Givery, Inc.

ハンズオンガイド Day2

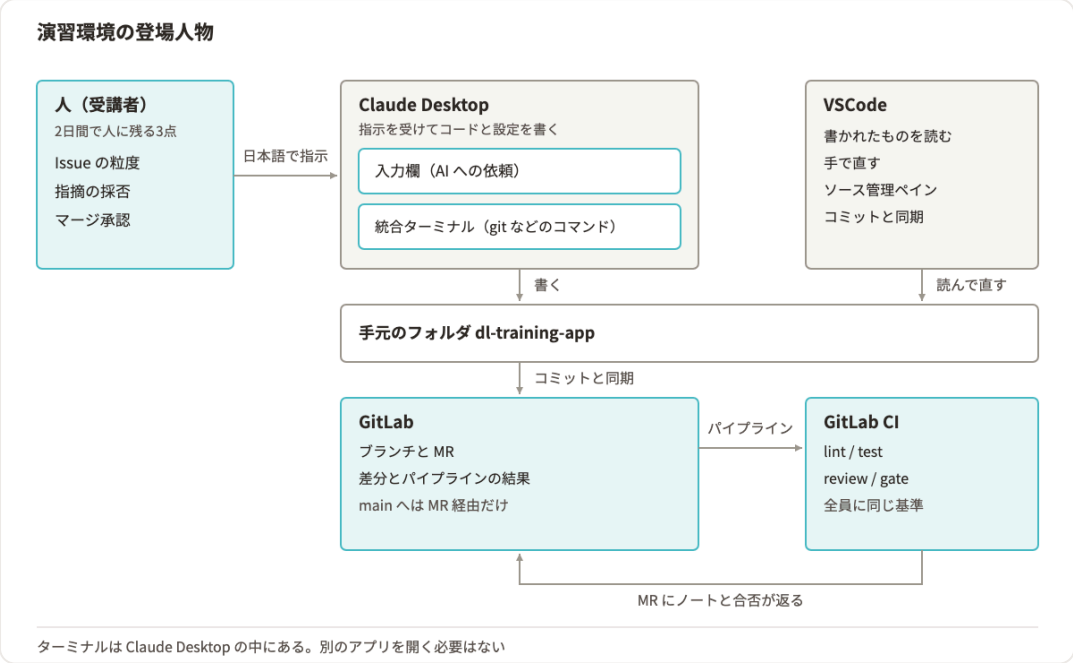
AIカスタム研修 ／ 2026年10月6日（火）

2日間の演習はプチ演習8本とハンズオン3本です。課題の正式な文面と提出先は GitLab の Issue にあり、このガイドはその進め方を1操作ずつ説明します。当日はこのガイドと Claude Desktop、VSCode、GitLab を並べて進めます。背景や補足は折りたたんであるので、必要なときだけ開いてください。

目次

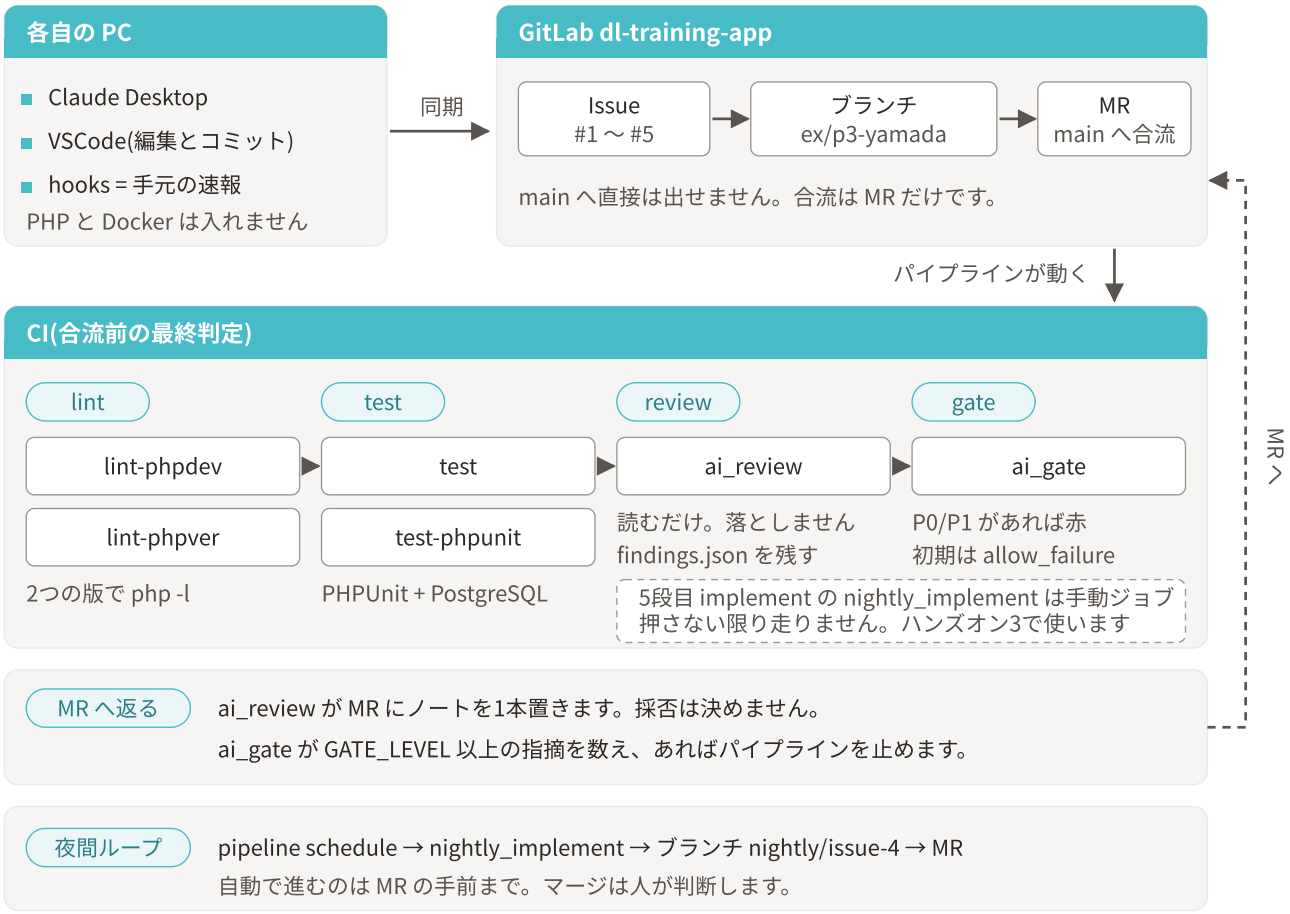
演習環境の全体像

2日間の演習は、Claude Desktop、VSCode、GitLab の演習リポジトリ、GitLab CI の4か所を行き来します。指示を出すのが Claude Desktop、書かれたものを読んで直すのが VSCode、変更を出して機械に判定させるのが GitLab と CI です。
資料に出てくる D2 は貴社の PowerShell 版ハーネス、smart3pm は貴社の bash 版ハーネスの呼び名です。



2日間で行き来する場所の全体像です。Claude Desktop の箱が入れ子になっているのは、統合ターミナルがその中にあるためで、コマンドを打つときに別のアプリを開く必要はありません。矢印が CI から MR へ戻るところまでで1周し、その1周が演習1本にあたります。人の箱に書いた3つだけは、最後まで機械へ渡しません。

登場人物	やること	出てくる場面
受講者	Issue の粒度を決め、指摘を採るか採らないかを決め、マージを承認します。2日間で人に残すのはこの3つです	全演習
Claude Desktop	プロジェクトを開いて指示を受け、コードと設定ファイルを書きます。統合ターミナルと worktree の操作も中にあります	全演習
VSCode	書かれたものを読み、手で直し、ソース管理ペインでコミットと同期を行います	全演習
GitLab	ブランチ、MR、差分、パイプラインの結果を Web 画面で見せます	ハンズオン1 から3
GitLab CI	lint / test / review / gate を全員の変更に同じ基準で当てます	ハンズオン1 から3

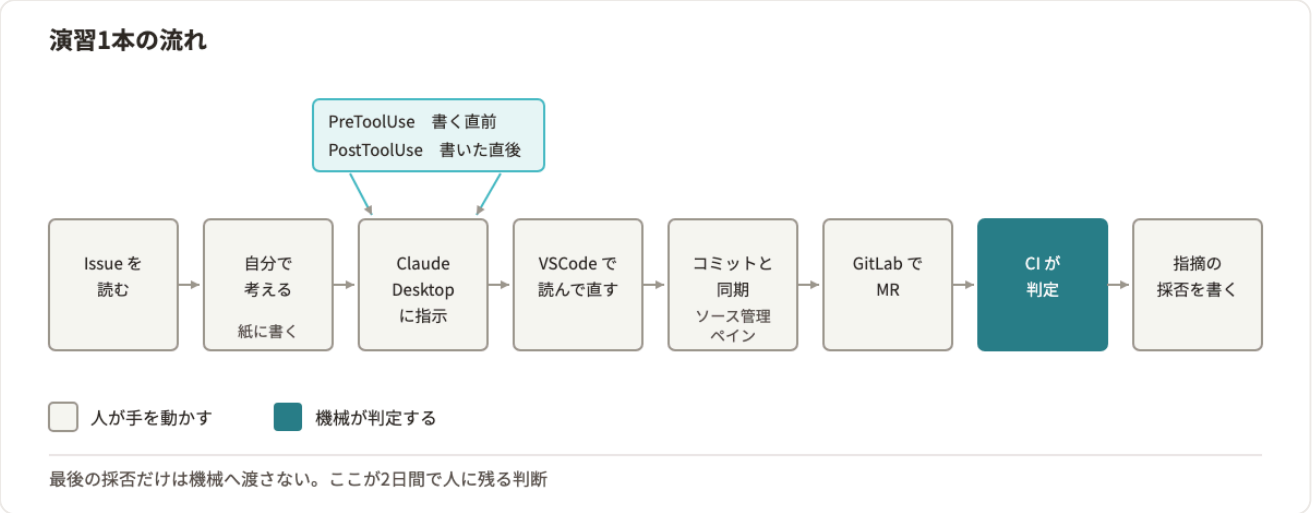


上段が手元と GitLab、中段が MR を出したときに動く CI のジョブです。中段右下の破線が5段目の implement で、手動でしか動きません。右端の点線は、CI の結果が MR へ戻ってくる経路を示します。最下段の夜間ループはハンズオン3で扱います。

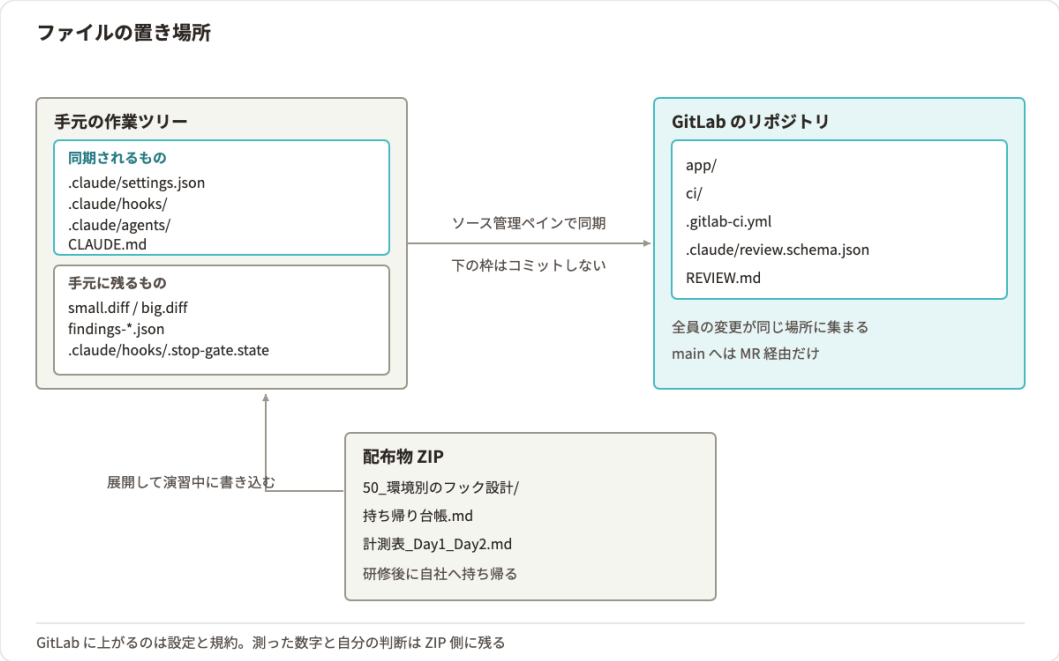
図に出てくる語のうち、3つだけ先に押さえてください。MR(Merge Request)は、自分のブランチを `main` に合流させる申請です。`ai_review` は MR の差分を AI に読ませ、指摘をコメントとして置くジョブです。`ai_gate` は、その指摘に重大なものがあればパイプラインを落とすジョブです。残りの語は末尾の用語集にあります。

この図で押さえる4点

- □ 手元で書いた変更は、ブランチと MR を経由してしか main に入らない
- □ 自動で走る CI は lint / test / review / gate の4段6本。5段目の implement は手動で、ハンズオン3まで押さない
- □ 指摘を出す ai_review と、落とす ai_gate は別のジョブになっている
- □ 夜間ループが自動で進めるのは MR を作るところまでで、マージは人が押す



プチ演習もハンズオンも、この8コマを1周します。塗りが濃いコマだけが機械の判定で、残りは人が手を動かす場所です。上から刺さる2本の矢印がフックの割り込む位置で、Claude Desktop が書く直前と直後にあたります。右端の採否は、どの演習でも機械へ渡しません。



演習中に触るファイルが、どの箱に属するかの一覧です。左の箱は上下2段に分かれていて、上段だけが同期され、下段の差分ファイルや `findings-*.json` は手元に残ります。下の配布物 ZIP は研修後に自社へ持ち帰るもので、リポジトリには入りません。測った数字と自分の判断が ZIP 側にあるのは、研修環境から切り離しても読める形にしておくためです。

手元のフックと CI に同じ検査を置く理由

同じコミットに2本のパイプラインが並ぶ仕組み

> **Tips：**自社のプロジェクトは、AI ツールが動く場所もコードの置き場所もランタイムの版も研修環境とは違います。配布物の `50_環境別のフック設計/環境別のフック設計.md` に、環境が違うときのフックの組み方をまとめています。プチ演習3 とプチ演習6 で該当箇所に触れます。

各演習の読み方

プチ演習8本とハンズオン3本は、同じ6つの見出しで書いてあります。先に「目的」と「達成状態」を読んで、何ができれば終わりかを決めてから手順に入ってください。

見出し	書いてあること
目的	この演習で何ができるようになるか。作業の説明ではなく、身に付くことが書いてあります
準備	始める前にそろっている状態。開いておく画面、いるファイル、前の演習の成果物です
手順	番号付きの操作。1項目1操作で、画面の名前とボタンの名前が入ります
達成状態	できたと自分で判定できる形。講師に聞かずに終わりを決められます
解説と補足	なぜこうするか、背景、別のやり方。折りたたみの中にあります
影響範囲	この変更が何に効くか。壊れるとしたら何が壊れるか。自社に写すときの置き場所です

手順の文には、どこで操作するかを書いています。書き方と画面の対応は次のとおりです。

手順の書き方	操作する場所
「Claude Desktop に」「次を送る」	Claude Desktop の入力欄。文章をそのまま貼り付けて送ります
「`/rewind`」のようにスラッシュで始まるもの	Claude Desktop の入力欄
「ターミナルで」	Claude Desktop の統合ターミナル。右上の `>_` アイコン、または `Ctrl` +バッククォートで開きます。WSL セッションで作業している方は統合ターミナルが使えないので、Windows Terminal で WSL を開き、リポジトリのルートへ移動してから同じコマンドを打ちます。その場合、フックのテストランナーは bash 版（`run_cases.sh`）です
「VSCode で開く」	左のエクスプローラーでファイルをクリックします。見えないときは **表示** > **エクスプローラー** です

手順の書き方	操作する場所
「ソース管理ペインで」	VSCode の左サイドバーにある枝分かれのアイコン。コミットと同期はここです。SourceTree などの Git クライアントやコマンドで進める方は、下の対応表で読み替えてください
「ブラウザで」	GitLab の画面。事前セットアップでログインした `https://gitlab-09291006aidev.give-app.net` です

Git の操作は VSCode のソース管理ペインで書いています。普段 SourceTree などの Git クライアントを使っている方、コマンドで進めたい方は、次の対応で読み替えてください。手順に出てくる操作はこの6種類だけです。

手順の操作	VSCode のソース管理ペイン	コマンド	SourceTree
変更されたファイルを見る	変更の一覧	`git status`	左の「作業コピー」の一覧
ファイルをステージする	ファイル名の横の ＋	`git add <ファイル>`	ファイルにチェックを付ける
行を選んでステージする	差分で行を選び「選択範囲をステージ」	`git add -p <ファイル>`	差分で行を選び「選択した行をステージ」
コミットする	メッセージを書いて「コミット」	`git commit -m "メッセージ"`	メッセージを書いて「コミット」
push する	「変更の同期」または「ブランチの発行」	`git push -u origin <ブランチ>`	「プッシュ」
main を最新にする	「…」メニューの「プル」	`git pull`	「プル」
ブランチを作る	左下のブランチ名から「新しいブランチの作成」	`git switch -c <ブランチ>`	「ブランチ」

初回の確認について　AI がファイルを書き換えたりコマンドを実行したりする直前に、実行してよいかの選択肢が出ます。演習では、その1回だけを許可する側の選択肢を選んでください。同じコマンドを以後確認なしで通す選択肢や、確認を出さないモードへ切り替える選択肢を選ぶと、プチ演習 2 と 3 で「止まる」ことを見る演習の結果が変わります。パーミッションモードの選び方は、このページの共通の進め方 STEP 1「Claude Desktop の基本動作」にあります。

確認が出ないモード

Mac をお使いの方の読み替え

共通の進め方

11本の演習は、この5つの操作の上に乗ります。プチ演習1から6で使うのは STEP 1 から STEP 3 までで、手元の設定を足してコミットするところまでです。プチ演習7と8はコミットもしないので、この5つの操作は出てきません。MR とパイプラインが出てくるのはハンズオンです。各カードの [Nmin] は当日の目安です。

> **先に決めておくこと**　このガイドの例では GitLab のユーザー名を `yamada` としています。ご自身の演習では、社用メールアドレスの「@ より前」をそのまま使ってください。ブランチ名に使うのは同じ文字列です。

STEP 1　Claude Desktop の基本動作

[4min]
研修では Claude Desktop を使います。演習の手順で「Claude Desktop に送る」と書いてあるところは、すべてこの4つの動作の組み合わせです。各演習ではこの節を前提にして、送る文章と見るところだけを書きます。

1. プロジェクトフォルダを選びます。新しいセッションを始めると、入力欄の上の行で環境、プロジェクトフォルダ、ブランチを選べます。プロジェクトフォルダに演習リポジトリ `dl-training-app` のフォルダを指定します。
2. パーミッションモードを **手動** にします。入力欄の下の行の左にあるモード名を押して選びます。新しいセッションを開いたときは、送る前にここを一度見てください。
3. 指示を出します。入力欄に日本語の文章を貼り付けて送ります。ファイル名は文章の中にそのまま書いて構いません。
4. 結果を見ます。書き換えたファイルの名前と差分が返信の中に並びます。実行前に確認が出たら、その1回を許可する側の選択肢を選びます。書かれた中身は VSCode で開いて自分の目で確かめてください。

パーミッションモードは画面の表示で **自動** / **手動** / **編集を受け入れる** / **プラン** の4つです。**自動** では確認が出ないため、プチ演習2 と3 の「止まる」ことを見る演習が成立しません。2日間は **手動** のまま進めてください。プチ演習2 の手順4 だけ、確認の出方を見るために一時的に **編集を受け入れる** に切り替えます。

設定を書き換えたら開き直してください。 `.claude/settings.json` とフックはセッションの開始時に読まれます。書き換えたあとは、セッションを開き直して初めて新しい設定で動きます。プチ演習1 から3 は、この開き直しを毎回はさみます。同じ設定ファイルは Claude Desktop と CLI の両方が読むので、どちらで書いても効きます。

新しいセッションを始める前の入力欄です。上の行が環境、プロジェクトフォルダ、ブランチとワークツリー、下の行の左がパーミッションモード、右がモデルと effort の表示です

この Step が終わった状態

- [ ] プロジェクトフォルダが `dl-training-app` になっている
- [ ] パーミッションモードが **手動** になっている

補足 CLI の位置づけ

研修では Claude Desktop を使います。同じことは CLI でもできるので、すでにターミナルで使っている方はそちらで進めていただいて構いません。設定ファイルは両方が同じものを読むため、`.claude/settings.json` の権限もフックも `CLAUDE.md` も、どちらで書いても効きます。演習の達成状態はどちらでも同じです。

腰を据えて使い込む段階に入ると、CLI のほうが自動化に載せやすくなります。差は1つで、対話画面を開かずに1回だけ走らせる形（`claude -p`）が CLI にしかありません。この形が取れると、出力をそのままファイルに残して CI のジョブや夜間のスケジュール実行に組み込めます。ハンズオン3 の夜間ループが、その実物です。

STEP 2 ブランチ名の形

[3min]
演習で作るブランチは `ex/<演習番号>-<ユーザー名>` です。演習番号は、プチ演習が `p1` から `p4`、ハンズオンが `h2` と `h3` です。プチ演習5 は `ex/p4-` のブランチのまま、プチ演習6 はハンズオン2 の `ex/h2-` のブランチのまま進めます。プチ演習3なら `ex/p3-yamada`、ハンズオン2 なら `ex/h2-yamada` になります。切り替えの操作は STEP 3 のソース管理ペインで行います。

ハンズオン1だけは例外です。Claude Desktop の入力欄の上の行で、ブランチ名の右にある **ワークツリー** にチェックを入れて開始すると、そのセッションだけが隔離した作業ツリーで動きます。名前の入力欄は無く、ブランチ名は `worktree-` で始まる名前が自動で付きます。入力欄の上に出たブランチ名を控え、以降この資料の `worktree-issue-1` は控えた名前に読み替えてください。作業ツリーは `.claude/worktrees/` の下に1つできます。

配布した `.claude/settings.json` には `worktree.baseRef` = `head` が入っているので、ワークツリーは開始したときに乗っていたブランチの HEAD から切られます。ハンズオン1 では `ex/p3-` の最後のコミットが分岐元です。コミットしていない変更は入りません。

演習	ブランチ	切る元	そこで足すもの
プチ演習1	`ex/p1-<ユーザー名>`	`main`	`model` と effort の設定、レビュアー定義2本
プチ演習2	`ex/p2-<ユーザー名>`	`ex/p1-`	`deny` 10本と `ask` 3本
プチ演習3	`ex/p3-<ユーザー名>`	`ex/p2-`	読み取りの `deny` 3本、PreToolUse フック
ハンズオン1	`worktree-` で始まる自動の名前	`ex/p3-` の HEAD	Issue #1 の改修
プチ演習4・5	`ex/p4-<ユーザー名>`	`ex/p3-`	PostToolUse フック、JSON ゲート
ハンズオン2	`ex/h2-<ユーザー名>`	`ex/p4-`	`REVIEW.md` 、観点別レビュアー、CI ゲート
プチ演習6	`ex/h2-<ユーザー名>` のまま	切らない	Stop フック
ハンズオン3	`ex/h3-<ユーザー名>`	`ex/h2-`	SessionStart フック、夜間ジョブ

main に戻らない理由

> **Tips：**ブランチ名の頭文字には意味を持たせています。 `ex/` は人が手で始めた演習、 `worktree-` は Claude Desktop のワークツリー、 `ai/` は `/implement-issue` が作ったもの、 `nightly/` は CI の夜間ジョブが作ったものです。MR の一覧を見たときに、誰が始めた変更かが名前だけで分かります。

STEP 3 VSCode のソース管理ペイン

[6min]
変更の確認、コミット、同期、ブランチの切り替えは、すべて VSCode の画面でできます。左サイドバーの枝分かれのアイコンがソース管理ペインです。アイコンの右上の数字が、いま変わっているファイルの数です。

1. ブランチを切り替えます。ウィンドウ左下にいまのブランチ名が出ています。そこをクリックして、一覧から選ぶか **新しいブランチの作成** で新規に作ります。
2. 差分を見ます。 **変更** の一覧でファイル名をクリックすると、左に変更前、右に変更後が並びます。AI が書いたものは必ずここで読んでください。
3. ステージに上げます。ファイル名の右の `+` を押します。変更したファイルだけを選んで押してください。フックが作った作業ファイルまで巻き込まないためです。
4. メッセージを書いてコミットします。入力欄に日本語の1行を書き、 **コミット** を押します。
5. 同期します。 **変更の同期** を押すと、手元のコミットが GitLab へ送られ、GitLab 側の更新が手元に入ります。

コミットは1つの変更意図につき1つです。10個の変更を1コミットにまとめると、そのうち1つを取り消したいときに手作業で切り分けることになります。メッセージは「見積一覧の顧客名検索をプレースホルダに置き換える」のように、何をどうしたかを書きます。ファイル名を並べたメッセージは、あとで履歴を読むときに何も足しません。

> **指示を出す前にコミットしてください。** `/rewind` で戻せるのは AI がファイルの編集として行った変更だけです。コマンドで動かした分、サブエージェントの変更、手で直した分は戻りません。全部まとめて効く保険は、作業の前後のコミットだけです。

切り替えで止まったときの見方

この Step が終わった状態

- [ ] 左下のブランチ表示が `ex/` で始まる名前になっている(ハンズオン1 だけは控えた `worktree-` で始まる名前)
- [ ] ソース管理ペインの **変更** に、前の演習の変更が残っていない

STEP 4 GitLab の Web 画面

[8min]
MR の作成、差分の確認、パイプラインの確認、マージの判断はブラウザで行います。`main` はブランチ保護がかかっており、合流の経路は MR だけです。

1. MR を作ります。同期したあとにプロジェクトの画面を開くと、上部に **Create merge request** の帯が出ます。出ていないときは左メニューの **Code** > **Merge requests** から **New merge request** を押します。

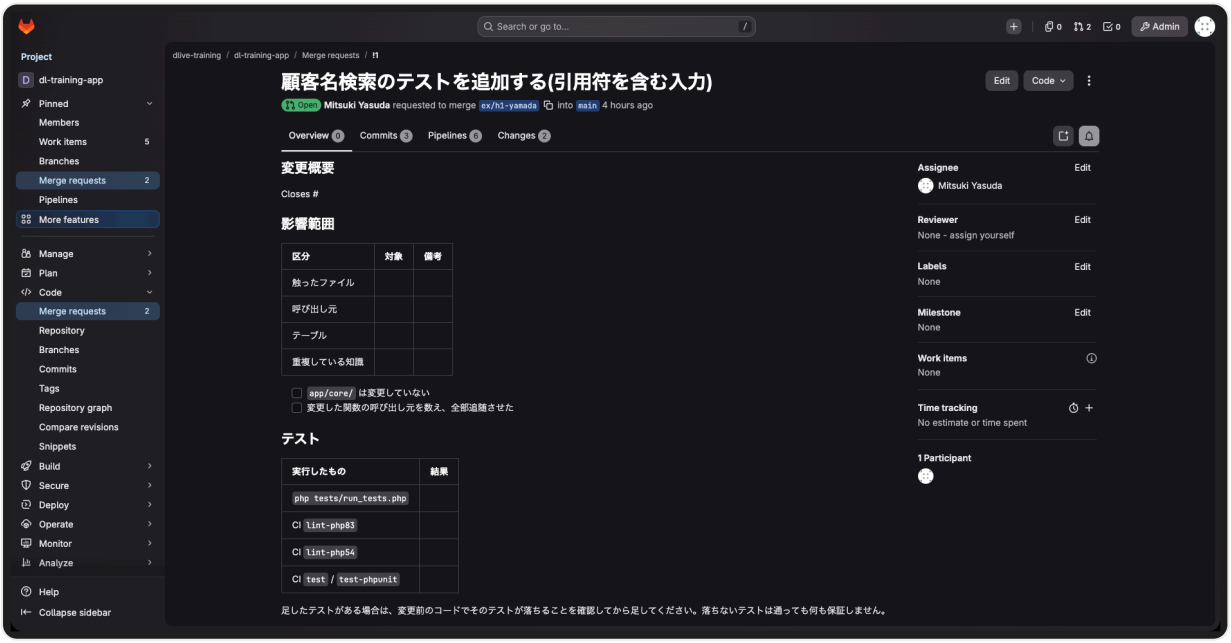
2. 合流先が `main` 、元が自分のブランチになっていることを確かめます。

3. 説明欄には、リポジトリの既定のテンプレート（`.gitlab/merge_request_templates/Default.md`）が最初から入っています。**変更概要** と **影響範囲** を先に埋め、**AI レビュー結果** の表は演習の途中で埋めます。

4. 差分は **Changes** タブで読みます。行の左に出るアイコンから、その行に直接コメントを付けられます。

5. パイプラインは **Pipelines** タブで見ます。落ちたジョブの名前をクリックするとログが開きます。

6. **Merge** は押しません。演習中は誰の MR もマージせず、開いたまま残してください。MR の一覧が2日間の記録になります。



> 説明欄の見出しがテンプレートどおりに入っていることと、右側の **Pipeline** が動き始めていることを見ます。

Pipelines タブには、パイプラインが2本ずつ並びます。1本目は変更を出すたびに走るブランチ用で `lint` と `test` の2段4ジョブ、2本目は MR に対して走る MR 用で `review` と `gate` の2段2ジョブです。どちらも前の段が落ちると後ろの段は動きません。ブランチ用のいちばん右の `implement` にある `nightly_implement` は再生ボタンが付いた手動ジョブで、ハンズオン3まで押しません。

> **merge request** のラベルが付いた行が review と gate、**branch** の行が lint と test です。Stages の丸にマウスを乗せるとジョブ名が出ます。赤い行は、テストだけを先に出した回のブランチ用パイプラインです。

落ちたジョブのログの読み方

マージを自分で押さない理由

> **Tips：** `Closes #1` を説明欄に残すと、マージしたときに Issue が閉じます。演習中は閉じないので、書き換えずにそのまま残してください。

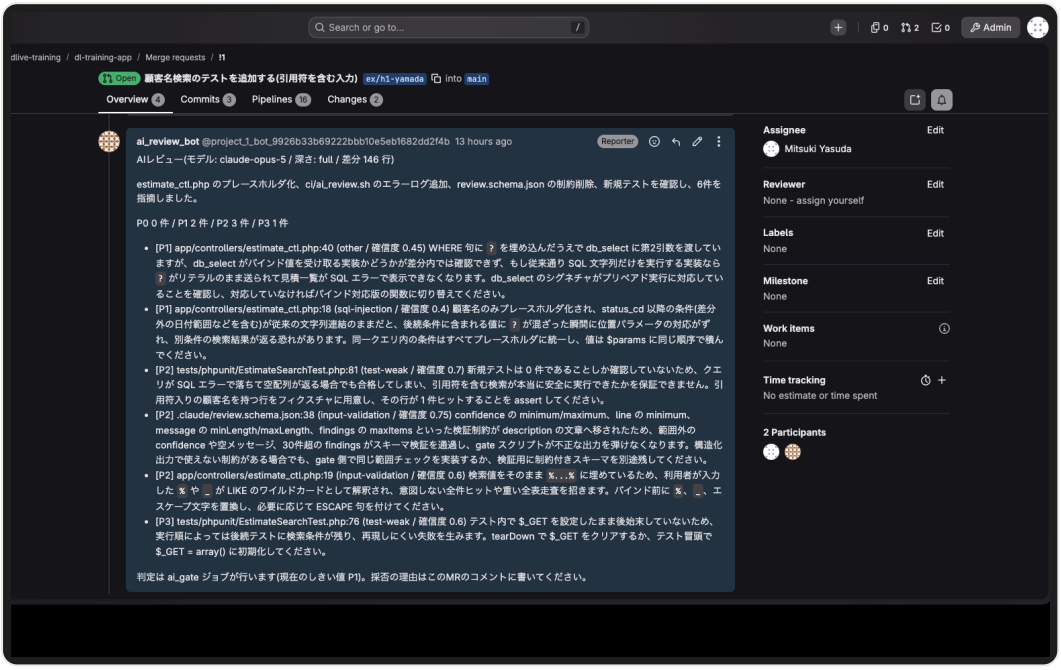
> **Tips：** `ai_gate` は初期状態で `allow_failure: true` です。落ちてもパイプライン全体は赤になりません。ハンズオン2でこの1行を外し、本当に合流を止めるゲートにします。 `test-phpunit` だけ他より2分ほど長くかかりますが、止まっているように見えても待ってください。

STEP 5　AI レビューのノートの読み方

[4min]

`ai_review` は MR にコメントを1本置きます。1件の指摘は `severity`、`confidence`、`file`、`line`、`message`、`category` の6つを持ちます。ノートは列挙するだけで、直すかどうかは決めません。

読む順番を決めておくと速くなります。`severity` で P0 と P1 だけを先に読み、次に `confidence` が 0.5 以下のものを「実物を確かめてから判断する」側に寄せ、最後に今回の変更の外を指している指摘を外します。この3手で、6件のノートが2件の判断に減ります。



> MR の **Overview** の下部に、`ai_review_bot` のコメントとして届きます。1行目にモデル名、深さ、差分の行数、その下に P0 から P3 の件数、続いて指摘の箇条書きです。

ノートの本文の例

P0 から P3 の線引き

> **件数を減らすことは目的ではありません。** このノートで見るのは、履歴を切ると指摘が何件どう変わるかです。ハンズオン1では同じ変更を3通りで読ませ、件数と所要時間を MR の表に並べます。CI の `ai_review` を加えると4行目になります。

補足 統合ターミナルとコマンド

画面から実行できないものは、Claude Desktop の統合ターミナルから打ちます。右上の `>_` アイコン、または `Ctrl` +バッククォートで開きます。セッションの作業ディレクトリで開き、Claude と同じ環境を共有するので、パスを移動する操作はいりません。2本目のタブが要るときは、ターミナルペーンの `+` で増やします。演習の手順で「ターミナルで」と書いてあるのは、

すべてこのターミナルです。コマンドを打つ場面は2日間で十数か所あり、どれも統合ターミナルにそのまま貼れる形で書いてあります。

やること	コマンド	使う演習
AI に読ませる差分をファイルに出す	`git diff main --output=review-input.diff`	プチ演習1、ハンズオン1
非対話で1回だけ走らせる	`claude -p "指示"`	ハンズオン3
消えたコミットを探す	`git reflog`	プチ演習2

> **Tips：**差分をファイルに出すのは、AI に「いまの変更だけ」を読ませるためです。リポジトリ全体を渡すと、今回触っていない場所の指摘が混ざります。 `>` で書き出さずに `--output=` を使うのは、PowerShell 5.1 の `>` が UTF-16 のファイルを作り、日本語が読めなくなるためです。

> **Tips：**統合ターミナルが使えるのは手元で動かしているセッションだけです。隔離した作業ツリーを作る操作はコマンドではなく、STEP 2 に書いた **ワークツリー** のチェックで行います。

1つのプロジェクトを全員で使う形

演習リポジトリ `dl-training-app` は1つで、受講者全員がここを使います。同じ場所を触るのに作業がぶつからないのは、人ごとにブランチが分かれていて、`main` が保護されているからです。演習に入る前に、この節でその形を押さえてください。

> 左が各自の手元、右が GitLab 側です。破線の `main` だけが全員の共有物で、演習中はここに何も入りません。右の実線の行が人ごとに分かれたブランチと、開いたままの MR です。下段は CI の待ちが出る理由で、これだけは全員で1台を分け合います。

作業がぶつからない理由

変更を書き込む先は、いつも `ex/<演習番号>-<ユーザー名>` という自分のブランチです。ユーザー名は社用メールアドレスの「@ より前」なので、全員分が重なりません。`main` にはブランチ保護がかかっており、受講者の権限では直接 push できません。合流の経路は MR だけです。ブランチを切り忘れて `main` のまま同期しようとした場合は、GitLab 側で弾かれます。その文面と直し方は「困ったとき」にあります。

他の方の手元への影響

演習中は誰の MR もマージしません。`main` が動かないので、他の方の変更がご自身の手元へ入ってくることはありません。同期を押しても、取り込まれるのは自分のブランチの分だけです。逆に、ご自身が書いたものが他の方の画面に現れることもありません。壊れる心配をせずに、思い切った書き換えを AI に頼んで構いません。

他の方のブランチと MR の見え方

GitLab の一覧には、受講者全員のブランチ、MR、パイプラインが並びます。見えてよいものです。むしろ MR が開いたまま残ることで、2日間に誰が何を試して、どの指摘をどう判断したかが一覧として残ります。守っていただく約束は1つで、他の方のブランチには切り替えない、他の方の MR はマージしない、それだけです。自分の行を探すときは、ブランチ名に入っているご自身のユーザー名で絞ってください。

パイプラインの順番待ち

CI のジョブを実行する runner は1台です。全員が同じ時間帯に同期すると、パイプラインは1台に並ぶので `pending` のまま待つ時間が出ます。異常ではありません。待っている間は、MR の説明欄の記入や記録の下書きなど、CI を待たない作業を先に進めてください。ハンズオン3 の手動ジョブは1本が最長30分 runner を使います。Step 3 は講師が `main` で1本だけ流して全員で同じログを読み、Step 5 で各自が自分のブランチの手動ジョブを押します。

マージを演習中に行わない理由

ブランチ名の接頭辞で見分けられること

記録台帳と計測表

演習の成果物は、ファイルそのものより「どこに置くと決めたか」の記録のほうが後で効きます。台帳と計測表の2つを、演習のたびに1行ずつ書き足してください。書き足すのは演習の最後ではなく、生成物ができた直後です。

台帳 持ち帰り台帳.md

2日間で足す機能を1行ずつ記録します。演習中に決めるのは置き場所と担当で、ここが空欄のまま研修が終わると、持ち帰ったファイルは端末の中に残ったままになります。最後の列は「未 / 書いた / 置き場所まで決めた」の3つから選びます。

台帳の書き方の例

置き場所の列は先に書いてください。 D2 は PowerShell 版、smart3pm は bash 版を正にします。同じガードを2言語で書き続けると、片方だけ直す事故が起きます。どちらを正にするかは Day1 の最後に各自が決めます。

> **Tips：** D2 と smart3pm の具体的なディレクトリ名は、この資料には書いていません。ご自身の端末で開いたハーネスの構成に合わせて記入してください。

計測 計測表

指摘の総数、うち P0 と P1、所要時間の3つを演習ごとに取り、採否を決めたら採用・見送りの件数を足します。Day2 の最後に、これが効果測定の初回の実測値になります。数字を取らないと、品質が上がったかどうかを言葉でしか言えません。

列	取り方	取るタイミング
指摘の総数	ノートの合計件数、または `findings.json` の `findings` の長さ	レビューを1回回すたび
うち P0 と P1	P0 と P1 の合計。ノートの見出し行にそのまま出ます	同上

列	取り方	取るタイミング
所要時間	レビューを頼んでから結果が返るまでの分数。CI はジョブ画面の `Duration`	同上
採用・見送り	「即修正」に仕分けた件数と、「将来課題」「意図した設計」に仕分けた件数を「1・3」の形で並べる	採否を決めた直後

計測表の書き方の例

> **Tips：**所要時間は秒単位で取らなくて構いません。「2分」「10分超」の粒度で足ります。比べるのは、読ませ方を変えたときの差です。

この章の持ち帰り物

- [ ] 持ち帰り台帳.md に、演習ごとの置き場所と担当が入っている
- [ ] 計測表に、ハンズオン1の3通りと CI の ai_review の4行がそろっている

ファイル	D2 での置き場所	smart3pm での置き場所
`持ち帰り台帳.md`	ハーネスの文書置き場(当日確定)	ハーネスの文書置き場(当日確定)
`計測表_Day1_Day2.md`	同上。効果測定の初回値として残します	同上

置き場所の具体名は、ご自身の端末で開いたハーネスの構成に合わせて決めてください。この2つは公開できる内容なので、社内で共有する側に置いても差し支えありません。

用語集

2日間の資料と演習で、説明なしに出てくる語をまとめています。手が止まったらここに戻ってください。

語	意味	出てくる場所
Claude Desktop	研修で使うデスクトップアプリです。セッションの開始時にプロジェクトフォルダを選ぶと、そのフォルダのファイルを読み書きします。統合ターミナルと worktree の操作も中にあります。設定ファイルは CLI と共通です。	全演習
ソース管理ペイン	VSCode 左サイドバーの枝分かれアイコンで開く画面です。変更の一覧、差分、ステージ、コミット、同期、ブランチの切り替えがここにまとまっています。演習の git 操作はこの画面で行います。	全演習
`CLAUDE.md`	リポジトリ直下に置く指示書です。セッションの開始時に読み込まれ、AI はこの文章を参考にして動きます。文章なので「守るとは限らない」のが前提で、演習では「文脈であって設定ではない」と呼びます。	プチ演習2、プチ演習4
`settings.json`	`.claude/settings.json` 。モデル、権限(`permissions`)、フック(`hooks`)をここに書きます。JSON が1文字でも壊れていると、ファイル全体が読まれずに既定値で動きます。セッションの開始時に読まれるため、書き換えたら `Ctrl+N` で新しいセッションを開きます。登録の確認は、このファイルを VSCode で開いて見ます。Claude Desktop の入力欄では `/permissions` と `/hooks` は使えません。	プチ演習1 から3
フック(hook)	決まった場面(ツールの実行前、実行後、応答の終了時など)で自動的に呼び出されるスクリプトです。 `settings.json` の `hooks` に登録します。スクリプトは標準入力で JSON を受け取り、終了コードで結果を返します。演習のフックは PowerShell 版(`.ps1`)と bash 版(`.sh`)が `.claude/hooks/` にあります。フックの単体テストは配布時点で22件あり、プチ演習3 とプチ演習4 で1件ずつ足します。	プチ演習3、プチ演習4、プチ演習6
環境別のフック設計	配布物の `50_環境別のフック設計/環境別のフック設計.md` です。AI ツールが動く場所、コードの置き場所、ランタイムの版が研修環境と違うプロジェクトで、フックをどう組むかをまとめています。	プチ演習3、プチ演習6
`exit 2` / `$LASTEXITCODE`	スクリプトが終わるときに返す数字が終了コードです。フックでは `0` が「通す」、 `2` が「止める」の合図です。 `2` の効き方はイベントで違い、PreToolUse はツールの呼び出しを止めて標準エラーの文章を AI に	プチ演習3、プチ演

語	意味	出てくる場所
	返し、Stop は応答を終わらせずに作業を続けさせます。PostToolUse は実行済みなので止められず、標準エラーの文章を AI に渡すだけです。SessionStart はセッションを止めず、標準エラーの文章は利用者の画面に出るだけで AI には届きません。`1` など `2` 以外の数字は止める合図にならず、処理はそのまま進みます。直前に動かしたスクリプトの終了コードは、PowerShell では `$LASTEXITCODE`、bash では `$?` と打つと表示されます。	習4、ハンズオン3
サブエージェント	別の会話として起こす、役割つきの AI です。定義は `.claude/agents/<名前>.md` に置き、使えるツールとモデルを絞れます。入力欄で `@` を打つと一覧に `<名前> (agent)` として出るので、そこから選んで依頼文を続けます。	プチ演習1、ハンズオン1
frontmatter	Markdown ファイルの先頭に `---` で囲って書く設定の部分です。サブエージェントの定義では `name` / `description` / `tools` / `model` / `effort` がここに入ります。閉じの `---` を消すと本文として読まれ、設定が効きません。	プチ演習1
`/rewind`	巻き戻し機能です。入力欄に `/rewind` と打つと、これまでのメッセージの一覧が開きます。地点を選ぶと、復元の範囲を コードと会話 / 会話のみ / コードのみ から選ぶ確認画面に進みます。Esc を2回押す開き方は Claude Desktop では効きません。戻せるのはファイルの編集として行われた変更だけです。	プチ演習2
パイプライン / ジョブ / ランナー	GitLab CI の用語です。変更や MR をきっかけに自動で走る一連の処理がパイプライン、その中の1つ1つ(`lint-phpver`、`ai_review` など)がジョブ、ジョブを実際に実行するサーバがランナーです。結果は MR の **Pipelines** タブに並びます。	ハンズオン1から3
`permission mode`	権限の効き方を決めるモード。Claude Desktop の画面に出るのは 手動 / 編集を受け入れる / プラン / 自動 の4つで、設定値ではそれぞれ default / acceptEdits / plan / auto です。設定値にはほかに dontAsk と bypassPermissions があり、bypassPermissions は Claude Desktop の設定で 権限をバイパス として出せます。dontAsk は画面に出ません。acceptEdits は作業ディレクトリ内の `rm` まで自動承認します。auto と bypassPermissions はプロジェクトの `.claude/settings.json` からは指定できません。	Day1の権限の話、ハンズオン3
`deny`	`permissions.deny` に書く拒否の規則。どのパーミッションモードでも効き、`allow` で例外を作れません。`Bash(...)` の規則は PowerShell ツールの呼び出しに当たらないので、Windows 向けには `PowerShell(...)` と対で書きます。コマンド文字列の照合なので、`/bin/rm` や `bash -c "..."` の形はすり抜けます。塞ぐのは PreToolUse フックの役目です。	プチ演習2、プチ演習3

語	意味	出てくる場所
`PreToolUse`	ツールを実行する直前に走るフック。`deny` と同じくどのパーミッションモードでも走り、`exit 2` で呼び出しを止めます。`matcher` が `Bash` だけだと PowerShell ツールの呼び出しでは走らないので、演習では `Bash|PowerShell` で登録します。演習では `block-destructive` がこれです。	プチ演習3
`PostToolUse`	ツールの実行後に走るフック。ファイルの編集の後に `check-phpver` を走らせ、このプロジェクトの対象バージョン(PHP 7.4)で動かない構文を `exit 2` で差し戻します。	プチ演習4
`Stop hook`	応答を終えようとしたときに走るフック。`{"decision":"block","reason":"..."}` を標準出力すると、作業を続けさせます。演習では `stop-gate` がテストの通過を見ます。	プチ演習6、ハンズオン3
`stop_hook_active`	Stop hook の入力 JSON に入る真偽値。継続で再び呼ばれたときに `true` になります。これを見ずに `block` を返し続けると、Claude Code が8回連続の block で打ち切るまで継続が繰り返され、そのぶんトークンを使います。	プチ演習6
`effort`	考える深さ。low / medium / high / xhigh / max の5段です。`settings.json` の `effortLevel`、skill と agent の frontmatter、環境変数 `CLAUDE_CODE_EFFORT_LEVEL` で決まります。CLAUDE.md には書けません。	プチ演習1
`worktree`	同じリポジトリの別の作業ツリー。Claude Desktop の入力欄の上の行で、ブランチ名の右にある **ワークツリー** にチェックを入れて開始すると作られます。名前の入力欄は無く、ブランチ名は自動で付きます。作業ツリーは `<プロジェクトルート>/.claude/worktrees/` の下にできます。分岐元は `settings.json` の `worktree.baseRef` で決まり、配布した設定の `head` は開始時のブランチの HEAD から切ります。終了したらサイドバーのアーカイブのアイコンで削除します。gitignore されたファイルを持ち込むときは、プロジェクトルートに `.worktreeinclude` を置きます。	ハンズオン1
非対話実行	対話画面を開かずに1回だけ走らせる使い方です。`claude -p "指示"` の形で、Claude Desktop の統合ターミナルか CLI から呼びます。Claude Desktop の画面操作としては用意されていないのがこれです。確認が要る操作はその場で拒否されて先へ進むため、`--permission-mode`、`--allowedTools`、`--max-turns` で何を許すかと回数の上限を決めて初めて無人で動かせます。夜間ループはこの形です。	ハンズオン3
`--bare`	CLAUDE.md もフックも読まずに起動します。制約が効かなくなるので実装側には使いません。履歴と文脈を切って読ませたいレビュー側にだ	ハンズオン1

語	意味	出てくる場所
	け使います。ログイン情報も読まないため、環境変数 `ANTHROPIC_API_KEY` が無い端末では起動しません。	の発展課題
`--json-schema`	出力を JSON Schema の形に固定します。 `.claude/review.schema.json` を渡すと `findings` の形で返り、そのまま判定スクリプトに流せます。	ハンズオン1の補足
`findings`	レビュー結果の配列。1件が `severity` / `confidence` / `file` / `line` / `message` / `category` を持ちます。CI では `findings.json` として artifacts に残ります。	プチ演習5、ハンズオン2
`P0 〜 P3`	指摘の重大度。P0 は動かないか壊すもの、P1 は実環境で落ちるか外部入力が素通しになるもの、P2 は指示との不一致と波及漏れ、P3 はそれ以外です。	全演習
`confidence`	0 から 1 の数値。実物を読んで確認できた度合いです。経路の一部が読めていない指摘は 0.5 以下になります。件数を絞るときの2番目の手がかりです。	プチ演習5、ハンズオン2
`GATE_LEVEL`	`ai_gate` が落とす重大度の下限。既定は `P1` で、P0 と P1 を止める意味です。 `.gitlab-ci.yml` の `variables` か CI/CD Variables で変えられます。	ハンズオン2
`allow_failure`	ジョブが落ちてもパイプライン全体を赤にしない設定。 `ai_gate` は初期状態で `true` です。ハンズオン2でこの1行を外します。	ハンズオン2
`artifacts`	ジョブが残すファイル。ジョブ画面の右側から取得できます。 `findings.json` と PHPUnit のレポートがここに入ります。	ハンズオン2、ハンズオン3
`pipeline schedule`	GitLab が決まった時刻にパイプラインを起動する仕組み。 **Build** > **Pipeline schedules** から作ります。夜間ループの起動側です。	ハンズオン3
プロジェクトアクセストークン	プロジェクト単位で発行するトークン。演習ではロール Developer、スコープ `api` と `write_repository` で発行しています。 `CI_JOB_TOKEN` では MR を作れないため、夜間ジョブはこちらを使います。	ハンズオン3
masked variable	CI/CD Variables のマスク指定。ジョブのログで値が伏せ字になります。API キーとトークンは必ずこの指定を付けて登録します。	Day1の秘密情報の話、ハ

語	意味	出てくる場所
		ンズオン3
MR	Merge Request。ブランチを `main` に合流させる申請です。演習では `main` が保護されており、合流の経路は MR だけです。	全演習
Issue	演習の題材。#1 から #5 まであり、本文は 背景 / 現状 / 期待する振る舞い / 受入条件 / 対象ファイル / 範囲外 の6節で書かれています。`/implement-issue` はこの6節の見出しをそのまま読みます。	ハンズオン1から3
`PHPCompatibility`	phpcs の規格。`--standard=PHPCompatibility --runtime-set testVersion 7.4` で対象バージョンに無い構文を検出します。`php -l` は文法として正しければ通してしまうため、`match` 式や `?->` はこちらでないと捕まりません。	プチ演習4

フック4イベントの位置

表の「フック」と「`exit 2`」の補足です。上段が4イベントの位置で、起動時が SessionStart、ツール呼び出しの直前が PreToolUse、実行の直後が PostToolUse、応答を終える直前が Stop です。下段が終了コードの違いで、`exit 0` は通し、`exit 2` は止めて標準エラーの理由をモデルへ返します。この止め方が効くのは PreToolUse で、PostToolUse は理由を渡すだけ、SessionStart は利用者の画面に出すだけです。Stop は `exit 2` と `decision: block` のどちらでも作業を続けさせられます。

> **Tips：**この表にない語が出てきたら、その場で講師にお伝えください。当日の質問はこの用語集に追記して、研修後に配り直します。

プチ演習4　PHP 7.4 制約の機械検査

D2-02 ／ 所要 [20min]

プロジェクトが対象とするバージョンを設定に書き、それを超える構文が書かれた瞬間に差し戻される状態を作ります。

Day2　**match を書かせようとしても switch に戻る状態**

目的

CLAUDE.md の文章で守っている制約を、フックの機械検査へ移せるようになります。対象のバージョンは設定に書く値として扱い、自社の版に差し替えて持ち帰れる形にします。

準備

開いておく画面	Claude Desktop（プロジェクトは `dl-training-app`）、VSCode、GitLab のプロジェクト画面
いるファイル	`CLAUDE.md` 、 `.claude/settings.json` 、 `.claude/hooks/check-phpver.ps1` 、 `app/libraries/util.php`
直前の演習の成果物	プチ演習3 で登録した `PreToolUse`（終えていない方もこの演習は進められます）
ブランチ	`ex/p3-<ユーザー名>` から `ex/p4-<ユーザー名>` を切ります。Step 1 の手順1 と 2 で作ります
パーミッションモード	**手動**。編集のたびに確認が出る状態で始めます。自動では確認が出ないまま編集が進むので、差し戻しの見え方が変わります。モードはフォルダごとに記憶されるので、`dl-training-app` を開いた状態で切り替えてください

このリポジトリが対象とする版は PHP 7.4 です。検査で止めるのは、7.4 に無い 8.x の構文です。どれも AI が指示なしで素で書く書き方なので、放っておくと混ざります。

止める書き方（8.x）	7.4 での書き方
`match` 式	`switch` か `if` の連鎖
`enum` 宣言	クラス定数を並べる
`readonly` プロパティ	`private` + 取得用のメソッド

名前付き引数	並び順で渡す
コンストラクタプロモーション	プロパティ宣言と代入を書く
`?->`	`isset` で受けてから呼ぶ
`str_contains` / `str_starts_with` / `str_ends_with`	`strpos` / `strncmp` / `substr` で書く

> **Tips：**バージョン番号は設定に書く値でしかなく、検査の骨格は同じです。自社へ持ち帰るときは、いま動いている版ではなく移行先の版を書いてください。移行先を対象にして書かせておくと、移行のときに直す量が減ります。

自分で考える　[3min]　AI に触る前に、手元のメモへ3つ書き出してください。自社の CLAUDE.md を持ち込んだ方は、その制約節を横に開いてください。

1. 自社の基幹システムで、これを書かれたら本番で落ちる、という構文を3つ挙げてください。バージョン番号ではなく構文の名前で書きます。

2. その3つを、禁止の列挙ではなく代替の書き方で1行ずつ書き直してください。「`match` は使わない」ではなく「`switch` で書く」の形です。

3. 3つのうち、機械で検査できるものに丸を付けてください。丸が付かなかったものは人が見る側に残ります。その理由も1行書きます。

手順

> 上の帯が D2-01 で挙げた3段、下の箱がその3段のどこに当たるかです。左端の `CLAUDE.md` は文脈なので、そこに書いただけでは差し戻しは起きません。止めているのは真ん中のフックと右端の CI です。差し戻しから編集へ戻る下の矢印が、人が何も言わないまま書き直しが始まる経路にあたります。

Step 0　代替の書き方を1行足す　[2min]

VSCode で `CLAUDE.md` の「実行環境の制約」の表を開き、自分で挙げた3つのうち表に無い構文を1行足してください。左の列に使えない書き方、右の列にこの環境での書き方を書きます。3つとも載っていた方は、右の列の言い方を自社の言葉に直してください。

Step 1　フックが無い状態で頼む　[3min]

1. VSCode で `dl-training-app` を開き、画面左下のステータスバーのブランチ名が `ex/p3-yamada` になっていることを確かめてください。違う名前なら、ブランチ名をクリックして一覧から `ex/p3-yamada` を選びます。main から切ると、Day1 に足した deny と PreToolUse が入りません。

2. もう一度ブランチ名をクリックし、「新しいブランチの作成」で `ex/p4-yamada` を作ってください。 `yamada` はご自身のユーザー名に置き換えます。いま乗っている `ex/p3-yamada` から切られます。

3. VSCode で `.claude/settings.json` を開き、 `"hooks"` の中に `"PostToolUse"` がまだ無いことを確認してください。

4. Claude Desktop を開き、セッション開始時のプロンプト領域で Project folder に `dl-training-app` を選んでください。入力欄に次をそのまま送ります。

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

5. 編集が通ったことを確認してください。差し戻しは起きません。CLAUDE.md には制約が書いてありますが、止める力は持っていません。

こう見えていれば正しい状態です。 `h()` は `util.php` の冒頭にある3行の関数で、書き換わるのはこの1行です。差分は VSCode のソース管理ペインでファイル名をクリックすると開きます。

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

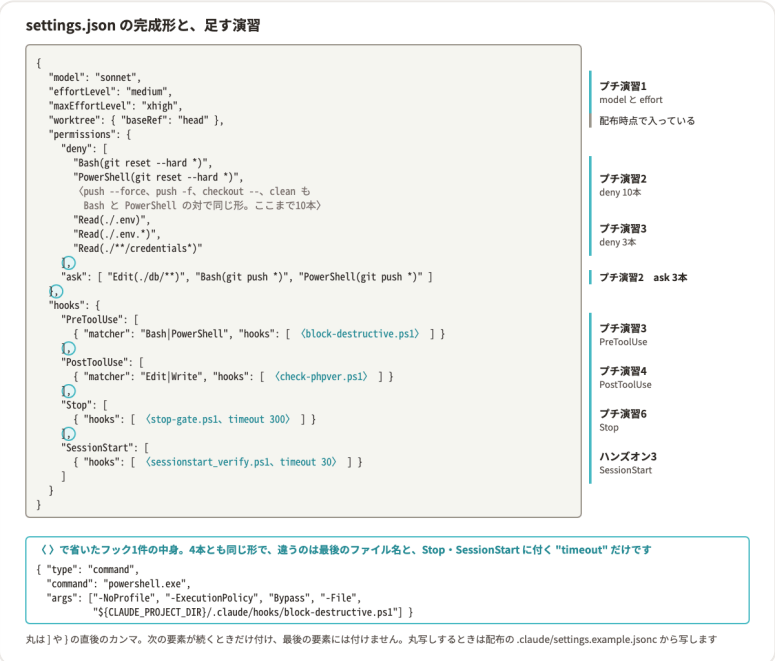
1回目で断られたとき

Step 2　PostToolUse に登録する　[3min]

1. VSCode で `.claude/settings.json` の `"hooks"` に次のブロックを足してください。 `.claude/settings.example.jsonc` の同じブロックと一字一句同じです。

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

足す場所は `"hooks": {` と、それを閉じる `}` の間です。プチ演習3 で PreToolUse を入れた方は、`"PreToolUse": [ ... ]` を閉じる `]` の直後にカンマを1つ置いてから、その下に貼ります。`"matcher": "Edit|Write"` は「Edit と Write の2つのツールの後だけ走らせる」という意味です。`command` に実行ファイル、`args` に引数を分けて書く形なので、パスに空白や日本語が入っても引用符の問題が起きません。



`.claude/settings.json` の完成形と、各演習で足す範囲です。`permissions.deny` の上10本(Bash と PowerShell の対5組)と `ask` がプチ演習2、deny の下3本がプチ演習3、`hooks` の PreToolUse がプチ演習3、PostToolUse がこの演習、Stop がプチ演習6、SessionStart がハンズオン3 です。〈〉で省いたフックの中身は4本とも同じ形なので、図の下に1件だけ全文を載せています。丸で囲んだ所が `]` や `}` の直後のカンマで、次の要素が続くときだけ付け、最後の要素には付けません。プチ演習6 Step 2 で Stop を足すときも、この図の位置に戻って確認します

2. 保存したら、Claude Desktop で `Ctrl+N` を押して新しいセッションを開いてください。`settings.json` はセッションの開始時に読まれます。登録されているかは、

VSCode で `settings.json` を開き、`"hooks"` の中に PreToolUse と PostToolUse が1つずつ並び、赤い波線が無いことで確かめます。

Step 3　同じ依頼をもう一度出す　[3min]

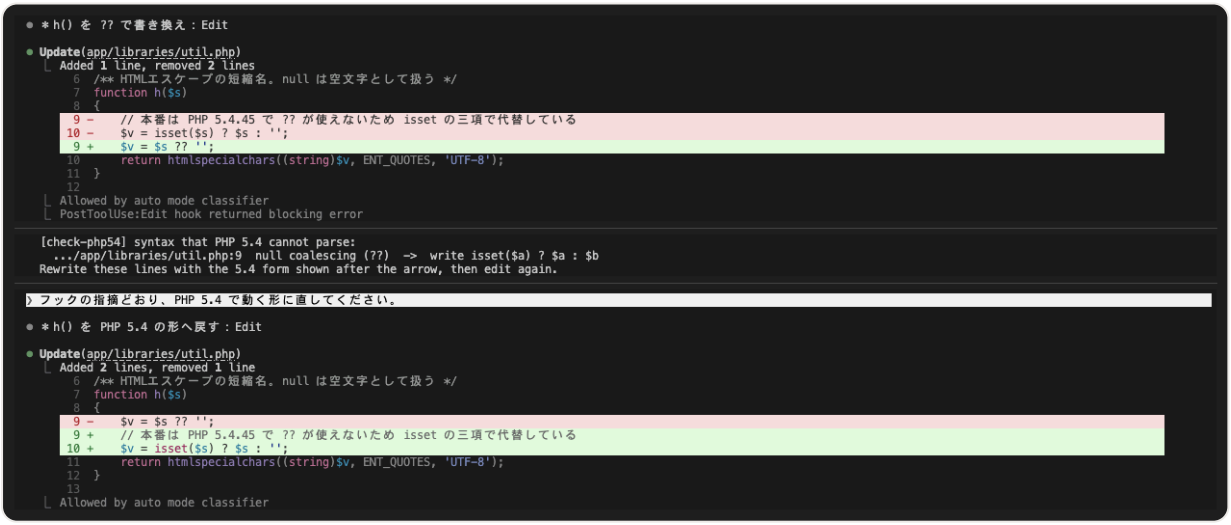
1. ソース管理ペインで `app/libraries/util.php` の行にカーソルを合わせ、右側の矢印アイコン(変更の破棄)で Step 1 の変更を戻してください。戻さないと、Claude が編集せずに「対応済みです」と答えることがあります。

2. Step 1 と同じ文をもう一度送ってください。文面は変えません。

3. 編集の直後にフックが走り、標準エラーの内容がモデルへ返ります。Edit ツールの結果の下に、終了コード 2 で止めた旨と、フックが標準エラーに書いた次の4行が返ります。2行目の先頭のパスと行番号は、ご自身の clone 先で変わります。

```
[check-phpver] code that PHP 7.4 cannot run:
  C:\...\app\libraries\util.php:9  match expression (8.0)  ->  use switch and assign in each branch
Rewrite these lines with the form shown after the arrow, then edit again.
Arrow functions, typed properties, ?? and the spaceship operator are all fine on 7.4.
```

4. そのまま待つと、モデルが `switch` の形へ書き直して編集をやり直します。人が指示を足さずに戻ることを確認してください。書き直しが終わったら、ソース管理ペインの差分に `match` が残っていないことを見ます。



> Step 3 の画面です。上半分が対象外の構文を入れた編集と、その直後に出たフックの差し戻し（`PostToolUse:Edit hook returned blocking error` と `[check-phpver]` の3行）、下半分が対象バージョンの形へ戻した書き直しです。この画面は、差し戻しのあとに人が一言送ってから書き直された版です。通常は一言を足さなくても、差し戻しの文がモデルへ返った時点で自動で書き直しが始まります。ご自身の画面では、その一言が無いまま下半分が出ることを確認してください

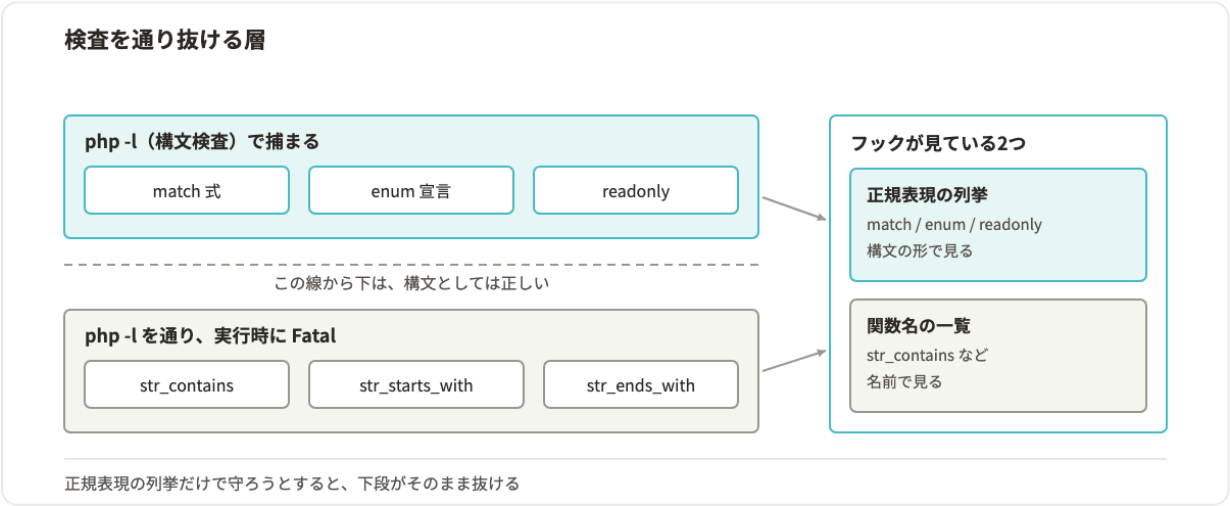
Step 4　検査そのものにテストを足す　[3min]

1. VSCode で `.claude/hooks/tests/fixtures/` に、`str_ends_with()` の呼び出しを1行だけ含む見本を `ng_str_ends_with.php.txt` という名前で作ってください。隣にある `ng_str_contains.php.txt` を開くと形が分かります。1行目が `<?php` 、2行目以降はASCIIだけで数行です。中身は例えば次の形です。

```
<?php
// str_ends_with() arrived in PHP 8.0. php -l on 7.4 does not report it.
function has_suffix($haystack, $needle)
{
    return str_ends_with($haystack, $needle);
}
```

2. `.claude/hooks/tests/cases.json` の `cases` 配列へ、次の1件を足してください。

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



> 点線の上下で、捕まる場所が変わります。下段の3つは構文としては正しいので、`php -l` だけでは素通りします。右の2つが、フックがこれを拾うために持っている判定で、正規表現の列挙が上段に、関数名の一覧が下段に対応します。

この1件は、配布済みの `phpver-ng-str-contains` と同じ側の性質を持ちます。`str_ends_with()` の呼び出しは構文としては正しいので `php -l` を通ります。7.4 には関数が無いので、落ちるのは実行時です。構文検査を通り抜けるものがあるので、フック側は関数名の一覧も見ています。正規表現の列挙だけで守ろうとすると、ここが抜けます。

cases.json の配列末尾に1件足す位置

```
{
  "cases": [
    ...
    {
      "id": "sessionstart-ok-always-zero",
      "hook": "sessionstart_verify",
      "input": { ... },
      "expect_exit": 0
    },
    {
      "id": "env-file-cat-blocked",
      "hook": "block-destructive",
      "input": { "tool_name": "Bash",
                 "tool_input": { "command": "cat .env" } },
      "expect_exit": 2
    }
  ]
}
```

ここにカンマを足す
左の線の範囲が足す1件

配列の要素と要素の間だけカンマ。
最後の要素の後ろに付けると
JSON 破損

最後の要素にはカンマを付けない

Step 4 の手順2 と 3 で触る位置です。左の縦線の範囲が足す1件で、配列の末尾に置きます。丸印の位置、つまり直前の要素を閉じる `}` の後ろにカンマを足し、足した要素の `}` の後ろには付けません。図の中身は説明用の別のケースなので、貼るのは上のコードブロックの1件です

3. 直前の要素の末尾にカンマが入っているかを見てください。JSON の配列は要素の間をカンマで区切り、最後の要素の後ろには付けません。

4. Claude Desktop の統合ターミナルを開いてください。右上の `>_` アイコンか、 `Ctrl` とバッククォートです。セッションの作業ディレクトリで開くので、移動は要りません。ここでテストランナーを回します。

```
powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\tests\run_cases.ps1
```

Step 5　MR を出す　[3min]

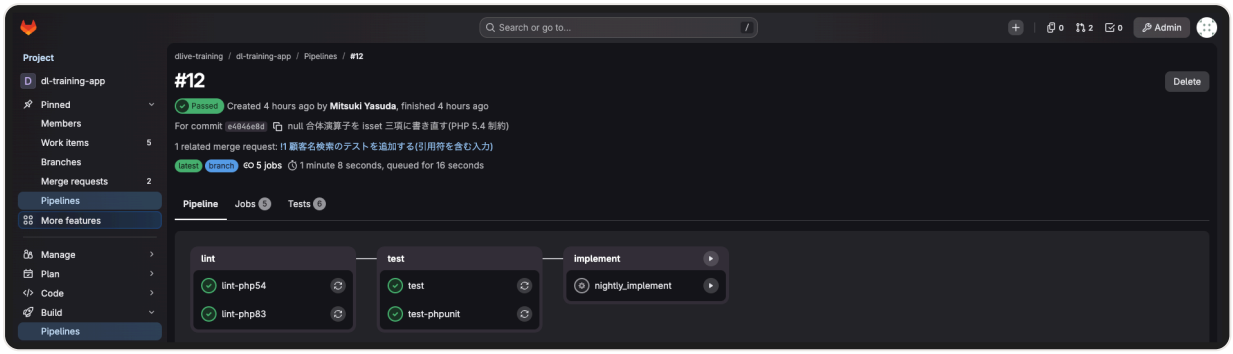
1. ソース管理ペインで変更されたファイルを確かめてください。 `app/libraries/util.php` は Step 3 で `switch` に戻った形のはずです。差分に `match` が残っていないことを見たら、 `util.php` は **変更の破棄** で元に戻します。 `h()` の書き換えは差し戻しを見るための題材なので、コミットには入れません。

2. 1つの変更意図ごとに1コミットで、3回に分けます。ソース管理ペインでファイル名の右の `+` を押して1つだけステージし、メッセージを書いてコミットします。順は、Step 0 の制約節、フックの登録、テストの追加です。

3. 3回コミットし終えたら、ソース管理ペインの「変更の同期」（または「ブランチの発行」）を押して push してください。

4. ブラウザで GitLab のプロジェクトを開き、 **Merge requests** から **New merge request** を選んでください。ソースに自分のブランチ、ターゲットに `main` を指定します。

5 MR の **Pipelines** タブで、パイプラインが動き出したことだけ確認して次へ進みます。結果はプチ演習5 の途中で見に戻ってください。



Step 5 で見る画面。**lint-phpver** の緑は、対象バージョンで動かない構文が混ざらなかったことの最終確認です。手元のフックはここまでの速報にあたります

達成状態

テストランナーの末尾がこう出ていれば、検査側は完成です。配布時点の22件に、プチ演習3 で足した1件と今回の1件が乗ります。プチ演習3 のケースを足していない方は23件です。

```
PASS  phpver-ng-str-ends-with  (exit 2)

24 cases: 24 passed, 0 failed
```

この演習が終わった状態

- [ ] フックを登録する前は `match` が通り、登録した後は同じ依頼が差し戻される。両方を自分の画面で見た
- [ ] 差し戻しの後、人が指示を足さずに `switch` へ書き直された
- [ ] ケースを1件足した状態で、テストランナーが24件すべて通る
- [ ] MR の `lint-phpver` の結果を、プチ演習5 の途中で見に戻って確認した

解説と補足

影響範囲

効く範囲	Edit と Write でのファイル書き換え。Bash 経由の書き換えは拾いません。最終判定は CI の `lint-phpver` です
壊れると何が壊れるか	`settings.json` の JSON が壊れると、フックがまとめて無効になります。画面には何も出ません。`.ps1` に非 ASCII が混ざると、誤検知か文字化けが起きます

会話の速度	編集のたびに走ります。検査が重いほど、書き換えのたびに待ちが増えます

自社ハーネスへ持ち帰るときの置き場所は、次の表で決めてください。D2 と smart3pm のどちらを正にするかは D1-12 の台帳に書いた判断に合わせます。

ファイル	D2 での置き場所	smart3pm での置き場所
`check-phpver.ps1` と `check-phpver.sh`	`.ps1` を正にして hooks 配下へ置き、編集後に走る検査の並びに1本足す	`.sh` を正にして hooks 配下へ置く。smart3pm 側は bash を正として運用されていると伺っているため
CLAUDE.md の制約節	既存の制約節を、代替の書き方の表に差し替える	同じ文面を規約側へ写す。2つのファイルで文面を割らない
追加したテストケースと見本ファイル	`hooks/tests/` の既存ケース集へ追記する	検査スクリプトと対になるテストを1本追加する。テストの無い検査を増やさない

プチ演習5　レビュー出力の JSON ゲート

D2-04 ／ 所要 [15min]

AI のレビューを JSON で受け取り、重大度で合否を返すスクリプトに流して、止める線を決めます。

Day2　**P0 と P1 が1件でもあれば exit 1 を返す判定スクリプト**

目的

レビューを読んで判断していた部分を、数えるだけの作業に変えられるようになります。止める線を自分で決め、手元と CI で同じ判定になる状態にします。

準備

開いておく画面	Claude Desktop（プロジェクトは `dl-training-app`）、その統合ターミナル、VSCode、配布物一式を展開したフォルダ（Windows のエクスプローラー）
いるファイル	`ci/gate.ps1`（bash は `ci/gate.sh`）、`.claude/review.schema.json`、配布物の `20_Day2の演習/プチ演習5_レビュー出力のJSONゲート/findings_sample.json`
ブランチ	プチ演習4 の `ex/p4-<ユーザー名>` のまま進めます。新しく切りません
統合ターミナルの開き方	右上の `>_` アイコン、または `Ctrl` とバッククォート。セッションの作業ディレクトリで開くので、移動は要りません。判定スクリプトはここで回します

API キーは使いません。判定スクリプトに流す入力は、配布の見本ファイルと、Claude Desktop に書き出させたレビューの2つです。

自分で考える　[3min]　スクリプトを触る前に、メモへ3つ書き出してください。

1. 自社の MR を止めてよい指摘はどれかを、重大度の言葉で決めてください。`.claude/review.schema.json` では、P0 が動かないか壊す、P1 が実環境で落ちるか外部入力が素通し、P2 が指示との不一致と波及漏れ、P3 がそれ以外です。

2. 判定できない入力（重大度が知らない値、`findings` が無い）を、通すか落とすかを決めてください。

3. 止める線を1段下げたとき、1日あたり何件の MR が止まると困るかを見積もってください。

手順

Step 1　見本ファイルを置く　[2min]

1. Windows のエクスプローラーで、配布物一式の `20_Day2の演習/プチ演習5_レビュー出力のJSONゲート` を開いてください。

2. `findings_sample.json` を、VSCode のエクスプローラーの `dl-training-app` のいちばん上（ルート）へドラッグしてコピーしてください。

3. VSCode でそのファイルを選んで `F2` を押し、名前を `findings.json` に変えてください。中身は P0 1件、P1 1件、P2 2件で、最後の1件の `confidence` が 0.5 です。

Step 2　既定の閾値で回す　[3min]

1. 統合ターミナルで、次の2行を上から順に打ってください。既定は `findings.json` と閾値 P1 です。

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

2. 1行目を `Enter` で実行してから、2行目を打ってください。2つを1行につなげると、`$LASTEXITCODE` が判定スクリプトの引数として渡り、`1` という名前のファイルを探しにいきます。

3. 1行目の出力の先頭が `gate: threshold=P1 total=4 blocking=2` 、2行目の終了コードが `1` であることを確認してください。CI はこの数字だけを見て、0 なら通し、それ以外なら落とします。

Step 3　閾値を1段下げる　[2min]

1. 閾値を P2 にして、同じ `findings.json` で回してください。

```
powershell -NoProfile -ExecutionPolicy Bypass -File ci\gate.ps1 -Threshold P2
```

2. `blocking` が 2 から 4 に増えることを確認し、増えた2件を読んでください。

3. 増えた指摘のうち、自社なら本当に MR を止めるものが何件あったかを数え、台帳に書いてください。

Step 4　判定できない入力を流す　[2min]

1. VSCode で `findings.json` の P2 の1件の `"severity": "P2"` を `"severity": "high"` に書き換えて保存し、Step 2 の2行をもう一度打ってください。`blocking` が 3 に増えます。知らない重大度は落とす側に数えるためです。

2. 書き換えを `"P2"` に戻してください。

> **未知の値の扱いを変えないでください**　`gate.ps1` と `gate.sh` は、知らない `severity` を落とす側に寄せ、`findings` の配列が無い入力は判定せずに終了コード 1 で落とします。判定できないものを通す作りにすると、ゲートが静かに無効になります。

Step 5　自分の差分のレビューを流す　[3min]

1. Claude Desktop の入力欄に次をそのまま送ってください。書き出しまで1分前後かかります。

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

2. Write ツールの確認が出たら承認してください。

3. 統合ターミナルで、書き出したファイルを判定にかけてください。

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

4. キーの欠けや形の崩れがあれば、判定スクリプトが終了コード 1 で落とします。落ちた場合は、標準エラーの1行で何が足りなかったかを読んでください。`findings.json` と `findings-mine.json` はコミットしません。

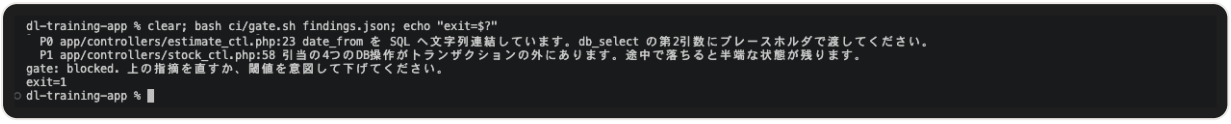
達成状態

見本ファイルを閾値 P1 で回すと、1行目がこう出て終了コードが 1 になります。続く2行は、止めた P0 と P1 の指摘です。

```
gate: threshold=P1 total=4 blocking=2
```

閾値 P2 では `gate: threshold=P2 total=4 blocking=4` になり、終了コードは同じく 1 です。止めたときの最後の行は `gate: blocked. fix the findings above or lower the threshold on`

`purpose.` です。 `.ps1` は ASCII だけで書く決まりなので、スクリプトが出す文が英語なのは正常です。



> 止まったときの出力例です。Mac で bash 版の `gate.sh` を流して撮ったもので、1行目の件数と終了コードは `gate.ps1` と同じです。最後の行だけ、bash 版は日本語で出ます

この演習が終わった状態

- [ ] 見本ファイルで、閾値 P1 と P2 の `blocking` が 2 と 4 に変わることを自分の画面で見た
- [ ] 知らない重大度を入れると、落とす側に数えられることを確かめた
- [ ] Claude Desktop に書き出させた自分のレビューを、同じ判定スクリプトに流した
- [ ] 止める線を自分の言葉で説明できる

解説と補足

影響範囲

効く範囲	MR を出す前の自己チェック。同じ判定がハンズオン2 の CI ゲート `ci/ai_gate.sh` になります
壊れると何が壊れるか	手元と CI で閾値が食い違うと、手元で通ったものが CI で落ちます。未知の `severity` を通す作りに変えると、ゲートが静かに無効になります
文字コード	PowerShell 5.1 の `>` は UTF-16 で書き出すため、判定スクリプトが読めません

ファイル	D2 での置き場所	smart3pm での置き場所
`gate.ps1`	PowerShell 版を正にする。編集後に自動で走らせず、MR を出す前に人が手で叩く位置へ置く	`gate.sh` を正にする。同じ引数の並びを保ち、2言語で判定を割らない
`review.schema.json`	レビューの出力形として1か所に置き、手元と CI の両方から同じファイルを参照する	同じファイルを共有する。カテゴリの語彙を増やすときは、指摘の集計側と一緒に直す
閾値と確信度の下限を決めた記録	持ち帰り台帳に、選んだ値とその理由を1行で残す	同じ値を使う。2つのハーネスで線を変えると、比較する数字が揃わなくなる

ハンズオン2　レビュー観点の切り分けと CI ゲート化

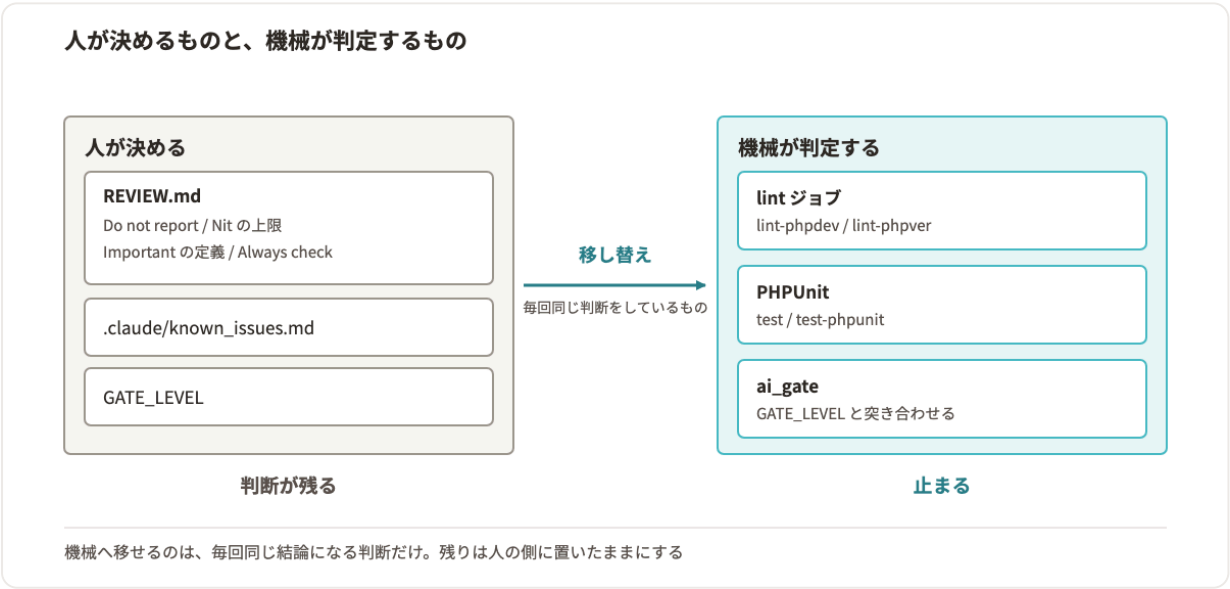
出てきた指摘を、人が読むものと機械が止めるものに振り分けられるようになります。

目的　**指摘の絞り込みと、機械判定への移し替え**

Day1 では、同じ変更を3通りで読ませると指摘の中身が変わることを見ていただきました。ここではその先で、指摘の置き場所を変えます。機械で判定できるものは CI へ、人の判断が要るものだけを AI の出力に残します。終わると、レビュー基準の実物(`REVIEW.md`)、観点を分けた3本のレビュアー、P0/P1 で合流を止める CI ゲートが、同じ1本の MR の上でつながります。

> **この演習の合格点**　ゲートで1回は落ちてください。一度も落ちないゲートは、設定してあるだけで効いていません。落ちた出力と、それをどう扱ったかが MR に残っていれば合格です。

> **Tips：**この演習は Claude Desktop の中の Claude Code で進めます。ファイルの編集とコミットは VSCode、MR とパイプラインの確認は GitLab のブラウザ画面です。判定スクリプトを手元で回すところだけ、Claude Desktop の中のターミナルを使います。



> この演習で動かすものの向きです。左は人が文章で決める場所、右は数字と終了コードで判定する場所で、演習中に作るファイルはどちらかに必ず属します。矢印に乗せてよいのは、毎回同じ結論になる判断だけです。そうでないものを右へ送ると、止まった理由を人が説明できなくなります。

達成状態　**終わったかどうかの判定**

- [ ] `REVIEW.md` の4つの見出しが埋まり、`.claude/known_issues.md` に3件ある
- [ ] 3本のレビュアーの findings が別々に残り、検出率とノイズ率の数字が出ている
- [ ] 足したテストが、実装前のパイプラインで赤だった履歴が MR に残っている
- [ ] `ai_gate` が1回赤になり、その出力と指摘ごとの採否と理由が MR に書いてある
- [ ] 最後の push で走った2本のパイプラインが緑になっている。ブランチ用(lint と test の4ジョブ)と MR 用(`ai_review` と `ai_gate`)の2本です

準備 始める前にそろっている状態

開いておく画面	Claude Desktop、VSCode、ブラウザで GitLab の演習プロジェクト
いるファイル	手元にクローン済みの `dl-training-app` 。`REVIEW.md` 、`.claude/agents/` の3本、`.gitlab-ci.yml`
直前の演習の成果物	プチ演習4・5 の変更が `ex/p4-<ユーザー名>` にコミット済みであること。Day1 の宿題の分類表があれば手元に出しておく
ブランチ	`ex/p4-<ユーザー名>` から `ex/h2-<ユーザー名>` を切ります。Step 1 の手順1 で作ります
紙かメモ帳	「自分で考える」で4つの欄を書きます

早見表 この演習で触るもの

触るファイル	`REVIEW.md` 、`.claude/known_issues.md` 、`.claude/agents/` の3本、`.claude/skills/review-route/SKILL.md` 、`.gitlab-ci.yml` (`ai_gate` の `allow_failure` と `GATE_LEVEL`)、`tests/phpunit/EstimateSearchTest.php` 、`app/controllers/estimate_ctl.php`
操作	レビュー基準を人が先に書き、観点別レビュアー3本を今の main に並列で当てて検出率を出す。ゲートを有効にし、テストを先に送って赤を見てから実装し、落ちた指摘を仕分けて緑まで回す
見る場所	Claude Desktop のバックグラウンドタスクの3本、findings の `file:line` の重なり、GitLab の **Pipelines** タブ、`ai_gate` のジョブログ、`計測表_Day1_Day2.md`

達成の状態	ゲートで1回落ちて直した記録が MR に残り、最後の push のブランチ用と MR 用の2本のパイプラインが緑。検出率とノイズ率が計測表に入っている
自社での使いどころ	自社のレビュー基準の初版と、機械で判定できる指摘を CI へ移す型。既知の許容の台帳
影響が及ぶ範囲	`allow_failure` を外すと、ゲートの赤で MR 用のパイプラインが赤になる。`GATE_LEVEL` は仮に P2 にしてから P1 に戻す。**Do not report** が広すぎるとゲートが空振りする

題材 Issue #2 と Issue #3

Issue	内容	この演習での使い方
#2	在庫の引当で半端な状態が残る（`app/controllers/stock_ctl.php` の `assign()`）	Step 2 で3本のレビュアーを当てる対象。実装はしません
#3	見積一覧の期間検索と CSV 出力（`app/controllers/estimate_ctl.php` の `index()`）	Step 2 のレビュー対象。Step 3 で受入条件1と3、受入条件2 の前半を実装して MR を通します

> **Tips：**Issue #3 の受入条件4(`EstimateCsvTest.php` の3本)は、この80分には含めていません。発展課題に回しています。

進め方 Step の構成と提出先

Step	すること	時間
自分で考える	レビュー基準に入れる観点と、止める重大度を紙に書く	[5min]
Step 1	レビュー基準を人が先に決める。`REVIEW.md` と既知の許容の台帳を書く	[10min]
Step 2	観点を分けた3本のレビュアーを、今のコードに並列で当てる	[30min]
Step 3	CI ゲートを有効にし、テストを1本足して MR を緑まで回す	[28min]

Step 3 補足	同じリファクタで、テストが無い場合と有る場合の差を測る	[7min]
合計		[80min]

提出先は GitLab の Merge Request です。ブランチ名は共通の進め方 STEP 2「ブランチ名の形」に合わせて `ex/h2-<ユーザー名>` にし、Step 1 から Step 3 までの成果物を同じブランチに載せて1本の MR で出します。

ブランチと MR の扱いの補足

自分で考える [5min] 人が先に決めること

紙かメモに書き出してください。ここを AI に相談してから始めると、出てきた案を評価する基準が手元に残りません。

1. 機械判定できる観点を3つ挙げます。CI の `lint-phpdev` と `lint-phpver` が既に見ているものは、AI に重ねて言わせる必要がありません。
2. 直近の改修で見送った指摘を3件思い出し、見送った理由を1行ずつ書きます。
3. 「機能テストの無い変更は Important として報告する」を自社の基準に入れるかどうかを決めます。
4. 合流を止める重大度の下限を決めます。P0 だけか、P1 までか。Step 3 の `GATE_LEVEL` がこの値になります。

手元に材料が無いときの探し方

ハンズオン2 Step 1　レビュー基準の起案

STEP 1 [10min]　REVIEW.md と既知の許容の台帳

演習リポジトリの `REVIEW.md` は、4つの見出しだけがあって中身が空です。ここが AI レビューの正になります。

1. VSCode の画面左下の表示が `ex/p4-<ユーザー名>` で、ソース管理ペインに未コミットの変更が残っていないことを確認します。違うブランチなら、ブランチ名を押して一覧から `ex/p4-<ユーザー名>` を選びます。そのあとブランチ名を押し、**新しいブランチの作成** で `ex/h2-<ユーザー名>` を作ります。
2. VSCode で `REVIEW.md` を開きます。埋める順番は **Do not report** 、 **Nit の上限** 、 **Important の定義** 、 **Always check** です。
3. **Do not report** に、自分で考える(1)で挙げた3つを移します。CI の lint が見ている範囲は、ここに入れます。
4. **Nit の上限** に件数を数字で書きます。「5件までを1つのコメントに」のように、まとめ方も書きます。
5. **Important の定義** を3件から5件に絞ります。 `lint-phpver` が見ている範囲(PHP 7.4 に無い構文と関数)は手順3で **Do not report** に入れたので、ここには入れません。書き出しに迷ったら、下の折りたたみ「Important の出発点の例」を開いてください。
6. **Always check** に、差分に出ていなくても毎回見させることを書きます。自分で考える(3)の判断をここに入れます。
7. `.claude/known_issues.md` を新規に作り、自分で考える(2)の3件を `K-001` から番号を振って書きます。1件は「場所」「指摘の内容」「見送った理由」の3行です。
8. `REVIEW.md` の **Do not report** から、この台帳を1行で参照します。
9. VSCode の **ソース管理** ペイン(左サイドバーの枝分かれアイコン)を開き、 `REVIEW.md` と `.claude/known_issues.md` の2つだけをステージに上げます。メッセージ欄に「handson2: レビュー基準の初版と既知の許容の台帳」と書いて **コミット** を押します。

ここで1つ決めていただきます　`ci/ai_review.sh` が読むのは `REVIEW.md` の1本だけです(スクリプト内の `RULES_FILE`)。手元のサブエージェントは `.claude/known_issues.md` を `Read` で開けますが、CI は開きません。K-00x を CI にも効かせるなら、要約を `REVIEW.md` に書き写すか、 `RULES_FILE` を2本の連結に変えるかのどちらかです。選んだほうと理由を MR の説明に1行書いてください。

この Step が終わった状態

- [ ] ソース管理ペインの差分で、2ファイルだけがコミットに入っている
- [ ] REVIEW.md の4つの見出しが全部埋まっていて、Important が5件以内に収まっている
- [ ] .claude/known_issues.md に K-001 から3件ある
- [ ] K-00x を CI に効かせる方法を1つ選び、MR の説明に書く内容が決まっている

Important の出発点の例

解説と補足　対象バージョンの書き方

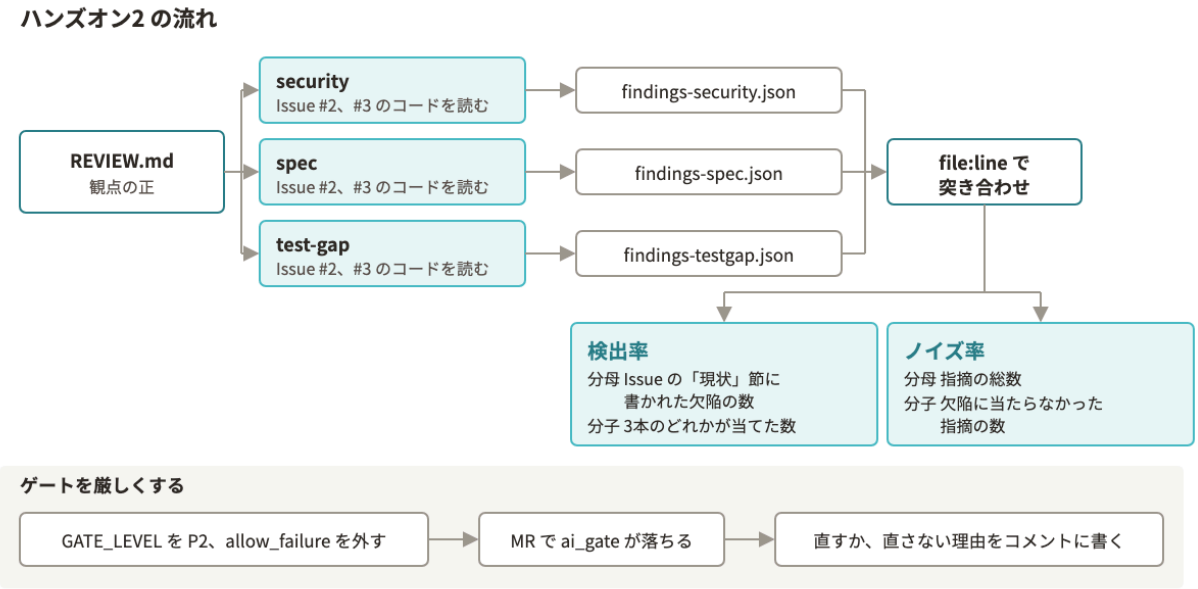
Important に何を入れるか決まりません

> **影響範囲：** `REVIEW.md` はこの後のレビュアー3本と CI の `ai_review` の両方が読むので、ここを緩めると Step 3 のゲートが空振りします。自社に写すときはリポジトリ直下に置き、レビュー用のエージェント定義と CI のスクリプトの両方から参照先として指定します。

ハンズオン2 Step 2　観点別レビュアーの並列適用

STEP 2 [30min]　3本の重なりと差

当てる先は、実装後の差分ではなく今の `main` のコードです。Issue #2 と #3 の欠陥が入ったままの状態に当てるので、既に分かっている欠陥をどれだけ拾うかが、そのまま数字で出ます。



> Step 1 で書いた `REVIEW.md` を観点の正として、security、spec、test-gap の3本が同じコードを並列で読み、`findings-security.json`、`findings-spec.json`、`findings-testgap.json` が残ります。それを `file:line` で突き合わせて検出率とノイズ率を出します。下段が Step 3 です

1. VSCode で `.claude/agents/` の3本を開きます。`security-reviewer.md`、`spec-reviewer.md`、`test-gap-reviewer.md` です。frontmatter の `model` と `effort` が3本とも違うことを確認してください。

2. 3本の「見るもの」節を、Step 1 の `REVIEW.md` に合わせて直します。**Do not report** に入れたものが「見るもの」に残っていたら消します。

3. 手順2 を保存し、Claude Desktop でこのプロジェクトを開きます。画面左上のプロジェクト名から演習フォルダを選びます。

4. Claude Desktop の入力欄に、次をそのまま貼って送信します。

```
security-reviewer と spec-reviewer と test-gap-reviewer の3つを並列で起動してください。
対象は app/controllers/stock_ctl.php の assign() と、
```

```
app/controllers/estimate_ctl.php の index() です。
関連する Issue は _issues/issue-2.md と _issues/issue-3.md にあります。
3本の findings を、レビュアーごとに分けてそのまま出してください。
要約もまとめもしないでください。
出し終えたら、3本の結果をそれぞれ findings-security.json、findings-spec.json、
findings-testgap.json という名前でリポジトリのルートに保存してください。
```

5 動いている間に、右上の⋮メニューの **バックグラウンドタスク** を開きます。3本が同時に動いていることと、モデル名が3本とも違うことを確認します。終わるまで数分かかります。

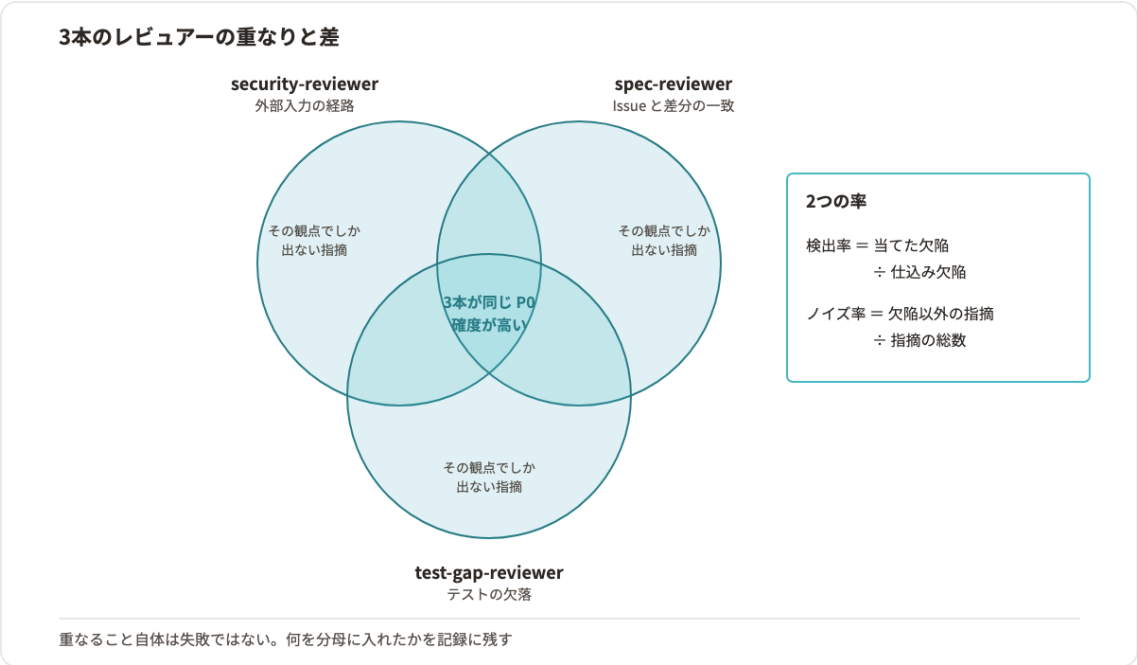
6 3本の結果が返ったら、VSCode で3つの JSON を開き、 `file` と `line` で突き合わせます。同じ場所を何本が指摘したかを数えます。この3ファイルは Step 3 の手元判定でも使います。コミットには入れません。

7 検出率とノイズ率を出します。検出率は、Issue #2・#3 の「現状」節に挙がっている欠陥の件数を分母、当てた件数を分子にします。欠陥として数えるのは、直さないと誤動作するものです。「条件の組み立てが `index()` の中にしかない」「CSV の入口がまだ無い」のような設計と未実装の話は数えません。ノイズ率は、指摘の総数を分母、仕込み欠陥以外の件数を分子にします。何を分母に入れたかも記録に残してください。

8 VSCode で `.claude/skills/review-route/SKILL.md` を新規に作り(フォルダも新規)、変更規模でレビュアーを選ぶ手順を書きます。50行未満は `review-light` 、50行以上または DB・権限・外部連携に触る変更は `review-full` 、観点を分けたいときは3本を並列、の3分岐です。先頭に次の frontmatter を置き、その下に本文を書きます。

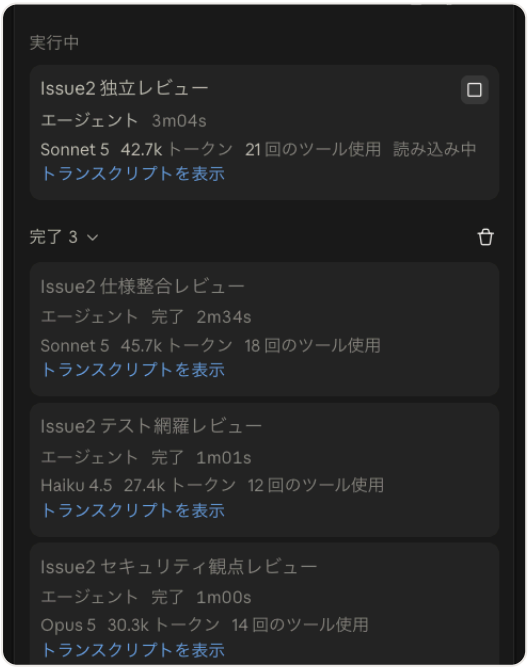
```
---
name: review-route
description: 変更の規模でレビュアーを選びます。
---
```

9 ソース管理ペインで `.claude/agents` と `.claude/skills/review-route` をステージに上げ、「handson2: 観点別レビュアーと振り分けの skill」でコミットします。 `findings-*.json` は手元の記録なのでステージに上げません。



3本を並列で走らせたときの、指摘の重なり方です。中央の重なりは無駄ではなく、別々の観点が同じ場所を指したという情報のほうに意味があります。円の外側の弧が、その観点を入れなければ出なかった指摘で、ここが分けた効果にあたります。右の2行は手順7で使う式です。

手順5の画面です。3つのエージェント名が同時に走り、下の一覧でトークン量が別々に増えていれば並列になっています。モデル名はバックグラウンドタスクのペインで確認できます



右上の：メニューの **バックグラウンドタスク** で開くペインです。実行中の本数と、完了した本数が分かれて並びます。1本ごとにモデル名とトークン数が出るので、3本のモデルが違うことをここで読みます

この Step が終わった状態

- [ ] 3本の findings が、レビュアーごとに分かれた形で手元にある
- [ ] 同じ file:line を何本が指摘したかを数えた表がある
- [ ] 検出率とノイズ率の分母と分子の数字が出ている
- [ ] .claude/skills/review-route/SKILL.md がコミットに入っている

解説と補足　サブエージェントと返ってくる形

3本が並列にならず、1本ずつ順番に動きます

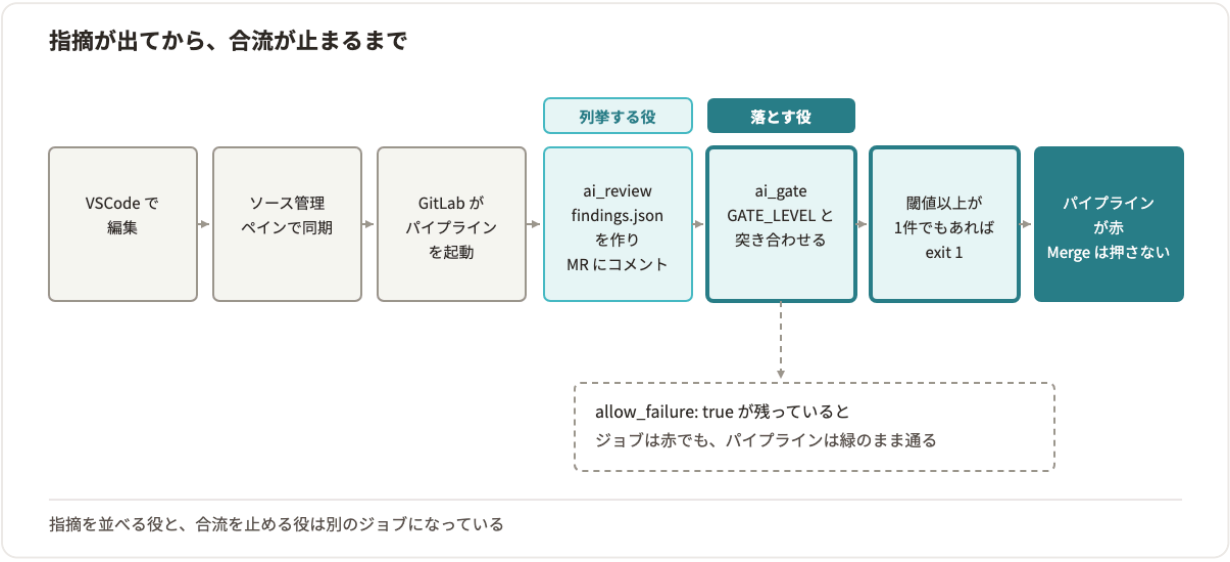
3本とも同じ指摘しか出しません

> **影響範囲：**観点を分けると1本あたりの指摘は減り、3本を足した総数は増えます。自社に写すときは `agents/` に3本を置き、どの変更でどれを呼ぶかを規約側に1行足します。3本とも常に呼ぶ運用にすると、レビュー待ちの時間が3倍になります。

ハンズオン2 Step 3　CI ゲートの有効化とテスト生成

STEP 3 [28min]　テスト先出しとゲートの順番

順番を1つだけ入れ替えます。実装より先にテストを送って、赤くなるところを見ます。赤くならないテストは、通っても何も保証しません。CI の履歴に赤が残っていることが、そのテストが今のコードを見ていた証拠になります。

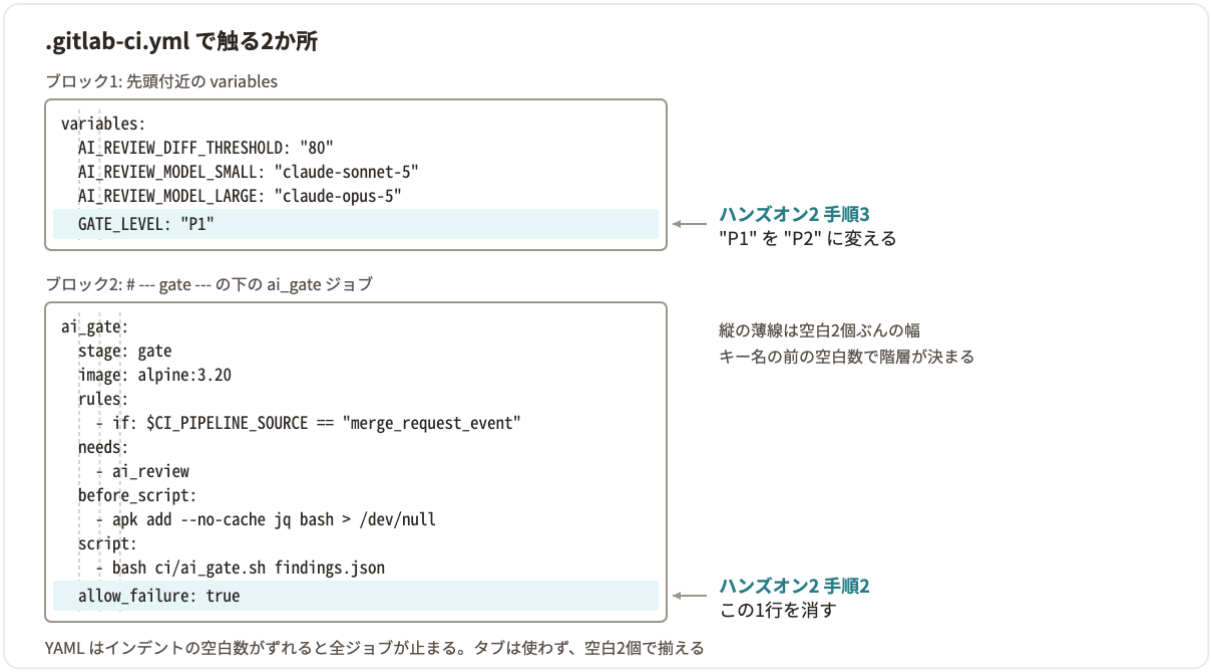


> 指摘が出てから合流が止まるまでの経路です。列挙する役の `ai_review` と落とす役の `ai_gate` が別のジョブなので、MR にコメントが付いただけでは Merge はできる状態のままです。下の破線が手順2で消す行にあたり、これが残っていると `ai_gate` が赤でもパイプラインは緑のまま通ります。

1. VSCode の画面左下で、ブランチが Step 1 で切った `ex/h2-<ユーザー名>` のままであることを確認します。

2. VSCode で `.gitlab-ci.yml` を開き、`ai_gate:` で始まるブロック(`# --- gate ---` の見出しの下)の最後の行 `allow_failure: true` を1行まるごと消します。他の行のインデントは変えません。

3. 同じファイルの先頭近く、`variables:` の下にある `GATE_LEVEL: "P1"` を、一度だけ `GATE_LEVEL: "P2"` にします。ゲートが本当に止めることを1回見るための仮の値で、手順11 で戻します。



手順2 と 3 で触る2か所です。上のブロックが先頭付近の `variables`、下のブロックが `# --- gate ---` の下の `ai_gate` ジョブです。縦の薄線が空白2個ぶんの幅で、キー名の前の空白の数で階層が決まります。行を消すときに上の行の空白まで巻き込むと、他のジョブごと止まります

4. Issue #3 の受入条件3を読み、`date_from` の値が条件として使われないとき一覧に何件出るかを、先に紙に書きます。`db/seed.sql` の見積は15件です。

5. Claude Desktop にテストを1本頼みます。

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

6. ソース管理ペインで、`.gitlab-ci.yml` だけをステージして1コミットにします。Step 2 の成果物は Step 2 の手順9 でコミット済みです。`.gitlab-ci.yml` を入れ忘れると CI 側は `allow_failure: true` のままで、ゲートは一度も落ちません。メッセージは「handson2: ai_gate の allow_failure を外し GATE_LEVEL を仮に P2 にする」です。

7. テストだけをさらに別のコミットにして、ソース管理ペインの **変更の同期** を押します。メッセージは「handson2: 期間検索が条件として使われないことのテストを追加」です。

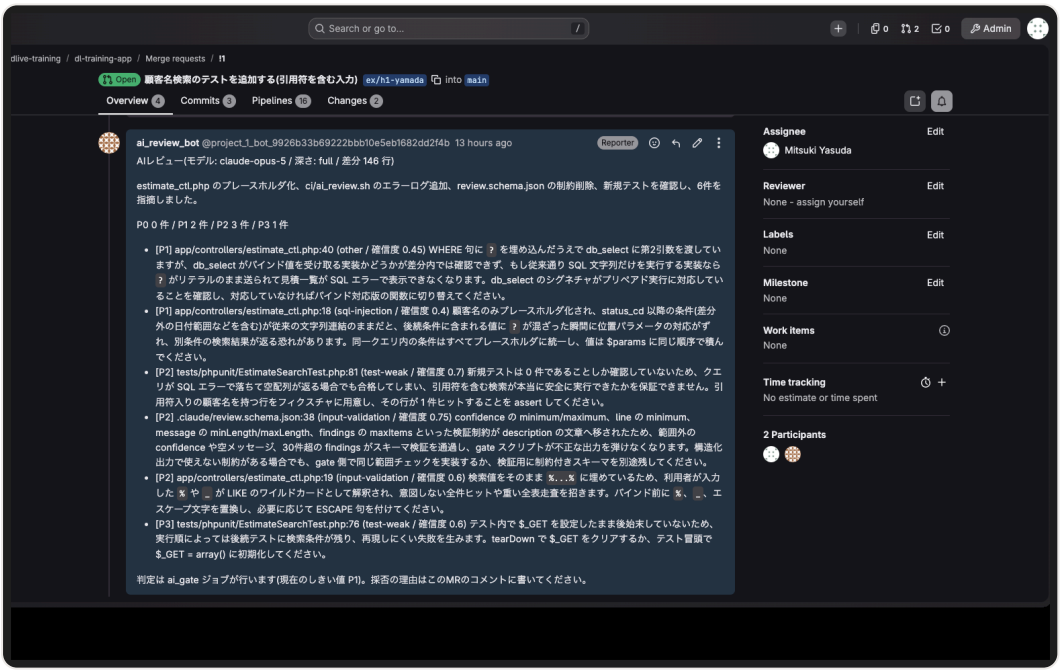
8. ブラウザで GitLab の演習プロジェクトを開き、画面上部に出る **Create merge request** を押します。出ていない場合は左のメニューの **Code** から **Merge requests** を開き、

New merge request で元を自分のブランチ、先を `main` にして作ります。

9 MR の **Pipelines** タブを開きます。**branch** のラベルが付いた行で `test-phpunit` が赤になっていることを確認します。赤いジョブ名をクリックするとジョブの画面が開くので、落ちた出力を MR の説明の「テスト」欄に貼ってください。

10 Issue #3 の受入条件1と、受入条件2 のうち private メソッドの切り出しまでを実装します。そのメソッドが返す2つの値(WHERE 句の文字列とパラメータ配列)の形を、先にご自身で紙に書いてから Claude Desktop に渡してください。依頼文に入れるのは、対象ファイル(`app/controllers/estimate_ctl.php` だけ)、Issue #3 の「期待する振る舞い」の 1 から 3 をそのまま、紙に書いたメソッドの戻り値の形、テストファイルは触らないこと、の4点です。

11 ソース管理ペインでコミットして **変更の同期** を押します。**branch** の行で `test-phpunit` が緑になり、**merge request** の行で `ai_review` が MR にコメントを1本置き、`ai_gate` が赤になります。緑のままなら下の「ai_gate が緑のままで、一度も落ちません」を開いてください。

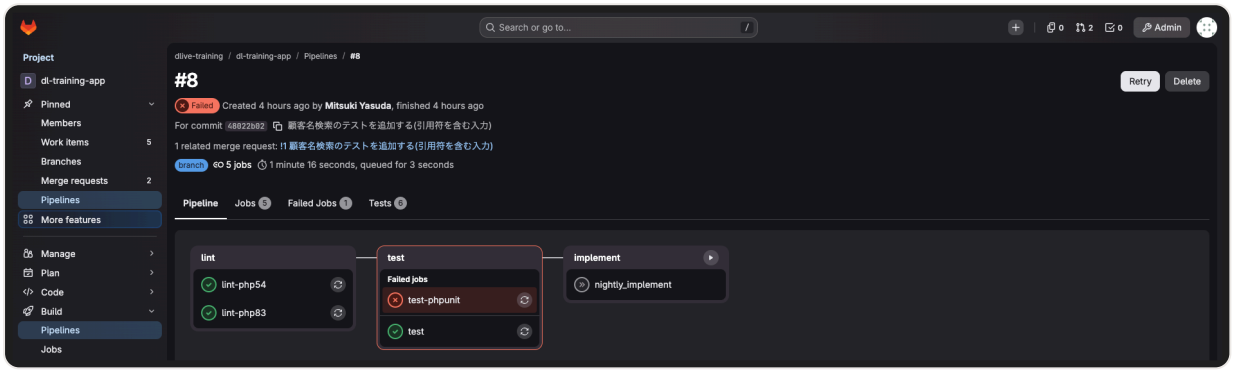


手順11 で MR の **Overview** タブに付く `ai_review_bot` のコメントです。1行目に使ったモデル、レビューの深さ、読んだ差分の行数が出ます。その下が P0 から P3 の件数で、続けて指摘が `[P1] ファイル:行（カテゴリ / 確信度）` の形で並びます。差分が `AI_REVIEW_DIFF_THRESHOLD` (既定 80)を下回ると軽いモデルに切り替わり、1行目のモデル名が変わります。判定はこのコメントではなく `ai_gate` ジョブが行います

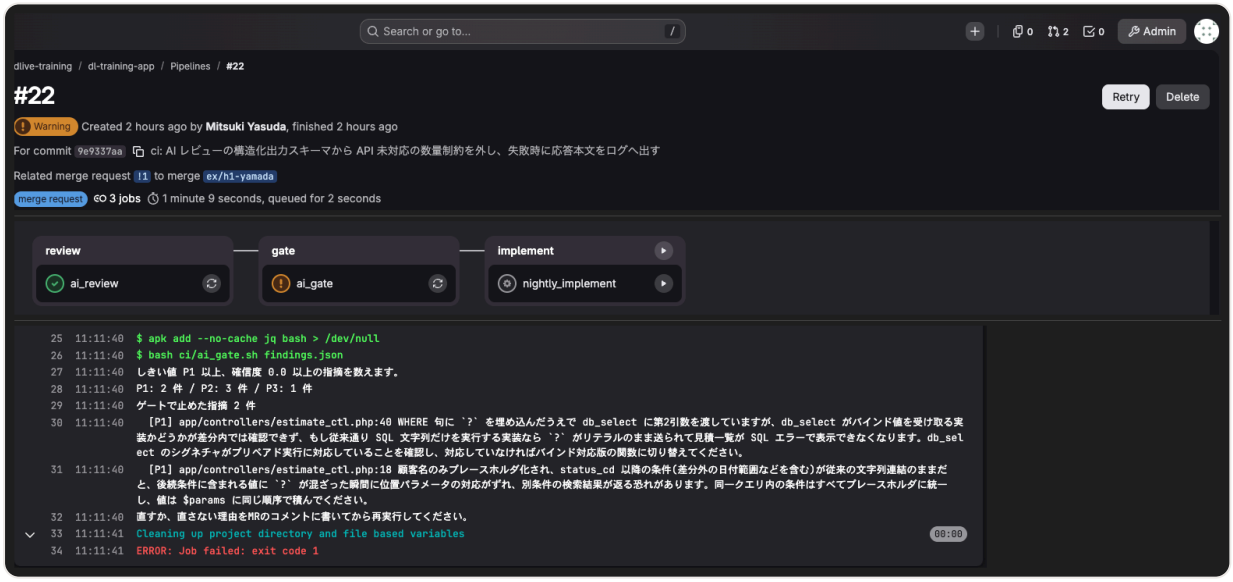
12 落ちた指摘を1件ずつ、即修正・要調査・将来課題・意図した設計の4つに仕分け、MR の説明の「人が判断した採否」の表に書きます。直すものを直し、「意図した設計」にしたものは **Do not report** に入れるかどうかを検討します。

13 `.gitlab-ci.yml` の `GATE_LEVEL` を `"P1"` に戻し、修正と合わせてコミットして同期します。この push で2本のパイプラインが走ります。**branch** の行のブランチ用(`lint-phpdev` 、 `lint-phpver` 、 `test` 、 `test-phpunit`)と、**merge request** の行の MR 用(`ai_review` と `ai_gate`)の両方が緑になったことを確認します。

> **落ちたときに全部直さないでください**　4件出たら4件直す、では Day1 の課題がそのまま戻ります。直さない判断をした指摘には理由を書いてください。理由が繰り返し同じであれば、それは `REVIEW.md` の **Do not report** か `.claude/known_issues.md` に移す合図です。



手順9で開くブランチ用のパイプラインです。実装前なので `test-phpunit` だけが赤になります。この赤が、足したテストが今のコードを見ていた証拠です



手順11で開く MR 用のパイプラインと、その下が `ai_gate` のジョブログです。 `ai_review` は緑のまま、 `ai_gate` がしきい値以上の指摘で止まっています。列挙する役と落とす役が分かれているためです。画面は `allow_failure` を外す前なのでオレンジの ! ですが、手順2 のあとは赤の × になり、MR 用のパイプライン全体が赤になります。**Merge** ボタンまで押せなくなるのは **Pipelines must succeed** を有効にしたプロジェクトで、演習のプロジェクトでは押せますが押しません

この Step が終わった状態

- [ ] 実装前のパイプラインで test-phpunit が赤だった履歴が MR に残っている
- [ ] ai_gate が1回赤になり、その出力が MR の説明に貼ってある

- [ ] 指摘ごとの採否と理由が MR の表に書いてある
- [ ] 最後の push で走ったブランチ用と MR 用の2本のパイプラインが緑になっている

解説と補足　ジョブのログの読み方

手元で先に同じ判定をかける方法

ai_review ジョブが動きません

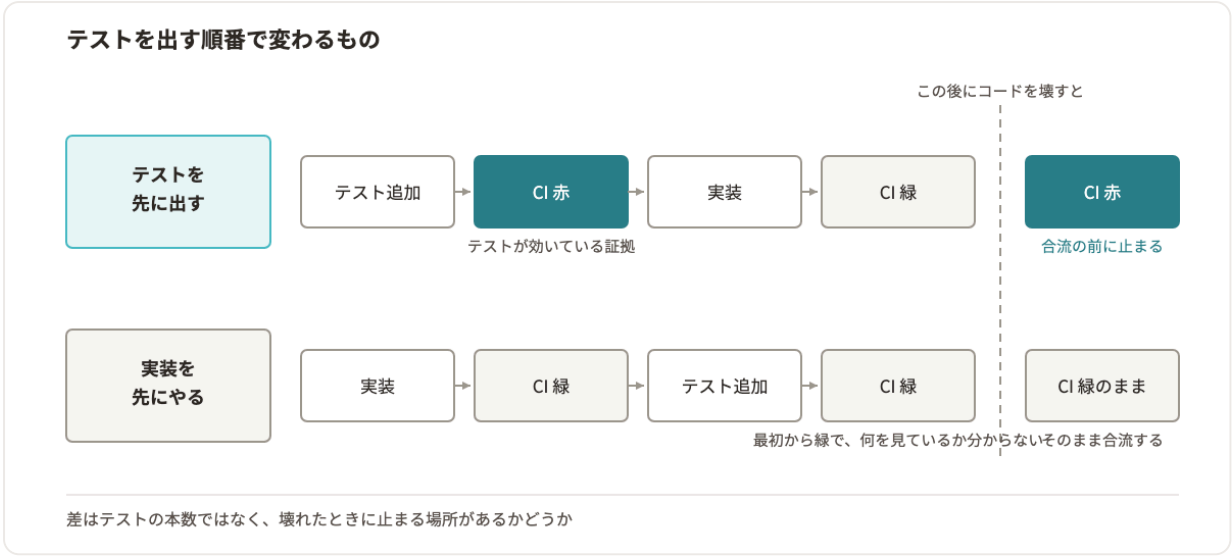
ai_gate が緑のままで、一度も落ちません

test-phpunit が赤ではなく、エラーで止まります

> **影響範囲：** `allow_failure` を外した時点から、ゲートが赤いと MR 用のパイプラインが赤になります。合流まで止めるには、プロジェクトの **Pipelines must succeed** と組にします。自社に写すときは、まず `allow_failure: true` のまま1週間まわして件数を見てから外すと、止まりすぎの調整が先に済みます。 `GATE_LEVEL` はチームごとに1つ、CI の変数に置きます。

STEP 3 補足 [7min]　テスト無しと有りの差

「機能テストを足せばリファクタを安全に進められる」という見立てを、ここで1回だけ実測します。緑が出たあとに壊してみる手順です。



同じテストを書いても、出す順番で残るものが変わります。上の帯には CI の履歴に赤が1回残り、そのテストが動いているコードを見ていた証拠になります。下の帯は最初から緑なので、その証拠が取れません。縦の点線から右が、この補足で実際に試す部分です。

1. 手順10 で入れた「 `YYYY-MM-DD` の形に合わない値を条件に使わない」判定を、1行だけ外したコミットを作ります。他は変えません。

2. 同期してパイプラインを見ます。 `lint-phpdev` 、 `lint-phpver` 、 `test` は緑のままです。既存の5本のテストも緑で、手順5で足した1本だけが赤になります。

3. この1本が無ければ、同じ壊れ方が合流まで届いていたことを確認します。

4. 確認できたら戻します。**ここはコマンドを使います。**コミット単位の打ち消しは画面から行えないので、Claude Desktop の中のターミナルで打ちます。

   ```
   git revert --no-edit HEAD
   ```

   `--no-edit` を付けると、コミットメッセージを編集する画面を開かずに打ち消しのコミットができます。そのあとソース管理ペインで **変更の同期** を押します。

戻すときは revert です `git reset --hard` は使いません。 `.claude/settings.json` の `permissions.deny` と PreToolUse フックで止まる設定になっているはずですが、止まる前に手で打たないでください。送信済みの変更を戻すのは `revert` です。

Tips：この差は、テストの本数ではなく「壊れたときに赤くなる場所があるかどうか」の差です。テストを増やすこと自体が目的になると、呼んでいるだけで結果を見ていないテストが増えます。1本足すたびに、実装前のコードで赤くなることを確かめてください。

ハンズオン2　OK 基準と持ち帰り物

OK 基準　ゲートで1回落ちて直した記録

完了の判定

- [ ] ai_gate が1回赤になり、その出力と、指摘ごとの採否と理由が MR に残っている
- [ ] 最後の push で走ったブランチ用と MR 用の2本のパイプラインが緑になっている
- [ ] 足したテストが、実装前は赤だったことがジョブの履歴から追える
- [ ] 配布の `計測表_Day1_Day2.md` の「Day2 検出率」に、検出率とノイズ率が入っている

マージまでは進めません。マージの判断は Day2 の最後にまとめて扱います。

生成物　ファイルの名前と置き場所

演習リポジトリでの置き場所と、自社ハーネスへ写すときの置き場所です。D2 は PowerShell が正、smart3pm は bash が正なので、判定スクリプトだけ行き先が分かれます。

ファイル	D2 での置き場所	smart3pm での置き場所
`REVIEW.md`	リポジトリ直下。コードレビュー側のエージェント定義から参照させる	リポジトリ直下。規約側に1行足して参照先にする
`.claude/known_issues.md`	リリースレビュー側にある同名の台帳を、コードレビュー側へ写す	同じ内容を1本置き、検査スクリプトからは参照しない
`.claude/agents/` の3本	`agents/` に追加。PowerShell に依存しないのでそのまま置ける	`agents/` に追加。同上
`.claude/skills/review-route/SKILL.md`	`skills/` に追加。既存の軽量パスの節と書き方をそろえる	`skills/` に追加

`.gitlab-ci.yml` の `ai_review` と `ai_gate`	CI 側へ新規。機械強制を定義どおりに戻す位置づけ	pre-receive の検査とは別に、CI 側へ新規
判定スクリプト	`ci/gate.ps1` (PowerShell 5.1、ASCII のみ)	`ci/gate.sh`
追加した PHPUnit テスト	テストの置き場所の規約に合わせる	同左
検出率とノイズ率の記録	計測表_Day1_Day2.md の「Day2 検出率」へ記入	同左。置き場所と担当だけ持ち帰り台帳.md に書く

> **Tips：**持ち帰り台帳には「置き場所」と「担当」の欄があります。ここで決めきれないものは、空欄のままで構いません。Day2 の最後に、置き場所と担当を全員ぶん確定させます。

追加と考察

検出率の記録と Day1 との比較

Step 2 で数えた値を、配布の `計測表_Day1_Day2.md` の「Day2 検出率」の表へ書き込んでください。表の行と列は次のとおりです。記入は計測表側で行います。

レビュアー	仕込み欠陥の件数	当てた件数	指摘の総数	検出率	ノイズ率
security-reviewer					
spec-reviewer					
test-gap-reviewer					
3本の合計(重複を除く)					

書き終えたら、次の3つを1行ずつ書いてください。全体共有ではここを聞きます。

1. Day1 の3通りレビューで数えた指摘件数と比べて、増えたか減ったか。減ったのであれば、それは `REVIEW.md` のどの行が効いた結果か。
2. 3本のうち、既知の欠陥を1件も拾えなかったレビュアーがいたか。いた場合、観点の書き方と当てた対象のどちらに原因があるか。
3. ゲートで止まった指摘のうち、直さない判断をしたものの割合。この割合が高いのであれば、`GATE_LEVEL` か **Important の定義** のどちらかが現場に合っていません。

検出率は指摘を絞っても動きません　分母が仕込み欠陥の件数なので、絞って動くのはノイズ率のほうです。両方下がったら `Do not report` に入れすぎです。ここで見たいのは、指摘を絞ったときに既知の欠陥を落としていないかどうかです。

発展課題　CSV 出力とレビュアーの4本目

早く終わった方から、上から順に進めてください。全部やる必要はありません。

1. Issue #3 の受入条件4(`tests/phpunit/EstimateCsvTest.php` の3本)を満たすところまで実装します。ここで受入条件2 の残り(`csv_text()` から条件組み立てのメソッドを呼ぶ)も満たします。テストを先に出して赤を見る順番は同じです。

2. Issue #2(在庫引当のトランザクションと行ロック)を、同じ流れで MR まで通します。`ai_review` は MR の差分しか読まないので、`stock_ctl.php` の指摘が出るのはこの MR からです。

3. `.gitlab-ci.yml` の `variables:` に `GATE_MIN_CONFIDENCE: "0.6"` を1行足して回し直し、止まる件数の差を見ます。確信度で切る運用が使えるかどうかの判断材料になります。

4. `AI_REVIEW_DIFF_THRESHOLD` を 80 から動かし、`ai_review` が使うモデルが切り替わる境界を測ります。

5. レビュアーの4本目を作ります。対象の PHP 7.4 に無い構文だけを見る1本です。プチ演習4で書いた `check-phpver` のフックと役割が重なるので、どちらを残すかまで決めてください。

6. 構造化出力を CLI で扱います。**ここはコマンドを使います。**Claude Desktop の中のターミナルで、スキーマを指定してレビュー結果を JSON に落とし、そのまま判定スクリプトへ渡す形を作ります。研修の端末には `claude` の CLI を入れていないので、研修後に CLI を入れた環境で試してください。PowerShell 5.1 の `>` は UTF-16 で書き出すので、`Out-File` で UTF-8 にします。

   ```
   claude -p --output-format json "REVIEW.md の基準で app/controllers/estimate_ctl.php を読み、.claude/review.schema.json の形で返してください" | Out-File -Encoding utf8 findings-cli.json
   ```

   `--output-format json` の出力は外側に包みがあるので、そのまま判定スクリプトに渡すと `findings` の配列が無いとして落ちます。中身の取り出し方はプチ演習5 の「研修

後に自分の API キーで CLI から試す場合」にあります。Claude Desktop は画面で結果を見るところまでで、ファイルを作って次の処理へ渡す形は CLI のほうが組みやすくなります。自動化に載せる段階でここに戻ってきてください。

> **Tips：**発展課題5は、そのまま「[A] で守っているものを [M] へ移す」作業です。フックで機械的に止まるのであれば、AI に同じことを見させる必要はありません。移した結果、レビュアーの観点が1つ減ることが望ましい姿です。

プチ演習6　止まる Stop hook

D2-07 ／ 所要 [15min]

テストが赤い間だけ作業を続け、決めた回数で必ず止まる Stop hook を登録します。

Day2　**テストが赤い間だけ1回続き、そこで止まる Stop hook**

目的

応答を終わらせてよいかの判定を機械に移し、止まらなくなる作りとの差を説明できるようになります。あわせて、戻す手間を減らすために変更の範囲を先に絞る考え方を持ち帰ります。

準備

開いておく画面	Claude Desktop、VSCode、講師の投影画面（Step 1 は見るだけです）
いるファイル	`.claude/settings.json` 、 `.claude/hooks/stop-gate.ps1` （bash は `stop-gate.sh` ）
直前の演習の成果物	プチ演習4 の PostToolUse と、ハンズオン2 までのコミット
ブランチ	ハンズオン2 の `ex/h2-<ユーザー名>` のまま進めます。新しく切りません。 `ex/h2-` は `ex/p4-` から切ってあるので、プチ演習4 の PostToolUse も入っています。ハンズオン3 の `ex/h3-` はこのブランチから切るので、ここで足す Stop がそのまま乗ります
確かめておくこと	統合ターミナルで `php --version` を打ち、手元に PHP があるか。研修用 PC には入れていないので、多くの方は「認識されていません」の表示になります。PHP が無い方は、Step 3 の継続を講師の投影画面で見ます。 `stop-gate` は PHP が無いとテストを回さずに終えるためです

自分で考える　[2min]　メモへ2つ書き出してください。

1. 自社で、AI の応答を終わらせてはいけない状態を1つ挙げてください。テストが赤い以外で書きます。

2. その判定にかかる時間を秒で見積もってください。フックは応答のたびに走ります。1回30秒かかる検査を置くと、会話の速度がその分だけ落ちます。

手順

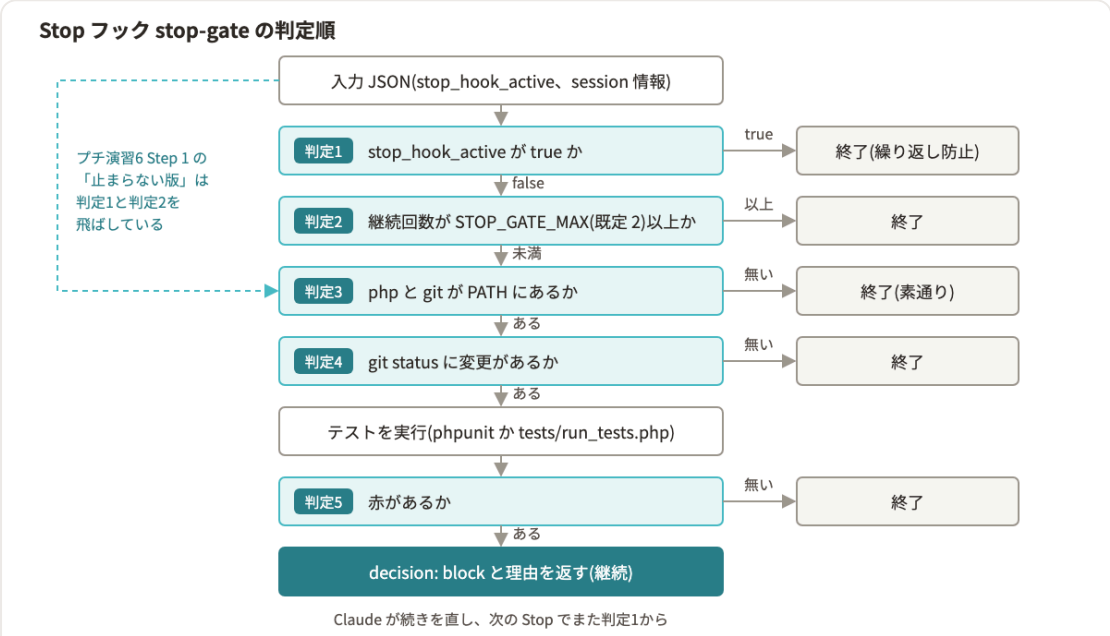
Step 1　止まらない版を見る　[3min]

講師が2つの版を並べて回します。受講者は手を動かさずに画面を見てください。止まらない版は各自の端末では回しません。止まらない版は、次の3行だけでできています。

```
# demo only. do not register this one.
$payload = [Console]::In.ReadToEnd() | ConvertFrom-Json
Write-Output '{"decision":"block","reason":"tests are red. fix and finish."}'
```

配布の版は、同じ処理の前に通す条件を置いています。どれかに当たった時点で、何も返さずに終わります。主な3つは次のとおりです。

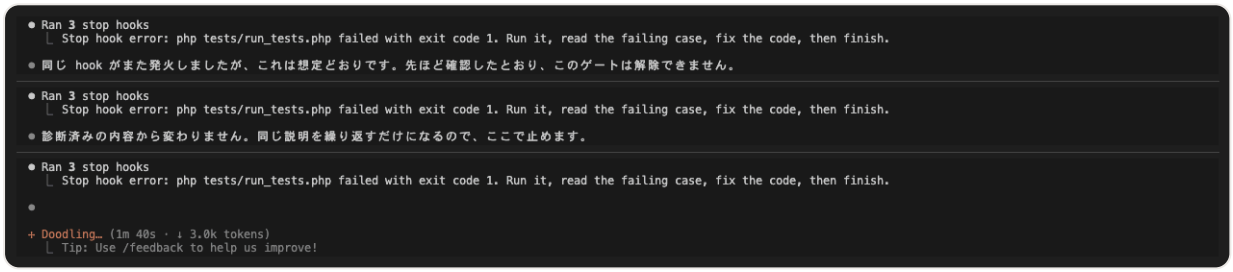
```
# Guard 1: already continued by this hook.
if ($payload.stop_hook_active -eq $true) { exit 0 }

# Cap: continuations counted per session in .stop-gate.state
if ($count -ge $max) { ...; exit 0 }

# Guard 2: nothing changed in the working tree
if ([string]::IsNullOrWhiteSpace(($changes | Out-String))) { ...; exit 0 }
```



> 配布の `stop-gate` の判定順です。上から、判定1 で `stop_hook_active` が true なら終了、判定2 で継続回数が `STOP_GATE_MAX` (既定 2)以上なら終了、判定3 で `php` と `git` が PATH に無ければ終了(php を入れていない端末はここで素通りします)、判定4 で作業ツリーに変更が無ければ終了、そこで初めてテストを回し、判定5 で赤があるときだけ `decision: block` を返します。左の点線が示すとおり、Step 1 の止まらない版は判定1 と判定2 を持たず、入力を読んだらすぐ判定3 以降へ進みます。継続のたびに同じ判定へ戻る理由がここにあります

観察ポイント	止まらない版	配布の版
継続が入った後の2回目の判定	同じ `block` を返す。判定の材料が前回と同じなので、結果も同じ	Guard1 が先に当たる。`stop_hook_active` が true なの

	になる	で、何も返さずに終わる
画面の見え方	同じ理由の文が続けて増える。Claude Code が8回連続の block で打ち切るまで、応答が終わらない	継続の文が1回だけ出て、その次で応答が終わる
止め方	人が止めるか、8回連続で打ち切られるのを待つ	放っておいても止まる
使用量	1ターンぶんの入力が継続のたびに積み上がる	1本の継続連鎖は Guard1 が必ず1回で切る。`STOP_GATE_MAX` は同じセッションで積み上がる継続回数の上限（既定2）



> Step 1 の前半。`Stop hook error` の同じ文が3回並んでいる部分を見てください。処理が進んでいるのではなく、同じ判定を繰り返しています。Mac の bash 版フックで撮った画面ですが、PowerShell 版でも文面は同じです

Step 2　正しい版を登録する　[3min]

1. VSCode で `.claude/settings.json` の `"hooks"` に Stop のブロックを足してください。`.claude/settings.example.jsonc` の同じブロックと一字一句同じです。プチ演習4と同じく、`"PostToolUse": [ ... ]` を閉じる `]` の直後にカンマを置いてから貼ります。Stop に `matcher` はありません。応答の終わりは1種類しか無いためです。

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

2. 統合ターミナルを開き（右上の `>_` アイコン、または `Ctrl` とバッククォート）、門の1つ目だけを単体で確かめてください。

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

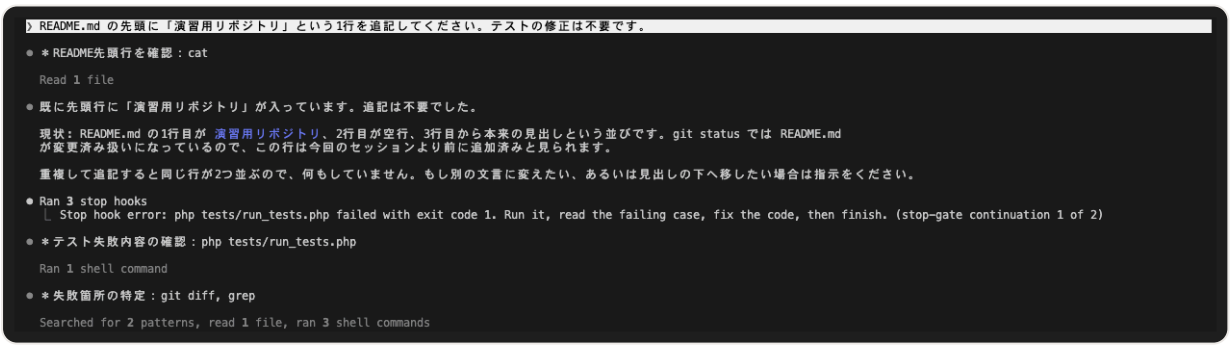
3. 何も表示されず、終了コードが 0 であることを確認してください。ここで `decision` を返す版が、Step 1 の止まらない版です。

4. Claude Desktop で `Ctrl+N` を押して新しいセッションを開き、VSCode の `settings.json` で `Stop` に1本入っていて赤い波線が無いことを確認してください。JSON が壊れていると読まれないので、カンマを見直します。

Step 3　継続の経路を見る　[4min]

1. Claude Desktop で何か1つファイルを編集させてから応答を終わらせてください。内容は何でも構いません。例えば「README.md の末尾に、今日の日付を1行足してください」です。

2. 手元に PHP がある方は、継続が1回入ってから応答が終わることを見てください。PHP が無い方の端末では継続が入らずに終わるので、同じ場面を講師の投影画面で見ます。



> Step 3 の画面。継続の文が1回だけであること、その後に応答が終わっていることを確認してください。末尾の `continuation 1 of 2` が回数の記録です

Step 4　上限の効き方を確かめる　[3min]

1. VSCode で `.claude/settings.json` を開き、先頭の `{` の次の行に次の1行を足して保存してください。`settings.json` の `env` に書いた値は、Claude Code が起動するフックにも環境変数として渡ります。0 は継続そのものを止める値です。

```
  "env": { "STOP_GATE_MAX": "0" },
```

2. Claude Desktop で `Ctrl+N` を押して新しいセッションを開き、Step 3 と同じようにファイルを編集させて応答を終わらせてください。PHP がある方の端末では、継続が入らずに応答が終わります。PHP が無い方は、講師の投影画面で同じ場面を見ます。

3. 確認が済んだら、手順1 で足した `"env"` の1行を消して保存し、`Ctrl+N` で新しいセッションを開いてください。消さないと、この後のハンズオン3 で継続が1回も入りません。

4. ソース管理ペインで `.claude/settings.json` だけをステージし、メッセージ欄に「プチ演習6: Stop フックを登録」と書いてコミットしてください。`"env"` の行が差分に残っていないこと、`.claude/hooks/.stop-gate.state` をステージに上げていないことを先に見ます。Step 3 と手順2 で書かせた `README.md` などの1行は、**変更の破棄** で戻してください。

5. 自社で使うときの上限を決め、台帳に書いてください。検査を残したまま継続だけ切りたいときが 0 です。

達成状態

Step 2 の単体確認では、何も出力されずに終了コードが 0 になります。Step 3 で継続が入る端末は、次の `reason` が画面に出てセッションが続きます。`exit code` の数字は環境で変わります。

```
{"decision":"block","reason":"php tests/run_tests.php failed with exit code 1. Run it,
read the failing case, fix the code, then finish. (stop-gate continuation 1 of 2)"}
```

PHP が無い端末では、フックは標準エラーに次の1行目を書いて終えます。Step 4 で上限を 0 にしたときは、PHP の有無にかかわらず2行目を書いて終えます。上限の判定は PHP を探す手前にあるためです。どちらも標準エラーなので、Claude Desktop の画面には出ないことがあります。

```
[stop-gate] php not found on PATH, letting the session stop
[stop-gate] already continued 0 time(s), the cap is 0. Letting the session stop.
```

この演習が終わった状態

- [ ] 止まらない版と配布の版の差を、通す条件3つで説明できる
- [ ] `stop_hook_active` が true のとき、フックが何も返さずに終わることを単体で確認した
- [ ] 継続が1回入って止まる場面を、自分の画面か講師の投影画面で見た
- [ ] `settings.json` の `env` で上限を 0 にすると、継続が入らずに応答が終わることを確認した
- [ ] `"env"` の行を消し、`.claude/settings.json` の Stop ブロックを1コミットで残した

解説と補足

影響範囲

効く範囲	応答のたびに検査が走ります（timeout 300秒）。夜間の無人実行で、テストが赤いまま終わらせない最後の門になります
壊れると何が壊れるか	`stop_hook_active` を見ない版を登録すると、Claude Code が8回連続の block で打ち切るまで継続が続き、そのぶん使用量が積み上がります
コミットしないもの	`.claude/hooks/.stop-gate.state`。手元の作業メモです

ここで登録したフックは、Claude Desktop から起動しても CLI から起動しても同じ設定ファイルが読まれ、同じように効きます。効く範囲が広いぶん、設定を別の環境へ持っていったときの差がそのまま出ます。同じフックを別の環境へコピーすると、次の4つで黙って効かなくなります。

起きること	結果
改行が CRLF に変わり、`#!/bin/bash` の行末に `\r` が付く	シェルが起動せず、判定が行われない
実行ビットが落ちる	フックが呼ばれずに素通りする
パスが移す前の環境のまま書いてある	保護対象に一致せず、素通りする
判定に使うコマンド（`jq` など）が入っていない	判定の途中で落ち、止まらない

4つとも「止まらずに通る」方向に倒れます。フックが動いていないことに気づかないまま、止めたかった操作が通ります。画面には何も出ません。環境は、AI ツールが動く場所とコードが置いてある場所とランタイムの版の3つの軸で別々に見てください。判定と実行を分け、止める対象を設定ファイルに出しておくと、環境が増えても入口を1本足すだけで済みます。組み合わせ別の設計は、配布物の `50_環境別のフック設計/環境別のフック設計.md` にまとめてあります。

ファイル	D2 での置き場所	smart3pm での置き場所
`stop-gate.ps1` と `stop-gate.sh`	`.ps1` を正にして hooks 配下へ置く。実行するテストの行を、自社のテストコマンドへ差し替える	`.sh` を正にして hooks 配下へ置く。既存の受入テストと同じ並びにテストを1本添える
`settings.json` の Stop ブロック	タイムアウトを自社の検査時間に合わせて書き換えてから写す	同じ検査を bash 側の設定へ写す。2言語で別々の判定を持たない

継続の上限（`STOP_GATE_MAX` の値）	既定値をスクリプト内で決め、台帳に理由を1行残す	同じ値にする。片方だけ緩いと、緩いほうが実質の上限になる

ハンズオン3　夜間ループの最小構成

GitLab のパイプラインからエージェントを起動し、実装から MR 作成までを人の操作なしで通します。この演習を終えると、夜間に任せてよい工程と人が残る工程を、自分の実行ログを根拠に線引きできます。
題材は Issue #4「見積状態コードの対応表の一本化」です。受入条件が全部機械で判定できる形なので、無人で流す1本目に向いています。人に残すのはマージだけです。

> **CI が外へ出られない場合**　GitLab の runner から `api.anthropic.com` へ到達できないときの進め方は、最後の「代替」のカードにまとめています。

目的　無人で流せる範囲と、人が残る境目の確定

Issue 1本を無人で流し、止まった場所を1つ選んで直し、夜間に回せる範囲を1枚に書くところまで進みます。

1. Issue 1本を無人で実装させ、MR が立つところまでを自分の手で1回通す。
2. 1回目に止まった箇所を記録し、止め金の4層のどれで直すかを選んで再実行する。
3. 今のハーネスで夜間に回せる範囲と、人が残る3点を自分の言葉で1枚に書く。

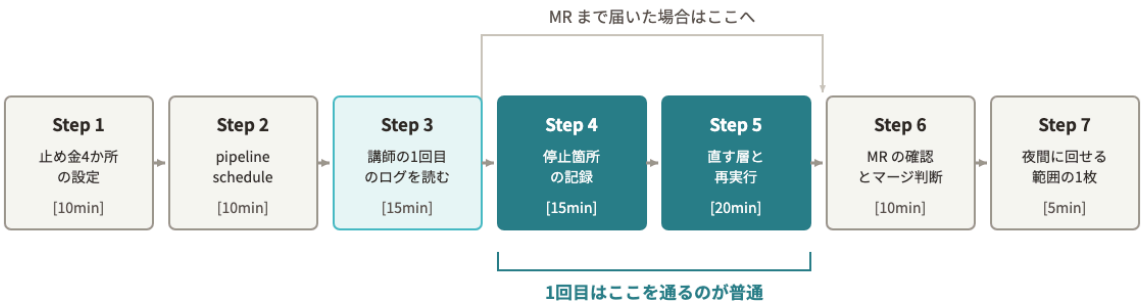
準備　始める前にそろっている状態

開いておく画面	Claude Desktop(`dl-training-app` を開いた状態)、VSCode(同じフォルダ)、ブラウザで GitLab の演習プロジェクト
触るファイル	`.claude/settings.json` (deny 13本とフック4本)、 `.gitlab-ci.yml` の `nightly_implement` 、 `ci/nightly_implement.sh` 、 `.claude/skills/implement-issue/SKILL.md`
作るもの	ブランチ `ex/h3-<ユーザー名>` 、pipeline schedule 1本(Inactive)、MR 1本、計測表の「Day2 夜間ループ」の節
ブランチ	`ex/h2-<ユーザー名>` から `ex/h3-<ユーザー名>` を切ります。ハンズオン2までの deny、PreToolUse・PostToolUse・Stop の3本、 `REVIEW.md` がそ

	のまま乗ります。SessionStart は Step 1 で足します
見る場所	ジョブログ末尾の `turns` と `cost_usd`、artifacts の `ci_logs/`、MR のパイプライン、**Code** > **Branches** の `main` の行
所要	[105min]。前提確認と「考える」で [5min]、Step 1 から Step 7 で [85min]、まとめで [15min] です

Step 3 は講師が `main` で1本だけ流し、全員が同じログを読んで Step 4 の記録を書きます。Step 5 は、各自が自分のブランチのパイプラインで手動ジョブを押します。
コマンドを打つ場面が3か所あります。いずれも Claude Desktop の中のターミナルから実行します。右上の `>_` アイコン、または `Ctrl` とバッククォートで開きます。セッションの作業ディレクトリで開くので、フォルダを移動する必要はありません。

ハンズオン3の進み方



止まった場所を記録して層を選び直すところが、この演習の本体。Step 1 から 7 で [85min]

> 分岐があるのは Step 3 の後だけです。上を通る細い線が MR まで届いた場合で、Step 4 と Step 5 を飛ばして Step 6 に合流します。1回目で通るほうが例外なので、濃い2つを通る前提で時間を見てください。
> 止まった場所を書いて層を選び直すところが、この演習の中身です。

Mac の方への読み替え

考える 止め金の置き場所の下書き

設定ファイルを開く前に、紙かエディタに書き出してください。5分で構いません。ここを飛ばすと、この後の Step 1 が設定を写すだけの作業になります。

1. Issue #4 を無人で流したとき、止まってほしい操作を3つ書き出します。
2. その3つを止める層を、下の4つから1つずつ指定します。

3 翌朝に最初に見る画面を1つ決め、そこで何を見てマージの可否を判断するかを3つ書きます。

(a) deny	`.claude/settings.json` の `permissions.deny`。文字列照合なので `bash -c` のような書き方はすり抜けます
(b) フック	PreToolUse フック。権限モードに関係なく先に走ります。変更してよい範囲の指定もここに置けます
(c) 回数と時間	`--max-turns` とジョブの `timeout`。操作の種類を見ずに打ち切ります
(d) ブランチ保護	GitLab の設定。エージェント側の設定から独立していて、設定ファイルを書き換えられても効きます

達成状態

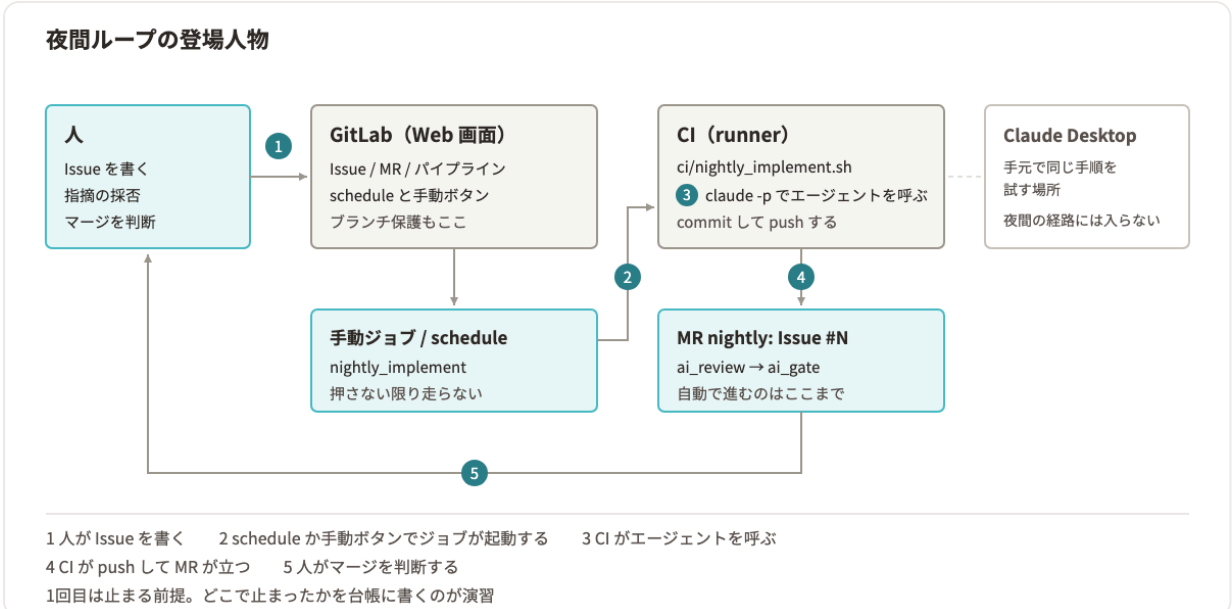
- [ ] 止まってほしい操作が3つ、具体的なコマンドか操作名で書けている
- [ ] 3つそれぞれに (a)〜(d) のどれかが割り当たっている
- [ ] 翌朝に見る画面が1つ決まり、判断材料が3つ挙がっている

同じ操作を2層で止めている箇所の見つけ方

部品 夜間ループの登場人物

この演習で触るファイルは4つです。すでにリポジトリに入っています。

`.gitlab-ci.yml`	`nightly_implement` ジョブ。schedule からは自動で、それ以外のパイプラインでは手動ボタンとして出ます
`ci/nightly_implement.sh`	ブランチを作り、エージェントを呼び、コミットを確かめ、push して MR を作るまでを1本にしたスクリプト
`SKILL.md`	`.claude/skills/implement-issue/`。Issue を読む・方針を3行で書く・影響範囲を調べる・実装する・テストを回す、の手順書
`.claude/settings.json`	止め金のうち、`permissions.deny` とフック登録を持っている場所



> 夜間に回すときの役割分担です。番号の順に進み、自動で進むのは4番の MR が立つところまでで、5番のマージ判断は人に残ります。右端の Claude Desktop は点線でつないであり、手元で同じ手順を試す場所としては使いますが、夜間の経路には入りません。

手元では Claude Desktop で対話しながら進め、夜間は CI が同じ設定のまま非対話で回します。Claude Desktop は非対話実行(`-p`)を持たないので、無人で回る部分は CI 側の CLI が受け持ちます。人が横にいる作業と無人で回る作業の境目が、そのままツールの使い分けに重なります。止め金をどこに置くかは、この境目のどちら側で効かせたいかで決まります。

スクリプトが進める4段と、エージェントに渡しているツール

この演習で初めて出てくる語

ハンズオン3　止め金と schedule の準備

Step 1 と Step 2 です。合わせて [20min]。ファイルの確認と編集は VSCode、エージェントへの指示は Claude Desktop、GitLab の操作はブラウザで行います。git の操作は VSCode のソース管理ペイン(左サイドバーの枝分かれのアイコン)を使います。

STEP 1　止め金4か所の設定 [10min]

「考える」で書き出した3つが、実際にどこで止まるかを4か所で確かめます。

手順

1. VSCode の左下の表示が `ex/h2-<ユーザー名>` で、未コミットの変更が残っていないことを確認します。違うブランチなら、ブランチ名を押して一覧から `ex/h2-<ユーザー名>` を選びます。そのあとブランチ名を押し、**新しいブランチの作成** で `ex/h3-<ユーザー名>` を作ります。名前の決め方は共通の進め方 STEP 2「ブランチ名の形」にあります。

2. VSCode で `.claude/settings.json` を開き、`permissions.deny` に次の13行があることを確認します。無ければ足します。

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

3. 足した場合は、ソース管理ペインでメッセージに 夜間ループの止め金: deny 13行 と入れてコミットします。

4. 同じ `settings.json` の `hooks` に、PreToolUse(`block-destructive`)、PostToolUse(`check-phpver`)、Stop(`stop-gate`)の3本が並ぶことを確認します。そのうえで、Stop を閉じる] の直後にカンマを1つ置き、その下に次の SessionStart のブロ

ックを足して保存します。 `.claude/settings.example.jsonc` の同じブロックと一字一句同じです。

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

保存したら Claude Desktop で `Ctrl+N` を押して新しいセッションを開き、`settings.json` に赤い波線が無いことを確かめて、 夜間ループの止め金: SessionStart でコミットします。Claude Desktop の入力欄では `/hooks` は使えないので、登録は `settings.json` で見ます。

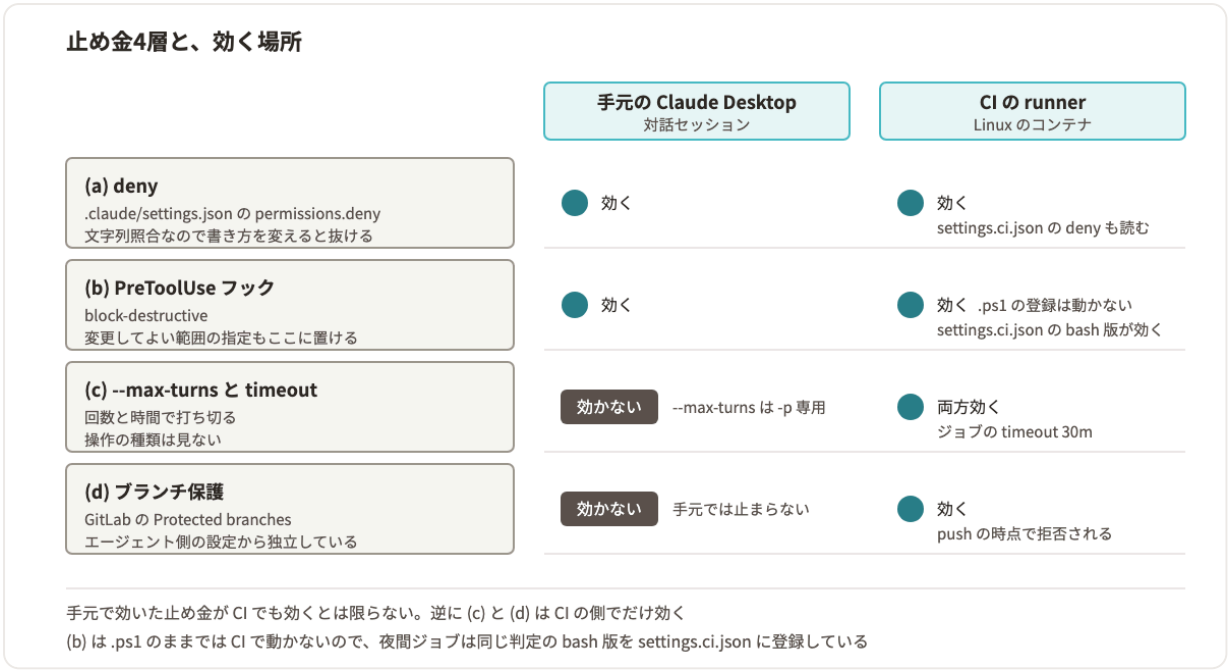
5 VSCode で `.gitlab-ci.yml` のいちばん下の `nightly_implement:` を開き、 `timeout: 30m` と `ISSUE_ID: "4"` があることを確認します。書き換えません。

6 VSCode で `ci/nightly_implement.sh` を開き、 `claude -p "$PROMPT" \` で始まる行に `--max-turns 15` が付いていることを確認します。値は上げないでください。

7 ブラウザで GitLab の演習プロジェクトを開き、 **Code** > **Branches** を選びます。 `main` の行に **protected** の表示が付いていることを確認します。見るだけです。Developer の権限では **Settings** のメニューは出ません。

達成状態

- [ ] ブランチが `ex/h3-<ユーザー名>` に切り替わり、VSCode の左下にその名前が出ている
- [ ] deny 13行、フック4本(SessionStart を足した後)、timeout、--max-turns、ブランチ保護の5つを自分の目で確認した
- [ ] 「考える」で書いた3つの操作のうち、どこでも止まらないものが1つ以上見つかっている



> 縦が止め金の種類、横が効かせたい場所です。(b) の行の CI 側は、`settings.json` に登録した `.ps1` が Linux のコンテナで動かないことを表しています。夜間ジョブは `ci/settings.ci.json` で同じ判定の bash 版を登録し直しているので、PreToolUse と PostToolUse は CI でも効きます。(d) は逆に手元では何も止めません。この2列は、片方だけを見ても埋まらないようにできています。

影響範囲

手元で効く `deny` とフックは、Claude Desktop と CLI が同じ設定ファイルを読むので、どちらで動かしても同じように効きます。Day1 に書いた設定をそのまま使えます。Claude Desktop のパーミッションモード(**手動** / **編集を受け入れる** / **プラン** / **自動**)を **自動** にしても、`deny` とフックは先に走ります。モードが決めるのは確認ダイアログを出すかどうかだけです。
CI の中では、`deny` 、ジョブの `timeout` 、ブランチ保護に加えて、bash 版の PreToolUse と PostToolUse が効きます。`nightly_implement` のイメージは Linux(`node:22-bookworm-slim`)で `powershell` が入っていないので、`ci/nightly_implement.sh` が `--settings ci/settings.ci.json` で bash 版のフックを登録しています。Stop は CI では登録していません。runner には DB が無く、テストが必ず赤になって継続を無駄に使うためです。

解説と補足

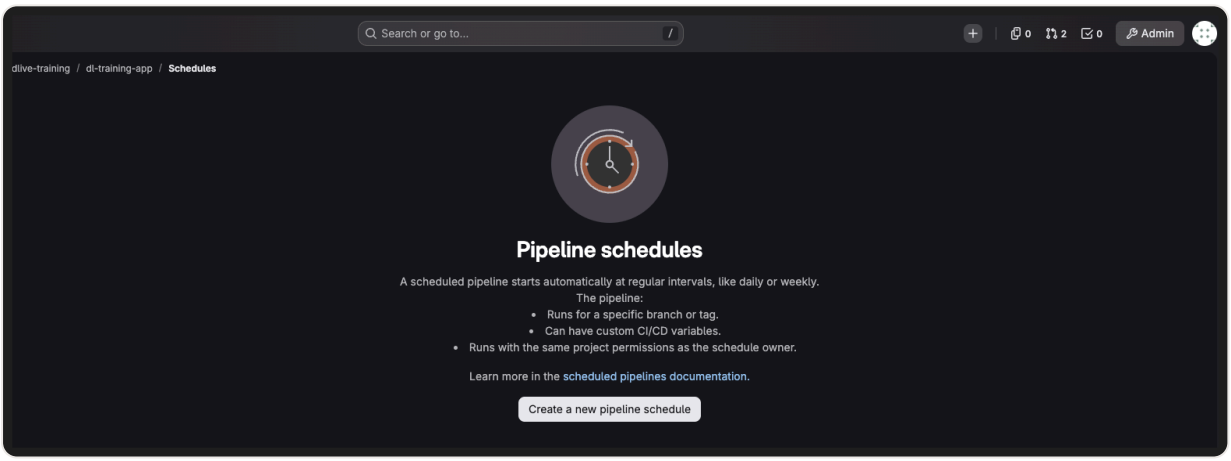
STEP 2 pipeline schedule の作成 [10min]

夜間に動かすための時刻の設定を作ります。当日は同時実行を避けるため手動ボタンから流しますが、schedule 自体は各自が作ってください。持ち帰るのはこの設定です。

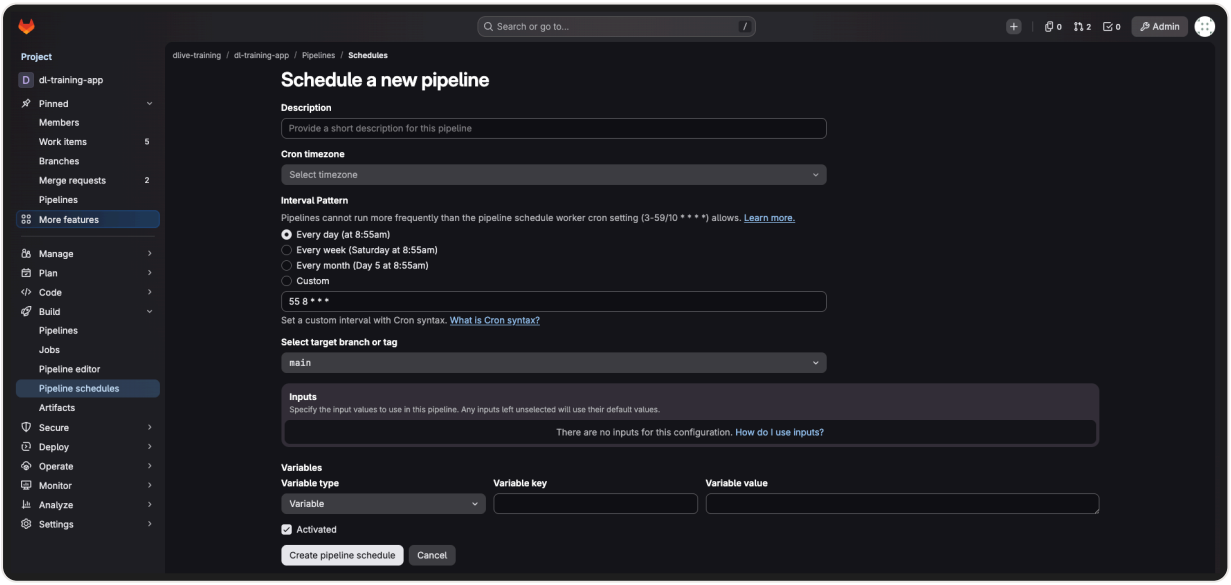
手順

1. GitLab で演習プロジェクトを開き、 **Build** > **Pipeline schedules** を選びます。

2. **New schedule**（1本も無いときは **Create a new pipeline schedule**）を押します。
3. **Description** に `nightly issue-4` と入力します。
4. **Interval Pattern** で **Custom** を選び、`0 2 * * *` と入力します。毎日 2:00 の意味です。
5. **Cron timezone** を Tokyo に、**Target branch** を `main` にします。
6. **Variables** に `ISSUE_ID = 4` を1行足します。
7. **Activated** のチェックを外し、**Create pipeline schedule** を押します。



手順1 で開く画面です。まだ1本も無いときはこの表示になります。



手順3 から手順7 で埋める入力画面です。開いた直後の状態で撮っています。

達成状態

- [ ] Pipeline schedules の一覧に自分の schedule が1本ある
- [ ] その行が **Inactive** になっている
- [ ] Variables に ISSUE_ID = 4 が入っている

影響範囲

schedule を有効にすると、その時刻に全員分が同時に走ります。runner の並列数と API の予算を一晩で使い切ります。時刻で動かすのは、自社のリポジトリに写してからです。

解説と補足

ハンズオン3　実行と停止箇所の記録

Step 3 と Step 4 です。合わせて [30min]。1回目は講師が `main` で1本だけ流し、止まる前提で全員が同じログを読みます。通すことではなく、止まった場所を特定して記録することがこの2つの Step の中身です。

STEP 3　講師の1回目のログを読む [15min]

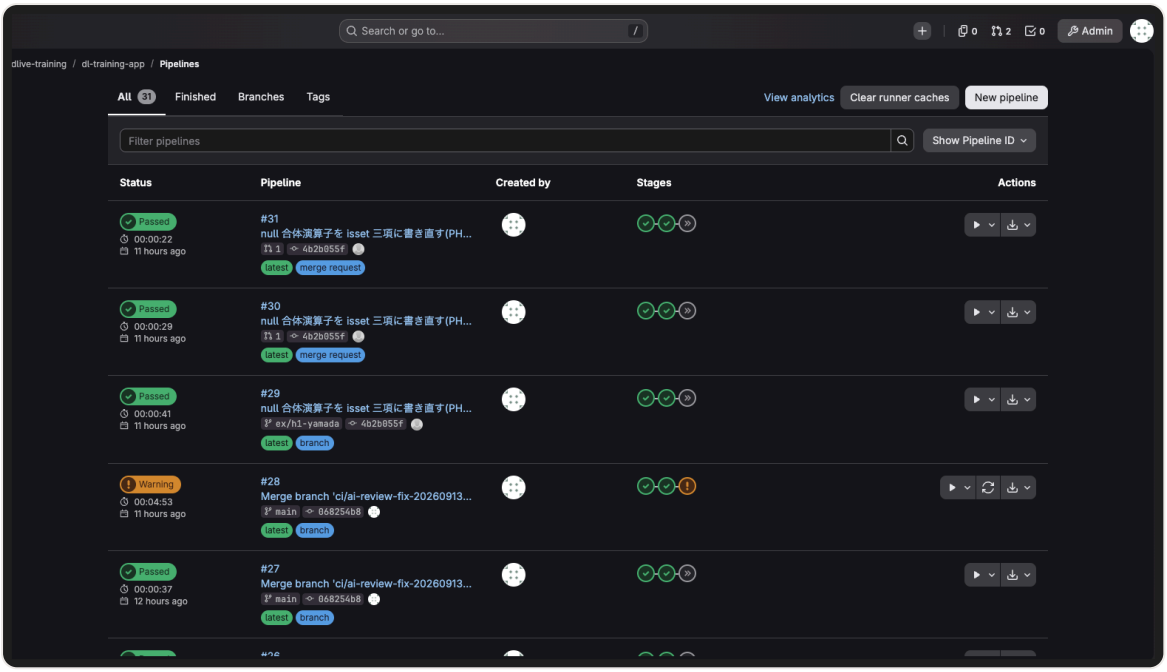
講師が `main` のパイプラインで `nightly_implement` を1本だけ流します。受講者は手動ボタンを押さず、全員が同じジョブのログを読みます。全員分を同時に走らせると、後のジョブが待ちに入って演習の時間内に終わらないためです。

手順

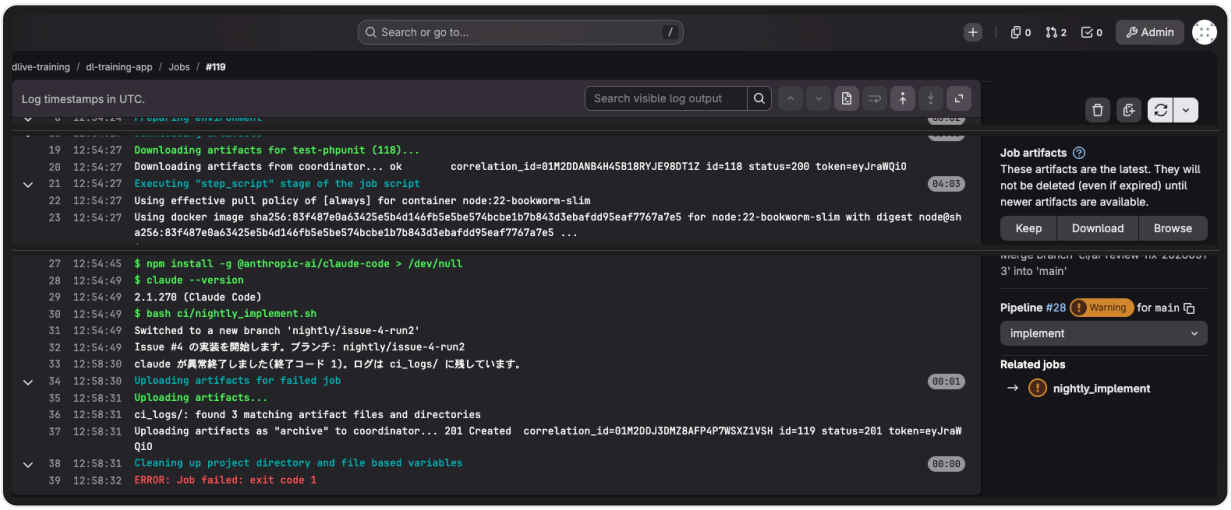
1. 講師が Zoom のチャットにパイプライン番号(`#` の付いた数字)を貼ります。GitLab の **Build** > **Pipelines** を開き、その番号の行を探します。ブランチが `main` の行です。
2. その行の **Stages** のいちばん右の丸(`implement`)を押し、 `nightly_implement` を選んでジョブの画面へ移ります。
3. 黒い領域のログを見ます。1〜2分の準備のあと `Issue #4` の実装を開始します。ブランチ: `nightly/issue-4` が出れば、スクリプトが動き始めています。
4. そのまま待ちます。数分から十数分のあいだログは動きません。画面上部の経過時間が進んでいれば動いています。
5. 終わったらログの末尾を読み、下の表のどの終わり方かを確かめて Step 4 へ進みます。

終わり方	ログの末尾
MR まで届いた	`turns: … / cost_usd: …` のあとに `View merge request for nightly/issue-4:` と URL、最後に `MRを作成しました。マージは人が判断します。`
動いたが差分が無い	`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/` に出ます
始まる前に終わった	`ANTHROPIC_API_KEY: CI/CD変数 ANTHROPIC_API_KEY が未設定です` 。 `ci_logs/` は残りません

1回目で MR まで届くことは多くありません。止まっているほうが普通です。



手順1 と 2 で見る一覧です。`#28` の行が `main` で夜間ジョブを流した例で、右端の丸を押してジョブへ進みます。



手順3 から手順5 で読むジョブのログです。この回は4分ほどで止まっています。

達成状態

- [ ] 講師が流した nightly_implement の行を、チャットの番号で開けた
- [ ] ジョブのログを最後まで読み、上の表のどの終わり方かが分かっている
- [ ] artifacts に ci_logs/ が残っている(環境変数の不足で止まった回だけは残りません)

影響範囲

このジョブには `allow_failure: true` が付いています。止まってもパイプライン全体は赤になりません。**failed** は赤ではなくオレンジの警告アイコンで出て、横に **Allowed to fail** が付きます。どちらの色で終わっても Step 4 へ進みます。

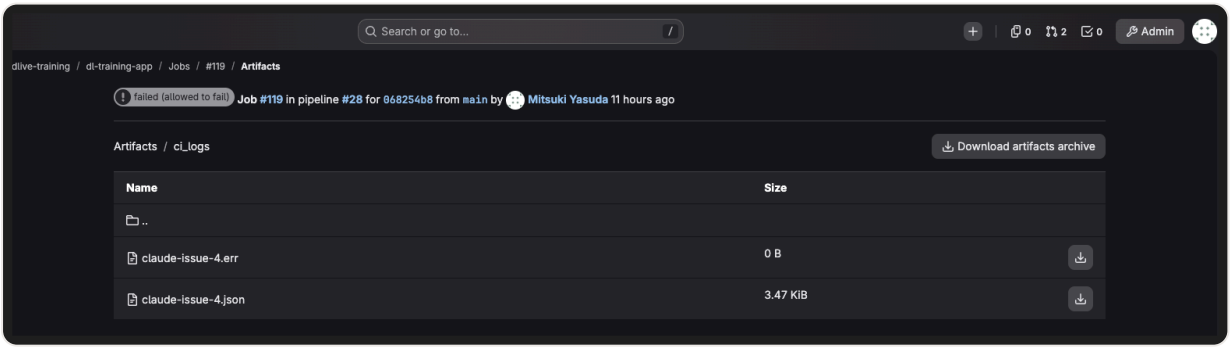
解説と補足

STEP 4　1回目の停止箇所の記録 [15min]

止まった場所を1つ特定して記録します。複数同時に起きていることもありますが、最初に効いた1つだけを選んでください。手を入れる層を1回に1つに絞るためです。

手順

1. ジョブ画面の右の **Job artifacts** の **Browse** を押し、`ci_logs` を開きます。
2. `claude-issue-4.json` を開き、`is_error` と `num_turns` を見ます。`total_cost_usd` は Step 7 で使うので控えます。
3. 同じ `claude-issue-4.json` の `permission_denials` を見ます。権限で拒否された場合は、ツール名とコマンドがここに入ります。`claude-issue-4.err` は CLI 自体のエラーの置き場所で、拒否はここには出ません。
4. 下の表から、自分の止まり方に一番近い行を1つ選びます。
5. VSCode で配布の `計測表_Day1_Day2.md` を開き、「Day2 夜間ループ」の節の上4つの欄(止まった箇所・止まった理由・どの層の設定で直すか・直した内容)を埋めます。下3つは Step 5 のあとに埋めます。



手順1 から手順3 で開く画面です。**Size** でおおよその中身が分かります。

止まり方	ログに出るもの	直す層
環境変数の不足で始まらなかった	ジョブログに `ANTHROPIC_API_KEY` の1行。`ci_logs/` が無い	CI 側(講師の設定)。受講者が直す層はなく、台帳には「環境」と書く

止まり方	ログに出るもの	直す層
Issue を読めずに終わった	`permission_denials` に `curl` か `glab` の拒否	プロンプト。`PROMPT` に `_issues/issue-4.md` を読ませる1行を足す。`curl` は `ci/settings.ci.json` の deny に入っているので、`--allowedTools` に足しても通りません
ブランチ操作で拒否された	`permission_denials` に `git switch` か `git push` の拒否	プロンプト側。SKILL.md の手順3と手順7を飛ばす添え書きが届かなかった場合に出ます
範囲外のファイルまで書き換わった	差分が `app/` と `tests/` の外に及んでいる	プロンプト側。`PROMPT` に触ってよい範囲を1行足す。戻す手間を減らすには、先に範囲を絞るほうが効きます
対象バージョンで動かない構文が入った	ブランチ用のパイプラインの `lint-phpver` が赤。`match` 式や `?->`、7.4 に無い関数の呼び出し	文脈。CI では止まっているので、夜間ジョブの bash 版 `check-phpver` をすり抜けた形です。CLAUDE.md の代替表を厚くする
テストが落ちたまま終わった	`test` または `test-phpunit` が赤	Stop hook。手元では `stop-gate.ps1` が継続させるが、CI の `ci/settings.ci.json` は Stop を登録していない。runner に DB が無く、テストが必ず赤になるため
レビューを受けずに終わった	`ai_review` のジョブが動いていない	CI 側。`ai_review` は `merge_request_event` でだけ動くので、MR ができるまで走らない
指摘を全部受け入れて設計が巻き戻った	差分が Issue の「範囲外」に及んでいる	REVIEW.md。Do not report と Always check の書き方。無人実行では採否を人が挟めない
MR の作成で止まった	コミットと push は通っているが、そのあとに 403	CI 側。`GITLAB_BOT_TOKEN` をプロジェクトアクセストークン(Developer、`api` と `write_repository`)に差し替える
ターン数の上限で打ち切られた	`num_turns` が 15。コミットが途中で終わっている	Issue の粒度。`--max-turns` を上げずに、Issue を割る

達成状態

- [ ] 止まった箇所を1つに絞り、ログの行を根拠として写した

- [ ] 直す層を表から1つ選んだ
- [ ] 計測表の「Day2 夜間ループ」の上4つの欄が埋まっている

影響範囲

ここで選んだ層が、Step 5 で直す1か所になります。2つ以上選ぶと、再実行でどちらが効いたか分からなくなります。

解説と補足

ハンズオン3　再実行とマージ判断

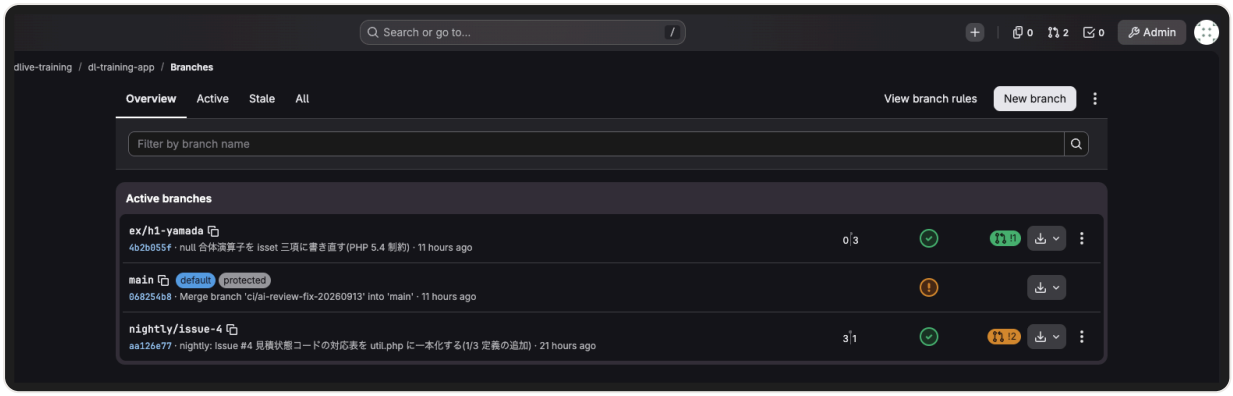
Step 5 から Step 7 です。合わせて [35min]。直す層を1つだけ直して流し直し、立った MR を人が読み、夜間に回せる範囲を1枚にまとめます。

STEP 5　直す層の選択と再実行 [20min]

Step 4 で選んだ層を1つだけ直して、もう一度流します。2回目は `main` ではなく、自分のブランチ `ex/h3-<ユーザー名>` の上で起動します。共有の `main` は変えません。

手順

1. VSCode で直す層のファイルを開き、1か所だけ変更します。
2. ソース管理ペインで、そのファイルだけをステージし、何を直したかが分かるメッセージでコミットします。
3. ソース管理ペインの **ブランチの発行**（2回目以降は **変更の同期**）を押して、自分のブランチを GitLab へ送ります。
4. GitLab の **Build** > **Pipelines** で、ブランチが `ex/h3-<ユーザー名>` の行を開きます。手順3 の push で動いたパイプラインです。
5. lint と test の段が終わると、いちばん右の `implement` の段に `nightly_implement` が再生ボタンの形で出ます。ご自身でこれを押します。Developer の権限でも、保護されていない自分のブランチの手動ジョブは押せます。`main` や他の方のブランチの行は押しません。
6. `nightly_implement` を押してジョブの画面へ移り、2回目のログを開きます。
7. 1回目に止まった箇所を通過していることを確かめます。別の箇所で止まった場合は、それも計測表に1行足します。



Code > **Branches** の画面です。`main` の **protected** が Step 1 で見たブランチ保護です。

達成状態

- [ ] 直した層が1つだけで、その変更が1コミットになっている
- [ ] 2回目の実行で、1回目の停止箇所を通過した
- [ ] 2回目で止まった箇所があれば、それも計測表に書いた

影響範囲

自分のブランチから流すと、そのブランチに入っている `.gitlab-ci.yml` 、`ci/nightly_implement.sh` 、`.claude/settings.json` で走ります。自分の修正が効いたかどうかを、共有の `main` を変えずに確かめられます。

解説と補足

STEP 6 MR の確認とマージ判断 [10min]

自動で立った MR を人が読んで、マージしてよいかを判断します。この演習で人が手を動かすのは、ここだけです。講師の1回目で MR まで届かなかった場合は、自分の2回目の MR で行います。

手順

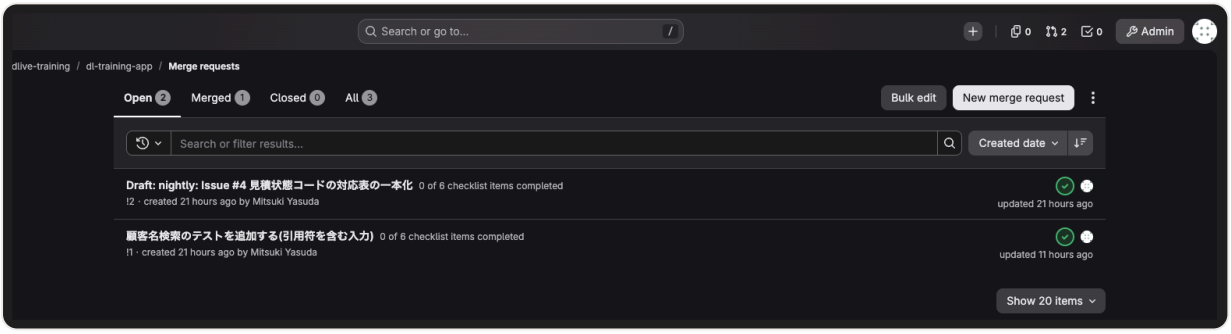
1. ジョブログの末尾に出た MR の URL を開きます。**Code** > **Merge requests** の一覧から探しても同じです。
2. MR の **Pipelines** タブで、`lint-phpdev` 、`lint-phpver` 、`test` 、`test-phpunit` の4つが緑であることを確認します。
3. **Overview** タブに戻り、`ai_review` が置いたコメントを読みます。

4 **Changes** タブで差分を Issue #4 の受入条件と突き合わせます。見るのは3点です。`app` 配下で `'作成中'` が出てくるファイルが `app/libraries/util.php` の1つだけになっているか(受入条件1)、`app/views/estimate_list.php` と `app/views/estimate_detail.php` の両方から `array(1, 2, 3, 4, 9)` が消えているか(受入条件2)、`tests/run_tests.php` にテストが3件増えているか(受入条件4)。`app/views/stock_list.php` にも同じ並びがありますが、在庫状態のプルダウンなので範囲外です。

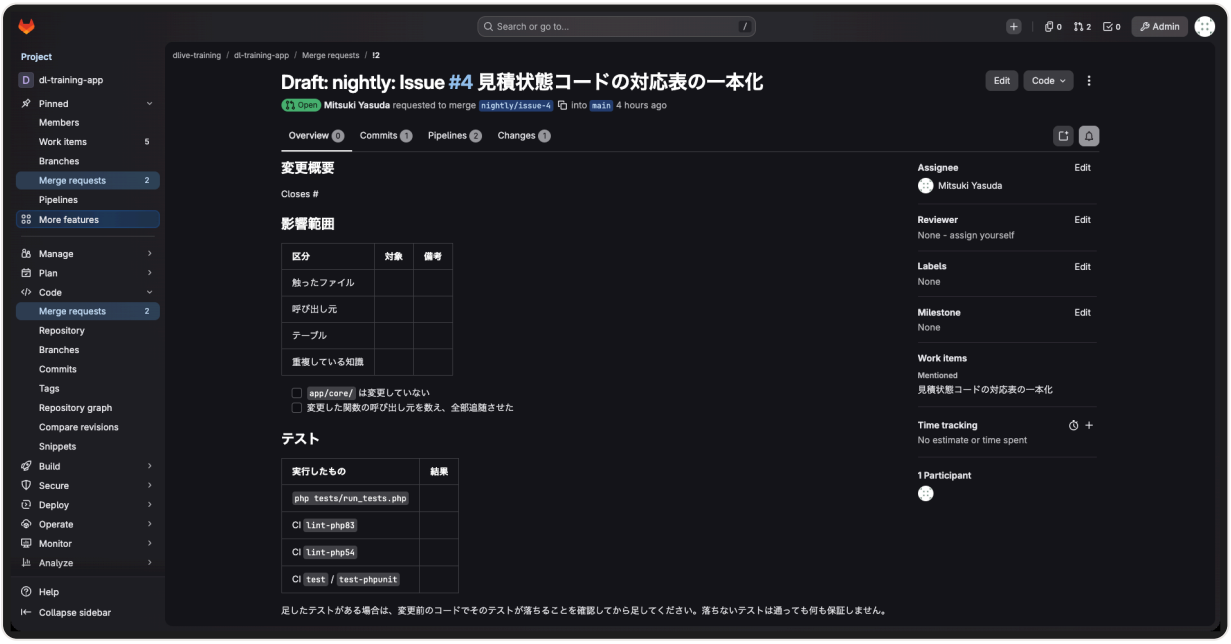
5 同じ **Changes** タブで、差分が `app/` と `tests/` の外に出ていないことを確かめます。出ている場合は、下の折りたたみの手当てを行います。

6 MR の説明に、変更方針3行と影響範囲の調査結果が入っていることを確認します。入っていない場合は、計測表に「人が残る作業」として書き足します。

7 判断の根拠3つを計測表の「Day2 夜間ループ」の行の末尾に書き、講師へ「マージ可」と伝えます。**Merge** のボタンは押しません。



手順1 で一覧から探すときの画面です。右端の丸がその MR のパイプラインの最新の状態です。



手順3 と手順6 で読む画面です。説明欄がテンプレートのまま残った例で、この状態なら手順6 の書き足しに当たります。

マージは講師が最後に1本だけ行います　全員が同じ Issue #4 を流しているので、誰かの MR が `main` に入った時点で、後から流した `nightly_implement` は「変更なし」で終わります。演習の最後に、講師が全員の判断を聞いたうえで1本だけマージし、残りの MR は開いたまま記録として残します。

達成状態

- [ ] MR のパイプラインの4ジョブが緑である
- [ ] 受入条件の3点を自分の目で確かめた
- [ ] 差分が `app/` と `tests/` の中に収まっている
- [ ] マージしてよいかを自分で判断し、根拠3つを計測表に書いて講師へ伝えた

影響範囲

`ai_gate` が赤でもマージボタンは押せる状態のままです。演習のプロジェクトには **Pipelines must succeed** を設定していないので、パイプラインの色にかかわらず **Merge** は押せます。`main` の `.gitlab-ci.yml` には `allow_failure: true` も残っていて、ハンズオン2で外した版は各自のブランチにしかありません。赤いままの MR を「押せるが押さない」と判断できるかどうかも、この Step で見ています。

範囲外の変更が混ざっていたときの戻し方

解説と補足

STEP 7　夜間に回せる範囲の1枚 [5min]

研修の後、社内で最初に説明する相手に見せるのはこの1枚です。

手順

1. VSCode で `計測表_Day1_Day2.md` の「Day2 夜間ループ」の節を開き、Step 4 で空けておいた下の3つの欄(再実行の結果・所要時間・ターン数)を2回目のログで埋めます。所要時間はジョブ画面の **Duration** の分数です。

2. 下の1つ目の表を埋めます。「今夜から回せる」に入れてよいのは、今日の実行で実際に通った工程だけです。

3. 2つ目の表の担当と目安を埋めます。項目は決まっています。埋めるのは、自社で誰がそれを持つかと、その判断に何分かけるかです。

4. 計測表の末尾の「最後に1枚書く欄」の3行に、1行目は1つ目の表の「今夜から回せる」に入れた工程名、2行目は2つ目の表の担当と目安、3行目は Step 4 で「直す層」に挙がったものを効いた順に書きます。

5. VSCode で `持ち帰り台帳.md` を開き、この演習の行を1行足します。「足したもの」の列に schedule の設定値を書きます。

工程	今夜から回せる	止め金として効いたもの
ブランチ作成		
Issue の読み取りと方針の作成		
実装		
対象バージョンの検査		
テスト		
レビューとゲート		
MR 作成		

人が残る作業	担当	1件あたりの目安
Issue の粒度決め		
マージ承認		
指摘の採否		

達成状態

- [ ] 「回せる」に入れた工程が、今日の実行ログで裏付けられている
- [ ] 人が残る3点に担当が入っている
- [ ] 計測表の「Day2 夜間ループ」の7つの欄が全部埋まり、ターン数の欄に turns と cost_usd が併記されている

影響範囲

schedule はファイルでは持ち出せません。`持ち帰り台帳.md` に書いた設定値が唯一の控えになります。

解説と補足

代替 CI が使えないときの進め方

runner から外部の API へ出られない場合は、Step 3 と Step 5 の CI 実走を講師環境で行います。受講者は同じ手順を Claude Desktop で回します。止め金の効き方は同じで、違うのは push と MR 作成を人が行うところだけです。

手順

1. VSCode の左下のブランチ名から **新しいブランチの作成** で `nightly/issue-4-<ユーザー名>` を作ります。CI から流すぶんと同じ名前にすると、push で衝突します。

2. Claude Desktop で `dl-training-app` を開き、パーミッションモードを **自動** にします。承認待ちで止まらなくなります。CI の `--permission-mode dontAsk` とは向きが逆です。`dontAsk` は許可の一覧に無い操作を聞かずに拒否し、**自動** は別のモデルが安全と見た操作を聞かずに通します。どちらのモードでも deny とフックは先に効きます。

3. 入力欄に次の1通をそのまま送ります。`/implement-issue 4` と、CI のスクリプトがプロンプトに添えているものと同じ指示を、1通にまとめた形です。2通に分けると、2通目が届く前に skill が手順3 のブランチ作成まで進むことがあります。

   > `/implement-issue 4` ブランチは既に作ってあります。ブランチの作成と `push` は人が行うので、`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。

4. 終わったら、Claude Desktop の画面で使ったターン数と、拒否されたツールがあればその行を控えます。CI の `ci_logs/` で見ていたものと同じ内容です。

5. VSCode のソース管理ペインで差分を確認し、**ブランチの発行** で GitLab へ送ります。

6. GitLab の **Code** > **Merge requests** で **New merge request** を押し、送ったブランチから `main` へ向けて MR を作ります。ここから先は Step 6 と同じです。

達成状態

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

影響範囲

手元で回すと、CI では動かなかったフックが動きます。`check-phpver` が Edit と Write の後に走り、対象の PHP 7.4 で動かない構文を差し戻します。同じ手順でも、手元と CI で通る範囲が変わることをこの差で確かめてください。Step 4 の表で「CI 側」と書いた行の理由がここにあります。

CLI から CI と同じ引数で流す場合

ハンズオン3　生成物と達成状態

達成状態　自分で判定できる完了の形

ここまで来ていれば完了

- ☐ 講師の1回目と自分の2回目の `nightly_implement` のログを追い、artifacts に `ci_logs/` が残っている
- ☐ 1回目に止まった箇所と、選んだ層と、直し方が台帳に1行で書かれている
- ☐ 再実行でブランチと MR ができ、`lint-phpdev`・`lint-phpver`・`test`・`test-phpunit` の4つが緑になっている(自分の2回目で届かなかった場合は、講師の1回目の MR で同じ4つを見た)
- ☐ Issue #4 の受入条件のうち、機械で判定できる3点を自分の目で確かめた
- ☐ マージしてよいかを人の判断として下し、根拠3つが計測表に残っている。自動マージの設定を入れていない
- ☐ Step 7 の2つの表が埋まり、「回せる」に入れた工程がログで裏付けられている

1回目が止まったまま時間切れになった場合も、止まった箇所の記録と層の選択まで書けていれば到達点に届いています。通すこと自体はこの演習の目的ではありません。

生成物　生成物の名前と場所

ブランチと MR	`nightly/issue-4` と、そこから立った MR `nightly: Issue #4`。GitLab の演習プロジェクトに残ります
実行ログ	`ci_logs/claude-issue-4.json` と `.err`。artifacts に1週間残ります。台帳に写す値は先に取り出してください
schedule	`nightly issue-4`（**Inactive**）。ファイルではなくプロジェクトの設定として残ります
計測表	「Day2 夜間ループ」の節。止まった箇所の行と、Step 7 の2つの表が入ります
直した層のファイル	Step 5 で直した1つ。`ci/nightly_implement.sh`、`.gitlab-ci.yml`、`REVIEW.md`、`.claude/settings.json` のいずれかです

影響範囲 自社ハーネスへの置き場所

持ち帰るのは5点です。D2 は PowerShell を正、smart3pm は bash を正として運用されていると伺っているので、同じファイルでも置き方が変わります。

ファイル	D2 での置き場所	smart3pm での置き場所
`.gitlab-ci.yml` の `nightly_implement` ジョブ	対象リポジトリの `.gitlab-ci.yml` に追記。`ISSUE_ID` の既定値は自社の Issue 番号へ	同じ。イメージに PHP が要らない案件では `before_script` から `php-cli` を外す
`ci/nightly_implement.sh`	`ci/` に置く。手元で流す運用にするなら、同じ引数のまま `.ps1` に写す。コメントと文字列は ASCII に保つ	`ci/` にそのまま。bash が正なので書き換え不要
`.claude/skills/implement-issue/SKILL.md`	`.claude/skills/implement-issue/` 。Issue の取得先を自社のプロジェクトパスへ	同じ場所。手順5の構文制約の節を、対象リポジトリの実行環境に合わせる
止め金の `.claude/settings.json` (deny・hooks・maxEffortLevel)	既存の `settings.json` へ deny を追記。hooks の `command` は `powershell -File` のまま	hooks を exec 形式のまま、`"command": "bash"` と `"args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/<名前>.sh"]` へ差し替える
pipeline schedule の設定(`ISSUE_ID` 、cron、Target branch)	対象プロジェクトの **Build** > **Pipeline schedules** 。ファイルでは持ち出せないので、台帳に設定値を控える	同じ。複数リポジトリに同じ schedule を作る場合は、台帳側に一覧を持つ

CI 側で登録する2つの変数

ハンズオン3　追加と考察

考察　時間が余ったときの追加

1. `ci/nightly_implement.sh` の `--max-turns` を 15 から 5 に下げ、同じ Issue を流します。どの手順で打ち切られたかを `num_turns` と差分から特定してください。打ち切りは失敗ではなく、Issue の粒度の測り方の1つです。

2. `.gitlab-ci.yml` の `GATE_LEVEL` を `P1` から `P2` に変えて MR を作り直し、`ai_gate` が落ちる件数の差を見ます。夜間に回すなら、しきい値をどちらに置くかを台帳に理由つきで書いてください。

3. `PROMPT` に書いた「触ってよい範囲」を `app/` だけに狭め、同じ Issue を流します。テストが書けずに止まるはずです。範囲を絞りすぎると何が起きるかを、自分の目で見てください。

4. 2回の実行の `turns` と `cost_usd` を並べます。1本あたりの費用が見えると、夜間に何本回せるかが決まります。人が残る3点の「1件あたりの目安」と合わせて、1晩の上限を出してみてください。

> **Tips：**しきい値を上げると MR は通りやすくなりますが、朝に読む指摘が増えます。下げると夜のうちに止まりますが、直すのは翌日の人です。どちらにしても作業は消えません。移動するだけだという前提で決めてください。

発展　研修後に手を付ける範囲

1. 予備の Issue #5 を同じ流れで1本回します。1本目と違う止まり方をしたら、それは Issue の書き方の差です。受入条件が機械で判定できる形になっているかを見比べてください。

2. Stop hook を CI 側でも効かせます。`ci/settings.ci.json` は bash 版の PreToolUse と PostToolUse だけを登録していて、Stop は外してあります。runner にテスト用の DB を用意したうえで `stop-gate.sh` を登録する形です。フックの登録を環境で切り替える書き方を、自社ハーネスの `settings.json` に持たせるところまでが課題です。環境ごとの

設計の勘所は、配布物の `50_環境別のフック設計/環境別のフック設計.md` にまとめています。

3. 自社リポジトリで schedule を1晩だけ有効にし、翌朝は MR の説明とテストの結果だけを読みます。

4. フック自身の受入テストを、bash 版にもそろえます。D2 では `hooks/tests/` に回帰テストの仕組みが用意されていると伺っています。同じ形を smart3pm 側にも置くと、二重運用のまま片方だけ検査が抜ける状態を抜けられます。

プチ演習7　自社ハーネスの診断

D2-09 ／ 所要 [15min]

配布した診断シートで自分のハーネスを30項目点検し、5層のうちどこが空いているかを番号で出します。

Day2　**空いている層が番号で出た状態**

目的

自分のハーネスを、印象ではなく確かめた結果で語れるようになります。次に足すものを1つに絞り、研修の翌週に着手できる形にします。

準備

開いておく画面	VSCode。配布フォルダを展開した場所を開いておいてください
いるファイル	`60_理想的なハーネスの雛形/自社ハーネス診断シート.md`。手元に写しを1つ作り、そちらへ書き込みます
点検の対象	1人1式に絞ります。自社のハーネスをお持ちの方はそれを、お持ちでない方は Day1 から今日までに演習リポジトリへ足した設定を対象にしてください
手元にあると早いもの	自社リポジトリの `CLAUDE.md` と `.claude/settings.json`。無くても進められます

判定は3つだけです。**「ある」は、効いていることを確かめた項目にだけ付けてください。**置いた記憶があるだけのものは「一部」です。ハーネスは、効いていないことが画面に出ません。思い込みで「ある」が付くと、そこだけ確認が止まります。

自分で考える　[2min]　シートの表を開く前に、先頭の予想欄を埋めてください。**先に書いてから表を開きます。**順番を逆にすると、表を読んだ印象がそのまま予想になり、あとで差が取れません。

1. 5層のうち、自分のハーネスでいちばん空いていそうな層を1つ書いてください。層は、文脈・権限・フック・CI・リポジトリ保護の5つです。

2. 30項目のうち「ある」が付きそうな数を、数字で書いてください。当てるためではなく、後で差を見るために書きます。

手順

Step 1　30項目を埋める　[9min]

1. 第1層から順に、判定の欄へ「ある」「一部」「無い」のどれかを書いてください。1項目あたり18秒の見当です。迷ったら「一部」に置いて先へ進みます。

2. 「どうやって確かめるか」の欄に書いてある方法で確かめられるものは、その場で確かめてください。設定ファイルの行数を数える、止めたい入力を1つ流す、といった数十秒で済むものだけです。

3. 確かめられなかった項目には、判定の横に印を1つ付けてください。持ち帰って確かめる分です。

> **Tips：**全部を確かめようとすると9分では終わりません。確かめた項目と、確かめていない項目が区別できていれば、この演習は成立します。区別が付いていない表は、埋まっていても使えません。

Step 2　3行にまとめる　[4min]

1. シート末尾の表へ、「無い」が一番多かった層を書いてください。層の名前と、その層で「無い」だった項目の番号を並べます。

2. 先頭に書いた予想と結果を見比べ、食い違った項目の番号を書いてください。**差が出た場所が、いちばん危ない場所です。**守れているつもりだったところだからです。

3. 次の1週間で埋める項目を、**1つだけ**選んで書いてください。番号と、その項目の「次の一手」の欄にあるファイル名を写します。

選ぶのは1つだけにしてください　5つ選んだ方は、1か月後にどれも終わっていません。1つ埋めて、効いていることを確かめてから次を選びます。

達成状態

30項目すべてに判定が入り、末尾の3行が埋まっている状態です。空欄が残っていてよいのは、判定の横に付けた「持ち帰って確かめる」の印だけです。

この演習が終わった状態

- [ ] 表を開く前に予想を書き、後から書き足していない
- [ ] 30項目に判定が入り、確かめた項目と確かめていない項目が区別できる
- [ ] 「無い」が一番多かった層を、番号付きで言える
- [ ] 予想と結果が食い違った項目の番号を書き出した

- [ ] 次の1週間で埋める項目が1つに決まっている

解説と補足

影響範囲

効く範囲	点検そのものは何も止めません。この後のプチ演習8で、ここで見つけた層を1つ書き換えます
判定を間違えると何が起きるか	確かめずに「ある」を付けた項目は、以後の点検から外れます。効いていないまま、守られている扱いで残ります
持ち帰り	埋めた診断シートは、そのまま自社の点検表として使えます。四半期に1回、同じ表を埋めると差分が見えます

自社ハーネスへ持ち帰るときの置き場所は、次の表で決めてください。

成果物	置き場所	次にすること
埋めた診断シート	自社リポジトリの `docs/` 配下。ハーネスと同じ場所に置き、設定を触った人が見に来られるようにする	四半期に1回、同じ表を埋め直す
空いている層の番号	持ち帰り台帳.md	プチ演習8 で、この中から1層を選ぶ
次の1週間で埋める1項目	持ち帰り台帳.md の先頭	埋めたら、その項目の「どうやって確かめるか」を流して「ある」に変える

プチ演習8　雛形の1層の置き換え

D2-09 ／ 所要 [20min]

診断で空いていた層を1つ選び、雛形の該当ファイルを自社の言葉に直して、テストで止まることを確かめます。

Day2　**自社の言葉に直した1層が、テストで止まる状態**

目的

雛形を読むだけで終わらせず、1層を自社の値に置き換えて動かすところまでを経験します。持ち帰った後は、同じ手順を残りの層へ繰り返すだけになります。

準備

開いておく画面	Claude Desktop（Project folder は配布フォルダの `60_理想的なハーネスの雛形` ）、その統合ターミナル、VSCode
いるファイル	`60_理想的なハーネスの雛形/template/` の一式。書き換えるのはこの中の1ファイルだけです
直前の演習の成果物	プチ演習7 で埋めた診断シートと、空いている層の番号
統合ターミナルの開き方	右上の `>_` アイコン、または `Ctrl` とバッククォート。セッションの作業ディレクトリで開きます。止まるかの確認はここで流します

書き換えるのは1層だけです　全部やろうとすると20分では終わりません。終わらなかった層は持ち帰りになり、手が付いていないので着手の記憶も残りません。1層を最後まで通したほうが、翌週に残ります。

自分で考える　[2min]　ファイルを開く前に、メモへ2つ書き出してください。

1. プチ演習7 で「無い」が多かった層のうち、どれを選ぶかを決めてください。迷ったら第3層です。書き換えた結果をテストで見られる層なので、20分で1周できます。

2. 書き換えた後、**何が止まるようになるか**を1行で書いてください。「安全になる」ではなく、「この操作を流すと終了コード2が返る」の形で書きます。この1行が、Step 4 で確かめる対象になります。

手順

Step 1　層とファイルを決める　[2min]

選んだ層に対応するファイルが1つだけあります。この表の右端が、今日触る1ファイルです。

層	自社の言葉に直すもの	触るファイル
第1層 文脈	変更の進め方の表。禁止の行それぞれに、自社で通る代わりの道を書く	`template/AGENTS.md`
第2層 権限	止める一覧と確認を挟む一覧。自社で実際に事故った操作を1件足す	`template/.claude/settings.json`
第3層 フック	触られたら困るパスと、止める操作の定義	`template/.claude/rules.json`
第4層 CI	落とす閾値。重大度と確信度の下限を自社の線に合わせる	`template/ci/gate.sh`
第5層 保護	戻す手順。配る手順と同じ粒度にそろえる	`template/docs/ROLLBACK.md`

> **Tips：**ファイルの冒頭に、そのファイルがどの原則のどの穴を塞ぐものかが1行で書いてあります。書き換える前にその1行を読むと、どこまで変えてよいかの線が分かります。

Step 2　直す前に止まり方を控える　[2min]

Windows の方は、雛形の入口 `guard.ps1` に入力を1件ずつ流して確かめます。Windows の PowerShell では `jq` も `bash` も使えないので、テスト一式を回す `guard.test.sh` は Mac と Git Bash を使える方向けです。

1. 統合ターミナルで、雛形のフォルダ `60_理想的なハーネスの雛形` にいることを確かめてから、止める側の入力を1件流してください。1行目を `Enter` で実行してから2行目を打ちます。

```
'{"tool_name":"Bash","tool_input":{"command":"rm -fr build"}}' | powershell -NoProfile -ExecutionPolicy Bypass -File template\.claude\hooks\guard.ps1
$LASTEXITCODE
```

2. 終了コードが `2` で、その上に `[guard] blocked` と止めた規則の名前が出ることを確認してください。

3. 通す側の入力も1件流し、終了コードが `0` であることを確認してください。

```
'{"tool_name":"Bash","tool_input":{"command":"ls -la"}}' | powershell -NoProfile -ExecutionPolicy Bypass -File template\.claude\hooks\guard.ps1
```

```
$LASTEXITCODE
```

4. 2件の結果を控えてください。これが基準線です。書き換えた後にこの2件の結果が変わったら、直したのではなく壊しています。

Mac と Git Bash を使える方

Step 3　1ファイルを自社の言葉に直す　[10min]

1. VSCode で Step 1 の表で決めたファイルを開いてください。

2. 自社の値へ直す箇所を2つから3つに絞ってください。第3層を選んだ方は、`write_rules` に自社で触られたら困るパスを1件足し、`deny_rules` に自社で実際に事故った操作を1件足します。全体を書き直す必要はありません。

3. 足した規則には、**止まったときに人が読むメッセージ**を必ず書いてください。止めるだけで代わりの道を示さない規則は、回避されます。

4. Claude Desktop に手伝わせても構いません。そのときは、ファイルの冒頭の1行と、直したい箇所だけを渡してください。ファイル全体を書き直させると、雛形が持っている判定の作りごと変わります。

> **Tips：**第3層を選んだ方は、規則を1本足すたびに、止めたい入力と通したい入力を1件ずつ用意してください。Windows の方は Step 4 で両方を流し、Mac と Git Bash の方は `guard.test.sh` へ2件とも足します。通す側を書かないと、全部止める規則を書いても緑になります。

Step 4　止めたいものが止まるか確かめる　[4min]

1. 統合ターミナルで、Step 2 の2件をもう一度流し、終了コードが `2` と `0` のまま変わらないことを確認してください。Mac と Git Bash の方は `bash template/.claude/hooks/guard.test.sh` を流し、件数が Step 2 以上で `NG` の行が出ていないことを見ます。

2. 「自分で考える」の2番に書いた1行を、実際に流して確かめてください。`ここに止めたい操作` を書き換えてから流し、終了コード 2 が返れば効いています。

```
'{"tool_name":"Bash","tool_input":{"command":"ここに止めたい操作"}}' | powersh
ell -NoProfile -ExecutionPolicy Bypass -File template\.claude\hooks\guard.ps1
$LASTEXITCODE
```

3 足した規則で止めたくない操作も1件流し、終了コード 0 が返ることを確認してください。通す側を確かめないと、全部止める規則を書いても気づけません。

4 持ち帰り台帳へ、直した層・触ったファイル・流した入力と終了コードの3つを書いてください。直す前と直した後の両方を書きます。

5 次に直す層を1つだけ書いてください。日付も入れます。

本数の宣言で落ちたとき

達成状態

基準線の2件の結果が変わらず、自分で書いた1行のとおりに止まる状態です。止めたい操作を流したあとの `$LASTEXITCODE` はこう出ます。

```
2
```

Mac と Git Bash で `guard.test.sh` にケースを足した方は、末尾が `guard.test(sh): 47/47 passed` のように、45 に足した件数を加えた数になります。

この演習が終わった状態

- [ ] 書き換えたファイルが1本だけで、他の層に手を付けていない
- [ ] 直す前に、止める側と通す側の2件の終了コードを控えてある
- [ ] 直した後も、その2件の終了コードが変わらない
- [ ] 止めたい操作を手で流し、終了コード 2 が返ることを自分の画面で見た
- [ ] 直した層・ファイル・件数と、次に直す層を台帳に書いた

解説と補足

影響範囲

効く範囲	雛形のフォルダの中だけです。演習リポジトリにも自社リポジトリにも影響しません
壊れると何が壊れるか	`rules.json` の JSON が壊れると、規則がまとめて読めなくなります。画面には何も出ません。テストを流すことが唯一の観測手段です
自社へ写した後	止める規則が増えるほど、誤って止まる操作も増えます。止まったときのメッセージに代わりの道を書いていないと、規則ごと外されます

ファイル	D2 での置き場所	smart3pm での置き場所
直した `template/` の1層	リポジトリ直下へコピーし、既存の `.claude` とは別名で置いてから1層ずつ差し替える	同じ手順で写す。2つのハーネスで規則の書き方を割らず、`rules.json` の形をそろえる
足したテストケース	既存の受入テストへ追記する。件数は README の本数宣言と一緒に直す	同じケースを足す。片方だけに足すと、通る操作が環境で変わる
直した層と次に直す層の記録	持ち帰り台帳.md。日付を入れる	同じ台帳にまとめる。ハーネスごとに台帳を分けない

※本教材の演習に登場する企業・人物・数値はすべて架空です。