり	出た症状	原因	直し先
findings.json が読めない	gate.ps1 が ConvertFrom-Json で止まる	PowerShell 5.1 の > は UTF-16 で書き出す	| Out-File -Encoding utf8 findings.json へ書き出す
認証エラー	claude --bare -p が API キー無しで止まる	--bare は OAuth やキーチェーンを読まない	$env:ANTHROPIC_API_KEY をセッションに入れる。settings には書かない
日本語の指示文が壊れる	指示文の一部が？に化けて severity の定義が伝わらない	PowerShell 5.1 のコンソールが CP932 で引数を渡す	プロンプトと gate.ps1 は ASCII のみ。日本語は REVIEW.md に書いて渡す

3行目の ASCII のみは D2 のフックと同じ制約です。D2-02 で足した検査を gate.ps1 にもかけられます。
出典 https://code.claude.com/docs/en/headless

ワーク

持ち帰り物と置き場所

schema と gate の2点を自社ハーネスへ

持ち帰り物	D2 PowerShell 5.1 正	smart3pm bash 正
review.schema.json	.claude/review.schema.json に置き、 d2-code-reviewer の出力形として 参照します	同じ内容を同じパスに置きます。 bash 側も同じファイルを 読みます
ゲート	.claude/gate.ps1 は今日の完成版。 hooks/tests/cases.json に P1 あり・なしの2ケースを足します	.claude/gate.sh を使います。 P0 と P1 の件数を jq で数え、 0 でなければ止めます

台帳の D2-04 行には review.schema.json・gate.ps1 または gate.sh を書きます。
採用した閾値と P1・P2 の件数も同じ行に残します。

D2-05 ではこの gate が .gitlab-ci.yml の ai_gate ジョブになり、CI では bash 版の jq 1行が走ります。

発展課題

confidence を第2のしきい値にする案

confidence で足切り

gate.ps1 に
-MinConfidence 0.7 を足し、
P1 でも confidence が低い
指摘は止めずに MR コメントへ
回します。止まった件数が
P1 の行と比べてどれだけ
減るかを見ます。

変更行数で閾値を自動選択

git diff main --numstat の
合計行数が 80 以下なら P1、
超えたら P2 で止めます。
D1-05 の review-light と
review-full の切り替えと
同じ境目に揃えます。

2ツールの突き合わせ

Claude と Codex の
findings.json を file と
line で突き合わせ、両方が
P0/P1 とした指摘だけで
止めます。片方だけの指摘は
confidence を 0.5 掛けにして
MR コメントへ回します。

いずれも gate.ps1 の書き換えだけで済みます。D2-05 のハンズオン中に余裕があれば試してください。
80 行は演習環境の ai_review.sh が quick と full を切り替えている境目と同じ値です。

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。機能テストの無い検索処理に PHPUnit のテストを生成させ、テストの有無でリファクタの壊れ方が見えるかを各自で確かめます。

[120min] ワーク

Codex のみの受講者は 089 と 098 の並行手順を使います

解説

全体図

人が決める20分と機械に載せる100分

Part1 は AI を起動しません。紙と REVIEW.md だけです。Part2 と Part3 は同じ差分(Issue #2 の初版)を使い続けます。差分を変えると比較が壊れます。

Part1 人が先に決める [20min]

- Step1 REVIEW.md 初版
- Step2 台帳と規則の移植

Part2 観点別サブエージェント [50min]

- Step3 security-reviewer
- Step4 残る2本の定義
- Step5 規模で選ぶ skill
- Step6 並列で当てる

Part3 CI ゲートとテスト生成 [50min]

- Step7 ai_gate の本物化
- Step8 MR を出す
- Step9 落ちて直す
- Step10 テスト生成
- Step11 破壊検知の比較

記録: 検出率

持ち帰り: REVIEW.md、agents 3本、skill、CI 2ジョブ、生成テスト、検出率

解説

題材 Issue の読み解き

欠陥を含む初版を読むところから始める演習

Issue	対象ファイル	変更の中身	既存テスト	初版パッチ
#2 見積一覧の検索条件の改修	app/controllers/estimate_ctl.php index()	customer_name / date_from / date_to の検索条件	無し 検索は未テスト	patches\issue2_draft.patch
#3 在庫引当の改修	app/controllers/stock_ctl.php	SELECT → UPDATE tr_stock → INSERT tr_bind_order_stock → UPDATE tr_order の引当	無し 引当は未テスト	patches\issue3_draft.patch

どちらも初版パッチは AI が書いた1回目という設定で、レビューで見つかるべき問題を含みます。
Part2 と Part3 の本筋は #2 です。#3 は test-gap-reviewer の当て先と発展課題に使います。

> **要確認**　Issue #2・#3 の本文文言とパッチの最終形は当日版の GitLab で確認します

ワーク

AI に渡す前に決める4つ

[7min]

自分の見送り率から逆算する Important の定義

Important の定義

宿題1で見送った指摘を思い出し、見送らなかった
指摘に共通する性質を1文で書きます。

Nit の上限

1 MR で読んでよい Nit の件数を数字1つで
書きます。理由も1行で書きます。

Do not report

宿題2の「機械判定できる」列から、lint や phpcs
に任せて AI に書かせない項目を3つ選びます。

機能テストの無い変更

機能テストの無い変更を Important に入れるか、
はい・いいえと、いいえなら誰が見るかを書きます。

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。
ここで書いた4つが Step1 の REVIEW.md に入ります。空欄のまま進むと公式例の写しになります。

ワーク

Step1 REVIEW.md の初版

Step1 [6min]

4見出しに 080 の答えを移すだけの起案

REVIEW.md(リポジトリのルート)

```
# Review rules
## Important(必ず報告)
- 5.4 非対応構文、SQL 直書きへの変数埋め込み、テストの無い DB 書き込み
## Nit(上限5件、まとめて1コメント)
- 命名と整形は phpcs が見る。ここには書かない
## Do not report
- 変更外の既存問題、スタイル、コメントの言い回し
## Always check
- 機能テストの無い変更は Important として報告する
```

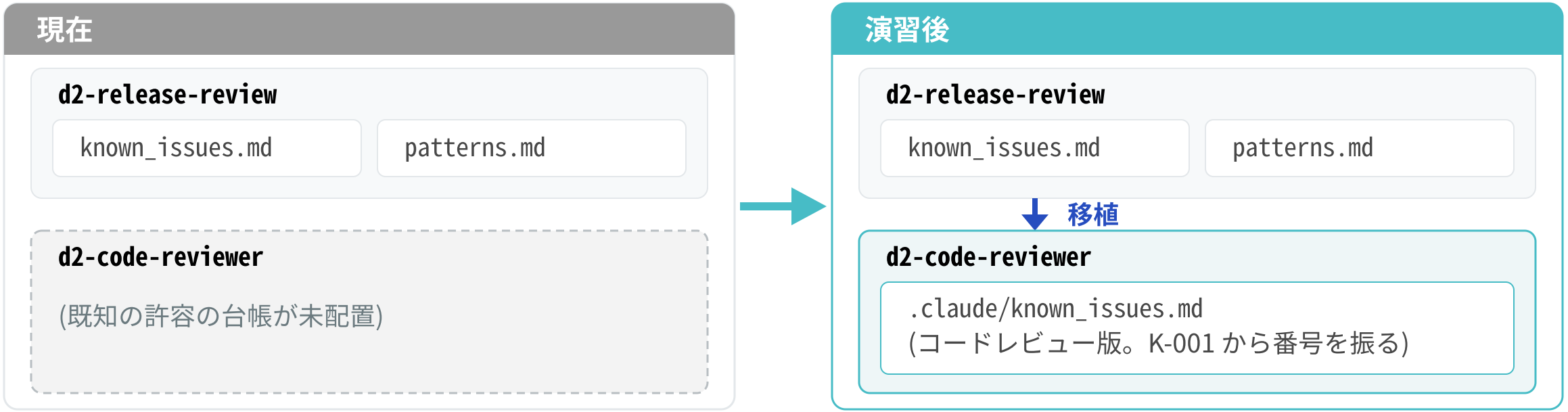
配布の雛形は空の4見出しです。上の9行は講師の例で、Important の1行目と Nit の上限と
Always check の1行を 080 の自分の答えに差し替えます。4見出しの骨格は、Anthropic の
Code Review が読む公式例と同じです。CI の ai_review はこのファイルを基準として渡します。
書いたら保存して Step2 へ進みます。まだコミットはしません。
出典: https://code.claude.com/docs/en/code-review

ワーク

Step2 台帳の移植とレビュー規則

Step2 [4min]

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



> Codex 保有者は同じ内容を AGENTS.md の ## Code Review Rules に写します

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くと、同じ対応不要の判断を改修のたびに繰り返さなくなります。今回作るのは .claude/known_issues.md の1本だけです。Day1 の MR で見送った指摘を K-001 として書きます。

出典: https://learn.chatgpt.com/docs/third-party/github

解説

観点別レビュアー3本の設計

観点1つにモデル1つを割る3本の分業

レビュアー	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

ワーク

Step3 security-reviewer.md

Step3 [8min]

外部入力の経路だけを追う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。
```

配布 ZIP の agents\security-reviewer.md を .claude\agents\ にコピーし、「見るもの」の節を開きます。
$_GET と $_POST から db_select 系の第1引数と app/views/ の echo に届く経路を追う、と
書かれています。この節の文を1つ、自社の基幹システムの言い方(使っている DB 関数名)に直します。
期待される画面: 保存後、Claude Code で /agents を打つと security-reviewer が一覧に出ます。
出典: https://code.claude.com/docs/en/sub-agents

ワーク

Step4 残る2本のレビュアー

Step4 [10min]

同じ形で観点とモデルだけ違う2本

Step4-1 spec-reviewer

配布の agents\spec-reviewer.md を
.claude\agents\ にコピーし、「指示にない
変更が混ざっていないか」の例を、自社で
起きた例に1つ差し替えます。

期待: model: sonnet と effort: high のまま

Step4-2 test-gap-reviewer

配布の agents\test-gap-reviewer.md を
コピーし、「見るもの」の3項目目「壊れた
ときに何が起きるか」の例を、自社の業務の
言葉に1つ直します。

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

ワーク

Step5 規模で選ぶ skill

Step5 [7min]

誘導は skill に置き保証は CI に置く

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

```
---
name: review-route
description: 変更規模で review-light か review-full を選び、観点別レビュアーを並列で呼ぶ
disable-model-invocation: true
---
変更規模: !`git diff --shortstat main`
1. 上の行数が 50 未満で、差分に db/ と app/core/ が無ければ review-light だけを呼ぶ
2. それ以外は review-full を呼び、3本の観点別レビュアーを並列で呼ぶ
3. 4本の結果を findings の形にまとめて人に返す。採否はここで決めない
4. 「50 未満」は D1-05 で自分が決めた境界に置き換える
```

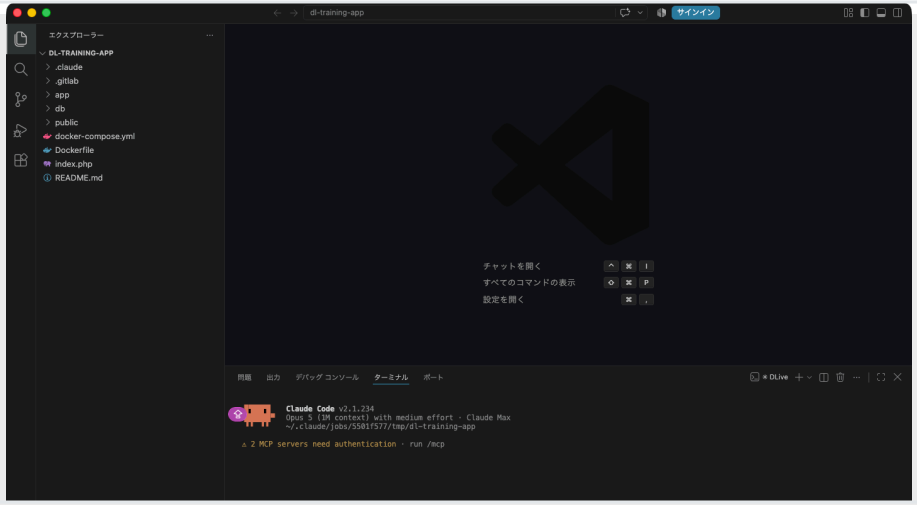
! で始まるバッククォートの行は、Claude が読む前にシェルで実行され出力に置き換わります。
この skill は誘導です。50 行を超えても light が選ばれることはあります。保証は Part3 の CI ゲートにだけ置きます。/review-route と打つと、/tasks に review-full と観点別3本が並びます。
出典: https://code.claude.com/docs/en/skills

ワーク

Step6 同じ差分への並列レビュー

Step6 [12min]

/tasks に4行並ぶ状態が正しい画面




```
review-full          sonnet   running
security-reviewer    opus     xhigh
spec-reviewer        sonnet   high
test-gap-reviewer    haiku    medium
```

画面: 統合ターミナルの実画面。下は /tasks の出力例です。

1 初版パッチの適用

統合ターミナルで
git apply patches\
issue2_draft.patch を実行

2 レビューの起動

入力欄に /review-route と
打ちます。4本が並列で
走ります。

3 4行の読み方

/tasks を打つと review-full と
3本のレビュアーの行に
モデル名と effort が出ます。
返った findings は 088 の表へ

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

要確認 /tasks に4本並んだ実画面へ差し替え(前夜撮影)。下の4行は出力の形です。

ワーク

指摘の重なりと差

[6min]

同じ差分で3本が違う場所を指す記録

指摘の場所(file:line)	security	spec	test-gap	review-full	Codex	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。Codex 列は 089 の並行手順を使った人だけ埋めます。
Codex は P2/P3 も出すので、件数をそのまま比べません。この表は 097 の分母ではありません。

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。

補足

Codex の並行手順(Part2)

観点を3回に分けた codex exec で同じ比較

PowerShell(リポジトリのルートで実行)

```
$rules = Get-Content REVIEW.md -Raw
codex exec -s read-only -m gpt-5.5 --output-schema .claude\review.schema.json -o sec.json `
  "$rules 観点は外部入力が SQL と画面に届く経路だけ。対象: git diff main"
codex exec -s read-only -m gpt-5.5 --output-schema .claude\review.schema.json -o spec.json `
  "$rules 観点は Issue の完了条件と差分の一致だけ。対象: git diff main"
codex exec -s read-only -m gpt-5.5 --output-schema .claude\review.schema.json -o gap.json `
  "$rules 観点は変更した関数にテストがあるかだけ。対象: git diff main"
```

- サブエージェントの model に Claude 系以外は指定できません。Codex では codex exec を観点の数だけ呼びます。
- -s read-only で作業ツリーは変わりません。--output-schema で findings の形に揃えます。
  3つの JSON を 088 の Codex 列に転記します。
- AGENTS.md の ## Code Review Rules は codex exec も読みます。Step2 で写した内容がここで効きます。

> **要確認**　-m に渡すモデル名は研修時点の Codex 側のラインアップで確定します

出典: https://developers.openai.com/codex/cli/reference,
https://learn.chatgpt.com/docs/config-file/config-reference

ワーク

委譲判定シート

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		ここは CI に落とす		☐ AI が直す ☐ 人が決める
要調査				☐ AI が直す ☐ 人が決める
将来課題				☐ AI が直す ☐ 人が決める
意図した設計				☐ AI が直す ☐ 人が決める

- 088 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、Part3 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、意図した設計が巻き戻ります。
  「意図した設計」の行が空の人は、見送る判断をしていない可能性があります。

ワーク

Step7 ai_gate を本物のゲートにする

Step7 [6min]

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
  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
```

- 配布済みの .gitlab-ci.yml には ai_gate の最後にある allow_failure: true の1行を消します。image の行は残します。
- 先頭の variables の GATE_LEVEL: "P1" を、080 で決めた Important の重さに合わせます。
- ai_review は findings.json を成果物に出し、ai_gate が jq で数えて exit 1 を返します。
- 落とす判断を書く場所は ci/ai_gate.sh の1箇所です。期待される画面は allow_failure が ai_gate の下に無い状態です。

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

ワーク

Step8 MR を出す

Step8 [6min]

push option 1本で MR が立つ手順

Step8-1

git switch -c issue-2-gate でブランチを切ります。
REVIEW.md・.claude/known_issues.md・agents 3本・skill・.gitlab-ci.yml と
Issue #2 の初版差分をまとめて追加し、設定と初版の2回に分けてコミットします。
期待される画面: git log --oneline -2 に2行

Step8-2

push option を使い、ブランチの push と MR の作成を1回で行います。

```
git push -u origin HEAD -o merge_request.create `
  -o merge_request.target=main `
  -o merge_request.title="Issue #2 検索条件の改修とレビューゲート"
```

期待される画面: ターミナルに MR の URL が1行出る

Step8-3

MR の URL を開き、Pipelines タブでジョブの並びを見ます。
lint-php83 / lint-php54 / test / test-phpunit / ai_review / ai_gate の6ジョブです。
ai_review が緑で MR にレビューのコメントが1本付き、ai_gate が赤になります。
緑のままの人は findings.json の中身を 093 の見方で確認します。

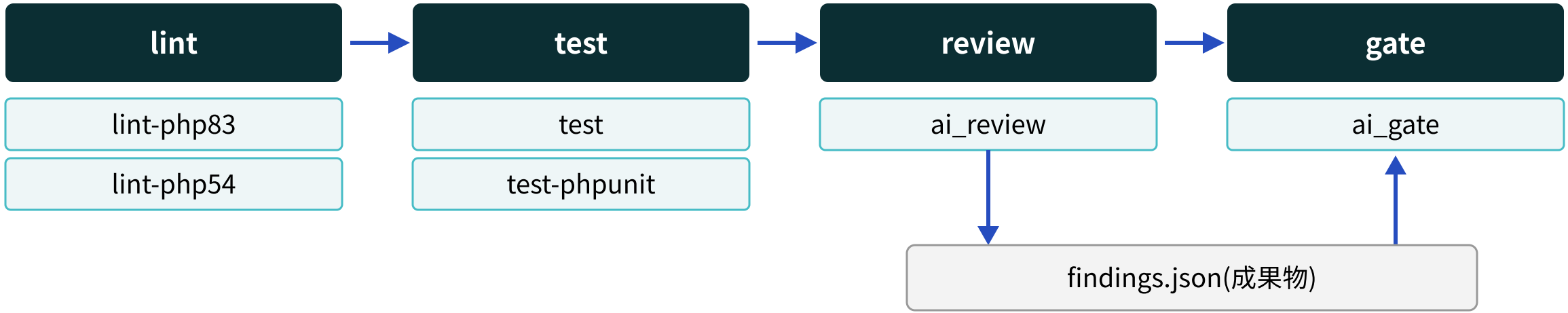
main は保護されていて直接 push できません。ai_gate が赤になるのがこの Step の期待どおりです。

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

解説

MR と CI の見方

赤の場所で直す層が決まる読み方



赤のとき直す場所

5.4 非対応構文 D2-01 の一覧を見る	生成テストか実装 どちらかが古い	API キーか REVIEW.md の形	指摘を直すか 採否の理由を MR に書く

ai_review の赤はレビューが走らなかった状態、ai_gate の赤は Important が残っている状態です。
ai_gate のログには「ゲートで止めた指摘 N 件」と [P0] file:line message の行が出ます。
findings.json は ai_review の成果物から1週間ダウンロードできます。

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

ワーク

Step9 ゲートで落ちて直す

Step9 [10min]

MR に残る1往復が合格の証拠

確認	記録する項目	どこに書くか
☐	ai_gate が止めた指摘の severity と file:line	MR コメント。ai_review の投稿の下に自分で返信します。
☐	その指摘の4分類。即修正・要調査・将来課題・意図した設計のどれか	同じ返信に1語
☐	「即修正」を直したコミットの SHA	git log --oneline -1 の出力を返信に貼ります。
☐	再実行後の ai_gate の結果(緑か、残った件数)	Pipelines の2回目を見て返信に追記します。
☐	「意図した設計」で見送った指摘の理由	MR コメントに1行。ゲートは通らないので GATE_MIN_CONFIDENCE を見直します。

止めた指摘は MR コメントの本文をそのまま貼り、「この1件だけ直す。他は触らない」と添えます。
1件1コミットで、直したら同じブランチに push します。MR は作り直しません。
見送った指摘でゲートが落ち続ける場合は、REVIEW.md の Do not report か .claude/known_issues.md に1行足します。

ワーク

Step10 機能テストの生成

Step10 [12min]

落ちるはずのケースを人が先に3つ書く生成

Step10-1 紙に3ケース

見積一覧の検索について、期待する結果を右の枠に
3つ書きます。AI には渡しません。

Step10-2 テストを生成させる

```
tests/phpunit/EstimateSearchTest.php を作る
PHPUnit 4.8 と PHP 5.4 で動く書き方
ケースは紙に書いた3つ。実装には触らない
db/seed.sql の初期データを前提にする
```

Step10-3 テストだけを push

test-phpunit ジョブを見ます。ケース3 が赤、
ケース1・2 が緑になれば正しい結果です。

テストを AI に書かせる前に、何が通って
何が落ちるべきかを人が決めます。

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

紙に書く3ケース

ケース1
customer_name の部分一致で返る行

ケース2
date_from と date_to の境界日(当日を含むか)

ケース3
customer_name に ' OR 1=1 -- を入れた
ときに返る行数

期待される画面

class EstimateSearchTest extends
PHPUnit_Framework_TestCase で始まる1本

生成物: tests\phpunit\EstimateSearchTest.php
CI では PHP 5.6 + PHPUnit 4.8.36 で走ります。

ワーク

Step11 破壊検知の比較

Step11 [8min]

同じリファクタで赤が出るのはテストがある側だけ

テスト無し

Issue #2 の初版

↓

リファクタ依頼

↓

lint 緑 / test 緑 / test-phpunit 緑
検索のテストが無い

↓

壊れていても気づけない

同じ依頼文

テスト有り(Step10 の3ケース)

Issue #2 の初版

↓

リファクタ依頼

↓

test-phpunit 赤
ケース1か2が落ちる

↓

壊れた場所が行番号で出る

講師が配る同じリファクタ依頼を、テストを外した状態と Step10 のテストがある状態で1回ずつ頼みます。
壊れなければ両方緑です。気づけるのは右だけという非対称は変わりません。080 のカード4 と突き合わせます。

要確認　リファクタ依頼の文言は、講師環境で赤が再現する形に当日確定します

ワーク

検出率の記録

既知の欠陥で測る当たりと取りこぼし

的中率 = 検出した仕込み欠陥 ÷ 指摘総数　読む時間の無駄がどれだけあるか

検出率 = 検出した仕込み欠陥 ÷ 仕込み欠陥の総数　取りこぼしがどれだけあるか

仕込み欠陥総数	検出数	指摘総数	的中率	検出率
講師が開示します	当たっていた件数	findings の件数	検出数÷指摘総数	検出数÷欠陥総数

ここで講師が Issue #2・#3 の仕込み欠陥を開きます。088 の表と findings.json で何件当たったかを数えます。
指摘総数は findings.json の findings 配列の件数です(186 の count_findings で数えます)。
088 の表は重なりを見るためのもので、分母には使いません。
この2つの数字は持ち帰り台帳に書き、D2-10 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: _____

補足

Codex の並行手順(Part3)

読む側だけ Codex に替えた同じ2ジョブ

.gitlab-ci.yml(ai_review を Codex で回す版)

```
ai_review:
  stage: review
  image: node:22-bookworm-slim
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  before_script:
    - npm install -g @openai/codex > /dev/null
  script:
    - codex exec -s read-only --output-schema .claude/review.schema.json
      -o findings.json "$(cat REVIEW.md) 対象: git diff
      origin/$CI_MERGE_REQUEST_TARGET_BRANCH_NAME"
  artifacts: { paths: [findings.json] }
```

- ai_gate は変えません。findings.json の形が同じなら、読む側が Codex でも ai_gate.sh はそのまま動きます。
- テスト生成(Step10)は codex exec -s workspace-write に同じ3ケースを渡します。

出典: https://developers.openai.com/codex/cli/reference, https://code.claude.com/docs/en/gitlab-ci-cd

要確認　CI 変数 OPENAI_API_KEY の登録と runner から api.openai.com への到達は当日版の環境で確認します

ワーク

合格の確認

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 が1行ある
☐	.claude\agents\ に security-reviewer / spec-reviewer / test-gap-reviewer があり、/tasks でモデルが3本とも違う
☐	MR で ai_gate が1回赤になり、直したコミットと 採否の返信が MR に残っている
☐	tests\phpunit\EstimateSearchTest.php が test-phpunit ジョブで走り、ケース3 が初版で赤になった
☐	097 の的中率と検出率が台帳に書けている

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、
D2-09 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

ai_review が赤
CI 変数 ANTHROPIC_API_KEY
の未登録

ai_gate が緑のまま
findings の confidence が低い。
GATE_MIN_CONFIDENCE を確認

test-phpunit が全部赤
::class か ?? の混入。
lint-php54 のログを先に見る

/review-route が light を選ぶ
skill は誘導。CI 側で止まって
いれば合格

持ち帰り物と自社への書き戻し

D2 と smart3pm に同じ形で置ける7点

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
AGENTS.md の節	既存 AGENTS.md に節を追加	同じ
agents 3本	.claude/agents/ に置く	同じ。md は ASCII 制約の対象外
review-route skill	.claude/skills/review-route/	同じ。! は bash で動く
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 4.8 の形で	検出率は台帳と D2-10 へ

持ち帰り台帳の D2-05 の行に7点と 097 の数字を書きます。
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 #3 の初版パッチを当て、test-gap-reviewer が引当の DB 書き込みを P1 で指すかを見ます。
ai_review_codex を別ジョブで足し、1つの MR に2社のレビューが付く状態を作ります。
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

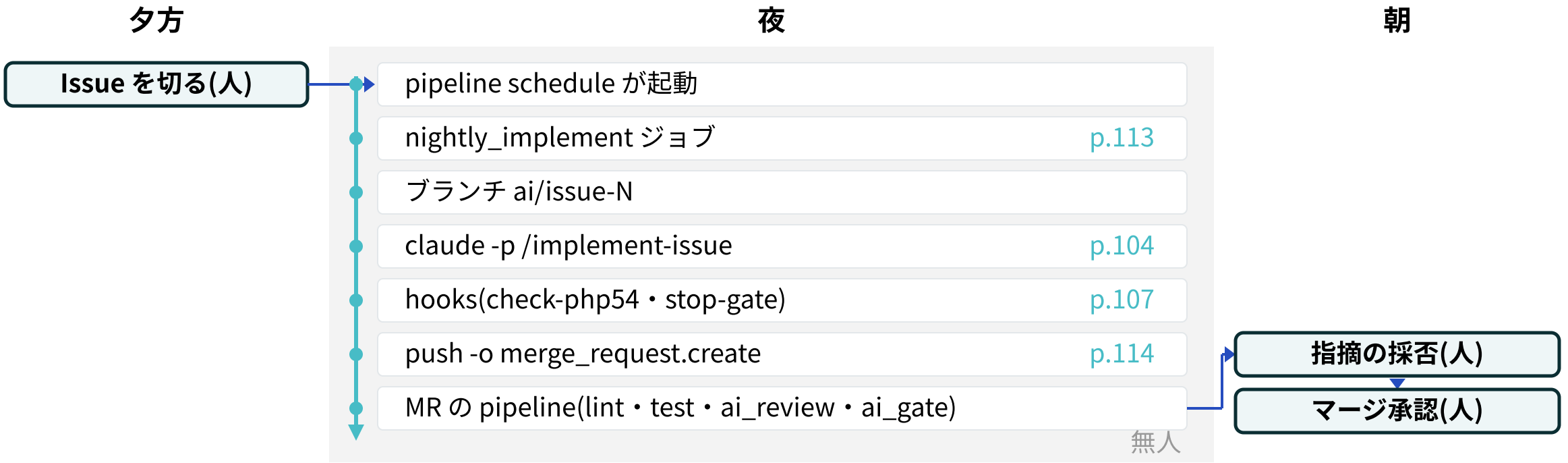
[30min]

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

解説

一晩の流れと部品の対応

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 auto で動いていても、-p の既定は全プランで default(Manual)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「何もしなかった」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(Manual)

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

解説

無人実行の最小セット

無人で動かすフラグは3本

PowerShell(nightly_implement ジョブの script 行)

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

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。
--output-format json で結果を構造化し、後続のジョブが読めるようにします。

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

Claude Code と Codex の無人実行

同じ3要素を別の名前で書く2つの CLI

Claude Code

無人実行

`claude -p "..."`

対話と同じコマンドに -p を付ける

権限の止め金

`--permission-mode dontAsk`

`--allowedTools`

書き込み範囲

道具単位で許可を列挙する

パスの制限は permissions 側に書く

Codex

無人実行

`codex exec "..."`

--full-auto は非推奨

権限の止め金

`approval_policy = "never"`

設定ファイルに書く

書き込み範囲

-s workspace-write は cwd と /tmp のみ

.git は読み取り専用。writable_roots に明示

要確認 --max-turns 相当は Codex 側に同名のものが無い

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

※ 各ロゴは各社の商標です。

補足

--bare で読まなくなるもの

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	5.4 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

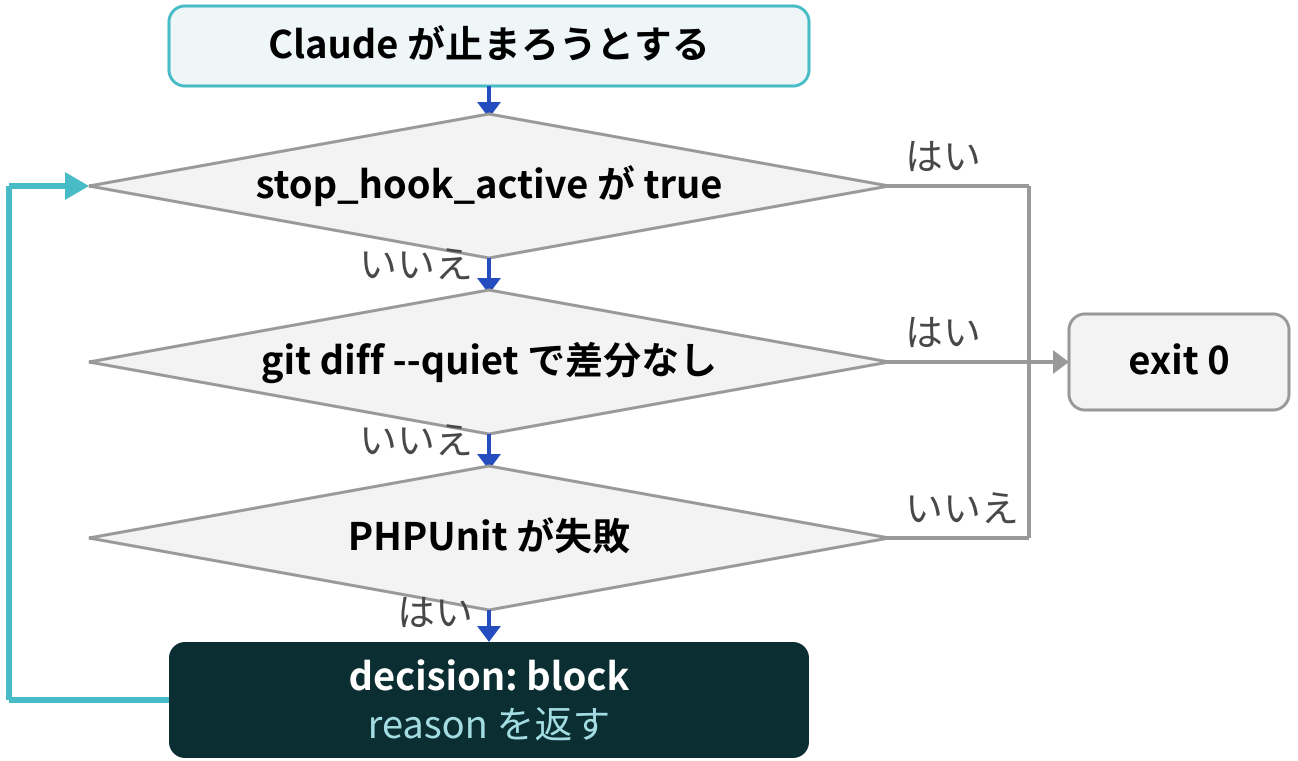
--bare は hooks・skills・CLAUDE.md・MCP・auto memory を全部スキップします。
夜間の実装ジョブに付けると check-php54 も stop-gate も動きません。
履歴を切って差分だけを見せたいレビュー側(ai_review)にだけ使います。将来 -p の既定になる予定です。

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

解説

Stop hook の判定順

stop_hook_active を見ないと永久に回る



1つ目の判定を外した版

block を返すたびに
Stop が再入して
止まりません

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 すると止まりません。

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

解説

stop-gate.ps1 の骨格

3つの判定と block を返す1か所

.claude/hooks/stop-gate.ps1

判定1 継続中か

stop_hook_active が
true なら何もせず
通します。

判定2 差分の有無

差分がゼロなら
何もせず通します。

判定3 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。

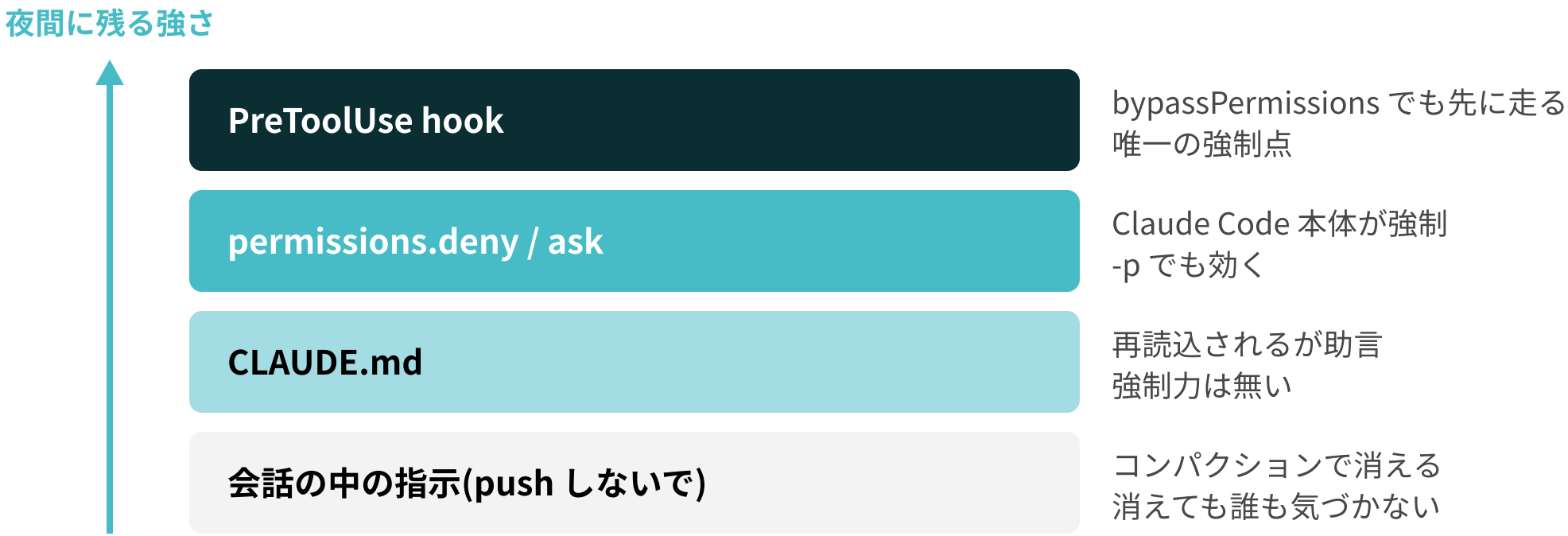
演習リポのテストは php tests/run_tests.php です。D2 の実運用ではこの行を
vendor\bin\phpunit に差し替えます(パスは当日確定)。
コードと文字列は ASCII だけで書きます。D2 の PowerShell 5.1 制約に合わせています。
スクリプト全文は講師用資料と配布 ZIP の .claude/hooks/ にあります。

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

解説

境界を書く場所の強さ

口頭の禁止はコンパクションで消える



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

解説

夜間ループの止め金一覧

配布のみ

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

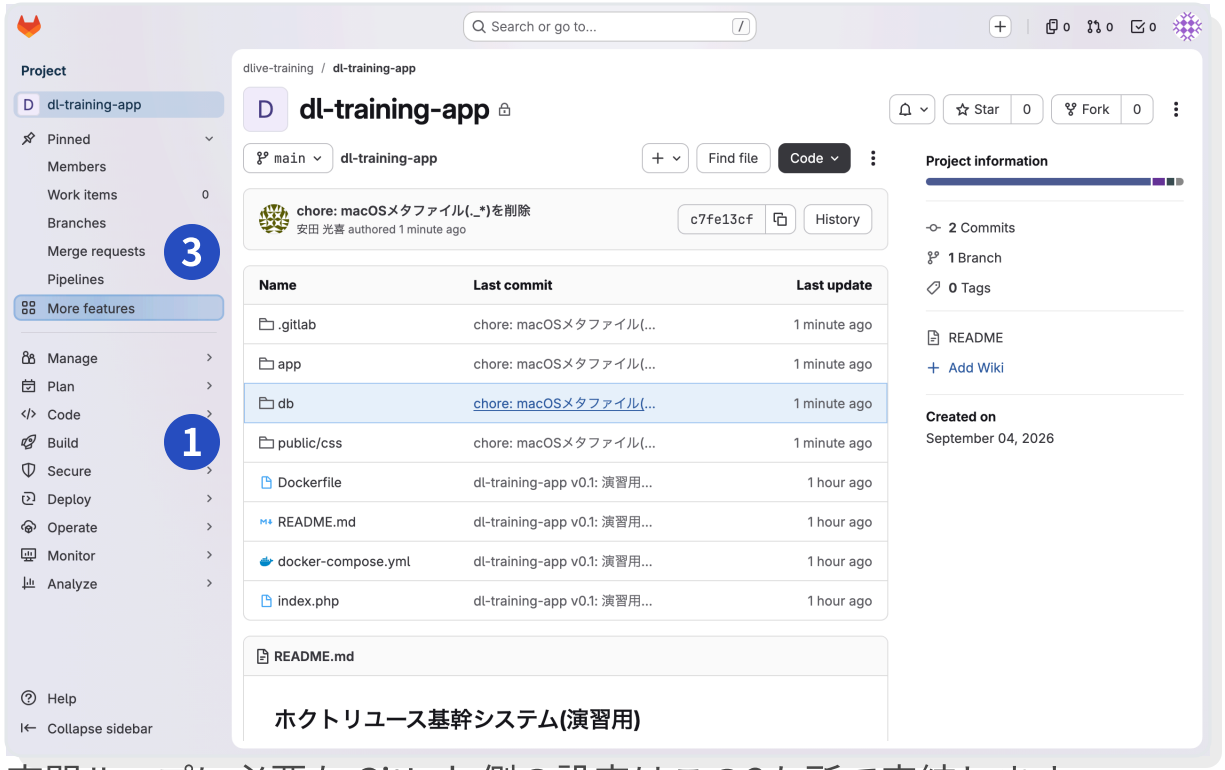
止め金を1か所に集めると、その1か所が外れたときに全部外れます。
deny は文字列照合です。/bin/rm や bash -c はすり抜けるのでブランチ保護と二重にします。
D2-08 ではこの5行を全部書いた状態でだけ schedule を流します。

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

実演

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/

解説

スケジューラの置き場所

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。

解説

nightly_implement ジョブ

schedule と手動の両方から起動できる1ジョブ

.gitlab-ci.yml(抜粋)

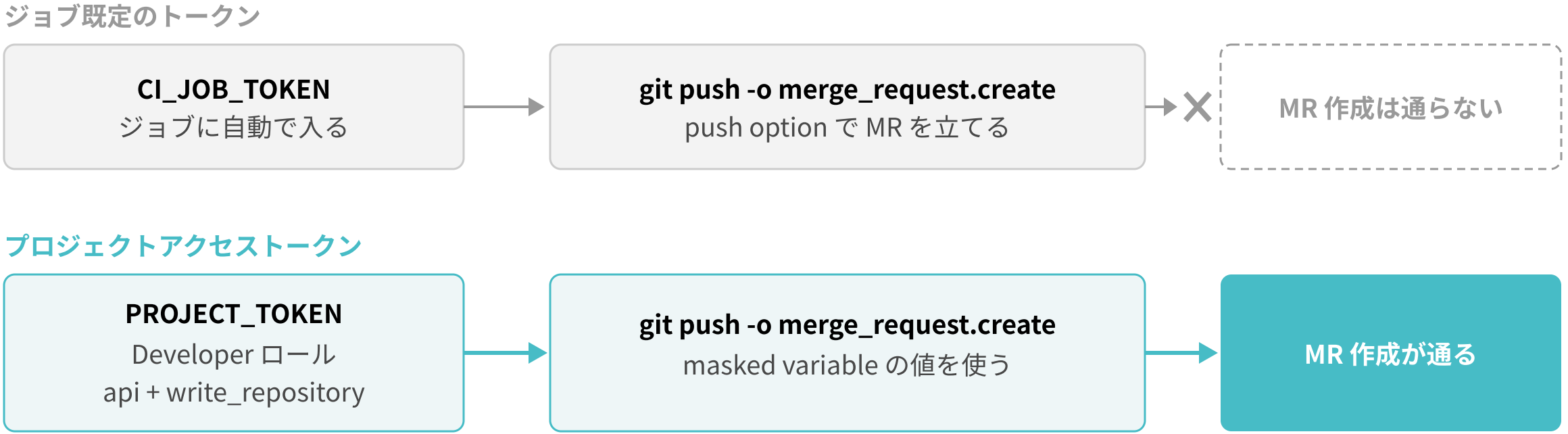
```
nightly_implement:
  stage: implement
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
    - when: manual
  timeout: 30m
  script:
    - git checkout -b "ai/issue-$ISSUE_IID"
    - claude -p "/implement-issue $ISSUE_IID"
      --permission-mode dontAsk --max-turns 15 --output-format json
      --allowedTools "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)"
    - git push "https://oauth2:${PROJECT_TOKEN}@${CI_SERVER_HOST}/${CI_PROJECT_PATH}.git"
      HEAD -o merge_request.create -o merge_request.target=main
```

rules の2行で、夜は schedule から、研修中は講師が手動で流します。
18名同時実行を避けるため、当日は when: manual で3名ずつ段階的に起動します。
最後の push に -o merge_request.create を付けると、push と同時に MR が立ちます。
出典: https://docs.gitlab.com/topics/git/commit/#push-options

解説

MR 作成に使うトークン

CI_JOB_TOKEN では MR が立たない



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、ジョブのログに出ないようにします。

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**

D2 の core.md は、全ルールにこの3つのタグを付けています。

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

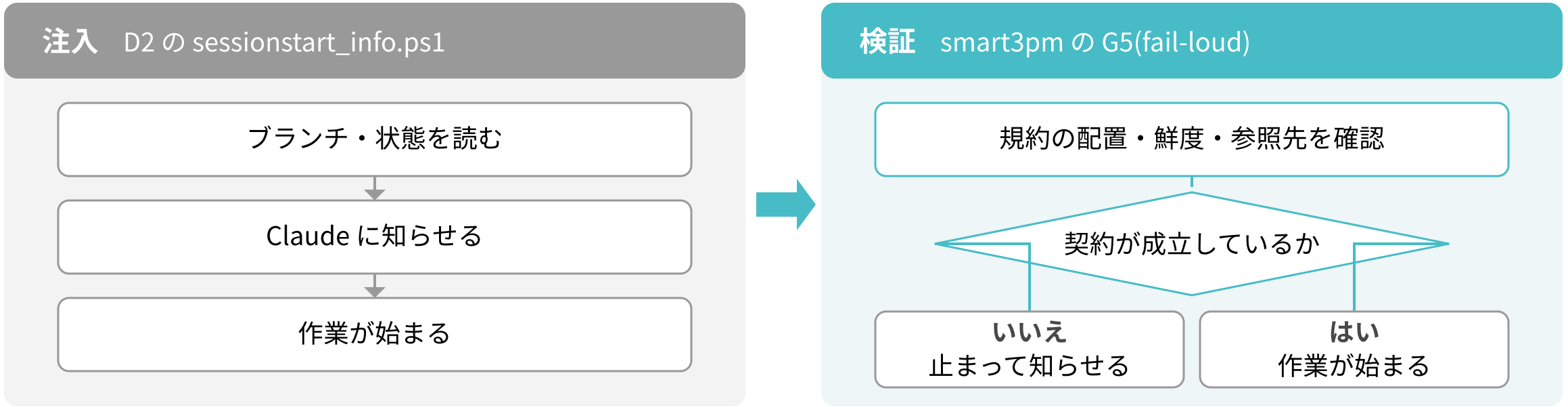
GitLab CI 後
ジョブが判定して終了コード

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すのが GitLab CI の役目で、ここが足すと効く箇所です。
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。

解説

SessionStart の注入と検証

注入は教えるだけで止まらない



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。
D2 の SessionStart は、smart3pm 側の fail-loud の考え方を足すと効く箇所です。

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。**

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。

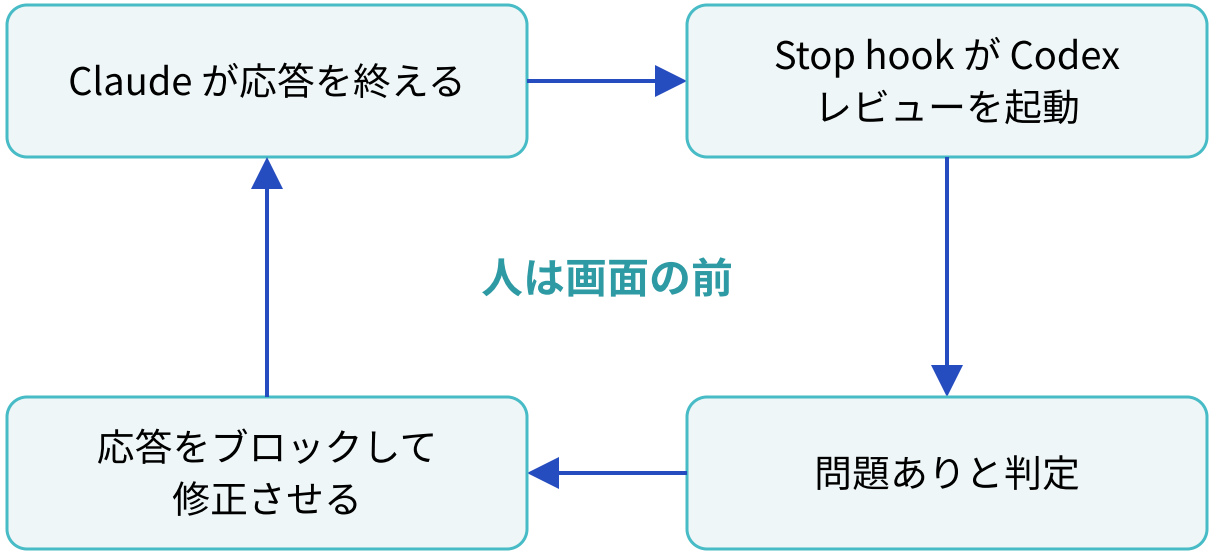
出典: https://claude.com/blog/how-anthropic-secures-its-ai-native-software-development-lifecycle (2026-07-21)

※ 各ロゴは各社の商標です。

解説

Review Gate の警告

監視下でだけ回すループもある



README の警告

長時間の Claude/Codex ループに
なり、使用量を急速に消費します。
監視できる時だけ有効化する、と
README 自身が書いています。

OpenAI 公式のプラグイン codex-plugin-cc(2026-03-30 公開)には Review Gate があります。
/codex:setup --enable-review-gate で有効化すると、応答のたびに Codex レビューが走ります。
設計した側が監視下限定と書いている事例です。夜間ループに組み込む部品ではありません。

出典: https://github.com/openai/codex-plugin-cc

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

Codex レビューを 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

結果

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

Claude Code が Codex レビューを飛ばす問題に、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)

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

01

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

朝

指摘の採否

02

ゲートを通った分だけ読む
ai_gate を通った P0・P1 だけ

朝

マージ承認

03

巻き戻せない変更を開く
含まれていれば必ず開く

ループの中でコードを読む係は置きません。

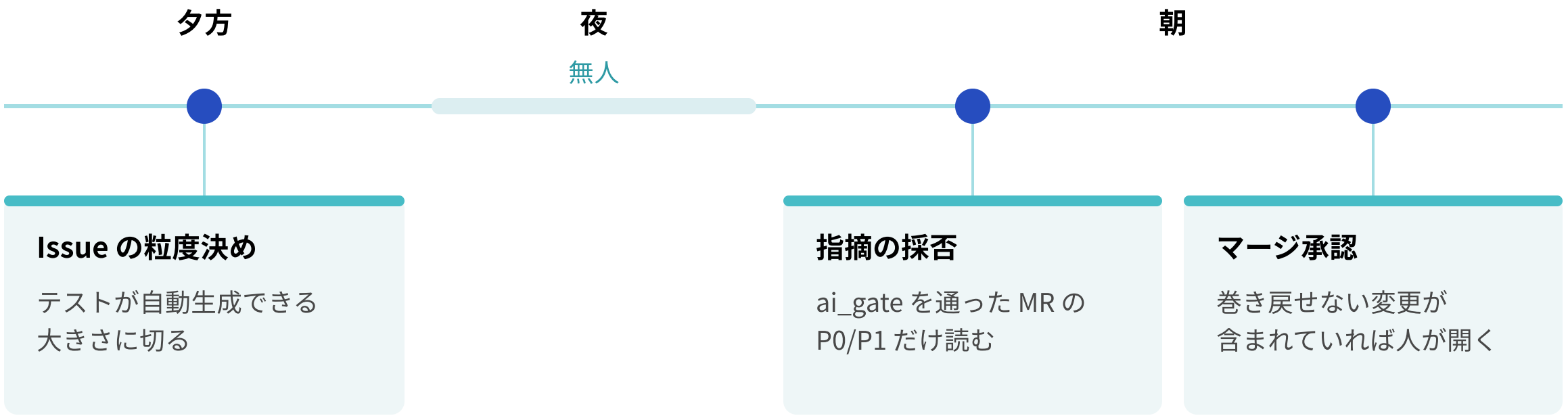
夜間ループで人が持つのは、何を任せるかを決める夕方と、出てきたものを通すかを決める朝の仕事です。
差分を1行ずつ読む係を置くと、夜間に回した分だけ朝の読み量が増えて自動化の意味が消えます。
057 で見たリスク加重の抽出は、夜間ループでは ai_gate を通った分だけに当てます。
巻き戻せない変更(DB・外部連携・権限)だけを人が開きます。

解説

一晩の中で人が触る3点

配布のみ

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。

解説

複数リポジトリを束ねる3つの形

配布のみ

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。

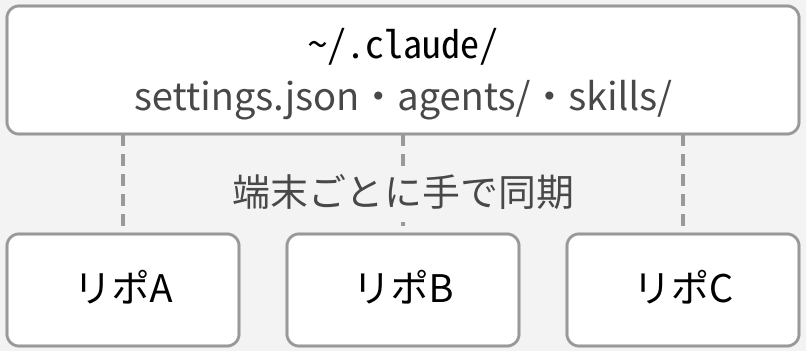
出典: https://code.claude.com/docs/en/agents

共有設定の置き方2案

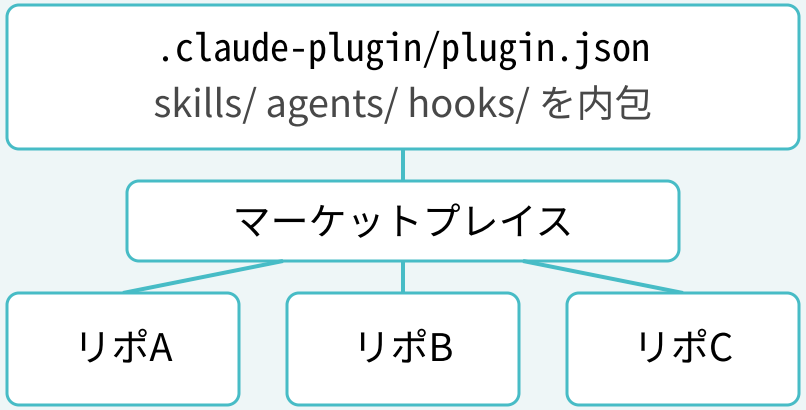
配布と版管理が要るなら plugin に束ねる

要確認 複数リポジトリの統括構成は 9/10 時点で未受領。受領後にこの2案へ当てます

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要ります。
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

解説

Codex で夜間ループを組むときの差

配布のみ

止まる仕組みは両方にあり名前だけ違う

部品	Claude Code	Codex
無人実行	claude -p --permission-mode dontAsk	codex exec -s workspace-write approval_policy = "never"
書き込み範囲	allowedTools で道具単位	cwd と /tmp のみ .git は writable_roots に明示
Stop hook	.claude/settings.json の hooks.Stop	.codex/hooks.json の Stop ※
block の返し方	{"decision":"block","reason":"..."}	同じ JSON 形式で block を返す
レビュー出力	--output-format json --json-schema	--output-schema と -o findings.json

※ プロジェクト配下の .codex は hooks が対話で発火しない既知 issue #17532

Codex にも同形の hooks.json があり、Stop で block を返すと継続プロンプトになります。
.codex/config.toml は trust_level が trusted のときだけ読まれます。効かないときはほぼこれです。
Codex のみの受講者は D2-08 で右の列の形で同じループを組みます。

出典: https://learn.chatgpt.com/docs/hooks

※ 各ロゴは各社の商標です。

ワーク

夜間に回せる範囲の自己チェック

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
☐	PROJECT_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3名ずつ流します。

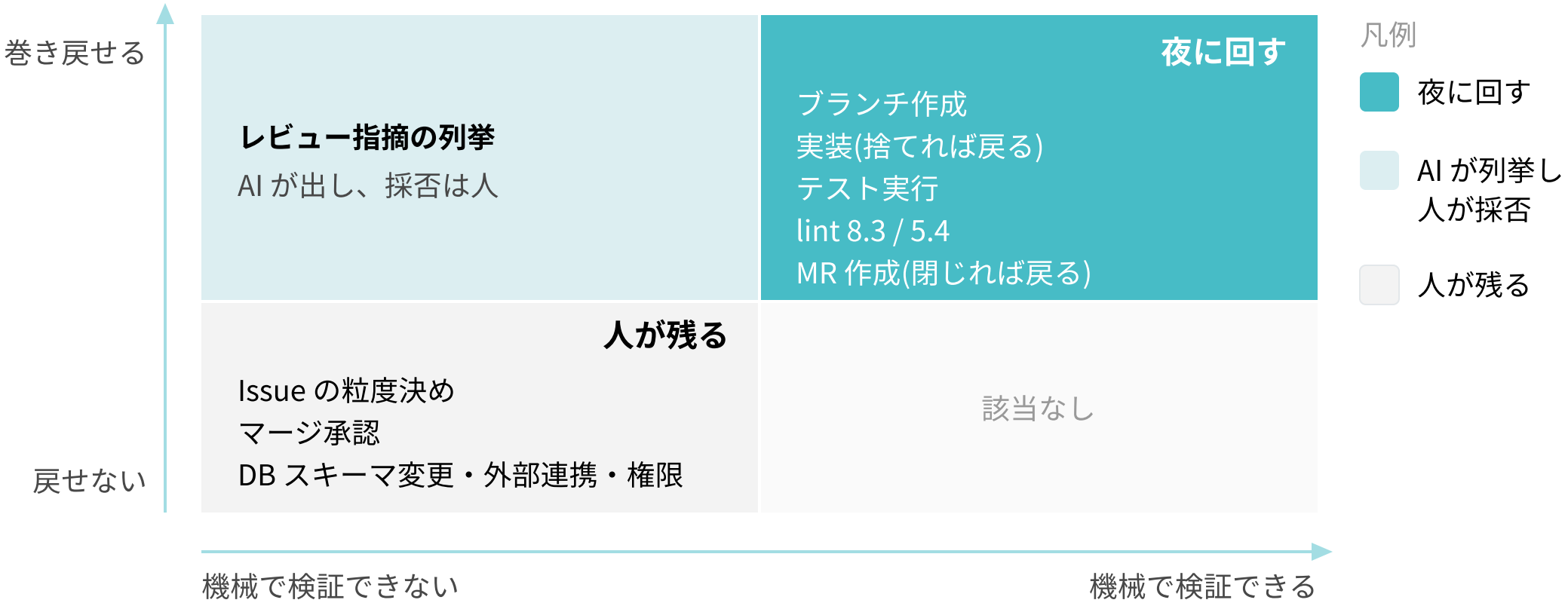
7行のうち1つでも空なら、schedule は流さず手動で回します。置き場所が5か所に分かれているのは 110 と同じ考え方です。
D2-10 の8層チェック表を塗り直すとき、この7行が到達した層の根拠になります。

解説

夜間ループの工程別の線引き

配布のみ

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

120 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

台帳の行番号は D2-09 で確定します。

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。統括構成を受領できれば 123 の2案に当てます。

D2-07

プチ演習6 止まる Stop hook

テストが通るまで止まらず、通ったら必ず止まる関所

つくるもの: .claude/hooks/stop-gate.ps1 と settings.json の Stop 登録
踏むもの: 止まらない版の永久ループ(講師実演)と正しい版の1回継続
持ち帰り: D2 向け stop-gate.ps1、smart3pm 向け stop-gate.sh、cases.json 2件

ワーク [15min]

ここで作るフックは D2-08 の夜間ジョブでそのまま動きます

解説

1ターンの中の Stop の位置

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

ワーク

前提確認

未コミットの差分がある状態で始める

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 --version`	2.1.267 以降	
演習フォルダ	VSCode で dl-training-app を開く	左ツリーに app/ と tests/	
未コミット差分	`git status`	作業ブランチに変更が残っている	
テストランナー	`php tests/run_tests.php`	11件の結果が出る	**要確認**
Codex 保有者	`codex --version`	版が出る。.codex/ を作る準備	

差分が無いと Stop hook は何もせず通ります。ハンズオン2で作った
作業ブランチの変更をそのまま残しておいてください。

> **要確認**　受講者PCに php があるかは研修3日前に1台で実機確認します。
> php が無い端末は講師配布の fake_tests.ps1 で挙動だけ踏みます(配布 ZIP の d2-07/)。

ワーク

Step0 紙に書く3行

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

Step0 [2min]

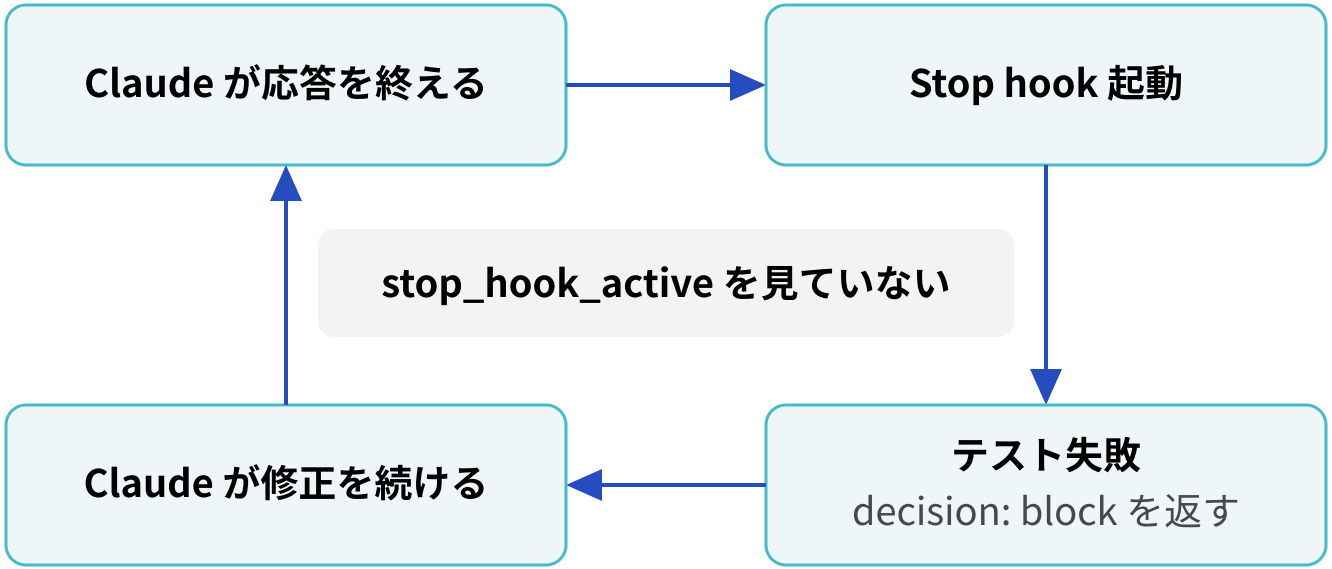
1. **通す条件**
   差分ゼロ・テスト通過・すでに継続中。この3つを自分の言葉で並べます
2. **継続中の見分け方**
   「すでに継続中」をフックはどの情報で知るか。入力 JSON のどのキーかを書きます
3. **継続の上限回数**
   何回まで続けさせるか。根拠はトークン・時間・夜間ジョブの timeout です

答えはこの後のページに出ます。先に自分の案を持つと、出てきたコードの
どの行が自分の案と違うかが見えます。

実演

止まらない版の挙動

継続中かどうかを見ないフックは永久に続く



止める手段
Ctrl+C

消えるもの
トークンと時間

講師環境で stop_hook_active の判定を外した版を回します
テストが常に落ちる状態なので、応答を終えるたび block が返ります
画面の反復回数と /usage の消費を見ていてください。受講者は操作しません

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

ワーク

Step1 stop-gate.ps1

通す3条件を先に書き最後だけ block を返す

.claude/hooks/stop-gate.ps1(ASCII のみ、PowerShell 5.1)

Step1 [3min]

```
1  # .claude/hooks/stop-gate.ps1 (ASCII only)
2  $in = [Console]::In.ReadToEnd() | ConvertFrom-Json
3  if ($in.stop_hook_active -eq $true) { exit 0 }
4  git diff --quiet; if ($LASTEXITCODE -eq 0) { exit 0 }
5  php tests/run_tests.php | Out-Null
6  if ($LASTEXITCODE -ne 0) {
7    $r = 'Tests failed. Run php tests/run_tests.php again.'
8    Write-Output ('{"decision":"block","reason":"' + $r + '"}')
9  }
10 exit 0
```

3行目
継続中なら通す

4行目
差分ゼロなら通す

7〜8行目
テスト失敗なら block

10行目
テスト通過なら通す

3行目が無いと永久ループになります。block は exit 0 のまま JSON を返します　出典: https://code.claude.com/docs/en/hooks

php が無いと $LASTEXITCODE が前の値のまま残るので、5行目の直前に $LASTEXITCODE = 0 を置きます

ワーク

Step2 Claude なしで単体テスト

フックはまず手で JSON を流して確かめる

Step2 [2min]

操作A 継続中の入力を流す

```
'{"stop_hook_active":true,"session_id":"t1"}' |
  powershell -NoProfile -File .claude/hooks/stop-gate.ps1; $LASTEXITCODE
```

期待される画面A

```
0      何も出力されず 0 だけが返ります
```

操作B 初回停止の入力を流す

```
'{"stop_hook_active":false,"session_id":"t1"}' |
  powershell -NoProfile -File .claude/hooks/stop-gate.ps1; $LASTEXITCODE
```

期待される画面B

```
{"decision":"block","reason":"Tests failed. Run php tests/run_tests.php again."}
0
```

Day1 プチ演習3と同じ手順です。この2本を D2 の hooks/tests/cases.json に追加してください。
smart3pm 担当は stop-gate.sh 用の .test.sh を同じ2ケースで書きます。

ワーク

Step3 settings.json への登録

登録は Stop イベント1本と timeout だけ

Step3 [2min]

Claude Code

.claude/settings.json

```
{ "hooks": { "Stop": [ { "hooks": [ {
    "type": "command",
    "command": "powershell -NoProfile
      -ExecutionPolicy Bypass -File
      .claude/hooks/stop-gate.ps1",
    "timeout": 120
} ] } ] } }
```

Codex

.codex/hooks.json

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

要確認 Codex の Stop 入力に stop_hook_active 相当があるかは未確認

Codex も同形の hooks.json を持ち、Stop で block を返すと継続プロンプトになります。リポジトリ直下の .codex 設定が対話セッションで発火しない既知の issue があるため、動かない時は ~/.codex/hooks.json に置きます。

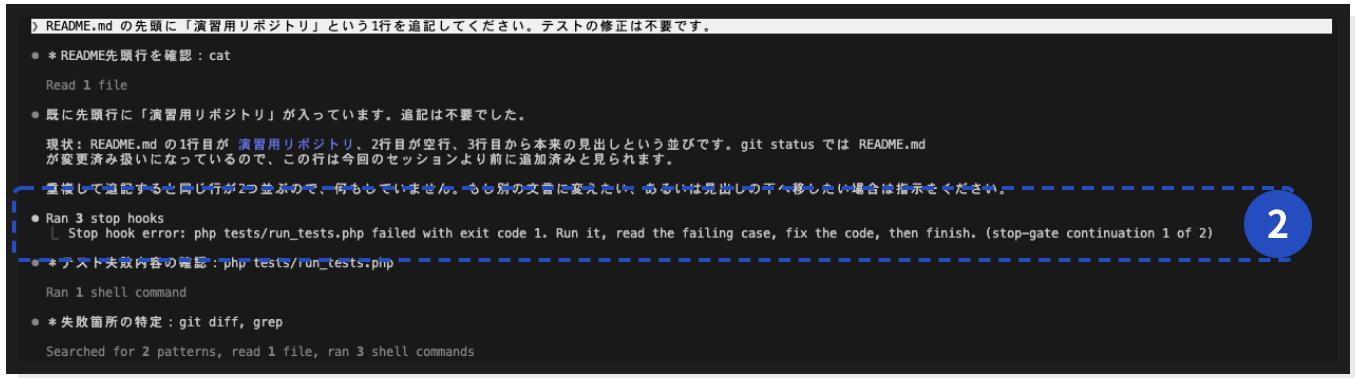
出典: https://code.claude.com/docs/en/hooks ・ https://learn.chatgpt.com/docs/hooks
https://github.com/openai/codex-plugin-cc

※ 各ロゴは各社の商標です。

ワーク

Step4 1回だけ継続される画面

1回目だけ続いて2回目で止まる



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。

Step4 [3min]

1. **入力は1行**
   「テストを通してください」と頼みます
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

操作: tests/ の1件を壊した状態で「テストを通してください」と頼みます
テストが通っていれば1回目で止まります。壊した状態なら1回続いてから止まります

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

画面: VSCode 統合ターミナルの Claude Code

補足

OK 基準とつまずき

合格は3つの画面が再現できること

OK 基準	確認の仕方	対処
OK1 単体テスト2ケース	Step2 の A と B が期待どおり	cases.json に追加済み
OK2 1回継続	Step4 で reason が1回出て止まる	settings.json の Stop 登録を確認
OK3 差分ゼロで素通り	コミット後に停止させても無出力	git diff --quiet の行を確認
つまずき	**見え方**	**対処**
実行ポリシーで停止	running scripts is disabled	-ExecutionPolicy Bypass を追加
JSON が読まれない	block されず止まる	exit 2 を混ぜていないか
全角文字で文字化け	reason が化ける	ps1 を ASCII だけにする

OK3 は忘れやすい確認です。差分ゼロで block を返すフックは、
何もしていないセッションを止められなくします。

持ち帰り物と発展

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
単体テスト2ケース	hooks/tests/cases.json に追記	stop-gate.test.sh を新規追加
テストコマンド	vendor\bin\phpunit に差し替え	自社のテストコマンドへ

要確認　D2 実運用のテストコマンド(vendor\bin\phpunit のパスと版)は当日確認します

発展 継続回数の上限

最大継続回数を環境変数で持たせます。stop_hook_active の早期 exit を、$env:TEMP 配下の session_id 別カウンタに置き換え、$env:STOP_GATE_MAX(既定 1)に達したら exit 0 で止めます。
ハンズオン3の夜間ジョブでは、この回数と --max-turns と timeout の3つが止め金になります。

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

[120min] ワーク

Codex 保有者は右カラムの並行手順で完走できます

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。止まった箇所を台帳に書き、直す層を選んで2回目を回します。
持ち帰りは5点。夜間ジョブ、ci/nightly_implement.sh、implement-issue skill、
止め金の settings.json、schedule の設定値です。

解説

全体の流れと所要時間

止め金を書いてから回す6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。
Step4 は待ち時間が出ます。自分の組の順番が来るまで Step5 の台帳の枠を埋めておきます。

Step1	Step2	Step3	Step4	Step5	Step6
止め金を設定に書く [15min]	ローカルで予行 [15min]	夜間ジョブと schedule [15min]	3名ずつ流す [20min]	台帳と再実行 [20min]	MR と委譲判定 [10min]

▲ 講師が組ごとに実行ボタンを押します

前提確認・題材・人が先に考える [10min]

まとめ 回せる範囲と持ち帰り [15min]

Step4 が1本5分を超えたときは、講師環境の1本を全員で読む形に切り替え、各自の実行は D2-09 の時間に回します。

ワーク

開始前の確認

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
☐	main にいて作業ツリーが clean	`git branch --show-current` `git status`
☐	D2-02 の check-php54 と D2-07 の stop-gate が /hooks の一覧に出る	Claude Code で /hooks
☐	ai_gate の allow_failure を外した .gitlab-ci.yml が main に入っている(D2-05 の MR)	GitLab の Repository 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 --version が 2.1.267 以降	PowerShell
☐	remote に nightly/issue-4 ブランチが無い	GitLab の Code > Branches で検索

行3 が未了の人は、D2-05 の MR をマージしてから進みます。allow_failure のままだと
Step6 の「レビューを飛ばす」が再現しません。行6 に残っている人は講師に知らせます。

ワーク

題材 Issue #4 の読み解き

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。対応表が util.php と estimate_ctl.php と estimate_list.php の3か所にあり、util.php の1か所へ寄せます。受入条件は次の6つです。

受入条件(Issue #4 の原文)	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. app 配下に array(1, 2, 3, 4, 9) が無い	grep
3. tests/run_tests.php に対応表のテストが3件増えている	CI
4. tests/phpunit/EstimateSearchTest.php の既存テストが全部通る	CI
5. lint-php83 / lint-php54 / test / test-phpunit が緑	CI
6. MR の説明に、対応表を寄せた理由と影響範囲の調査結果が書かれている	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。

> **要確認**　Issue #4 本文の実画面は当日朝に差し替えます。条件の文言は Issue の原文です。

ワーク

人が先に考える 止まる箇所の予想

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	
2 止まる原因は「権限」「5.4 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	
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	
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	

紙かメモに5分で書きます。AI には渡しません。
Step4 のログと突き合わせ、予想が外れた行だけ 153 の台帳に理由を残します。

解説

Step1 止め金の4層

Step1 [15min]

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 Settings > Repository 講師が設定済みです
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`	Claude のターン数の上限 Codex に相当する上限はありません(161)	ci/nightly_implement.sh の --max-turns 15
permissions.deny と hooks	reset --hard / push --force / .env 参照 編集直後の 5.4 検査とテスト未通過での停止	.claude/settings.json .claude/hooks/

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段目は手元では止め金として効きますが、CI runner(Linux)では deny だけが効きます。
CI 側の 5.4 検査とテストは lint-php54 / test / test-phpunit のジョブが受けます。

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

ワーク

Step1 settings.json の止め金

Step1 [15min]

口頭の禁止を deny と hook へ

.claude/settings.json(permissions ブロック)

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

.claude/settings.json(hooks ブロック)

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

> Codex は ~/.codex/config.toml の
> sandbox_mode = "workspace-write" が
> 同じ役です。.git は読み取り専用なので
> コミットはスクリプト側で打ちます。

D1-10 で足した秘密情報の3本も同じ deny です。
Read(./.env) / Read(./.env.*) /
Read(./**/credentials*) の3本が並びます。

Step1-1 4種5本の deny が .claude/settings.json にあることを /permissions で確かめます。

Step1-2 Stop と D1-10 の PreToolUse、D2-02 の PostToolUse が /hooks に出ることを確認します。

Step1-3 settings.json を git add して git commit します。CI が読むのはコミット済みの版です。

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

ワーク

Step2 ローカルでの予行

Step2 [15min]

CI と同じ引数で踏む1回目の停止

dontAsk は allowedTools の外を全部拒否します。承認待ちで止まらず、拒否は結果に残ります。

Step2-1 VSCode の統合ターミナルで dl-training-app のルートに居ることを確認し、
mkdir ci_logs を打ちます。ci_logs フォルダができれば正しい状態です。
Step2-2 次を1つの命令として打ちます。ターン上限だけ 5 に下げ、他は CI と同じです。

```
$p = "/implement-issue 4 ブランチの作成と push はCIジョブ側で行うので、" +
     "skill の手順3と手順7は実行せず、コミットまでで終えてください。"
$tools = "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)"
claude -p $p `
  --permission-mode dontAsk `
  --allowedTools $tools `
  --max-turns 5 `
  --output-format json > ci_logs\claude-issue-4.json
```

Step2-3 git status を打ちます。ci_logs/ 以外に
変更が無くブランチが main のままなら正しい状態です。
記録する項目は終了までの秒数と git status の結果です。

> **Codex**
> codex exec -s read-only で
> AGENTS.md を読ませ、実装手順を
> 番号付きで出させます。実装はしません。

出典: https://code.claude.com/docs/en/headless, https://code.claude.com/docs/en/permission-modes

解説

Step2 結果 JSON の読み方

朝に見るのは4つの値だけ

キー	意味	朝の判断	予行で見えた値
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	
session_id	セッション ID	claude --resume <id> で続きを対話で開ける	

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 143 の予想1を見比べます。
権限で拒否されたときは、拒否されたコマンドが ci_logs\claude-issue-4.err に出ます。

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

ワーク

Step3 夜間ジョブの定義

Step3 [15min]

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"
    - when: manual
  timeout: 30m
  variables:
    ISSUE_ID: "4"
  script:
    - bash ci/nightly_implement.sh
```

Step3-1 VSCode で .gitlab-ci.yml を開き、nightly_implement を読みます。
rules の1行目が schedule、2行目が手動ボタンです。allow_failure: true が付いています。
1回目は止まる前提なのでパイプライン全体を赤にしません。ISSUE_ID の既定は 4 のままです。

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

解説

Step3 スクリプトと止め金の位置

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-$USER_TAG`	
3	`claude -p`（手順3・7を飛ばす添え書き）	`--max-turns` / deny(settings.json) / 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 を作らずに終わります。朝に見るのは ci_logs/ の JSON です。
同じ Issue を流し直すときは remote の nightly/issue-4-$USER_TAG を先に消します。
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/

ワーク

Step3 変数と schedule の登録

Step3 [15min]

作るだけで流さない今日の schedule

Step3-2 GitLab で dl-training-app を開き、Build > Pipeline schedules > New schedule を押します。

Step3-3 5つの欄を次のとおり埋め、Activated にチェックを入れて保存します。

New schedule

Description	`nightly issue-4`
Interval Pattern	`Custom 0 2 * * *`
Cron timezone	`Tokyo`
Target branch	`main`
Variables	`ISSUE_ID=4 USER_TAG=自分のイニシャル`

Codex

schedule は GitLab 側の設定です。
AI ツールの違いは出ません。
同じ操作をします。

Step3-4

一覧の自分の行に Run
pipeline schedule の再生
ボタンがあることを確認し
今日は押しません。

保存すると一覧に nightly issue-4 の行が増え、Next Run に次回の実行予定時刻が出ます。
当日はこの schedule からは起動しません。Step4 で使うのは手動ボタン側だけです。
USER_TAG はブランチ名に入ります。runner の並列数は当日朝に講師が伝えます。自社では並列数と API 予算を先に決めます。

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

実演

Step4 3名ずつ流す運用

Step4 [20min]

runner と予算を使い切らない段階実行

講師が Build > Pipelines > Run pipeline で main を選び、nightly_implement の手動ボタンを組ごとに押します。
ジョブ名は全員 nightly_implement です。講師が板書した実行時刻と USER_TAG で自分の行を探します。
待っている間は 153 の台帳の左2列を埋めます。

棒1本 = 受講者1名の nightly_implement

組の全員のジョブが終わってから次の組へ

講師が組ごとの開始時刻をホワイトボードに書きます

要確認 1本あたりの所要と費用は講師環境の事前計測で確定し、組の札に [Nmin] が入ります。
1本が5分を超えた場合、Step4 は講師環境の1本を全員で読む形に切り替えます。
各自の実行は D2-09 の時間に回します。リモート参加者はチャットで順番を受けます。

ワーク

Step4 ジョブログの観察

Step4 [20min]

止まり方が出るログの末尾3行

Step4-1

Pipelines 一覧で自分の組のパイプラインを開き、nightly_implement のジョブ名を押します。
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。

Step4-2

```
# 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
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。
```

Step4-3

Job artifacts の Browse から ci_logs/claude-issue-4.json を開き、result を読みます。
期待される画面: 止まった理由の文章。拒否されたコマンドは .err に出ます。

記録するのは、要約行の3つの数と result の最初の1文、MR ができたかどうかです。そのまま 153 に写します。

> **要確認**　turns と cost_usd は表示例です。講師環境の実走で出た値に差し替えます。

ワーク

Step5 止まった箇所の台帳

Step5 [20min]

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	.err に拒否されたコマンド。curl なら手順1、git switch や push なら手順3・7	-p のフラグ スクリプト プロンプト	allowedTools に Bash(curl *) を足す または Issue をファイルで渡す
5.4 の構文が混入	MR の lint-php54 が赤。 ?? や ::class の混入	手元の hooks と CI のジョブ	手元は check-php54。CI は phpcs のジョブを足す
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。 bash 版の stop-gate を CI に
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	自動修正ループを作らない。 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 は直し方が2通りあります。行4 の多くは、MR がまだ立っていないだけです。

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

ワーク

Step5 層を選んで再実行

Step5 [20min]

remote のブランチを消してから2回目

Step5-1

153 で選んだ直し方を、1か所だけ実装します。
allowedTools に Bash(curl *) を足すか、スクリプトで
Issue 本文を取ってファイルで渡す2行を足します。
期待される画面: 差分は ci/nightly_implement.sh の数行

Step5-2

直した ci/nightly_implement.sh は自分のブランチに置いたままにします。
`git switch -c fix-nightly-<タグ>`
コミットして push し MR を作ります。マージは講師が1本だけです。
期待される画面: 自分の MR の Changes に ci/nightly_implement.sh の数行

Step5-3

1回目の自分のブランチだけ消します。他の人のは消しません。
`git push origin --delete nightly/issue-4-<タグ>`
GitLab の Code > Branches から消しても構いません。
期待される画面: 自分のタグの付いたブランチが Branches に出ない

Step5-4

講師に「2回目の準備ができた」と伝えます。組ごとに再実行されます。
期待される画面: 「MRを作成しました。マージは人が判断します。」

Codex

Step5-1 で直すときに
claude -p の行を 161 の
codex exec に差し替える
人は、その差し替えも
同じ MR に入れます。

直す層の考え方と
Step5-2 以降の手順は
変わりません。

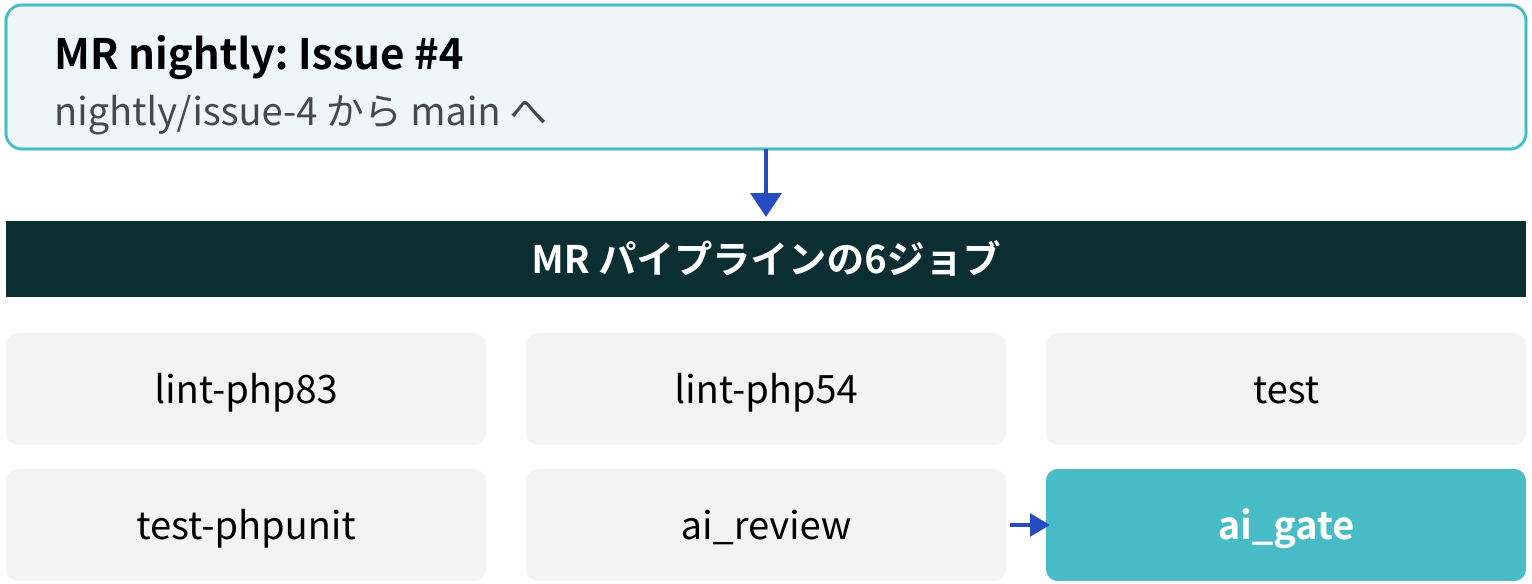
2回目の MR はブランチ名に自分のタグが入ります。別の行に当たった人は、その行にも印を付けます。
出典: https://code.claude.com/docs/en/headless

解説

Step6 MR と CI の見方

Step6 [10min]

朝に人が開く場所は3か所



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. **6ジョブの色**
   Pipelines タブで見ます
3. **Changes の差分**
   1 と 2 が揃ってから

押すボタンは1つだけです。

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。
夜間ジョブの MR も、人が作った MR と同じ6ジョブを通ります。夜間用の道はありません。

D2-05 で allow_failure を外した人の MR は、P0 か P1 が残っているとマージボタンが押せません。
受入条件の1と2は Changes タブの検索で数十秒で確かめられます。朝の確認がこの速さになります。

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

解説

Step6 委譲判定シート 夜間版

Step6 [5min]

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
3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)**

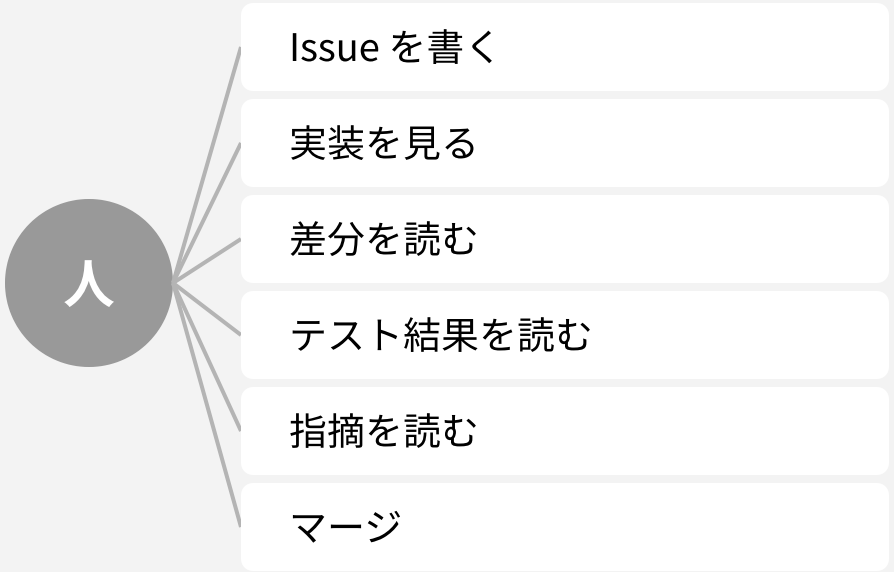
この表は埋めません。143 で書いた「夜間に任せない変更」がどの行に当たるかを読み合わせます。
行4 は Issue の範囲外です。skill は提案で止まり、広げるかどうかは朝に人が決めます。
残りの時間は 153 の台帳記入に回します。

解説

人が残る3点

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(120 と同じ3点)

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

ワーク

今のハーネスで夜間に回せる範囲

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			
5.4 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。157 の残る3点は、残す前提なので表に入れません。

補足

UTM 未解除時の切り替え

CI は講師環境・手元は同じ -p スクリプト

CI 側(講師環境で実走)

講師が Step4 を講師アカウントで
実走し、画面を共有します。

受講者は 152 のログと 155 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 PowerShell)

自分でブランチを作ってから回します。

```
git switch -c nightly/issue-4
```

Step2 と同じコマンドを --max-turns 15 に
戻します。allowedTools は CI と同じです。

MR は push option で作ります。

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

※ 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 5.4 検査も効き、その差自体が教材になります。

> **要確認**　UTM のカテゴリ遮断の解除可否。当日までに先方の手続きの進み方を確かめます。

ワーク

OK 基準とつまずき

4つの OK と4つの詰まりを照合

OK 基準	確認方法
☐ 1回目の result と 143 の予想を見比べた(講師のジョブログの4値でも可)	153 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、自分の判断でマージを押した	Changes とマージ済み
☐ 158 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 159 へ
2回目も「変更がありません。」で終わる	直した層が main に無い	main と result を読み直す
push が branch already exists で失敗	remote にブランチが残る	Step5-3 をやり直す
lint-php54 が赤で MR が立った	5.4 構文の混入	直さず 153 の行2 に印

OK3 の3点は、'作成中' が util.php だけ、array(1, 2, 3, 4, 9) が無い、run_tests.php が3件増えた、です。
OK が4つ揃った人は、持ち帰り5点を台帳に書きます。OK2 が出ない人はつまずき2 から見ます。

> **要確認**　つまずき3 のエラー文は、講師環境の実走で出た文言に差し替えます。

補足

Codex で回す場合

コミットと push はスクリプト側の仕事

ci/nightly_implement.sh

```
codex exec -s workspace-write -C . \
  "AGENTS.md の制約に従って
   Issue #${ISSUE_ID} を実装する" \
  > "${LOG_DIR}/codex-issue-${ISSUE_ID}.log"
```

claude -p の呼び出しを、この形に
差し替えます。手順と push は変えません。
結果は ci_logs/codex-issue-4.log に出ます。

PowerShell(手元の予行)

```
codex exec -s workspace-write `
  "AGENTS.md の制約に従って
   Issue 4 を実装する"
git add -A
git commit -m "nightly: Issue #4 (codex)"
```

workspace-write は .git を読み取り専用に
します。コミットは人かスクリプトが打ちます。

制約の正は AGENTS.md です。CLAUDE.md と同じ内容を保ちます。片方だけ直すと出来上がりが変わります。
スクリプトのコメントが指す .codex/implement-issue.md は、同じ手順を Codex 向けに書いたものです。
止め金は timeout 30m とブランチ保護の2層です。ターン数の上限は Codex 側にありません。
夜間に流す Issue は1本に絞り、145 の deny は .codex/config.toml にも同じ内容を書きます。

> **要確認**　Codex CLI に --max-turns 相当の打ち止めがあるかは未確認です。timeout 30m を前提に運用します。

出典: https://developers.openai.com/codex/cli/reference

持ち帰り5点と置き場所

D2 と smart3pm の両方に置ける5点

持ち帰り物	D2(PowerShell 5.1・ASCII)	smart3pm(bash 正)
.gitlab-ci.yml の nightly_implement ジョブ	対象リポジトリに追記。ISSUE_ID の既定を自社の Issue 番号へ	同じ。PHP が要らない案件は before_script の php-cli を外す
ci/nightly_implement.sh	ci/ に置く。手元で回すなら ps1 に写し、文字列は ASCII に保つ	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	手順1 の Issue 取得先を自社のプロジェクトパスへ	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
止め金の settings.json deny 4種5本・hooks 4種	既存の settings.json に追記。command は powershell -File のまま	command を bash 版の行に差し替えた 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。
持ち帰り台帳の D2-08 の行に、この5点と 152 の turns と cost_usd を書きます。

まとめ 自社への書き戻し

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。
- bash 版のフックを CI でも効かせます。フック登録だけの JSON を用意し、--settings で渡します。PostToolUse の 5.4 検査が runner 上でも走ります。
- ai_review の後に P0/P1 を直すジョブを足したくなったら、153 の行5 を読み直してから決めます。
- 発展1〜3 は D2-09 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

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14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。

[25min]

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14点の出どころ

プチ演習6本とハンズオン3本の成果物14点

Day1 9/29

D1-05	D1-07	D1-10	D1-11
2	2	2	1
規模別レビュアー2本 モデルと effort	deny 5本 CLAUDE.md の戻し方節	deny 3本 block-destructive	実装とレビューの分離定義

Day2 10/6

D2-02	D2-04	D2-05	D2-07	D2-08
2	1	2	1	1
実行環境の制約節 check-php54	schema と gate CI の2ジョブ分割	REVIEW.md と規約 観点別 agents 3本	stop-gate	夜間ジョブ一式 implement-issue

Day1 7点
Day2 7点

14点

各演習の最後に書いた台帳の行を、この14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	規模別レビュアー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 codex-review か review-other	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「Codex レビュー結果」欄も含めます。

解説

台帳の1行の書き方

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-php54(ps1 / sh) + cases.json | post_edit_checks | hooks/check-php54.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 7点の置き先

CI に触る2行はGitLab 導入後に置く7点

No	持ち帰り物	出どころ	D2 の置き先	smart3pm の置き先
8	CLAUDE.md と AGENTS.md の 実行環境制約節	D2-02	CLAUDE.md の制約節	AGENTS.md の制約節
9	PostToolUse フック check-php54 (ps1 / sh)と cases.json の追加	D2-02	post_edit_checks (PostToolUse)	hooks/check-php54.sh
10	REVIEW.md(自社版初版)と Code Review Rules	D2-05	リポジトリ直下と AGENTS.md	REVIEW.md と AGENTS.md
11	観点別サブエージェント3本と known_issues・patterns	D2-05	.claude/agents/ d2-code-reviewer の隣	.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

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。

解説

14点を当てはめる先の一覧

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	Codex 保有者
文脈	CLAUDE.md	core.md と CLAUDE.md	AGENTS.md
設定	.claude/settings.json	.claude/settings.json	~/.codex/config.toml
フック	**hooks/*.ps1 (ASCII のみ) hooks/tests/cases.json**	hooks/*.sh	hooks.json
エージェントとスキル	.claude/agents/ .claude/skills/	.claude/agents/ .claude/skills/	AGENTS.md に観点を書く
CI	.gitlab-ci.yml(3つとも共通の1本)		

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

14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。
Codex 保有者はプロジェクト配下の .codex/config.toml が trusted の時だけ読まれます。
社内共有するものは .claude/settings.json に置きます。2026-09-09 時点の 2.1.267 で確認したパスです。

出所: code.claude.com/docs (settings-precedence)、learn.chatgpt.com/docs (config-reference)

ワーク

ガイドを書く前に紙に書く3問

[3min]

最初に触る1点と壊れたときの気付き方

1 14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。

ワーク

社内展開ガイド A4 の雛形

[5min]

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-php54.md

```
# check-php54.ps1 の運用
## 何を
Edit/Write の後に PHP 5.4 非対応構文を検査し、見つけたら exit 2 で差し戻すフック
## どこに
D2: hooks/check-php54.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-php54.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-php54.sh に、検証を bash 版に置き換えます。

ワーク

作成者以外が読めるかの5項目

[3min]

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 Code

統合ターミナルで claude を起動します。

```
社内展開ガイド_check-php54.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。
```

Codex

codex を起動して同じ文を送ります。
codex exec -s read-only でも可

ワーク

作成者以外の1名による30秒説明

[4min]

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。

研修後の伸びしろ3点

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
フックの回帰テストの非対称	D2 側はテストが揃い、smart3pm 側は一部のみ	D1-10 の単体テスト手順を smart3pm の検査全本に足す	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

要確認　一斉切替の時期は「年末から年明け」までです。具体的な日付は当日確認します。

2日間で触らなかった3点です。新しく作るものはなく、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。

発展

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-php54
- 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。174 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。

作成者以外が外し方を言える14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14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。

補足

D2-09 の持ち帰り

台帳とガイドと宿題3点を D2-10 へ

持ち帰り台帳 確定版

14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13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。
D2-10 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: D2-10 効果測定と8層チェック表の塗り直し [20min]

D2-10

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。

[20min]

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

解説

レビュー工数を測らない理由

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。

解説

MR データから取る4指標

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

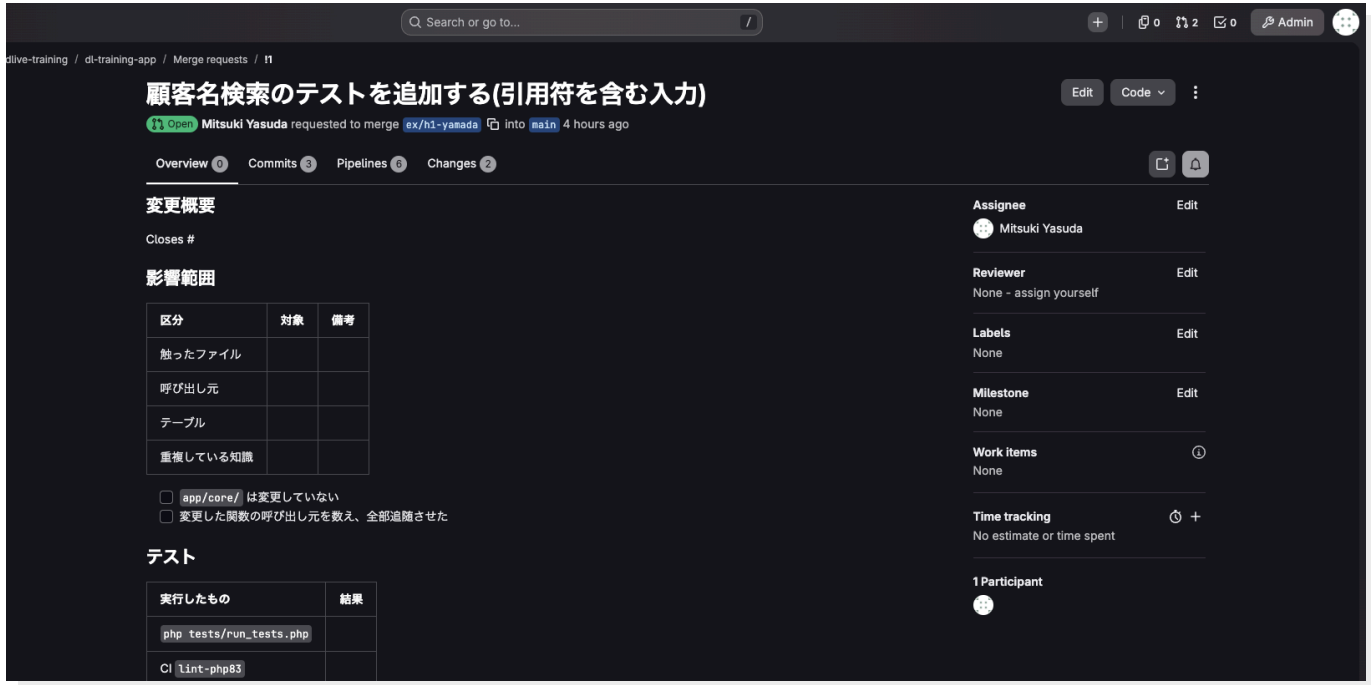
4つとも MR とパイプラインに残る記録だけで数えられ、人の申告を使いません。
指標2 だけは人が1行書く手間が要ります。その1行が「AI は列挙し人が判断する」切り分けの記録になります。

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

実演

MR 画面で拾う数字の位置

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。

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

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。

解説

ビフォー値8週分との突き合わせ

比較できるのは件数と時間の2指標だけ

ビフォー 8週

先方が研修前に計測
改修1件ごとのレビュー記録
出どころは Excel と
チャットワーク

単位を揃える

改修1件 = MR 1本
指摘1件 = コメント1本

アフター 8週

自社 GitLab の MR から
4指標を毎週拾う

指標1 比較する	指標2 初期値として置く	指標3 初期値として置く	**指標4 比較する**

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

的中率 = 検出した仕込み欠陥 ÷ 指摘総数

検出率 = 検出した仕込み欠陥 ÷ 仕込み欠陥の総数

仕込み欠陥総数	検出数	指摘総数	**的中率**	**検出率**

> **要確認 / 要記載**　D2-05 で各自が記録した値を当日記入

今日のハンズオン2 で記録した的中率と検出率が、アフター値の最初の2点になります。
patterns.md のタグ別集計は、指摘の中身を見るときの補助として残します。主指標には入れません。

ワーク

数字を拾う人と画面

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。

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

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。

補足

findings.json の件数集計

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。
Codex の codex exec --output-schema も同じ review.schema.json に揃えているため、同じスクリプトで数えられます。

出典: https://code.claude.com/docs/en/headless, https://developers.openai.com/codex/cli/reference

ワーク

8層チェック表の塗り直し

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	☐	☐	プチ演習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。

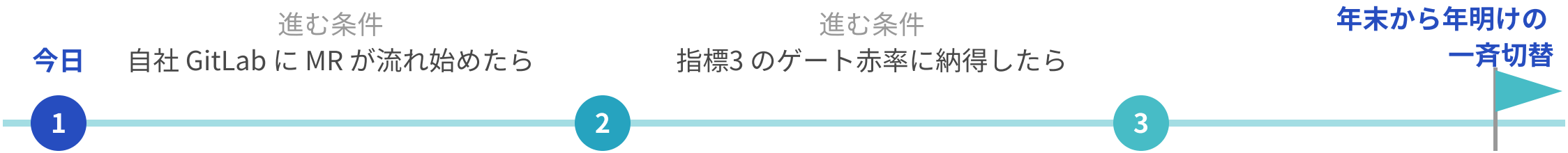
第4層の sandbox は Windows 非対応のため、持ち帰るのは permissions です。担当を書いた層は 189 に移します。

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

解説

年末切替までの3段

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はありません。

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

ワーク

自社版の順番1枚

進む条件を4指標の数字で1つ書く

1 Step 1 [1min]

188 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、181 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 D2-09 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。

補足

研修後の確認事項

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14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)

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(D2-09 の手順) 台帳14点とガイド A4 を自社へ持ち帰る
書き戻し	14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