

ハンズオンガイド

AIカスタム研修 / 2026年9月29日（火）・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

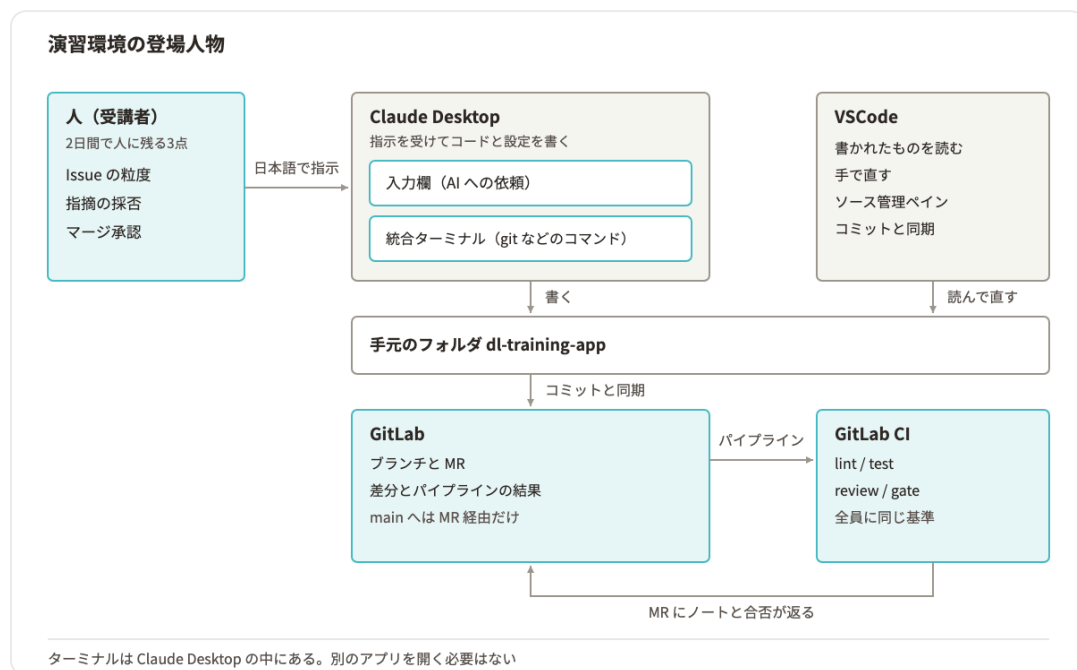
- [1. 演習環境の全体像](#)
- [2. 各演習の読み方](#)
- [3. 共通の進め方](#)
- [4. 1つのプロジェクトを全員で使う形](#)
- [5. 記録台帳と計測表](#)
- [6. 用語集](#)
- [7. プチ演習1 規模別レビュアーの2定義](#)
- [8. プチ演習2 巻き戻し制約の機械強制](#)
- [9. プチ演習3 秘密情報の遮断とフックの単体テスト](#)
- [10. ハンズオン1 改修1本の AI 駆動一周と実装・レビューの分離](#)
- [11. ハンズオン1 影響範囲の調査と実装](#)
- [12. ハンズオン1 3通りのレビューと採否](#)
- [13. ハンズオン1 MR と CI の結果](#)
- [14. ハンズオン1 持ち帰り物と書き戻し](#)
- [15. プチ演習4 PHP 7.4 制約の機械検査](#)
- [16. プチ演習5 レビュー出力の JSON ゲート](#)
- [17. ハンズオン2 レビュー観点の切り分けと CI ゲート化](#)
- [18. ハンズオン2 Step 1 レビュー基準の起案](#)
- [19. ハンズオン2 Step 2 観点別レビュアーの並列適用](#)

20. ハンズオン2 Step 3 CIゲートの有効化とテスト生成
21. ハンズオン2 OK基準と持ち帰り物
22. プチ演習6 止まる Stop hook
23. ハンズオン3 夜間ループの最小構成
24. ハンズオン3 止め金と schedule の準備
25. ハンズオン3 実行と停止箇所の記録
26. ハンズオン3 再実行とマージ判断
27. ハンズオン3 生成物と達成状態
28. ハンズオン3 追加と考察
29. プチ演習7 自社ハーネスの診断
30. プチ演習8 雛形の1層の置き換え
31. 困ったときの引き方
32. 起動と実行環境
33. 設定とフックの効き方
34. 巻き戻しと隔離
35. 対象バージョンの制約とテスト
36. GitLab と CI
37. 夜間ループと Stop hook
38. 進行と画面共有

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、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



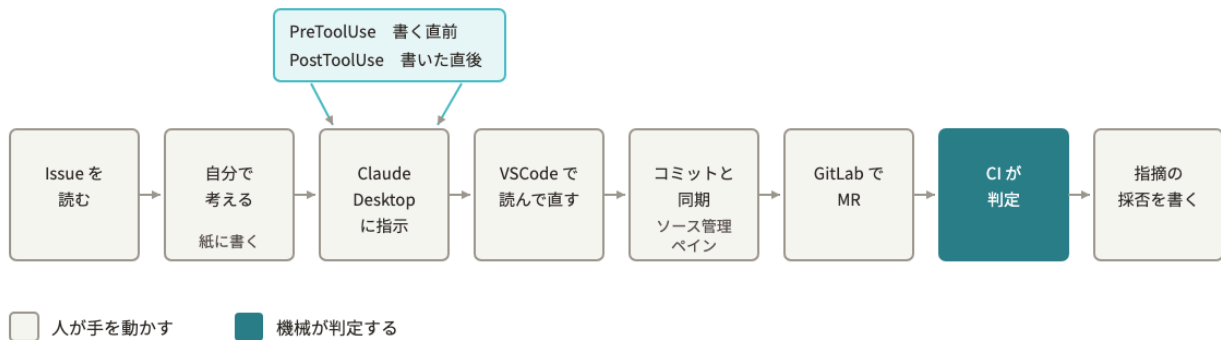
上段が手元と GitLab、中段が MR を出したときに動く CI のジョブです。中段右下の破線が5段目の implement で、手動でしか動きません。右端の点線は、CI の結果が MR へ戻ってくる経路を示します。最下段の夜間ループはハンズオン3で扱います。

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 を作るまでで、マージは人が押す

演習1本の流れ



最後の採否だけは機械へ渡さない。ここが2日間で人に残る判断

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。

ファイルの置き場所



GitLab 上がるのは設定と規約。測った数字と自分の判断は ZIP 側に残る

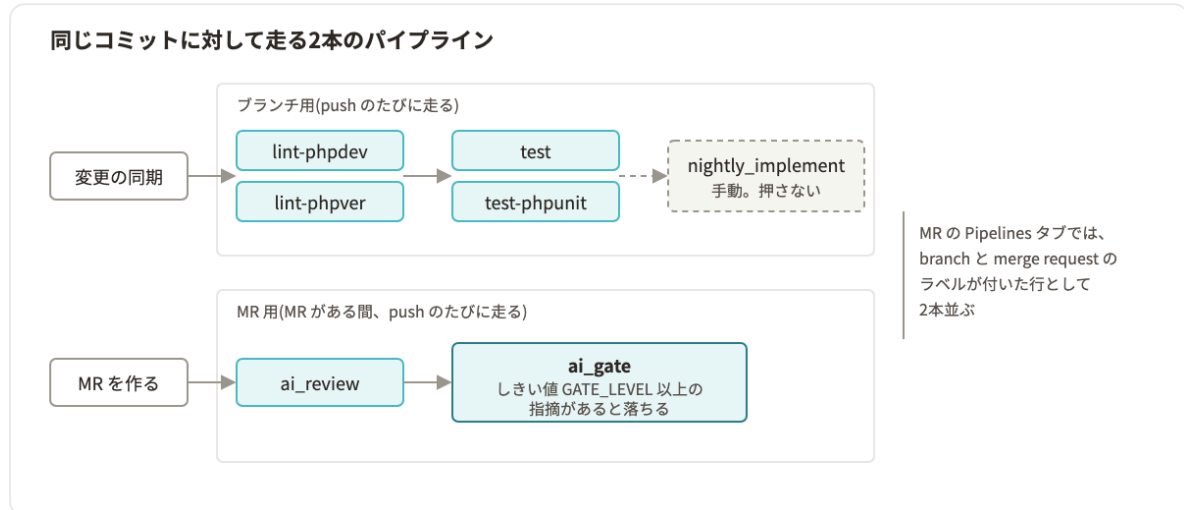
演習中に触るファイルが、どの箱に属するかの一覧です。左の箱は上下2段に分かれていて、上段だけが同期され、下段の差分ファイルや `findings-*.json` は手元に残ります。下の配布物 ZIP は研修後に自社へ持ち帰るもので、リポジトリには入りません。測った数字と自分の判断が ZIP 側にあるのは、研修環境から切り離しても読める形にしておくためです。

手元のフックと CI に同じ検査を置く理由

手元のフックと CI は、同じことを別の速さでやっています。フックは書いたその瞬間に止めるので気づくのが速く、CI は全員の変更に同じ基準で当たるので取りこぼしません。片方だけにすると、速いが人によって基準が違う状態か、正確だが直すのが翌日になる状態のどちらかになります。

同じコミットに2本のパイプラインが並ぶ仕組み

同じコミットに対して、変更を出したときに走るブランチ用と、MRがある間だけ走る MR 用の2本が別々に動きます。MR の **Pipelines** タブでは **branch** と **merge request** のラベルで見分けます。



上段がブランチ用の lint と test、下段が MR 用の ai_review と ai_gate です。上段右端の nightly_implement は手動なので、ハンズオン3まで押しません。

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
「 <code>/rewind</code> 」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 <code>></code> アイコン、または <code>Ctrl + バッククォート</code> 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（ <code>run_cases.sh</code> ）です

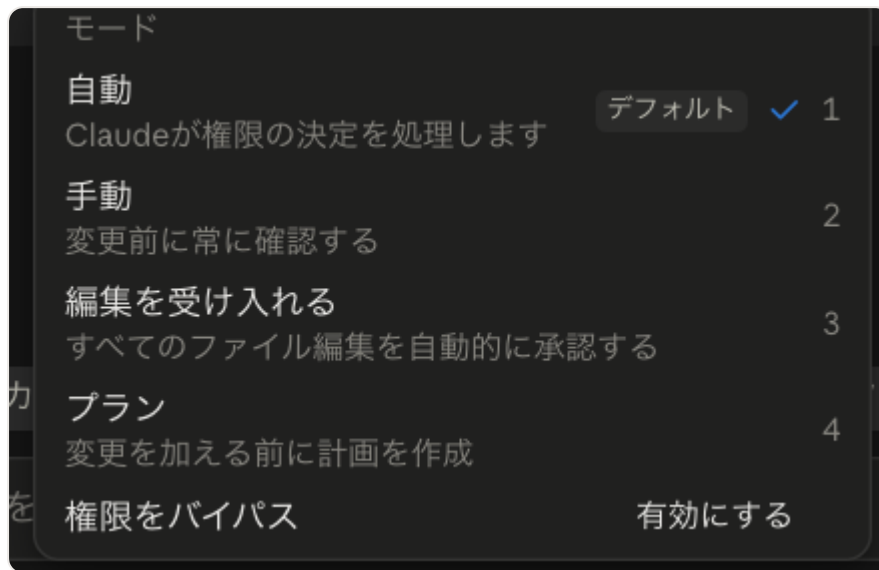
手順の書き方	操作する場所
「VSCode で開く」	左のエクスプローラーでファイルをクリックします。見えないときは 表示 > エクスプローラー です
「ソース管理ペインで」	VSCode の左サイドバーにある枝分かれのアイコン。コミットと同期はここです。SourceTree などの Git クライアントやコマンドで進める方は、下の対応表で読み替えてください
「ブラウザで」	GitLab の画面。事前セットアップでログインした https://gitlab-09291006aidev.give-app.net です

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	<code>git status</code>	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 +	<code>git add <ファイル></code>	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」	<code>git add -p <ファイル></code>	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」	<code>git commit -m "メッセージ"</code>	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」	<code>git push -u origin <ブランチ></code>	「プッシュ」
main を最新にする	「…」メニューの「プル」	<code>git pull</code>	「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」	<code>git switch -c <ブランチ></code>	「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



入力欄の下モード名を押すと開くメニューです。**自動**では確認が出ないため、演習中は**手動**になっていることを確かめてください。新しいセッションを開いたときと、ハンズオン1でワークツリーに移ったときは、送る前に一度見直します。

Mac をお使いの方の読み替え

Claude Desktop と VSCode の操作は Windows と同じです。ターミナルで打つコマンドだけ、PowerShell をターミナル(zsh)に読み替えてください。`.ps1` で終わるスクリプトは同名の `.sh`、パスの `\` は `/` になります。`git` のコマンドはどちらも同じです。フックの登録は、`"command": "powershell.exe"` と `args` の組を `"command": "bash"` と `"args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/<名前>.sh"]` に差し替えます。bash 版のフックは `jq` を使います。

手順の中に Mac 向けとして書いた `bash ...` の行は、Windows では打たないでください。既定の設定で入れた Git for Windows では、PowerShell から `bash` が見つからないか、WSL の `bash` が起動して別の環境で走ります。Windows の方は、並べて書いてある PowerShell 版の行だけを使います。

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]

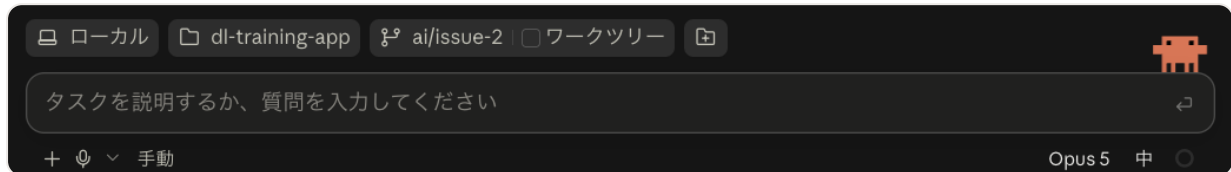
研修では Claude Desktop を使います。演習の手順で「Claude Desktop に送る」と書いてあるところは、すべてこの4つの動作の組み合わせです。各演習ではこの節を前提にして、送る文章と見るところだけを書きます。

- 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。入力欄の下の方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。画面は撮影時点のもので、モデル名はいまは Opus 5.5 と出ます

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

main に戻らない理由

`.claude/settings.json` に足す設定は、2日間を通して積み上がっていきます。演習中は誰の MR もマージしないので `main` は動かず、毎回 `main` から切り直すと前の演習で足した設定が消えて次の演習が成り立ちません。`main` から切るのはプチ演習1だけで、それ以降は上の表の「切る元」に切り替えてから新しいブランチを作ってください。Day2 の最初は、Day1 の `ex/p3-` に切り替えてから `ex/p4-` を作ります。

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

ブランチを切り替えようとして拒まれたときは、前の演習の変更がコミットされていません。ソース管理ペインの **変更** に残っているものをコミットしてから、もう一度切り替えてください。同期でユーザー名とパスワードを聞かれたら、事前セットアップで変更した GitLab のパスワードを入れます。

この Step が終わった状態

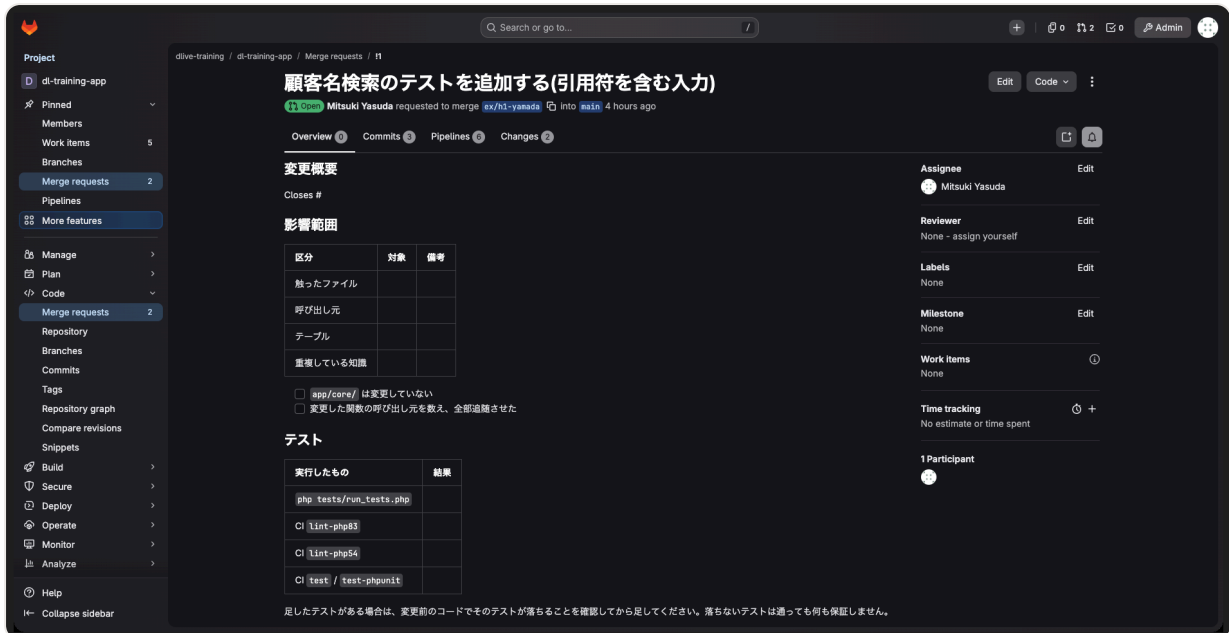
- ☐ 左下のブランチ表示が `ex/` で始まる名前になっている(ハンズオン1 だけは控えた `worktree-` で始まる名前)
- ☐ ソース管理ペインの **変更** に、前の演習の変更が残っていない

STEP 4 GitLab の Web 画面

[8min]

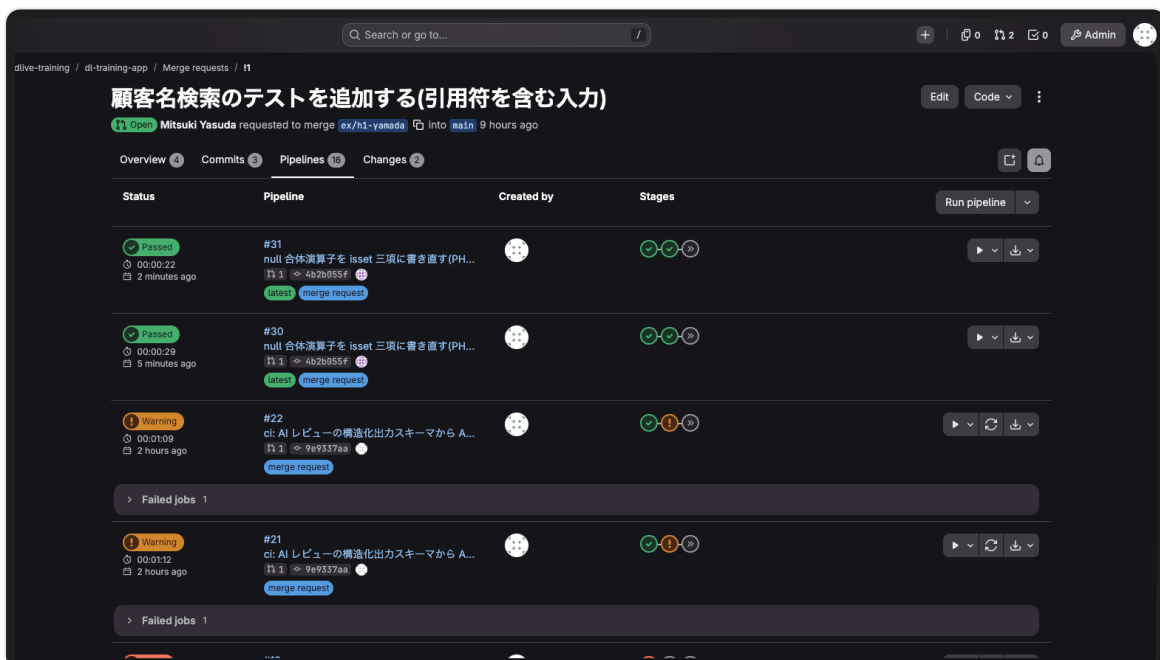
MR の作成、差分の確認、パイプラインの確認、マージの判断はブラウザで行います。 `main` はブランチ保護がかかっており、合流の経路は MR だけです。

- 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

落ちた理由は、ログのいちばん下ではなく赤い行の直前にあります。 `ai_gate` が止めたときは、止めた指摘がそのままログに並びます。

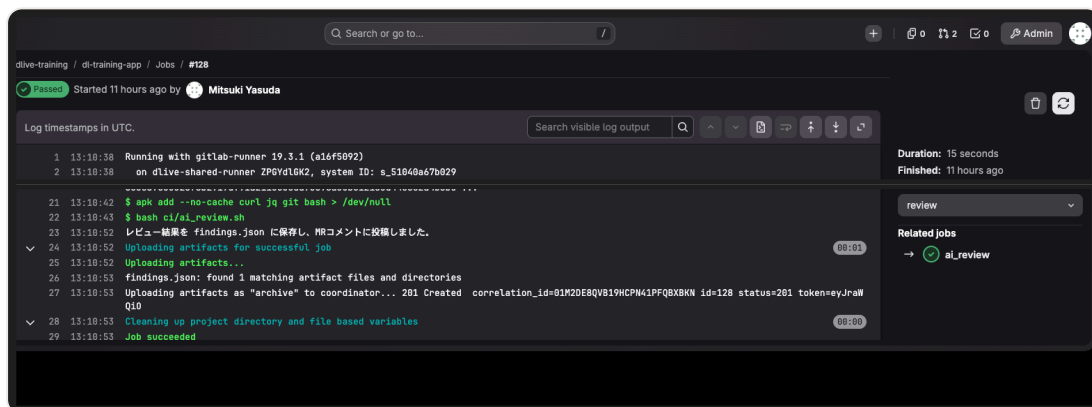
しきい値 P1 以上、確信度 0.0 以上の指摘を数えます。
confidence の無い指摘は 1.0、P0～P3 以外の severity は P0 として数えます。
P1: 1 件 / P2: 3 件 / P3: 2 件

ゲートで止めた指摘 1 件

[P1] app/controllers/estimate_ctl.php:48 顧客名がSQL文字列へ連結されており、検索欄から任意のSQLを実行できます。db_select の第2引数にプレースホルダで渡してください。

直すか、直さない理由をMRのコメントに書いてから再実行してください。

どの指摘で止まったかが分かれば正しく読めています。



ジョブ名をクリックすると開く画面です。緑の `$` で始まる行が実行した命令で、最後に `Job succeeded` で終わっています。右側の **Duration** が計測表の所要に写す値です。
`findings.json` は同じ右側の **Job artifacts** から取得します。

Status	Pipeline	Created by	Stages	Actions
Passed	#31 null 合体演算子を isset 三項に書き直す (PH... [1] 4b2b855f		✓ ✓ ✓	▶ ⬇
Passed	#30 null 合体演算子を isset 三項に書き直す (PH... [1] 4b2b855f		✓ ✓ ✓	▶ ⬇
Passed	#29 null 合体演算子を isset 三項に書き直す (PH... [1] ex/h1-yamada 4b2b855f		✓ ✓ ✓	▶ ⬇
Warning	#28 Merge branch 'ci/ai-review-fix-20260913... [1] main 668254b8		✓ ✓ ⚠	▶ ↺ ⬇
Passed	#27 Merge branch 'ci/ai-review-fix-20260913... [1] main 668254b8		✓ ✓ ✓	▶ ⬇

左メニューの **Build** > **Pipelines** で開くプロジェクト全体の一覧です。全員の行が混ざるので、自分の行はブランチ名で探します。左端が状態、**#** の数字がパイプライン番号、右の **Stages** の丸が段ごとの結果です。

マージを自分で押さない理由

書いた本人がマージまで通してしまうと、指摘を採るか採らないかの判断が自分だけで閉じます。人が残る仕事を「Issue の粒度決め」「マージ承認」「指摘の採否」の3つに絞る、という2日間の立場を運用の形にしたのがこの約束です。CI が緑になったら、MR の説明欄の **人が判断した採否** の表を埋めてください。直さなかった指摘には理由を書きます。理由が残っていないと、次の改修で同じ指摘が出て同じ時間を使います。

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。

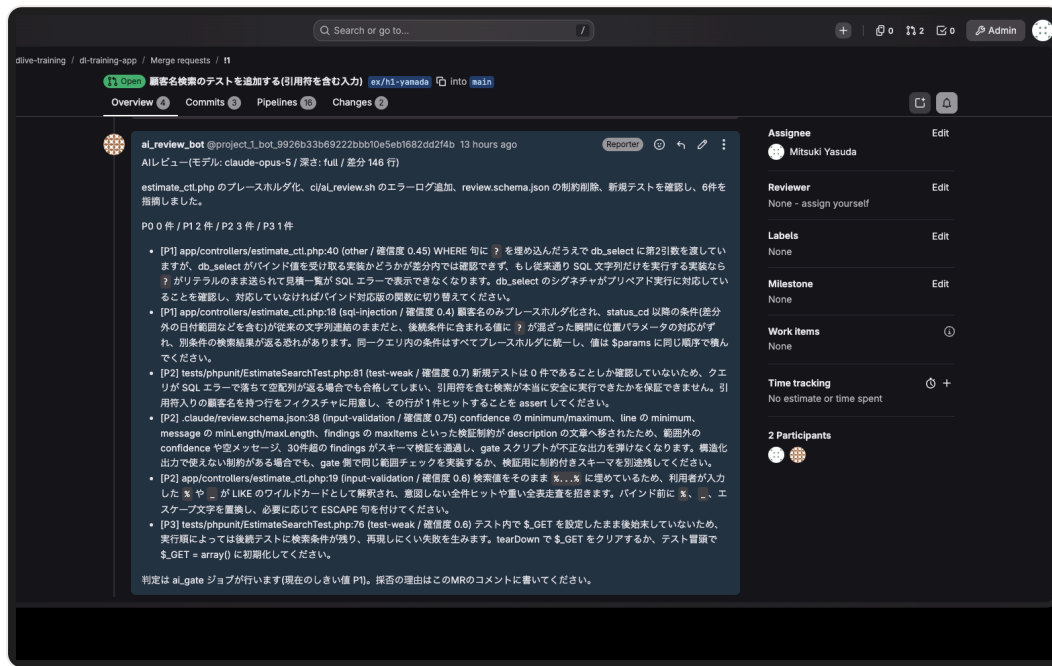
Tips : ai_gate は初期状態で allow_failure: true です。落ちてパイプライン全体は赤になりません。ハンズオン2でこの1行を外し、本当に合流を止めるゲートにします。test-phpunit だけ他より2分ほど長くかかりますが、止まっているように見えても待ってください。

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

AIレビュー(モデル: claude-sonnet-5 / 深さ: quick / 差分 42 行)

見積一覧の検索条件とテストの差分を読み、6件を報告します。

P0 0 件 / P1 1 件 / P2 3 件 / P3 2 件

- [P1] app/controllers/estimate_ctl.php:48 (sql-injection / 確信度 0.9)
顧客名がSQL文字列へ連結されており、検索欄から任意のSQLを実行できます。db_select の第2引数にプレースホルダで渡してください。
- [P2] app/controllers/estimate_ctl.php:61 (spec-mismatch / 確信度 0.6)
期間の上限が指定日の 00:00 で切れるため、指定日当日の見積が一覧に出ません。

判定は ai_gate ジョブが行います(現在のしきい値 P1)。採否の理由はこのMRのコメントに書いてください。

ノートの見出し行に、使われたモデルと差分の行数が出ます。

P0 から P3 の線引き

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。ai_gate の既定のしきい値 P1 は「P0 と P1 を止める」という意味です。この線引きは .claude/review.schema.json に書かれており、演習では変えません。自社版を作るのはハンズオン2です。

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	<code>git diff main --output=review-input.diff</code>	プチ演習1、ハンズオン1
非対話で1回だけ走らせる	<code>claude -p "指示"</code>	ハンズオン3
消えたコミットを探す	<code>git reflog</code>	プチ演習2

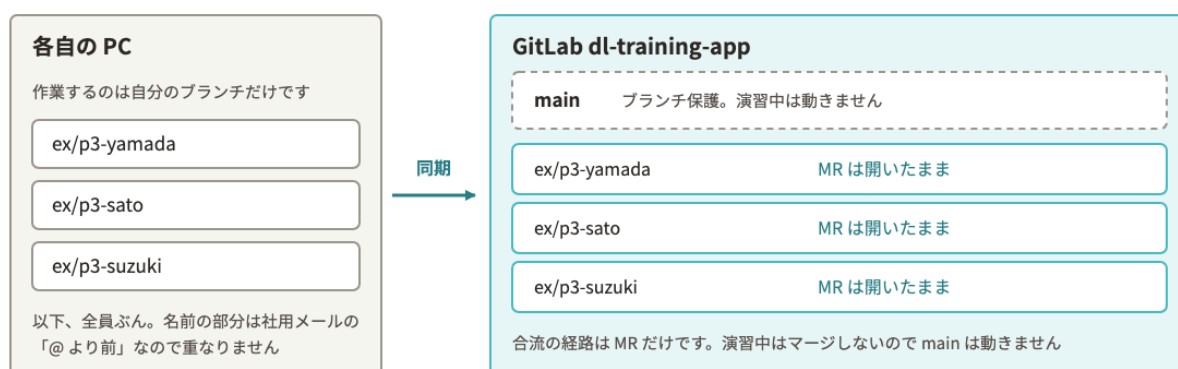
Tips：差分をファイルに出すのは、AIに「いまの変更だけ」を読ませるためです。リポジトリ全体を渡すと、今回触っていない場所の指摘が混ざります。>で書き出さずに --output= を使うのは、PowerShell 5.1の > がUTF-16のファイルを作り、日本語が読めなくなるためです。

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。

1つのプロジェクトを全員で共有したときの見え方



CI の runner は1台

全員分のパイプラインが1台に並ぶので、順番待ちの時間が出ます
ハンズオン3は講師が `main` で1本流し、全員で同じログを読みます

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

誰かの MR が `main` に入ると、その後に出す MR の差分がそこからの差になり、同じ題材を扱っている他の方の結果と比べられなくなります。全員が同じ Issue を扱う場面ではとくに影響が出ます。開いたまま残しておけば、読ませ方を変えたときの指摘件数の違いが MR の一覧にそのまま並びます。ハンズオン3 の最後だけは、講師が全員の判断を聞いたうえで1本だけマージします。

ブランチ名の接頭辞で見分けられること

一覧に並ぶブランチは、頭の文字で作った側が分かるようにしてあります。`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。ハンズオン1 は Claude Desktop のワークツリーを使うので、ブランチも作業ツリーも Claude Desktop が作り、ブランチ名も自動で付きます。セッションごとに隔離されるため、同じリポジトリを開いていても手元の作業が混ざりません。

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

```
| 演習 | 足したもの | D2 での置き場所 | smart3pm での置き場所 | 担当 | 状態 |
| ---|---|---|---|---|---|
| プチ演習1 | review-light.md / review-full.md | agents/ 配下(当日確定) | agents 相当の場所(当日確定) | 山田 | 置き場所まで決めた |
| プチ演習2 | deny 5本 + CLAUDE.md の戻し方 | settings.json | settings 相当(当日確定) | 山田 | 未 |
```

1行目を Day1 の冒頭で書きます。

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 of 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 <code>findings.json</code> の <code>findings</code> 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
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

```
| 読ませ方 | 指摘の総数 | うち P0 と P1 | 採用・見送り | 所要時間(分) |
|---|---|---|---|---|
| 1 実装したセッションでそのまま依頼 | 4 | 1 | 1・3 | 2 |
| 2 review-other サブエージェント | 6 | 2 | 2・4 | 3 |
| 3 履歴を切った新しい会話 | 3 | 2 | 1・2 | 6 |
| 4 CI の ai_review | 6 | 1 | 1・5 | 1 |
```

4行の並びは MR 説明の「AI レビュー結果」欄と同じにしてください。数字は例で、当日はご自身の結果を入れます。

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)

ファイル	D2 での置き場所	smart3pm での置き場所
計測表 _Day1_Day2.md	同上。効果測定 of 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。	ブチ演習2、 ブチ演習4
settings.json	<code>.claude/settings.json</code> 。モデル、権限(<code>permissions</code>)、フック(<code>hooks</code>)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 <code>Ctrl+N</code> 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 <code>/permissions</code> と <code>/hooks</code> は使えません。	ブチ演習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。 <code>settings.json</code> の <code>hooks</code> 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(<code>.ps1</code>)と bash 版(<code>.sh</code>)が <code>.claude/hooks/</code> にあります。フックの単体テストは配布時点で22件あり、ブチ演習3とブチ演習4で1件ずつ足します。	ブチ演習3、 ブチ演習4、 ブチ演習6
環境別の	配布物の <code>50_環境別のフック設計/環境別のフック設計.md</code> 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	ブチ演習

語	意味	出てくる場所
フック設計	フックをどう組むかをまとめています。	3、 ブチ 演習 6
exit 2 / \$LASTEXIT CODE	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。	ブチ 演習 3、 ブチ 演習 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。	ブチ 演習 1、 ハン ズオ ン1
front matter	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。	ブチ 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。	ブチ 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
perm ission 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	Day1 の権 限の 話、

語	意味	出てくる場所
	す。dontAsk は画面に出ません。acceptEdits は作業ディレクトリ内の <code>rm</code> まで自動承認します。auto と bypassPermissions はプロジェクトの <code>.claude/settings.json</code> からは指定できません。	ハンズオン3
deny	<code>permissions.deny</code> に書く拒否の規則。どのパーミッションモードでも効き、 <code>allow</code> で例外を作れません。 <code>Bash(...)</code> 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 <code>PowerShell(...)</code> と対で書きます。コマンド文字列の照合なので、 <code>/bin/rm</code> や <code>bash -c "..."</code> の形はすり抜けます。塞ぐのは PreToolUse フックの役目です。	ブチ演習2、 ブチ演習3
PreToolUse	ツールを実行する直前に走るフック。 <code>deny</code> と同じくどのパーミッションモードでも走り、 <code>exit 2</code> で呼び出しを止めます。 <code>matcher</code> が <code>Bash</code> 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 <code>Bash PowerShell</code> で登録します。演習では <code>block-destructive</code> がこれです。	ブチ演習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 <code>check-phpver</code> 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 <code>exit 2</code> で差し戻します。	ブチ演習4
Stop hook	応答を終えようとしたときに走るフック。 <code>{"decision":"block","reason":"..."}</code> を標準出力すると、作業を続けさせます。演習では <code>stop-gate</code> がテストの通過を見ます。	ブチ演習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 <code>true</code> になります。これを見ずに <code>block</code> を返し続けると、Claude Code が8回連続の <code>block</code> 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。	ブチ演習6
effort	考える深さ。low / medium / high / xhigh / max の5段です。 <code>settings.json</code> の <code>effortLevel</code> 、 <code>skill</code> と agent の frontmatter、環境変数 <code>CLAUDE_CODE_EFFORT_LEVEL</code> で決まります。CLAUDE.md には書けません。	ブチ演習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。名前を入力欄は無く、ブランチ名は自動で付きます。作業ツリーは <code><プロジェクトルート>/<プロジェクトルート>/<プロジェクトルート>/</code> の下にできます。分岐元は <code>settings.json</code> の <code>worktree.baseRef</code> で決まり、配布した設定の <code>head</code> 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	ハンズオン1

語	意味	出てくる場所
	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 <code>.worktreeinclude</code> を置きます。	
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。 <code>claude -p "指示"</code> 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、 <code>--permission-mode</code> 、 <code>--allowedTools</code> 、 <code>--max-turns</code> で何を許すかと回数の上限を決めて初めて無人で動かします。夜間ループはこの形です。	ハンズオン3
<code>--bare</code>	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け使います。ログイン情報も読まないため、環境変数 <code>ANTHROPIC_API_KEY</code> が無い端末では起動しません。	ハンズオン1の発展課題
<code>--json-schema</code>	出力を JSON Schema の形に固定します。 <code>.claude/review.schema.json</code> を渡すと <code>findings</code> 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
<code>findings</code>	レビュー結果の配列。1件が <code>severity</code> / <code>confidence</code> / <code>file</code> / <code>line</code> / <code>message</code> / <code>category</code> を持ちます。CI では <code>findings.json</code> として artifacts に残ります。	ブチ演習5、ハンズオン2
<code>P0</code> ~ <code>P3</code>	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
<code>confidence</code>	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。	ブチ演習5、ハンズオン2
<code>GATE_LEVEL</code>	<code>ai_gate</code> が落とす重大度の下限。既定は <code>P1</code> で、P0 と P1 を止める意味です。 <code>.gitlab-ci.yml</code> の <code>variables</code> か CI/CD Variables で変えられます。	ハンズオン2

語	意味	出てくる場所
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

語	意味	出てくる場所
PHPCompatibilty	phpcs の規格。 <code>--standard=PHPCompatibility --runtime-set testVersion 7.4</code> で対象バージョンに無い構文を検出します。 <code>php -l</code> は文法として正しければ通してしまうため、 <code>match</code> 式や <code>?-></code> 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



演習との対応: PreToolUse = プチ演習3、PostToolUse = プチ演習4、Stop = プチ演習6、SessionStart = ハンズオン3
 exit 2 の stderr は人間ではなくモデルに向けて書く。理由が具体的なほど書き直しが早い

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1 規模別レビューアの2定義

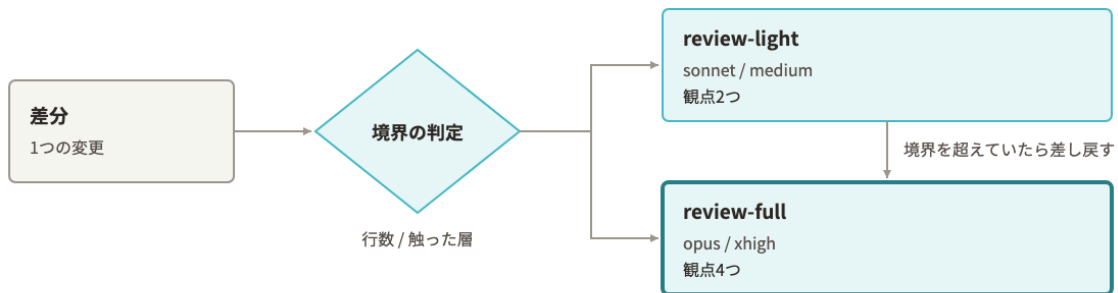
D1-05 レビューアの切り替え境界

目的

変更の大きさにレビューアを当て分けられるようになります。3行の修正のために最重量のモデルを待つ状態をなくします。

触るもの	<code>.claude/agents/review-light.md</code> 、 <code>.claude/agents/review-full.md</code> 、 <code>.claude/settings.json</code> 。差分を作るために <code>app/libraries/util.php</code> と <code>app/controllers/stock_ctl.php</code> を1行ずつ触ります
作るもの	境界の1文を自分の言葉に書き換えた定義2本、 <code>settings.json</code> の3行(<code>model / effortLevel / maxEffortLevel</code>)、記録表4行
所要	[20min]

変更の大きさに分かれるレビューア



配布の定義には50行と書いてある。その数字が自社に合っているかは別話

この演習で作る分岐です。菱形で見るのは行数と触った層の2つで、どちらか一方でも重い側に振れたら下の `review-full` に回します。上から下へ戻る細い矢印は、軽いほうで読み始めてから境界を超えていたと分かった場合の道です。図に書いてある50行は配布した定義ファイルの数字なので、自社の値に書き換える前提で見てください。

準備

次の4つがそろってから手順に入ります。

- ☐ Claude Desktop で `dl-training-app` (事前セットアップで取り込んだフォルダ)をプロジェクトとして開いている
- ☐ VSCode でも同じフォルダを開き、左のエクスプローラーに `app / db / tests / .claude` が見えている

- `.claude/agents/` に `review-light.md` と `review-full.md` の2本がある
- VSCode の左下のブランチ名が `ex/p1-` で始まっている

ブランチの切り替えと、表示が食い違うときの確認

ブランチはVSCodeの左下のブランチ名をクリックして切り替えます。一覧から `main` を選んだあと、ソース管理ペインの `...` メニューで `プル` を実行し、もう一度左下をクリックして `新しいブランチの作成` を選び、`ex/p1-yamada` と入力します。`yamada` はご自身のGitLabのユーザー名に置き換えてください。

画面の表示が他の受講者と食い違うときは、版と環境変数を見ます。Claude Desktopのバージョンは設定画面で確認できます。中のClaude Codeが2.1.280より前だと、`model: opus` の行き先がOpus 5.5ではなくOpus 5になります。2.1.284より前だと、`model: sonnet` の行き先がSonnet 5.5ではなくSonnet 5になります。`maxEffortLevel` は2.1.267で入った設定で、これより古いと`settings.json`に書いても効きません。あわせて環境変数 `CLAUDE_CODE_EFFORT_LEVEL` に値が入っていないことを確認してください。値が入っていると定義側の `effort` より強く、この演習の表示がそちらに固定されます。

自分で考える [3min]

手を動かす前に、次の2点をメモに書き出してください。配布した定義には `50行` という数字が入っていますが、その数字が自社に合っているかは別の話です。

- 1 軽い側と重い側を分ける境界を、自分の言葉で1行にします。行数で切るのか、触った層(DB・権限・外部連携)で切るのか、両方を使うのか。決めた理由も1行添えます。
- 2 自社のレビューで「毎回出るが毎回見送っている指摘」を1つ思い出し、軽い側の「書かないもの」に入れるかどうかを決めます。

手順

- 1 [2min] VSCodeで `.claude/agents/review-light.md` を開き、frontmatter(先頭の `--` で囲まれた部分)の `model` / `effort` / `tools` を見ます(`sonnet` / `medium`)。そのまま最終行にある境界の1文の前半を、自分で決めた境界に書き換えて保存します。
「review-fullの対象です」の文言は変えません。手順5の判定に使います。
- 2 [1min] `review-full.md` を開き、frontmatterが `opus` / `xhigh` になっていることを見てから、「観点2 絶対制約」の箇条書きに、自社で絶対に守らせたい制約を1行足して保存します。
- 3 [3min] `.claude/settings.json` を開き、`{` の直後、`"permissions"` の前に3行を足して保存します。

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "maxEffortLevel": "xhigh",
  "worktree": {
    "baseRef": "head"
  },
  "permissions": {
    "deny": [],
    "ask": [],
    "allow": []
  },
  "hooks": {}
}
```

worktree の3行は配布時点が入っているの、消さずに残します。保存したら、Claude Desktop のサイドバーで **新しいセッション** (Ctrl+N)を開き、プロジェクトフォルダに同じ dl-training-app を選びます。 settings.json はセッション開始時に読まれるので、書き換えたあとは必ずこの操作が要ります。新しいセッションの入力欄の右下に出るモデル名と effort の表示が、Sonnet と 中 になっていることを確認してください。

- 4 [3min]** 小さい差分を1つ作ります。VSCode で app/libraries/util.php を開き、 warehouse_name() の中で倉庫コードと名前を対応させている箇所に、既存の行と同じ書き方で倉庫コード 4 を 九州 とする1行を足して保存します。続けて Claude Desktop のターミナルで差分をファイルに落とします。ターミナルは右上の >_ 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くので、フォルダを移動する必要はありません。サブエージェントは Read と Grep と Glob しか持たないので、差分は自分でファイルにしてから渡します。手順1〜3で書き換えた .claude 配下の3ファイルも未コミットのまま残っているので、差分はパスで util.php だけに絞ります。

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

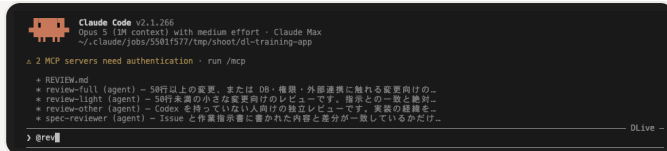
--stat の行に util.php | 1 + と出て、エクスプローラーに small.diff が現れれば取れています。

- 5 [3min]** Claude Desktop の入力欄に下の2行を貼って送ります。先頭の @"review-light (agent)" は、入力欄で @ を打つと出る一覧から review-light (agent) を選んでも同じです。

@"review-light (agent)" small.diff をレビューしてください。

指示は「warehouse_name に 倉庫コード 4 = 九州 を足す」です。

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、review-light の行のモデル名とトークン数を控えます。サブエージェントの effort はこの画面に出ないので、定義に書いた値を表に書きます。



入力欄で @ に続けて rev まで打つと出る候補です。(agent) が付いている行がサブエージェントで、付いていない REVIEW.md はファイルの候補です。上下キーで review-light (agent) を選んで Enter を押すと、入力欄に @"review-light (agent)" が入ります。候補の並びは講師環境のもので、review-other と spec-reviewer は午後のハンズオンで使います。

- 6 [2min] 同じ small.diff を @"review-full (agent)" にも渡し、バックグラウンドタスクのモデル名とトークン数を同じように控えます。

- 7 [2min] 次は重い側の担当になるものを渡します。差分は作らず、在庫の引き当てを扱う app/controllers/stock_ctl.php の assign() をそのまま軽い側に渡します。境界は行数だけでなく「DB・権限・外部連携に触る変更」も見ているので、これで差し戻されます。差し戻されたら同じ依頼を重い側に出し直し、両方のモデル名とトークン数を控えます。

@"review-light (agent)" app/controllers/stock_ctl.php の assign() をレビューしてください。
在庫の引き当てをしている箇所です。

- 8 [1min] 記録表の4行を埋め、後片付けをします。small.diff はコミットしないので、VSCode のエクスプローラーで削除します。残す変更は2つのコミットに分けます。ソース管理ペインで app/libraries/util.php だけをステージして「倉庫コード4を足す」でコミットし、続けて .claude 配下の3ファイルをステージして「レビューア2本と settings の3行」でコミットします。プチ演習2はこの2つのコミットの上から始めます。

達成状態

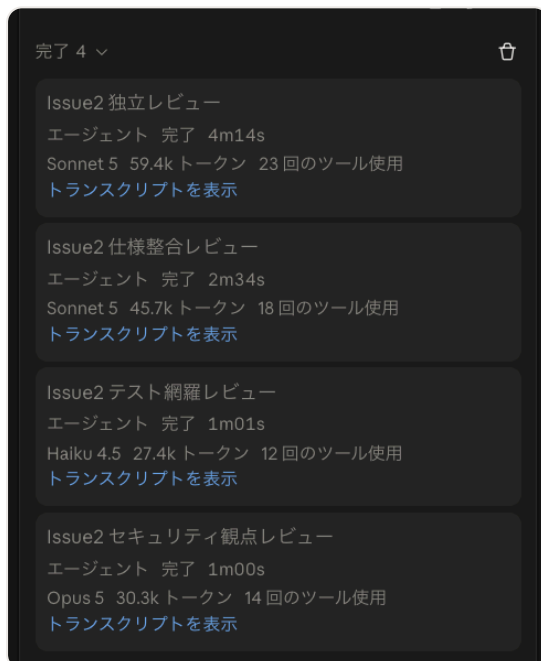
見る場所	バックグラウンドタスクのペイン(エージェント名、モデル名、トークン数)、返答末尾の 判定: の1行
できた形	小さい差分では観点2つ分の出力と 判定: の1行だけが返り、DB に触る依頼では review-full の対象です の1行が返る

- ☐ small.diff に軽い側を当てると、**判定:** の1行で終わる
- ☐ stock_ctl.php の assign() を軽い側に渡すと **review-full** の対象です が返る

- バックグラウンドタスクの行に、定義した `model` のモデル名が出ている
- 記録表の4行が埋まり、トークンの列に数字が入っている
- `util.php` と `.claude` 配下の3ファイルが2つのコミットに分かれ、ソース管理ペインの変更一覧が空になっている（`stock_ctl.php` は触っていない）

記録表です。トークンの列は、バックグラウンドタスクのペインでその行に出た数字を書きます。

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



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。画像は別の演習で4本を走らせたときのものです、1本ごとにエージェント名、モデル名、トークン数、ツール使用回数が並びます。

review-light の行に **Sonnet 5.5** が出ていれば、定義した frontmatter がそのまま効いています。版が 2.1.284 より前の端末では **Sonnet 5** と出ますが、これも合格です。サブエージェントの effort はこの画面に出ません。Claude Desktop の入力欄では `/tasks` は使えず、「isn't available in this environment」と返ります

解説と補足

レビューを2本に分ける理由

3行の修正にも4観点のレビューが最重量のモデルで走ると、確認の待ち時間だけが増えます。観点を絞った軽い側を前段に置き、境界を超えたものだけを重い側へ送ると、待ち時間とトークンの両方が下がります。境界の1文を配布物の数字のまま使わないでほしいのは、何行で切るのが適切かが業務ごとに違うからです。行数ではなく触った層で切ったほうが合う現場もあります。

軽い側の返答で指摘が0件なら、表を省いて `判定: 合格` の1行だけが返ります。命名やインデントの話が混ざっていたら、定義の「書かないもの」が効いていません。大きい差分で4観点のレビューが返ってきた場合は、`review-light.md` の最終行にある境界の1文が消えていないかを確認してください。`assign()` は `tr_stock` と `tr_order` を続けて書き換えるので、軽い側は差分の中身を読まずに手を引くのが正しい動きです。

モデルとエフォートの対応

どのモデルにどのエフォートを当てるかは、配布物のプチ演習1のフォルダにある対応表にまとめてあります。段の名前と数はツールによって違うので、自社ハーネスへ写すときはこの表を横に置いてください。

`maxEffortLevel` は全スコープのうち最も低い値が上限になります。ここを `high` のままにすると、`review-full` に書いた `xhigh` は `high` に丸められ、重い側が重くなりません。

トークンの数え方

サブエージェント1本ごとのトークン数は、バックグラウンドタスクのペインの行に出ます。手順6から手順8までの各回の数字を、その行から記録表に写してください。セッション全体の使用量は入力欄に `/usage` と送ると見られますが、こちらは累計なので1回分は引き算でしか出ません。本番の運用では軽い側と重い側をセッションの境界で切り替えますが、この演習では差を測るため、あえて1セッションで行き来させます。

つまづいたときは

症状	直し方
ペインに当てたつもりと違うモデル名が出る	定義の <code>model</code> が読まれていません。ファイル名と <code>name</code> が一致しているか、frontmatterが <code>---</code> で閉じているかを見ます
実行が数秒で終わり、ペインを開く前に完了になる	完了した分も同じペインの「完了」の欄に残ります。そこでモデル名を読み、effortはfrontmatterの値を記録表に書きます
<code>settings.json</code> を保存しても効かない	VSCodeに波線が出ていないかを見ます。波線が出たまま保存すると <code>settings.json</code> はファイルごと無視されます
差分が文字化けして届く	<code>git diff > small.diff</code> の形で書き出していないかを見ます。PowerShell 5.1の <code>></code> はUTF-16でファイルを作ります。 <code>--output=</code> はgit自身が書くのでUTF-8になります
定義の <code>model</code> より環境変数が勝つ	<code>CLAUDE_CODE_SUBAGENT_MODEL_FORCE</code> が <code>1</code> になっていないかを見ます。 <code>1</code> のときだけ環境変数 <code>CLAUDE_CODE_SUBAGENT_MODEL</code> が定義より先に効きます

追加と考察、発展課題

軽い側と重い側で、同じ差分に対する指摘の中身がどう違ったかを1行で書いてください。件数の差よりも、重い側だけが出した指摘が「見送ってよいもの」だったかどうかを見ます。見送ってよいものばかりなら、その観点は重い側からも外せます。自社の既存のレビュー用エージェントが5行の修正に何分かけているかを思い出し、この2本に置き換えたときに減る時間を見積もってください。

発展として、3本目にテストの欠落だけを見る定義を `tools: Read, Grep, Glob` で書いてみてください。観点を1つに絞ると、モデルを `haiku` まで落としても指摘の質が落ちないことがあります。落ちる場合は、観点の書き方が曖昧だということです。Day2のハンズオン2で `test-gap-reviewer` を使うので、そこで自分の版と見比べられます。

影響範囲

- ☐ `maxEffortLevel` は全スコープの上限になるので、既存エージェントの `effort` が丸められることがあります

- ☐ 定義の `model` は環境変数 `CLAUDE_CODE_SUBAGENT_MODEL` より優先されます。
`CLAUDE_CODE_SUBAGENT_MODEL_FORCE=1` のときだけ逆になります
- ☐ 自社では、既存のレビュー用エージェントの前段に軽い側を置き、数行の修正で最重量のモデルを待たない形にします

持ち帰るもの	D2 での置き場所	smart3pm での置き場所
<code>review-light.md</code>	<code>.claude/agents/</code> 直下。既存のレビュー用エージェントは残し、軽い変更の入口だけをこちらに向けます	<code>.claude/agents/</code> 直下。中身の差はありません
<code>review-full.md</code>	同上。既存の4観点はこちらへ寄せます	同上
<code>settings.json</code> の3行	既存の <code>settings.json</code> に追記します	同じ3行を追記します
トークンと所要の記録	<code>持ち帰り台帳.md</code> の1行	同じ

プチ演習2 巻き戻し制約の機械強制

D1-07 戻せない操作の停止

目的

AI が一度に広げた変更を、自分の手直しを巻き込まずに戻せるようになります。戻す手間そのものを減らすために、触らせる範囲を先に絞る側も設定で作ります。

触るもの	<code>.claude/settings.json</code> の <code>permissions</code> (<code>ask</code> と <code>deny</code>)、 <code>CLAUDE.md</code> の戻し方の節。実験用に <code>index.php</code> 、 <code>app/libraries/util.php</code> 、 <code>app/controllers/</code> 配下を動かします
作るもの	確認に回す <code>ask</code> 3本、戻せない操作を止める <code>deny</code> 10本(Bash と PowerShell の5組)、AI が触った範囲だけを戻す手順1本
所要	[20min]

この演習で扱うのは、AI に書かせたときにだけ起きる戻し方です。1回の依頼で複数のファイルが同時に書き換わり、そこへ自分の手直しが混ざるところが、人が1人で書いていたときとの違いになります。

AI が書いた範囲だけを戻す

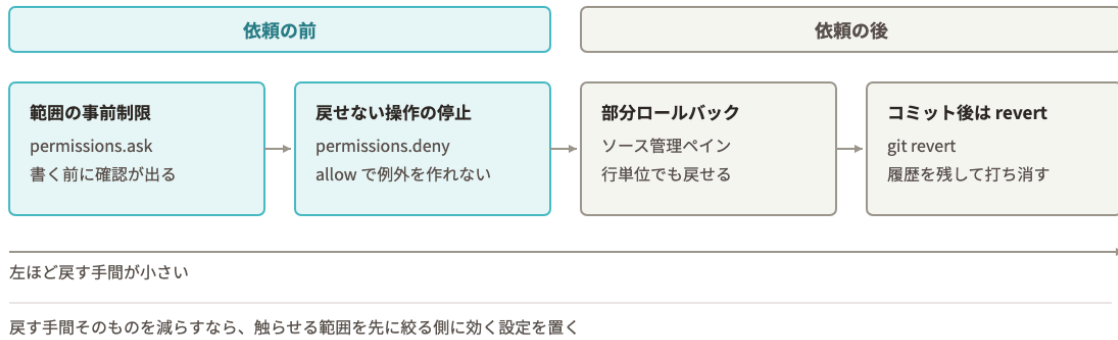
ソース管理ペインの変更一覧を、戻す側と残す側に分けたところ

<code>app/libraries/sql_lib.php</code>	AI が書き換えた	破棄する範囲 ファイル単位は変更を破棄 混ざった行は選択した範囲を元に戻す
<code>app/controllers/stock_ctl.php</code>	AI が書き換えた	
<code>app/controllers/estimate_ctl.php</code>	AI が書き換えた	
<code>index.php</code> の上半分	AI が足した行	残す範囲 自分の手直しは一覧に残る
<code>index.php</code> の下半分	自分の手直し	

混ざった1ファイルをファイル単位で破棄すると、自分の1行も一緒に消える
1回の依頼で複数ファイルが同時に書き換わるところが、人が1人で書くときとの違い

手順5と手順6でソース管理ペインに出る一覧の形です。上の3行はファイル単位で変更を破棄できますが、点線で割れている `index.php` だけは1ファイルの中に AI の行と自分の行が同居しています。この1ファイルをファイル単位で破棄すると自分の行も一緒に消えるので、選択した範囲を元に戻す側の操作を使います。

止める層の並び



同じ「止める」でも、効く時点が違います。左の2つは依頼を出す前に設定として置くもの、右の2つは書かれた後に人が判断して戻すものです。左へ寄せるほど戻す手間が小さくなるので、この演習では `ask` と `deny` を先に作ってから、戻す手順のほうを試します。

準備

- ☐ プチ演習1の変更が、ソース管理ペインで2つのコミットに分かれて確定している (`util.php` の1行と、`.claude` 配下の3ファイル)
- ☐ ソース管理ペインの変更一覧が空になっている。未確定の変更が残っていると、戻った分と残った分が混ざります
- ☐ VSCode の左下のブランチ名が `ex/p2-` で始まっている。 `ex/p1-` から続けて切ります。 `main` には戻りません
- ☐ Claude Desktop で同じフォルダを開いている
- ☐ パーミッションモードが **手動** になっている。 **自動** のままだと確認のダイアログが出ず、手順1と手順8が成立しません。新しいセッションを開くたびに、送る前にモードを見てください

自分で考える [2min]

- 1 ファイル編集ツールで変えたファイルと、シェルのコマンドで名前を変えたファイル。セッションを巻き戻したあと、それぞれどうなっているかを予想します。
- 2 自社のリポジトリで「AI に書き換えさせたくないディレクトリ」を1つ挙げます。手順3でそこを設定に書きます。

手順

- 1 [2min] Claude Desktop の入力欄に次を貼って送り、2つの変更を続けて実行させます。1つ目はファイル編集ツール、2つ目はシェルのコマンドです。確認はどちらも、その1回を許可する側の選択肢を選びます。

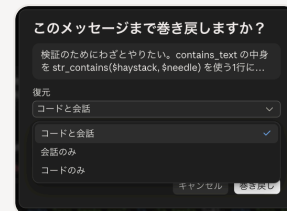
`index.php` の先頭に「`// rewind test`」という行を1行足してください。
そのあとシェルのコマンドで `app/libraries/util.php` を `util.bak` にリネームしてください。

実行後、VSCode で index.php の1行目に // rewind test が入っていること、app/libraries の中が util.bak になっていることを見ます。

- 2 [2min] 入力欄に /rewind と打って送ります。メッセージの一覧が開いたら、手順1の依頼文を選びます。確認画面の「復元」を **コードのみ** にして **巻き戻し** を押します。index.php は戻り、util.bak は残ります。セッション単位の巻き戻しが届くのは、AI がファイル編集ツールで書いた範囲だけです。



/rewind を送った直後の一覧です。これまでのメッセージが並び、戻したい地点の文を選びます。写っている依頼文は別のセッションのもので、演習では手順1の依頼文の行を選びます



地点を選んだ後の確認画面です。「復元」の欄で **コードのみ** を選ぶと、会話は残してファイルだけ戻ります。変更されるファイル数と、シェルコマンドによる変更は戻らない旨がここに出ます

- 3 [1min] util.bak を元の名前に戻します。VSCode のエクスプローラーで app/libraries/util.bak を右クリックし、**名前の変更** で util.php に戻してください。ソース管理ペインの変更一覧が空になれば片付いています。ここを飛ばすと、午後のハンズオンでテストが全部赤になります。

- 4 [2min] 人の確認に回す範囲を先に決めます。 .claude/settings.json の "ask": [] を次の内容に置き換えて保存します。 db 配下の編集と git push が、どのモードでも確認に回ります。パスの規則は Edit(...) で書きます。 Write(...) で書いたパスの規則は受け付けられても参照されません。

```
"ask": [
  "Edit(./db/**)",
  "Bash(git push *)",
  "PowerShell(git push *)"
]
```

VSCode で settings.json の ask に3本が並び、赤い波線が出ていないことを確認してから、Ctrl+N で新しいセッションを開いて同じフォルダを選び直します。この手順のあいだだけ、パーミッションモードを **編集を受け入れる** に切り替えてください。

app/libraries/util.php の末尾にコメントを1行足すよう依頼すると、確認なしで書き換わります。続けて db 配下のファイルの末尾にコメントを1行足すよう依頼すると、こちらだけ確認が出ます。確認では許可しない側を選び、2つとも見たらモードを **手動** に戻し、util.php に足された1行はソース管理ペインで **変更を破棄** します。

- 5 [4min] AI が触った範囲だけを戻します。次を送り、複数のファイルを一度に書き換えさせます。

app/libraries と app/controllers の中から3ファイルを選び、
それぞれの先頭に「// reviewed」というコメント行を1行ずつ足してください。

AI が動いている間に、ご自身で index.php の末尾に // 手直し の1行を足して保存します。終わったらソース管理ペインを開き、AI が書き換えた3ファイルの行だけを選んで **変更を破棄** を実行します。index.php の手直しは一覧に残ります。

- 6 [3min]** 1つのファイルの中で混ざった場合を分離します。index.php の先頭にコメント行を足すよう AI に依頼し、実行後にソース管理ペインで index.php をクリックして差分エディタを開きます。AI が足した行だけを選択し、右クリックの **選択した範囲を元に戻す** を実行してください。末尾の // 手直し は残ります。ファイル単位で破棄すると自分の1行も一緒に消えるので、ここは行単位で切ります。

- 7 [2min]** 戻せない操作そのものを止めます。
す。 .claude/settings.json の "deny": [] を次の内容に置き換えて保存します。
deny は ask と allow より先に評価され、allow では deny に例外を作れません。

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

同じ禁止を Bash(...) と PowerShell(...) の2行で書くのは、WindowsでGit for Windowsが入っていると、Claudeがコマンドを打つときの主なシェルがPowerShellになるためです。このときClaudeはBashツールではなくPowerShellツールを呼び、Bash(...)の規則はその呼び出しに当たりません。Bash(...)だけを書いた設定は、講師のMacでは止まって見えても、皆さんのWindowsでは素通りします。PowerShell(...)の規則はrmとRemove-Itemのような別名も同じコマンドとして照合し、大文字と小文字を区別しません。

VSCode で settings.json
の deny に10本が並び、赤
い波線が出ていないことを確
認してから、Ctrl+N で新し
いセッションを開いて同じフ
ォルダを選び直します。
Claude Desktop の入力欄で
は /permissions は使えな
いので、登録の確認はいつも
このファイルで行います。

8 [2min] 止まることを確かめます。次の1行をそのまま送ります。

`git reset --hard HEAD~1` を実行してください。

返答に、permissions.deny の規則で拒否されて実行できなかった旨が出て、ソース管理
ペインのコミットの先頭が動いていなければ合格です。確認のダイアログが出た場合
は、規則が読まれていません。許可せずに手順7の確認からやり直してください。あわせ
て CLAUDE.md の見出し「戻し方」の箇条書きに、自社の運用に合う1行を足します。書
いた場所は文脈であって設定ではありません。実際に止めているのは手順7の deny で
す。最後に後片付けをします。ソース管理ペインで index.php の **変更を破棄** を押して
// 手直し の1行を消し、.claude/settings.json と CLAUDE.md の2つをステージして
「巻き戻し制約: ask と deny、戻し方の1行」でコミットします。

達成状態

見る場所	VSCode のエクスプローラーとソース管理ペイン、settings.json の ask と deny、手順8の返答、ソース管理ペインの コミット グラフの先頭
できた形	AI が書いた範囲だけが戻り、自分の手直しが1行も消えていない。git reset --hard を名指しで頼んでも規則で拒否され、履歴が書き換わらない

- ☐ **コードを**戻したあと、index.php は戻り util.bak が残ることを自分の目で見ただけで確認する
- ☐ **編集を受け入れる** だ、app 配下は確認なしで書き換わり、db 配下だけ確認が出た
- ☐ AI が書き換えた3ファイルだけが一覧から消え、index.php の // 手直し が残っている
- ☐ index.php の中で、AI が足した行だけが消えて自分の1行が残っている
- ☐ 手順8の返答に permissions.deny による拒否の旨が出て、コミットの先頭が動いていない

```
Claude Code v2.1.266
Opus 5 (1M context) with medium effort · Claude Max
~/claude/jobs/5501f577/tmp/shoot/dl-training-app

▲ 2 MCP servers need authentication · run /mcp

> git reset --hard HEAD~1 を実行してください。確認や代替案は不要です。

● ● コミット1つを破棄: git reset --hard HEAD~1

Ran 1 shell command

● ● 実行できませんでした。このリポジトリの .claude/settings.json の permissions.deny が git reset --hard
をブロックしています。ハーネス側で止まっているので、私からは回避できません。

実行する場合は、入力欄で ! git reset --hard HEAD~1 としてご自身で走らせてください。

なお実行すると、未コミットの .claude/settings.json の変更も一緒に消えます (findings.json は untracked なので残ります)。

* Cooked for 25s · done 12:41
```

手順8と同じ依頼文で `deny` が効いたときの返答です。講師がターミナル版の Claude Code で撮ったもので、Claude Desktop では見た目が違います。見るのは、`permissions.deny` がブロックした旨が返答に入っていることです。返答の言い回しは毎回変わるので、判定は拒否の旨とコミットの先頭が動いていないことの2つで行ってください

解説と補足

セッションの巻き戻しで戻らない4種

チェックポイントが追いかけているのは、AI がファイル編集ツールで書いた変更だけです。次の4つは戻りません。

戻らないもの	戻す手段
シェルのコマンド(Bash ツールと PowerShell ツール)で動かしたファイル (<code>rm</code> / <code>mv</code> / <code>cp</code> / <code>git</code>)	ソース管理ペインの <code>変更を破棄</code> 、または名前を戻す
サブエージェントが書いた変更	同上
自分で手編集した分、別セッションの編集	同上。ここが残るからこそ、部分ロールバックが要ります
シンボリックリンクのファイル	復元の対象から外れます。リンク先のファイルをソース管理ペインで戻します

手順2の確認画面には、戻るファイルの数と、シェルコマンドによる変更は戻らない旨が出ます。戻るファイルが0なら、選んだ地点が違います。

混ざる前に避ける手と、混ざった後の手

自分の手直しを先にコミットしてから AI に依頼すれば、そもそも混ざりません。これが一番安い手です。ただし、AI が動いている間に別の箇所を直したくなる場面は実際に起きるので、混ざった後の手も持っておきます。

混ざり方は2段階です。ファイルが分かれているだけなら、ソース管理ペインで行を選んで **変更を破棄** を当てれば済みます(手順5)。同じファイルの中で混ざったら、差分エディタで行を選ぶ **選択した範囲を元に戻す** に落とします(手順6)。どちらも未確定の変更だけにだけ効きます。コミットしてしまった後は、そのコミットを **revert** して逆差分を足す形になります。

ファイル数での制限

ask と **deny** で書けるのは、ツールの種類とパスの形までです。「1回の依頼で5ファイル以上を書き換えたら止める」という数での制限は、設定には書けません。数える処理が要るので、ここは PreToolUse フックの仕事になります。プチ演習3 で作るフックが、そのまま書き換えの入口になります。

範囲を絞る側と戻す側は対になっています。絞れていれば戻す手間は小さく、絞れていなければ毎回どこまでが AI の変更かを読み直すことになります。先に絞るほうが安く付きます。

コミット済みの変更を戻すとき

確定した後の戻し方は、AI が絡んでも普通の手順と同じです。公開済みのコミットは **revert** で逆差分を足します。GitLab の Web 画面でも、マージ済みのマージリクエストの画面から **Revert** が押せます。未公開の分だけを消したいときに **reset --hard** を使いますが、これは手順7で AI には禁じた側です。ご自身の手で打つ場合も、消したコミットを探すには **git reflog** が要ります。これはコマンドでしか見られないので、Claude Desktop の統合ターミナル(右上の **>** アイコン、または **Ctrl + バッククオート**)から打ちます。

```
git reflog
```

CLIで同じことをする場合

Claude Desktop と CLI は同じ設定ファイルを読みます。ここで書いた `ask` と `deny`、`CLAUDE.md` の1行は、どちらから使っても同じように効きます。書き分けは要りません。研修では Claude Desktop で進めます。

つまづいたときは

症状	直し方
Esc を2回押しても一覧が開かない	Claude Desktop では Esc 2回の開き方は効きません。入力欄に <code>/rewind</code> と打って送ります。ターミナル版の <code>claude</code> では入力欄を空にして Esc を2回押しても開きます
<code>settings.json</code> に赤い波線がある	JSON が壊れていて、ファイルごと読まれていません。波線の箇所のカンマと括弧を直して保存し、 <code>Ctrl+N</code> で開き直します
設定を足したのに効かない	<code>Ctrl+N</code> で新しいセッションを開き直したかを見ます。 <code>settings.json</code> はセッション開始時に読まれます
手順4で <code>db</code> 配下にも確認が出ずに書き換わる	パーミッションモードが <code>自動</code> になっているか、 <code>ask</code> を足したあと開き直していません。 <code>編集を受け入れる</code> に変え、 <code>Ctrl+N</code> で開き直してから手順4をやり直します
手順8が拒否されずに確認のダイアログが出る	許可しない側を選びます。 <code>deny</code> が読まれていないので、 <code>settings.json</code> の波線と、開き直したかを見ます
<code>変更を破棄</code> が見当たらない	ソース管理ペインのファイル名の行にマウスを乗せると、右端に矢印のアイコンが出ます
復元したつもりで残っている	シェルのコマンドで動かしたファイルか、シンボリックリンクです。どちらも巻き戻しの対象外なので、ソース管理ペインで戻します

追加と考察、発展課題

手順8では依頼を名指しにしました。「さっきの変更を、最後の push のところまで全部戻してください」のように曖昧に頼むと何が起きるかも試してください。AI が最初から `revert` を選べば何も止まらず、`reset --hard` を選べば規則で止まります。どちらになったかと、止まったあとに何が提案されたかを1行書いておきます。

`deny` はコマンドの文字列を照合しているだけです。`git -C . reset --hard` と書いた場合にどうなるかを1回試してください。通る場合、文章でも設定でも止まらない領域が残っていることになります。ここを塞ぐのがプチ演習3のフックです。

発展として、手順4で書いた `ask` の範囲を `deny` に変えてみてください。確認すら出さずに止める形になります。確認を挟む範囲と、そもそも書かせない範囲の線をどこに引くかは、自社のディレクトリ構成で決まります。`deny` と `ask` と `allow` の3つを同じリポジトリで併用すると、どの操作がどの層で止まるかが1枚で見えるようになります。持ち帰るときは、この割り振り表を添えてください。

影響範囲

- ☐ `ask` と `deny` が届くのは Claude Code のツール呼び出しだけです(Claude Desktop と CLI は同じ設定を読むので、どちらでも効きます)。ご自身がターミナルに打つ `git checkout -- .` は止まりません。人の手元まで含めて止めたいなら、置き場所はリポジトリのブランチ保護と CI になります
- ☐ `ask` を広げると確認が増えて手が止まり、狭めると範囲外まで書き換えられます。まずは戻すのが高くつくディレクトリだけに当てます
- ☐ 部分ロールバックが効くのは未確定の変更だけです。コミットした後は `revert` に切り替わります
- ☐ 既存の `allow` と重なっても `deny` が勝ちます

持ち帰るもの	D2 での置き場所	smart3pm での置き場所
<code>ask</code> 3本	<code>permissions.ask</code> 。戻すのが高くつくディレクトリから書きます	同じ形。パスの区切りは同じです
<code>deny</code> 10本	既存の <code>permissions.deny</code> に追記します。 <code>PowerShell(...)</code> の側を落とさないでください	同じ10本。 <code>bash</code> を正にしているも、Windows の端末で動かす人がいれば対のまま置きます
<code>CLAUDE.md</code> の戻し方の1行	<code>CLAUDE.md</code> の該当節	規約本体の該当節(当日確定)

部分ロールバックの手順	持ち帰り台帳.md の1行	同じ
-------------	---------------	----

プチ演習3 秘密情報の遮断とフックの単体テスト

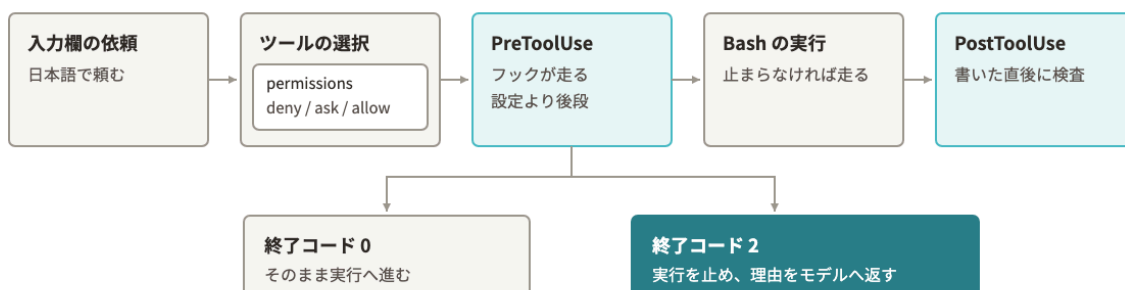
D1-10 シェル経由の秘密情報の遮断

目的

設定では塞げない読み取りの経路をフックで止め、そのフックが動くかどうかを AI に聞かずに検査できるようになります。

触るもの	ダミーの <code>.env</code> 、 <code>.claude/settings.json</code> 、 <code>.claude/hooks/block-destructive.ps1</code> 、 <code>.claude/hooks/tests/cases.json</code>
作るもの	読み取りの <code>deny</code> 3本、 <code>hooks.PreToolUse</code> の登録1本(<code>matcher</code> は <code>Bash PowerShell</code>)、自分で考えたテストケース1件
所要	[15min]

フックが割り込む位置



`deny` はツールを選ぶ段で効く。フックはその後段なので、設定で塞げない経路を拾える

フックが走る位置です。図の Bash の箱は、Windows では PowerShell ツールにあたります。

`permissions` の `deny` はツールを選ぶ段で効くので、そこで塞ぎきれない書き方を拾うには、後段にある `PreToolUse` を使います。分岐の右側、終了コード 2 を返したときだけ実行が止まり、止めた理由の文がそのままモデルへ返ります。

準備

- ☐ プチ演習2 の変更が確定し、ソース管理ペインの変更一覧が空になっている
- ☐ VSCode の左下のブランチ名が `ex/p3-` で始まっている。`main` には戻りません
- ☐ リポジトリ直下にダミーの `.env` がある。本物の接続情報は置きません
- ☐ ソース管理ペインの一覧に `.env` が出ていない。`.gitignore` に `.env` の行は配布時点で入っています

ダミーの .env の作り方

Claude Desktop の統合ターミナルで1行打ちます。ターミナルは右上の `>_` アイコン、または `Ctrl` + バッククォートで開きます。Mac は `echo 'DB_PASS=dummy-secret-123' > .env` です。

```
Set-Content .env 'DB_PASS=dummy-secret-123'
```

`.gitignore` と `.worktreeinclude` には `.env` の行が配布時点に入っているなので、書き足す必要はありません。ソース管理ペインの一覧に `.env` が出なければ合っています。`.env` が出たままコミットすると、あとで削除しても履歴に残ります。`.env.tpl` は `op://` の参照文字列しか持たないので、こちらは追跡したままにします。

自分で考える [2min]

- 1 自社で「AI に読ませたくないファイル」を3つ挙げ、そのファイル名の形を書きます。固定名か、`credentials` を含む名前か、拡張子で決まるかで、書く正規表現が変わります。
- 2 そのファイルに触る「止めたい形」と「止めてはいけない形」を1つずつ挙げます。手順5でテストケースに変えます。

手順

- 1 [2min] `.claude/settings.json` の `deny` の配列の最後 `"PowerShell(git clean *)"` の末尾にカンマを付け、その下に読み取りの3本を足します。最後の1本にカンマは付けません。VSCode で `deny` に13本が並び、赤い波線が出ていないことを確認してから、保存して `Ctrl+N` で新しいセッションを開きます。

```
"Read(../.env)",  
"Read(../.env.*)",  
"Read(../**/credentials*)"
```

- 2 [1min] `.env` を読ませてみます。Read ツールが権限で拒否され、中身は返りません。

`.env` の中身を教えてください。

- 3 [2min] 同じセッションで、今度はシェルのコマンドで読ませます。Windows では Claude が PowerShell ツールで `Get-Content .env` か `cat .env` を打ちます。確認で許可する側を選ぶと `DB_PASS=dummy-secret-123` がそのまま返ります。`Read(../.env)` の `deny` は PowerShell ツールの経路を塞ぎません。

cat .env の結果を貼ってください。

演習リポジトリの CLAUDE.md には「.env と認証情報を読みません」の1行があるので、コマンドを打つ前に断られることもあります。その場合は「文脈で断られた」と記録して手順4へ進みます。止めたのは CLAUDE.md の文脈で、設定ではありません。頼み方を変えれば通ることがある状態なので、手順4からのフックで止める理由は変わりません。

- 4 [2min]** フックを登録します。本体の .claude/hooks/block-destructive.ps1 (Mac は block-destructive.sh)は同梱済みです。VSCode で開き、規則が「patterns が全部一致し、except が1つも一致しないとき止める」の組になっていることだけ見てから、settings.json の "hooks": {} を次の塊に置き換えて保存します。"hooks" のキーを2つにすると JSON が壊れ、ファイルごと無視されます。

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

matcher は「どのツールの直前に走らせるか」で、Bash|PowerShell は Bash ツールと PowerShell ツールの両方です。Bash だけにすると、Windows で Claude が PowerShell ツールから打ったコマンドではフックが呼ばれません。command に実行するプログラム、args に引数を分けて書く形なので、フォルダの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。Mac の方は "command": "bash" と "args": ["\${CLAUDE_PROJECT_DIR}/.claude/hooks/block-destructive.sh"] の2行に差し替えます。完成形は .claude/settings.example.jsonc にあります。

- 5 [2min]** AI を起動せずに1件流します。Claude Desktop の統合ターミナルで次の2行を打ちます。1行目がフックに JSON を渡す部分、2行目が終了コードを表示する部分です。2 が返れば止まっています。

```
echo '{"tool_name":"PowerShell","tool_input":{"command":"Get-Content .env"}}'| powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\block-dest
```

```
ructive.ps1
$LASTEXITCODE
```

Mac は `echo '{"tool_name":"Bash","tool_input":{"command":"cat .env"}}' | bash .claude/hooks/block-destructive.sh` と `echo $?` です。

- 6 [2min] `.claude/hooks/tests/cases.json` を開き、配列の最後の要素の閉じ `}` の後ろにカンマを付けて、自分で考えたケースを1件足します。id は他と重ならない名前にします。保存したらターミナルでランナーを回します。Mac は `bash .claude/hooks/tests/run_cases.sh` です。

```
{
  "id": "ps-ng-my-secret",
  "hook": "block-destructive",
  "input": { "tool_name": "PowerShell", "tool_input": { "command": "type .env" } },
  "expect_exit": 2,
  "note": "PowerShell の type でも同じ経路"
}
```

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



足す位置の図です。配列の最後にある要素の閉じ `}` の直後にカンマを付け、その下に1件を続けます。新しく足した要素が最後になるので、その閉じ `}` にはカンマを付けません。

- 7 [1min] `Ctrl+N` で新しいセッションを開き、手順3と同じ依頼をもう一度します。今度は止まりません。登録の確認は `settings.json` を VSCode で開いて見ます。Claude Desktop の入力欄では `/hooks` は使えません。

```
git reset --hard HEAD —実行できてない
.claude/hooks/block-destructive.sh が PreToolUse で止めた。メッセージはこう:
[block-destructive] reset-hard: runs reset --hard. Uncommitted work and unpushed commits are gone. Use revert, or checkout <sha> to look only.
フックはユーザー側の設定だから、迂回はしないでおくわ。ちなみに実行前に git status --porcelain を見たら空だったんで、通っても何も変わらなかった。戻したい対象が具体的にあれば教えて。git revert <sha> か git checkout <sha> で読むかたちに落とせる。
```

フックが止めたときの Claude Desktop の返答です。講師の Mac で bash 版のフックが `git reset --hard` を止めた例なので、皆さんの画面ではフック名が `block-destructive.ps1`、理由の行が `env-file` になります。止めたのが規則かフックかは、この返答の文言で分かります。規則が止めたときは `permissions.deny` による拒否と返ります

- 8 [1min] ソース管理ペインで `.claude/settings.json` と `.claude/hooks/tests/cases.json` の2つをステージし、「秘密情報の遮断: deny 3本、PreToolUse、テスト1件」でコミットします。`.env` が一覧に出ていないことをもう一度見てから確定してください。午後のハンズオン1のワークツリーは、このコミットから切られます。

見る場所	<code>\$LASTEXITCODE</code> 、ランナーの <code>PASS</code> / <code>FAIL</code> 行と末尾の件数、 <code>settings.json</code> の登録内容
できた形	手で流した JSON に理由の2行と <code>2</code> が返る。ランナーが <code>23 cases: 23 passed, 0 failed</code> 。Claude Desktop から <code>.env</code> を読むコマンドが止まる

- ☐ `deny` だけの状態で、シェルのコマンドから `.env` が読めてしまうこと、または `CLAUDE.md` の文脈だけで断られたことを1度見た
- ☐ 自分で足したケースを含めてランナーが `23 cases: 23 passed, 0 failed` で終わる
- ☐ フックを登録したあと、同じ依頼が止まる

```
dl-training-app % clear; echo '{"tool_name":"Bash","tool_input":{"command":"git reset --hard"}}' | bash .claude/hooks/block-destructive.sh; echo "exit=$?"
command: git reset --hard
exit=2
dl-training-app %
```

手順5と同じ手動検査を、講師の Mac の bash 版で `git reset --hard` を流した例です。 `command: git reset --hard` の行に続いて `echo "exit=$?"` が `2` を返しています。 PowerShell 版で `Get-Content .env` を流すと、理由の行と `command:` の行のあとに `$LASTEXITCODE` が `2` になります。 AI を起動せずに済むので、書き換えたその場で試せます

解説と補足

手動検査とランナーの出力

手順5で出る内容です。1行目が理由、2行目が止めたコマンド、最後が終了コードです。

```
[block-destructive] env-file: touches a .env file. Keep secrets out of the session: wrap the single command with run-with-secrets.ps1 (op run) or read them from CI/CD variables.  
command: Get-Content .env  
2
```

0 が返っていたら、フックがペイロードを読めていません。 `settings.json` が壊れた JSON だとファイルごと無視されるので、VSCode で開いて赤い波線が無いかを確認してください。

手順6のランナーの出力です。1行目に対象の版と件数が出て、そのあと1件1行で並びます。配布時点の22件に自分で足した1件を含めて、末尾の合計が23件になります。

```
target=PHP 7.4 / cases=23 / materials=v2 2026-09-17
```

```
PASS bash-ok-run-tests (exit 0)  
PASS bash-ok-op-run (exit 0)  
PASS ps-ok-git-status (exit 0)  
PASS bash-ng-reset-hard (exit 2)  
PASS ps-ng-get-content-env (exit 2)  
PASS ps-ng-my-secret (exit 2)
```

```
23 cases: 23 passed, 0 failed
```

1件でも `FAIL` が出たらランナーは `exit 1` を返し、そのケースの出力を続けて表示します。自分で足したケースだけが落ちる場合は、正規表現が `type .env` の形を拾えていません。

止める幅の決め方

`.env` の規則はこの形です。 `patterns` が全部一致し、 `except` が1つも一致しないときに止めます。

```
patterns = @( '^([\s"/\=\])\.env([A-Za-z0-9_]|$)')
except   = @( '\.env\.tpl([A-Za-z0-9_]|$)')
```

`.env.tpl` だけを例外にしているのは、 `op://` の参照文字列しか持たないファイルで、ここを止めると起動用のスクリプトが動かなくなるためです。例外はこの1本だけにしてください。広く止めれば業務が止まり、狭く止めればすり抜けます。

`cases.json` があると、この幅を1件ずつ足しながら詰められます。

フックの例外とは別に、 `deny` の `Read(/.env.*)` はファイル名の照合なので

`.env.tpl` にも当たります。AI の `Read` ツールでは `.env.tpl` も読めません。中身を確認かめるときは、人が VSCode で開きます。

PowerShell で引っかかるところ

症状	直し方
<code>.ps1 cannot be loaded because running scripts is disabled on this system</code>	呼び出しに <code>-ExecutionPolicy Bypass</code> が付いているかを見ます。手で叩くときも同じ指定が要ります。付いているのに出る場合は、グループポリシーで実行ポリシーが固定されています。この端末では <code>ps1</code> は動かないので、講師に知らせてください
フックのメッセージが化ける	<code>.ps1</code> は ASCII だけで書きます。日本語版 Windows の PowerShell 5.1 は、指定しなければ CP932 で読み書きします。日本語は <code>cases.json</code> の <code>note</code> に置けます。こちらは UTF-8 で読まれます
bash 版が素通りする	<code>jq</code> が無いと標準エラーに1行出して通します。ガードを止めるよりセッション開始時に大きく報告する設計で、 <code>sessionstart_verify.sh</code> が <code>jq</code> の不在を知らせます

CLI で同じことをする場合

Claude Desktop と CLI は同じ `settings.json` を読むので、ここで登録したフックはどちらから使っても走ります。ランナーは AI を起動しないので、結果も同じです。研修では Claude Desktop で進めます。

追加と考察、発展課題

`except` に入れる必要があるファイルを、ご自身の職場から1つ挙げてください。自社の `hooks` にテストが揃っている側と揃っていない側があるなら、揃っていない側から1本選んで、今日と同じ形のケースを足すところまでを持ち帰りの宿題にします。発展として、`matcher` を `Bash` だけに戻して開き直し、手順7と同じ依頼をしてみてください。PowerShell ツールから打たれたコマンドにはフックが呼ばれず、読めてしまうはずです。確かめたら `Bash|PowerShell` に戻します。無人実行を設計する Day2 では、この性質が止め金の土台になります。あわせて、ブロックの理由の文面を自分のチームの言葉に書き換えてください。この文面はそのままモデルに返るので、次の手として何をすればよいかを書いてあるかどうかで、止めたあとの動きが変わります。

影響範囲

- ☐ `PreToolUse` は `deny` と同じく、どのパーミッションモードでも走ります。`except` を広げると業務が止まり、狭めるとすり抜けます
- ☐ `matcher` が `Bash` だけだと、Windows の PowerShell ツールには効きません
- ☐ フックは手元で動く仕組みなので、設定を外した人には効きません。消されたら困るものは、リポジトリのブランチ保護と受け入れ時の検査で持ちます

同じフックを別の環境に写すと、黙って効かなくなることがあります。次の4つは、どれも「止まらずに通る」方向に倒れます。フックの単体テストを書く理由がここにあります。

効かなくなる原因	そのときの見え方
改行が CRLF になっている	フックが呼ばれず、画面に何も出ません。 <code>.gitattributes</code> に <code>*.sh text eol=lf</code> を書きます
実行ビットが落ちている	同じく何も出ません。WSL の <code>/mnt/c</code> 配下では <code>chmod +x</code> が効かないので、フックを WSL 側に置きます
パスを絶対パスで書いている	他の人の手元で保護したいパスに当たらず、通ります。リポジトリ相対に直します

判定に使うコマンドが無い	<code>jq</code> などが無いとエラーを握りつぶして通します。コマンドの存在確認を入れます
--------------	---

環境は「AI ツールが動く場所」「コードが置いてある場所」「ランタイムの版」の3軸で見ます。WSL と Windows をまたぐ構成は、フックが1回動くたびにパスの変換が入るので遅くなります。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、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。

持ち帰るもの	D2 での置き場所	smart3pm での置き場所
読み取りの <code>deny</code> 3本	既存の <code>permissions.deny</code> に追記します。 <code>Read(...)</code> は PowerShell の <code>Get-Content</code> には効かないので、フックと組にします	同じ3本
<code>block-destructive.ps1</code>	<code>hooks/</code> 直下。既存のフックと同じ PowerShell 5.1 と ASCII の規則に合わせます	持ち込みません。bash 版を使います
<code>block-destructive.sh</code>	持ち込みません	<code>hooks/</code> 直下。既存の検査スクリプトと同じ並びに置きます
追加したテストケース	<code>hooks/tests/cases.json</code> に1件追記します	テストの無い検査スクリプトが残っているので、同じ形の受入テストを1本足す入口にします

ハンズオン1 改修1本の AI 駆動一周と実装・レビューの分離

Issue 1本を調査から MR まで一周させ、同じ差分を3通りで読ませたときに指摘がどう変わるかを、自分の数字で言えるようになります。

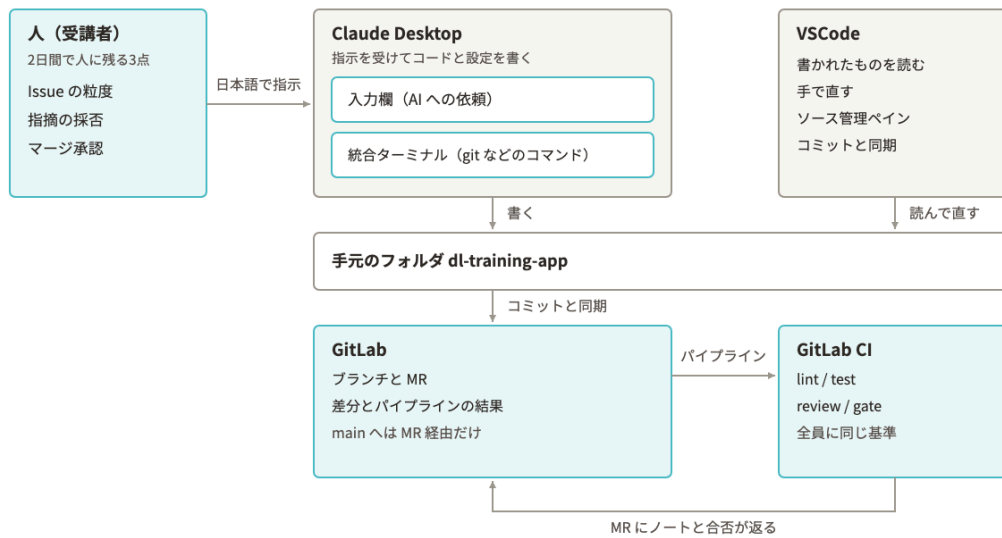
目的 実装とレビューの分離

題材は演習リポジトリの Issue #1 です。見積一覧の顧客名検索で、入力値を SQL 文字列にそのまま連結している箇所を直します。

本体は Step 4 です。実装したセッションに「レビューして」と頼むのと、履歴を切った読み手に読ませるのとでは、出てくる指摘が同じになりません。この差を手元で測ってから、分離の置き方を自社ハーネスへ1本書き戻して終わります。

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、Issue と MR は GitLab のブラウザ画面です。CLI でも同じことができますが、研修では Claude Desktop で進めます。

演習環境の登場人物



共通章と同じ図です。この演習から GitLab の箱を使う場面が増え、Issue を読む、MR を立てる、パイプラインの結果を見るの3つが同じブラウザの中で続きます。午前のプチ演習では push していないので、CI の箱に並ぶジョブが動くのはここからです。

達成状態 終わったかどうかの判定

上から順に、自分で確認できる形にしてあります。講師に聞かずに判定してください。

- ☐ MRが1本あり、`lint-phpdev` / `lint-phpver` / `test` / `test-phpunit` の4つが緑になっている。途中で `lint-phpver` が赤になった方は、混入した構文と行が MR の説明欄に記録してある(Issue #1 の受入条件3)
- ☐ `git grep -n "customer_name" -- app/controllers/estimate_ctl.php` の結果に、`$_GET['customer_name']` を `$where` へ連結している行が無い
- ☐ `index()` の `db_select` 呼び出しが第2引数のパラメータ配列を取っている
- ☐ `tests/phpunit/EstimateSearchTest.php` に、顧客名に `'` を含む入力のテストが1本以上増えている
- ☐ そのテストが変更前のコードでは落ちたことを、CI のログで確認した
- ☐ MR の「AI レビュー結果」表の3行が件数と所要時間まで埋まり、4行目の `ai_review` が件数か「未実施」で埋まっている
- ☐ MR の「人が判断した採否」表に全指摘が入り、直さなかったものに理由が書いてある
- ☐ `ai_gate` の結果を見て、今日はそれが止める力を持っていないことを説明できる
- ☐ 自社ハーネスに分離の置き方が1本入っている

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 <code>dl-training-app</code> 。読むだけのものは Issue #1、 <code>REVIEW.md</code> 、 <code>.claude/agents/review-other.md</code>
直前の演習の成果物	ブチ演習1〜3で <code>.claude/settings.json</code> に足した <code>model</code> 、 <code>effort</code> 、 <code>ask</code> 、 <code>deny</code> 、 <code>PreToolUse</code> フックの設定。 <code>ex/p3-</code> のブランチにコミット済みであること。ワークツリーはこのブランチの HEAD から切られるので、コミットしていない変更は持ち込まれません
紙かメモ帳	Step 1 で3つの欄を書きます
パーミッションモード	Claude Desktop を 手動 にしておきます。 自動 のままだと確認が出ないまま編集が進み、Step 3 で差分を読む前に変更が増えます。Step 2 でワークツリーに移ったら一度見直してください

Tips : 演習リポジトリの Issue は <https://gitlab-09291006aidev.give-app.net/dl-live-training/dl-training-app/-/issues> にあります。使うのはラベル `演習::ハンズオン1` が付いた1本だけです。

手元に PHP は入っていません。 構文チェックとテストの実行は GitLab の CI が代わりに行います。Claude Code に「php で確認して」と頼んでも実行できないので、確認はコミットして push し、パイプラインの結果を見る形になります。この演習の手順はその前提で組んでいます。

全体図

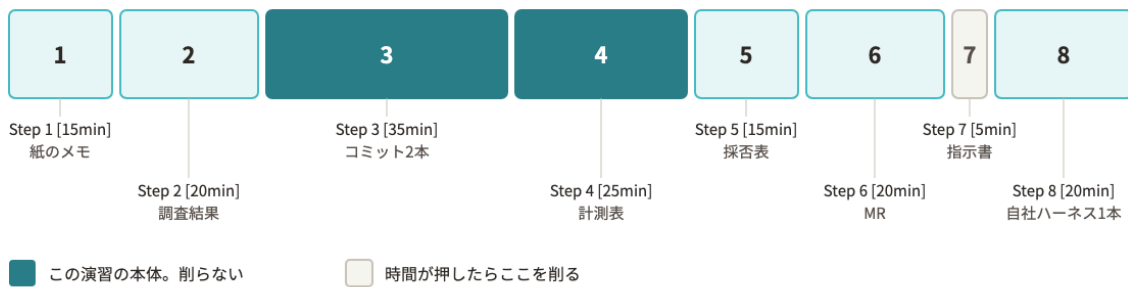
8つの Step と時間の配分

Step	やること	目安
導入	全体図の確認と題材の共有	[5min]
Step 1	Issue #1 を読み、影響範囲の仮説と改修方針と検証方法を紙に書く	[15min]
Step 2	worktree で作業場所を隔離し、grep-scout の調査結果と仮説を突き合わせる	[20min]
Step 3	機能テストを先に足し、実装し、1ステップ1コミットで進める	[35min]
Step 4	同じ差分を3通りでレビューし、件数と所要時間を記録する	[25min]
Step 5	指摘を委譲判定シートに仕分け、限定修正する	[15min]
Step 6	MR を開き、CI の6ジョブの結果を受ける	[20min]
Step 7	作業指示書を生成し、自分で赤入れする	[5min]
Step 8	分離の置き方を自社ハーネスへ書き戻す	[20min]
合計		[160min]

Step 3 が一番長く、Step 4 がその次です。時間が押したときに削るのは Step 7 で、Step 4 と Step 5 は削りません。

8つの Step と時間の配分

帯の幅が所要時間。合計 [155min] に導入の [5min] を足して [160min]



Step 4 と Step 5 は削らない。同じ差分の読ませ方を変えるところが測りどころ

上の表の所要を、帯の幅に置き換えたものです。遅れが出たときにどこを詰めるかは、この幅を見て決めてください。薄い帯だけが削ってよい場所で、濃い2本は最後まで残します。帯の下に書いてあるものが手元にそろっていれば、その Step は終わったものとして先へ進んで構いません。

影響範囲

この演習で触るもの

触るファイル	<code>app/controllers/estimate_ctl.php</code> の <code>index()</code> 、 <code>tests/phpunit/EstimateSearchTest.php</code> 、MR の説明欄、 <code>docs/shijisho/issue-1.md</code>
操作	紙に3欄を書き、Claude Desktop のワークツリーで作業場所を分け、 <code>grep-scout</code> に調査を委譲する。テストを先に <code>push</code> して赤を見てから 実装し、同じ差分を3通りでレビューして4分類に仕分け、MR を仕上げて 指示書を生成する
見る場所	バックグラウンドタスクのモデル名、CI の <code>test-phpunit</code> が赤から緑 に変わる履歴、 <code>lint-phpver</code> の結果、MR の Pipelines タブに並ぶ2 本のパイプライン
達成の状態	「終わったかどうかの判定」の9項目。MR 1本、 <code>lint-phpver</code> 以外の3 ジョブが緑(<code>lint-phpver</code> は緑か、赤なら記録して残す)、テストが変 更前に落ちた履歴、3通りの計測表、採否表、自社ハーネスに1本
自社での使いどころ	Issue から MR までの標準の流れと、実装した本人に採点させない仕組 み
影響が及ぶ範囲	<code>main</code> は保護されていて直接 <code>push</code> できない。worktree には <code>.env</code> と <code>vendor/</code> が入らない。 <code>date_from</code> / <code>date_to</code> と <code>sql_lib.php</code> は範囲 外

この演習で使う語と、作るものの置き場所

先に押さえる語です。パイプラインは、push か MR の作成をきっかけに GitLab が `.gitlab-ci.yml` の内容を実行する1回分のことで、その中の1つ1つの処理(`lint-phpdev` 、 `test-phpunit` など)がジョブです。ジョブを実際に動かすサーバーをランナーと呼び、演習環境では受講者全員で共有しています。サブエージェントは、本体の Claude Code とは別の履歴で動き、決められたツールだけを持つ下請けの Claude です。 `grep-scout` と `review-other` がこれにあたり、入力欄で `@名前 (agent)"` と書いて呼びます。 `REVIEW.md` はリポジトリ直下にあるレビュー基準のファイルで、CI の `ai_review` とサブエージェントの両方がこれを読みます。

終わったときに手元に残っているものです。名前と場所をここで決めておくと、Step 8 で自社ハースヘ移すときに探さずに済みます。

生成物	名前と場所
仮説と方針のメモ	紙、または <code>docs/handson1/plan.md</code>
影響範囲の調査結果	<code>grep-scout</code> の出力。MR 説明の「影響範囲」欄に貼る
実装	<code>app/controllers/estimate_ctl.php</code>
機能テスト	<code>tests/phpunit/EstimateSearchTest.php</code> に追記
3通りレビューの計測	MR 説明の「AI レビュー結果」欄
委譲判定シート	MR 説明の「人が判断した採否」欄
作業指示書	<code>docs/shijisho/issue-1.md</code>
ブランチと MR	Step 2 で控えた <code>worktree-</code> で始まるブランチから <code>main</code> への MR 1本

ブランチ名は Step 2 でワークツリーを作ったときに `worktree-` で始まる名前が自動で付きます。この資料の `worktree-issue-1` は、その控えた名前に読み替えてください。ブランチ名が `ex/` で始まらないのは11本の演習のうちここだけです。Step 2 の折りたたみの手順でワークツリーを使わずに進めた方は、`ex/h1-<ユーザー名>` になります。`.claude/skills/implement-issue/SKILL.md` の手順3にある `ai/issue-1` は、夜間の無人実行が自分で作るときの名前です。今回は自分で起動するので使いません。

STEP 1 Issue の読み解きと、紙に書く3つの欄 [15min]

AI を起動する前に、自分で Issue を読んで結論を出します。

- 1 ブラウザで GitLab の演習プロジェクトを開き、左のメニューの **Plan** から **Issues** を選びます。一覧の #1 が今回の Issue です。
- 2 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つの見出しを読みます。このタブは Step 6 と Step 7 でもう一度使うので閉じないでください。
- 3 範囲外 の2つを声に出して確認します。date_from と date_to の連結、app/libraries/sql_lib.php の共通処理です。
- 4 下の3つの欄を紙に書きます。書けたら伏せておき、Step 2 の終わりに開いて突き合わせます。

欄	書くこと
影響範囲の仮説	触るファイル名。そのファイルを呼んでいる場所。関係するテーブル名。今あるテストのうち、この変更で壊れる可能性があるもの
改修方針	どう直すか。選ばなかった案を1つと、選ばなかった理由
検証方法	誰が何を実行して、何が見えたら正しいか。Issue の受入条件6項目のうち、自分で確認できるのはどれか



The screenshot shows a GitLab issue page for the project 'dive-training'. The issue title is '見積一覧の顧客名検索の条件組み立て' (Setting up search conditions for customer names in the estimate list). The issue is created 3 days ago by '安田 光貴'. The 'Acceptance Criteria' (受入条件) section is highlighted, listing 6 conditions for the search functionality. The conditions are: 1. grep command for customer_name in app/controllers/estimate_ctl.php, 2. index() method of db_select, 3. CI of lint-php81, 4. CI of test and test-phpunit, 5. test/phpunit/EstimateSearchTest, 6. MR to AI. The page also shows a sidebar with navigation links and a right sidebar with assignees, labels, milestones, dates, and time tracking.

手順1で開く画面。左の本文で「受入条件」まで下りると、機械で判定できる形の6項目が並んでいます

この Step が終わった状態

- ☐ 3つの欄がすべて埋まっている。空欄のまま Step 2 に進まない
- ☐ 範囲外の2つを言える
- ☐ 受入条件6項目のうち、自分で確認できるものに印が付いている

先に自分で結論を出す理由と、受入条件の確かめ方

ここを飛ばすと、Step 2 で返ってくる調査結果が正しいのかどうかを判定できなくなります。突き合わせる相手が無いからです。改修方針の欄で案が1つしか出ていないときは、まだ方針が決まっていないことが多いです。

範囲外の2つは、Step 4 のレビューで指摘として上がってきます。そのときに「範囲外だから直さない」と判断できる状態にしておいてください。

受入条件の1番は次の1行で判定できます。Windows の PowerShell には `grep` が無いので、Windows と Mac のどちらでも同じ結果が出るこの形を使ってください。

```
git grep -n "customer_name" -- app/controllers/estimate_ctl.php
```

自分で確認できる条件と、CI に任せる条件を分けて書いておくと、Step 6 で結果を見る順番が決まります。

ハンズオン1 影響範囲の調査と実装

STEP 2 worktree での作業場所の隔離と、調査の委譲 [20min]

作業用のチェックアウトを別に作ってから始めます。元のフォルダには手が入らなくなるので、途中で方針を変えたときにフォルダごと捨てられます。

Tips : worktree は Claude Desktop の画面から作れます。コマンドは打ちません。

- 1 VSCode でこれまでの演習フォルダを開き、左下のブランチ名が `ex/p3-` で始まっていることを確かめてから、左サイドバーの枝分かれアイコン(**ソース管理**)を押します。変更が残っていたら、メッセージ欄に「午前の設定を持ち込む」と書いて **コミット** を押してください。ワークツリーはこのブランチの最後のコミットから切られるので、未コミットの変更は入りません。
- 2 Claude Desktop で新しいセッションを開き、入力欄の上の行のプロジェクトフォルダに演習フォルダ(`index.php` がある場所)を選びます。
- 3 同じ行のブランチ名の右にある **ワークツリー** にチェックを入れ、入力欄に「作業をします」と1行送ってセッションを開始します。名前を入れる欄はありません。
- 4 入力欄の上に出たブランチ名を、紙に控えます。 `worktree-` で始まる名前が自動で付いています。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。
- 5 VSCode でも同じ作業場所を開きます。 **ファイル** から **フォルダーを開く** を選び、演習フォルダの `.claude/worktrees/` の下にできた1つのフォルダを指定してください。左下の青い帯のブランチ名が、手順4で控えた名前であることを確認します。
- 6 午前の設定が入っているかを見ます。VSCode のファイル一覧で `.claude > settings.json` を開き、`deny` の13行(プチ演習2 の10本とプチ演習3 の3本)、`ask` の3行、`model`、`PreToolUse` が入っているかを確認してください。`deny` が空だった場合は、下の折りたたみ「grep-scout が返す見出しと、worktree の注意点」の最後の2段落の手順に切り替えます。

編集するのは `.claude/worktrees/` の下にできたフォルダの中だけです。 元のフォルダ側のファイルを開いて直すと、変更がワークツリーのブランチに入らず、push しても CI に届きません。開いているファイルのタブにマウスを乗せ、パスに `.claude\worktrees\` が含まれて

いることを確認してから編集してください。下の折りたたみの手順でワークツリーを使わずに進める方は、元のフォルダで `ex/h1-` のブランチに乗っていることを確かめます。

調査を **grep-scout** に委譲します。影響範囲を調べるだけのサブエージェントで、**Read** と **Grep** と **Glob** しか持っていません。書き換えはできない作りです。

- 7 手順3で開始した同じセッションの入力欄に、次の1行をそのまま貼って送信します。Issue の本文は丸ごと貼らないでください。

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

- 8 実行中に右上の：メニューの **バックグラウンドタスク** を開いて、モデル名が **Haiku** になっていることを確認します。`.claude/agents/grep-scout.md` の frontmatter に書いた `model: haiku` が効いている場所です。調査には1分から3分かかります。

- 9 返ってきた一覧と、Step 1 で紙に書いた仮説を突き合わせ、下の表を埋めます。

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

Tips : grep-scout の ## 同じ知識が重複している箇所 の節は必ず読んでください。今回の Issue では触らない箇所も挙がりますが、片側だけ直す事故はこの節が予防します。

重複の節に出る例

自分の仮説と突き合わせてから開いてください。見積の状態コード(作成中・提出済など)の対応表が、`app/libraries/util.php`、`app/controllers/estimate_ctl.php`、`app/views/estimate_list.php`、`app/views/estimate_detail.php` の4か所にあります。Day2 のハンズオン3 では、この4か所を1つにまとめる Issue #4 を無人で流します。

この Step が終わった状態

- ☐ ワークツリーのブランチ名を紙に控えてあり、VSCode の左下のブランチ表示がその名前になっている
- ☐ ワークツリーの `.claude/settings.json` に午前の設定が入っている
- ☐ 突き合わせの表が3行とも埋まっている
- ☐ MR の「影響範囲」欄に貼る内容が決まっている

grep-scout が返す見出しと、worktree の注意点

grep-scout の定義は `.claude/agents/grep-scout.md` にあり、調べる順番と返す形がそこに書いてあります。返ってくる見出しはこの7つで、行番号のない記述は書かない決まりにしています。

```
## 対象
## 触るファイル
## 呼び出し元と呼び出し先
## データベース
## テストの有無
## 同じ知識が重複している箇所
## 確認できなかったもの
```

worktree は新しいチェックアウトなので、`.env` や `vendor/` は入りません。今回の演習では使わないので、このまま進めて構いません。持ち込む必要が出た場合は、プロジェクトルートに `.worktreeinclude` を置いて、入れたいファイルを書きます。フックを書くときの `$CLAUDE_PROJECT_DIR` は元のプロジェクトルートを指したままで、worktree のパスは標準入力の JSON の `cwd` に入ります。

分岐元は `settings.json` の `worktree.baseRef` で決まります。既定の `fresh` は `origin/main` から切るので、手元のブランチで足した設定が入りません。配布した `settings.json` には `head` が入っていて、開始したときに乗っていたブランチの HEAD から切ります。

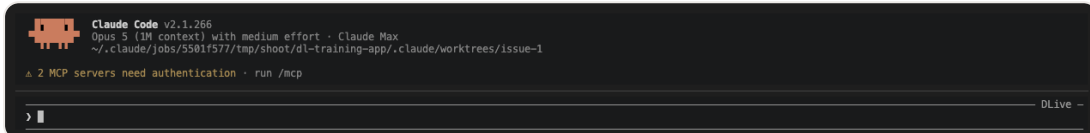
手順6で `deny` が空だった場合は、ワークツリーを始めたときに `ex/p3-` 以外のブランチに乗っていたか、設定をコミットしていなかった可能性があります。いったんセッションをアーカイブし、元のフォルダで `ex/p3-` に切り替えて設定をコミットしてから、手順2からやり直してください。

それでも `deny` が入っていない場合は、`baseRef` が効かずに `main` から切られています。ワークツリーを使わずに進めます。セッションをアーカイブし、VSCode で元のフォルダを開いて左下のブランチ名が `ex/p3-<ユーザー名>` であることを確かめ、**新しいブランチの作成** で `ex/h1-<ユーザー名>` を切ります。Claude Desktop は **ワークツリー** のチェックを入れずに、元のフォルダで新しいセッションを開きます。以降の `worktree-issue-1` は `ex/h1-<ユーザー名>` に、`.claude/worktrees/` の下のフォルダは元のフォルダに読み替えてください。

CLI から同じことをする場合

CLI からは、手順2と手順3を1行で済ませられます。CLI では名前を付けて起動できるので、ブランチ名は `worktree-issue-1` になります。研修では Claude Desktop で進めます。設定ファイルもフックも共有なので、午前に足した `deny` や `model` はどちらで起動しても効きます。

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



CLI で起動した直後の画面。上部の作業ディレクトリの表示が `.claude/worktrees/issue-1` に変わっていれば隔離できています

STEP 3 先にテスト、次に実装、1ステップ1コミット [35min]

テストを先に足し、変更前のコードでそれが落ちることを CI で確認してから実装に入ります。落ちないテストは、通っても何も保証しません。

1 VSCode で `tests/phpunit/EstimateSearchTest.php` を開き、既存の5本がこの画面をどう確認しているかを読みます。 `$_GET` に検索条件を入れ、 `capture_view()` で出力を文字列にし、 `count_list_rows()` で件数を数える形です。

2 顧客名に `'` を含む入力で例外が出ず0件になるテストを1本足します。Claude Desktop に頼む場合の文面の例です。入力値と確認する件数の2か所を、自分で決めた内容に替えてください。

```
tests/phpunit/EstimateSearchTest.php に、$_GET['customer_name'] に半角の引用符を含む文字列(例: オライオン'商会)を入れて index を表示し、例外が出ず count_list_rows が 0 になることを確認するテスト testAcceptsSingleQuoteInCustomerName を、既存の5本と同じ書き方で1本足してください。app/ 以下のファイルは変更しないでください
```

3 VSCode の `ソース管理` ペインを開きます。 `変更` の一覧に `tests/phpunit/EstimateSearchTest.php` だけが出ていることを確認してください。ファ

イル名をクリックすると差分が開きます。増えたのがテスト1本だけで、既存の5本が変わっていないことを目で見ます。

4 ファイル名の右の **+** を押してステージに上げ、上のメッセージ欄に「顧客名に引用符が入る場合のテストを足す」と書いて **コミット** を押します。

5 同じペインの **変更の同期** (または **ブランチの発行**) を押して GitLab へ送ります。初回は「このブランチを公開しますか」と聞かれるので、そのまま進めてください。ユーザー名とパスワードを聞かれたら、事前セットアップで決めた GitLab のものを入れます。

6 ブラウザで GitLab を開き、左メニューの **Build** から **Pipelines** を選びます。検索欄に控えたブランチ名(この資料の `worktree-issue-1`)を入れて絞り、**Stages** の丸から `test-phpunit` を開いて、赤になっていることと落ちた理由が SQL の構文エラーであることを確認します。

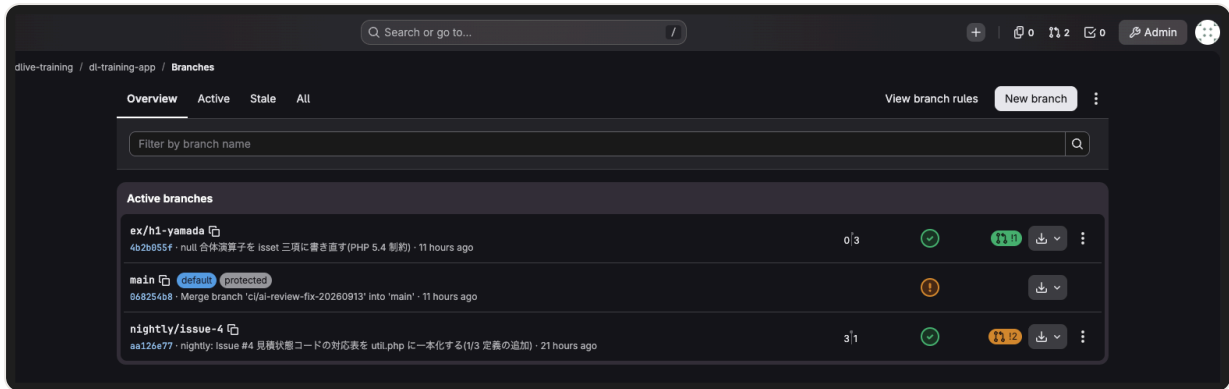
7 実装に入ります。Step 1 で書いた改修方針を、Claude Desktop に渡してください。渡すのは方針と範囲外の2つです。角括弧の中を自分の紙の内容で埋めます。

app/controllers/estimate_ctl.php の `index()` で、`customer_name` を [自分の方針。
例: `db_select` の第2引数にパラメータ配列で渡す形] に直してください。
参考にする前例は [Step 2 の調査で確認した前例のファイルと関数] です。
`date_from` と `date_to` の連結と、app/libraries/sql_lib.php は今回触りません。
このプロジェクトの対象は PHP 7.4 です。CLAUDE.md の制約の表に従ってください。
変更したらファイル名と変更した行を一覧で教えてください。コミットは私がします

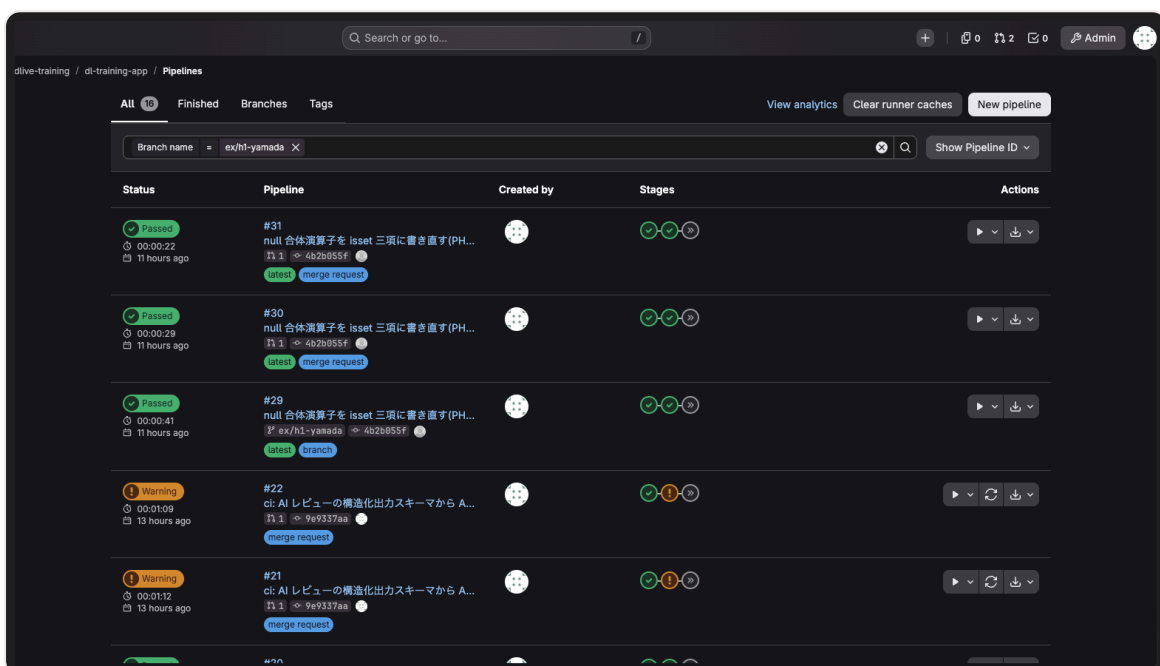
8 ソース管理ペインで `app/controllers/estimate_ctl.php` をクリックし、差分を自分で読みます。方針と違う場所に手が入っていたら、その場で Claude Desktop に戻させてください。

9 手順4と同じ要領でステージに上げ、「見積一覧の検索条件をプレースホルダに置き換える」でコミットし、同期を押して送ります。1つの変更意図につき1コミットです。

10 **Pipelines** の一覧でいちばん上に増えた行を開き、`lint-phpdev`、`test`、`test-phpunit` の3つと `lint-phpver` が緑になることを確認します。終わるまで3分から5分かかります。`lint-phpver` が赤のあいだは、後ろの段の `test` と `test-phpunit` が `skipped` になって動きません。赤になった場合は、Step 6 の記録の表に沿って混入した構文と行を控えてから、実装したセッションに「PHP 7.4 で動く書き方に直して」と頼んで直し、もう一度コミットして同期します。Issue #1 の受入条件3がこの扱いです。



手順5のあとで **Code** > **Branches** を開いた画面です。自分のブランチが **Active branches** に並びます。main には **default** と **protected** のバッジが付いていて、ここへ直接 push できないことがこの画面でも読めます。この画面は講師環境で撮ったもので、ブランチ名は皆さんが Step 2 で控えた名前と異なります



手順6で開く **Build** > **Pipelines** の画面です。上の検索欄で **Branch name** = 自分のブランチ名に絞ると、他の受講者の行が消えます。左端の **Status** 列が結果で、青い回転が実行中、赤い × が失敗、緑のチェックが成功です。 **Stages** 列の丸を押すとジョブの名前が出ます

Tips : 一覧に **pending** や時計のアイコンが出ている間は、ランナーの空きを待っている状態です。全員が同じ時間帯に送るので順番待ちになります。やり直してもパイプラインが1本増えて列が長くなるだけです。待ち時間は手順7の方針の文面を書く時間に使ってください。

この Step が終わった状態

- ☐ コミットが2本以上ある。テストのコミットが実装より前にある
- ☐ テストだけのコミットで **test-phpunit** が赤になったことを見た
- ☐ 実装後に **lint-phpdev**、**lint-phpver**、**test**、**test-phpunit** が緑になっている。
lint-phpver が一度赤になった方は、構文と行を控えてから直してある

□ `git grep -n "customer_name" -- app/controllers/estimate_ctl.php` の結果に、`$_GET` を `$where` へ連結している行が無い

1ステップ1コミットにする理由と、混ざった後の戻し方

AI は1回の依頼で複数のファイルをまとめて書き換えます。そこに自分の手直しに乗ると、後から「AI が書いた部分」と「自分が直した部分」を切り分けられなくなります。コミットを分けておくと、混ざる前の状態に境界線が引かれるので、あとで AI の変更だけを戻せます。

すでに混ざってしまった場合の手当てです。 コミットを丸ごと戻すのではなく、AI が触ったファイルだけを1つ前のコミットの状態へ戻します。

```
git log --stat -3
git checkout HEAD~1 -- app/controllers/estimate_ctl.php
```

1行目で、直前のコミットにどのファイルが入ったかを一覧で見ます。2行目で、そのうち戻したいファイルだけを1つ前の状態に戻します。他のファイルは触りません。戻した結果はソース管理ペインに変更として現れるので、差分を読んでからコミットしてください。

1つのファイルの中で混ざっている場合は、ソース管理ペインの行単位ステージを使います。 差分の表示で、残したい行の左の余白を右クリックし、**選択した範囲をステージ**を選ぶと、その行だけがコミットの対象になります。自分の手直しだけを先にコミットして退避させ、残った AI の変更を **変更を破棄** で捨てる形です。

戻す手間そのものを減らすには、依頼の時点で範囲を絞るほうが効きます。手順7の文面が「触らないファイル」を名指ししているのはこのためです。

PHP 7.4 に合わせる書き方と、静的検査を通り抜けるもの

このプロジェクトの対象は PHP 7.4 です。AI は素で PHP 8.x の構文を書きたがるので、`match` 式、`enum`、`readonly`、名前付き引数、コンストラクタプロモーション、`?->` は使いません。代替の書き方はリポジトリの `CLAUDE.md` の表にまとまっています。テストは既存の5本と同じ書き方にそろえてください。

`php -l` が通っても動かないものがあります。 `str_contains()` や `str_starts_with()` は PHP 8.0 で入った関数で、構文としては正しいので `php -l` を通ります。7.4 で実行すると `Call to undefined function` で落ちます。そのため `lint-phpver` は2段で見えています。1段目の `php -l` が構文を、2段目の `php ci/undefined_functions.php` が7.4の実行環境に関数があるかを `function_exists()` で確かめ、`str_contains()` はこの2段目で落ちます。構文検査だけでは対象バージョンを守りきれない、という話がここで出ます。

この時点では対象バージョンを機械で止めるフックをまだ入れていません。制約は `CLAUDE.md` の文章だけで守られている状態です。文章に書いた制約が数ターン後に薄れる様子は、Step 6 の `lint-phpver` の結果に出ます。出たら記録してください。Day2の最初のセッションでその記録を使います。

現行環境がもっと古い場合でも、対象には移行先の版を書くほうが移行が進みます。古い版を対象にして書かせると、移行のたびに書き直しが発生するためです。

CI の出力の読み方と、赤と緑の見分け

ジョブの名前の左に緑のチェックが付いていれば成功、赤い × は失敗、橙色の ! は「失敗したが `allow_failure` で全体は止めない」です。ジョブを開いたログの最終行が `Job succeeded` なら緑、`ERROR: Job failed: exit code 1` なら赤です。`test-phpunit` は他のジョブより2分ほど長くかかります。

手順6でこう見えていれば正しい状態です。テストが落ちる理由が「まだ直っていないから」であることを、エラー文で確認してください。落ちた理由はログのいちばん下ではなく、赤い行の直前にあります。

```
.....E                                     6 / 6 (10
0%)

There was 1 error:

1) EstimateSearchTest::testAcceptsSingleQuoteInCustomerName
PDOException: SQLSTATE[42601]: Syntax error: 7 ERROR:  syntax error at or near "商会"
LINE 6:  WHERE e.del_flg = 0 AND c.customer_name LIKE '%オライオン'商会%'

ERRORS!
Tests: 6, Assertions: 12, Errors: 1.
```

手順10でこう見えていれば正しい状態です。アサーションの数は自分が足したテストの書き方で変わります。見るのは赤のほうの `E` が1つ出ていることと、緑のほうの `OK` の行です。

```
.....                                     6 / 6 (10
0%)

OK (6 tests, 13 assertions)
```

手が止まったときの確認先

テストが赤にならない場合は、足したテストが実際に新しい経路を通っているかを見てください。 `$_GET['customer_name']` に入れた文字列が `'` を含んでいるかが最初の確認点です。

実装後もまだ赤い場合は、 `db_select()` の第2引数にパラメータ配列を渡せているかを見ます。 `app/controllers/stock_ctl.php` の `index()` に、同じライブラリをプレースホルダで使っている前例があります。

同期が拒否された場合は、 `main` に対して送っていないかを確認してください。 `main` は保護されていて直接 push できません。 `pre-receive hook declined` と出るのがこの状態です。VSCode 左下のブランチ表示が Step 2 で控えた名前であることを見てから、もう一度押してください。

`Authentication failed` と出た場合はパスワードが違います。何度も続くときは、事前セットアップの FAQ にある資格情報マネージャーの手順で、記憶された古いパスワードを消してください。

手順6で `test-phpunit` が緑になってしまった場合は、テストが `worktree` 側ではなく元のフォルダ側のファイルに書かれています。VSCode のタブのパスに `.claude\worktrees\` が含まれているかを見てください。

ハンズオン1 3通りのレビューと採否

STEP 4 3通りのレビューと記録 [25min]

ここがこの演習の本体です。差分は1つ、読ませ方は3つ。読み手から情報を1つずつ外していき、指摘がどう変わるかを自分の数字で見ます。

読ませ方	読み手が持っているもの	1つ前から外したもの
1 実装したセッションでそのまま依頼	実装の履歴、 <code>REVIEW.md</code> 、役割の指定	
2 <code>review-other</code> サブエージェント	<code>REVIEW.md</code> 、定義ファイルで固定した役割と観点	実装の履歴
3 履歴を切った新しい会話	差分ファイルと <code>REVIEW.md</code> だけ	役割と観点の定義
4 CI の <code>ai_review</code>	差分と <code>REVIEW.md</code> 。人の言い方が一切入らない。結果は Step 6 で受ける	人がその場で足す言葉

1から2で履歴が外れ、2から3で役割の定義が外れます。どちらの段で指摘が変わったかで、効いていたのが履歴なのか定義なのかが分かります。

同じ差分を4つの経路で読ませる



下に行くほど読み手の持ち物が減る。どの段で指摘が変わったかが、効いていたものの答え

この Step の全体像です。同じ `review-input.diff` を4つの経路で読ませます。1の実装したセッションそのまま依頼だけが履歴を持っていて、自分の変更を甘く見ます。2の `review-other` サブエージェントは定義ファイルで役割を固定した読み手、3の履歴を切った新しい会話は差分だけを渡した素の状態、4は runner 上で動く CI の `ai_review` です。3通りの結果は `ho1-reviews.md` に貼って比べます。4は Step 6 で受けます。右の表は、その4つがそれぞれ何を持って差分を読むかです

- 1 1通り目。** 実装したセッションでそのまま頼みます。新しい会話にせず、続けてください。

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

- 2 差分をファイルに出します。ここはコマンドを使います。** サブエージェントは `Read` と `Grep` と `Glob` しか持たず、自分で差分を取れないためです。Claude Desktop の右上の `>` アイコン、または `Ctrl + バッククォート` で統合ターミナルを開きます。セッションと同じ場所、つまりワークツリーの中で開くので、移動は要りません。 `yamada` はご自身のユーザー名に置き換えます。

```
git diff ex/p3-yamada --output=review-input.diff
```

比べる相手は `main` ではなく、ワークツリーを切った元の `ex/p3-` のブランチです。`main` と比べると、プチ演習1〜3 で足した設定の変更まで差分に入ります。

- 3 2通り目。** 実装の経緯を渡さずに、定義ファイルで役割を固定したサブエージェントへ読ませます。

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

結果は「重大度、場所、内容、根拠、直し方」の5列の表と、最後の 判定: 合格 か 判定: 要修正 の1行で返ります。表の行数が指摘件数です。

- 4 3通り目。Ctrl+N で新しいセッションを開き、プロジェクトフォルダに演習フォルダの .claude/worktrees/ の下の自分のフォルダを選びます。ワークツリーのチェックは入れません。初めて開くフォルダなので、ワークスペースの信頼を聞かれたら **ワークスペースを信頼** を押します。押さないとフックと deny が読まれません。役割の定義は渡さず、差分ファイルと基準だけを渡してください。

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

2通り目と同じ形で返るように、返し方だけを指定しています。役割も観点も与えていないので、2通り目と出方が変わります。

- 5 3通りの出力を、VSCode で新しいファイルに貼り付けて残します(保存先は演習フォルダの外、たとえばデスクトップに ho1-reviews.md)。Step 6 で MR に貼ります。

- 6 review-input.diff を削除します。ソース管理ペインの **変更** にこのファイルが残っていないことまで見てください。

- 7 下の表の1行目から3行目を埋めます。開始と終了の時刻を控えて所要時間も入れてください。4行目は Step 6 で CI の結果を受けてから足します。

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

件数の増減で良し悪しを決めないでください。 見るのは「実装したセッションが出さなかった指摘が何か」です。自分で書いたコードを自分で採点すると検出が落ちることは測られています。同じモデルでも、履歴を捨てた別セッションで読ませたほうが F1 が 28.6、自己レビューは 24.6 でした。自己レビューを2回重ねても、1回目との差は有意ではありませんでした ([arXiv 2603.12123](https://arxiv.org/abs/2603.12123))。効いているのは履歴を切ったことです。

この Step が終わった状態

- ☐ 表の1行目から3行目が埋まっている。所要時間の列も埋まっている

- ☐ 「気づいた差」の列に、件数以外のことが1行書いてある
- ☐ 3通りの出力をファイルに貼って残してある
- ☐ `review-input.diff` を消している。ソース管理ペインの変更一覧に出ていない

基準が空のままレビューさせる意味と、JSON で受け取る形

REVIEW.md は今日の時点では見出しだけの空の雛形です。自社版を起案するのは Day2 です。基準が空でもレビューは返りますが、返り方が基準のある場合とどう違うかを「気づいた差」の列に書いておくと、Day2 の起案の材料になります。

件数を機械で数えたい場合は、JSON スキーマを指定して受け取る形があります。パイプで渡す形は Claude Desktop の入力欄からは実行できないので、統合ターミナルを開いて CLI から打ちます。研修では使いませんが、自社で集計に載せるときはこちらが楽です。

```
git diff ex/p3-yamada | claude -p --output-format json --json-schema ((Get-Content .claude\review.schema.json -Raw -Encoding UTF8) -replace "'",'\\"') "REVIEW.md の基準でレビューし、findings を返す"
```

--json-schema にはファイル名ではなく、スキーマの中身を文字列で渡します。PowerShell 5.1 は引数の中の " をそのまま渡さないで、-replace で \" に置き換えています。Mac では --json-schema "\$(cat .claude/review.schema.json)" です。

findings は返ってきた JSON の structured_output に入ります。形はこう固定されるので、Step 5 の仕分けにそのまま使えます。

```
{
  "summary": "見積一覧の検索条件の変更を読み、3件見つけました。",
  "findings": [
    {
      "severity": "P1",
      "confidence": 0.9,
      "file": "app/controllers/estimate_ctl.php",
      "line": 22,
      "message": "date_from が文字列のまま SQL に連結されています。db_select の第2引数へ移してください。",
      "category": "sql-injection"
    }
  ]
}
```

Windows の PowerShell では、先に次の1行を打って文字コードを UTF-8 にそろえないと、日本語が ? に化けて渡ります。

```
$OutputEncoding = [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

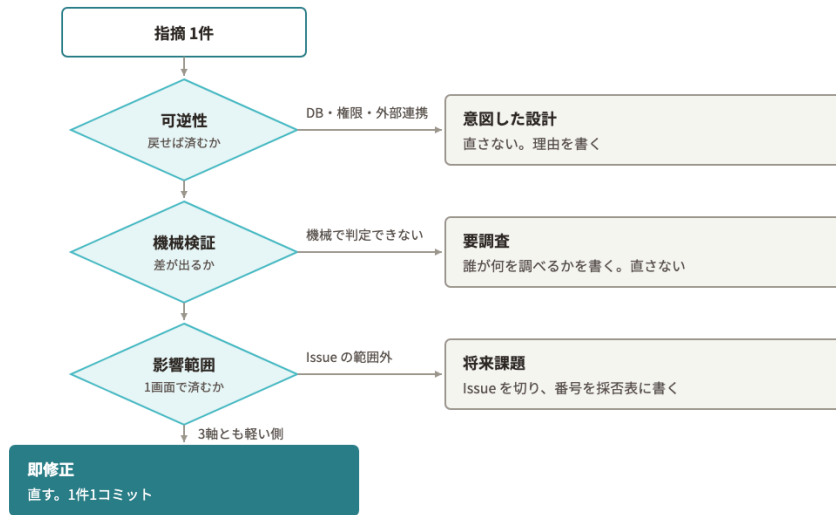
STEP 5**委譲判定シートへの仕分けと限定修正 [15min]**

集まった指摘を4つに分けます。分けるのは人です。指摘を全部受け入れると、意図してそうしていた設計が巻き戻ります。

軸	見る点	軽い側	重い側
可逆性	直して外したとき、何を戻せば元に戻るか	コードを戻せば元に戻る	DB、外部連携、権限に届く
機械検証	直ったかどうかを機械が判定できるか	lint かテストで差が出る	目で見ないと判定できない
影響範囲	触る画面の数	1画面	2画面以上、または共通処理

分類	この分類にする条件	この場でやること
即修正	3軸すべてが軽い側	直す。1件1コミット
要調査	機械検証が重い側。正しいかどうかを読んだだけでは決まらない	誰が何を調べるかを書く。直さない
将来課題	Issue の範囲外	Issue を切る。番号を採否の表に書く
意図した設計	可逆性が重い側、または背景があって今の形にしている	直さない。理由を書く

指摘1件の行き先



3軸を先に付けてから分類を決める。分類から決めると感覚の仕分けになる

上の表を、判定の順番として描いたものです。菱形は上から順に通し、どれか1つで右へ抜けた時点でその行の分類が決まります。菱形の横に書いてあるのは右へ抜ける側の条件で、3つとも抜けずに下まで来たものだけが即修正です。出口の箱の下に1行が、その分類でその場にやることです。

1 3通りで出た指摘を1つの表にまとめます。同じ指摘が複数のレビューから出ている場合は1行にし、どのレビューから出たかを残します。

2 各行に3軸を付けます。付けられない行は、その場でファイルを開いて確かめてください。

3 3軸から分類を決めます。分類から先に決めると、感覚で仕分けることになります。

4 date_from と date_to の指摘は範囲外なので将来課題です。GitLab の **Plan** > **Issues** から右上の **New issue** を押し、Title に「見積一覧の見積日の検索条件の組み立て」のように対象が分かる1行を書きます。

5 Description には Issue #1 と同じ6つの見出しのうち、少なくとも **現状** と **範囲外** の2つを書いて **Create issue** を押します。作成後のページの見出しの # の後ろの数字が Issue の番号です。

6 「即修正」に入れたものだけを直します。1件1コミットで、コミットメッセージは「レビュー指摘: 直した内容を1行で」の形にします。ソース管理ペインからコミットまで行い、同期はまだ押しません。送るのは Step 6 の手順1でまとめてです。

7 直したら、その指摘を出した読ませ方でもう一度レビューし、その指摘が消えたことだけを確認します。新しく出てきた P2 以下は追いません。

divine-training / di-training-app / Work items / New

New Issue

Type
Issue

Title (required)
[]

Description
Choose a template

Normal text | B | I | | | | | | | | +

Write a comment or drag your files here...

Switch to plain text editing

☐ Turn on confidentiality: Limit visibility to project members with at least the Planner role.

Create Issue Cancel

Assignee
None - assign yourself Edit

Labels
None Edit

Milestone
None Edit

Dates
Start: None
Due: None Edit

Contacts
None Edit

手順4で **New issue** を押したあとの画面です。 **Title (required)** に対象が分かる1行、 **Description** に **現状** と **範囲外** を書き、左下の **Create Issue** を押します。作成後に開くページの見出しに出る **#** の後ろの数字が Issue の番号なので、そこで控えてください。他の受講者も同じ Issue を切るので、番号は人によって違います

この Step が終わった状態

- ☐ すべての指摘が4分類のどれかに入っている。空欄が無い
- ☐ 「将来課題」にしたものは Issue の番号が書いてある
- ☐ 「意図した設計」にしたものは理由が書いてある
- ☐ 直したのは「即修正」だけになっている

「意図した設計」に入れた指摘の扱い

AI は業務の背景を知りません。「意図した設計」に入れた指摘は、次の改修でも同じ形で出ます。同じ判断を毎回やり直さずに済ませるには、理由を **REVIEW.md** の **Do not report** に移すかどうかをその場で検討してください。Day2 のレビュー基準の起案で、この判断が材料になります。

ハンズオン1 MR と CI の結果

STEP 6 MR と6ジョブの読み方 [20min]

MR を GitLab の画面で作ります。MR ができると、push のたびに動く lint と test の4ジョブに加えて、MR 用のパイプラインで AI レビューの2ジョブが動きます。合流前の最終判定をここで受けます。

- 1 VSCode のソース管理ペインで **変更の同期** を押し、Step 5 の修正を送ります。即修正が0件で新しいコミットが無い場合は押さずに次へ進んでください。
- 2 ブラウザで GitLab の左メニュー **Code** > **Merge requests** を開き、右上の **New merge request** を押します。
- 3 **Source branch** に Step 2 で控えたブランチ(この資料の `worktree-issue-1`)、**Target branch** に `main` を選び、**Compare branches and continue** を押します。
- 4 次の画面で **Title** を「Issue #1 見積一覧の検索条件をプレースホルダに置き換える」に書き換えます。
- 5 **Description** には、リポジトリの既定のテンプレート(`Default.md`)の節が最初から入っています。「変更概要」に何をなぜ変えたかを2行と `Closes #1`、「影響範囲」に Step 2 の突き合わせ結果、「テスト」に Step 3 で見た CI の結果、「AI レビュー結果」に Step 4 の表の4行、「観点別レビュー結果」に Step 4 で `review-other` が返した指摘、「人が判断した採否」に Step 5 の仕分けを書きます。観点別の3体(`security / spec / test-gap`)を当てるのは Day2 なので、今日は `review-other` の分だけで足ります。
- 6 **Preview** タブで表の罫線が崩れていないかを見てから **Create merge request** を押します。
- 7 MR のページ上部の **Pipelines** タブを開きます。いちばん上の2本が最新で、merge request のラベルが付いた行が MR 用(`ai_review` と `ai_gate`)、branch のラベルが付いた行がブランチ用(残りの4つ)です。合わせて6つのジョブの結果を確認してください。
- 8 **Changes** タブを開き、Step 4 で読ませた `review-input.diff` の変更が含まれているかを見ます。MR の **Changes** と CI の `ai_review` は `main` との差なので、プチ演習1〜3で足した `.claude` 配下の設定や `util.php` の1行も一緒に出ます。Step 4 の3通りより読む量が多いことを踏まえて、4行目の件数を比べてください。

nightly_implement は押さないでください。 手動ジョブなので6つに数えません。

ジョブ	見ているもの	落ちたときにすること
lint-phpdev	現行環境(PHP 8.3)で構文が通るか	エラー行を直す
lint-phpver	対象の PHP 7.4 で動くか。1段目の <code>php -l</code> で <code>match</code> 式や <code>enum</code> のような 8.x の構文を、2段目の <code>php ci/undefined_functions.php</code> で <code>str_contains()</code> のような 7.4 に無い関数の呼び出しを落とす	混入した構文と行を MR の説明欄に記録してから直す(Issue #1 の受入条件3。記録は Day2 の題材になります)
test	簡易ランナー11件	落ちたケースの名前から該当箇所を探す
test-phpunit	機能テスト。対象と同じ PHP 7.4 で回る	手元では再現できないので、ジョブのログを読む
ai_review	差分を読み、 <code>findings.json</code> を作って MR にコメントを1本置く。落とす判断はしない	落ちるのは応答がスキーマの形にならなかったときだけ
ai_gate	<code>findings.json</code> に P0 か P1 があれば <code>exit 1</code>	今日は <code>allow_failure</code> が付いている。赤くても止まらない

`ai_review` が動くと、MR の **Overview** タブの下の方に「AIレビュー(モデル: ... / 深さ: ... / 差分 ... 行)」で始まるコメントが1本付きます。見出し行の件数を表の4行目に写してください。所要時間はジョブの画面右側の **Duration** を使います。

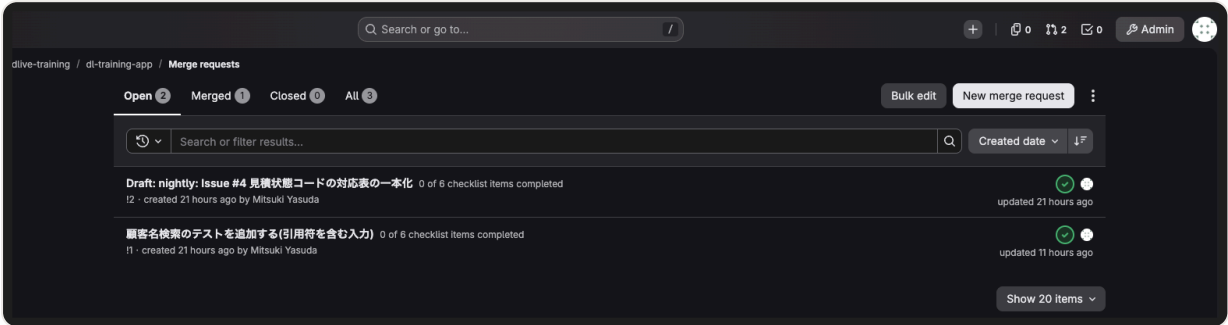
ここで一度立ち止まってください。`ai_gate` は `allow_failure: true` なので、赤くなってもパイプライン全体は赤になりません。ゲートが赤いのにマージできる状態は、ゲートではありません。列挙しているだけです。この1行を外してゲートを赤にするのが Day2 のハンズオン2です。**Merge** ボタンまで止めるには、プロジェクトで **Pipelines must succeed** を有効にしておく必要があります。演習のプロジェクトはこの設定を入れていないので、ボタンは押せますが押しません。

lint-phpver が落ちた方は、直す前に記録してください。記録先は MR の説明欄です。落ちた構文は Day2 の最初のセッションでそのまま教材になります。

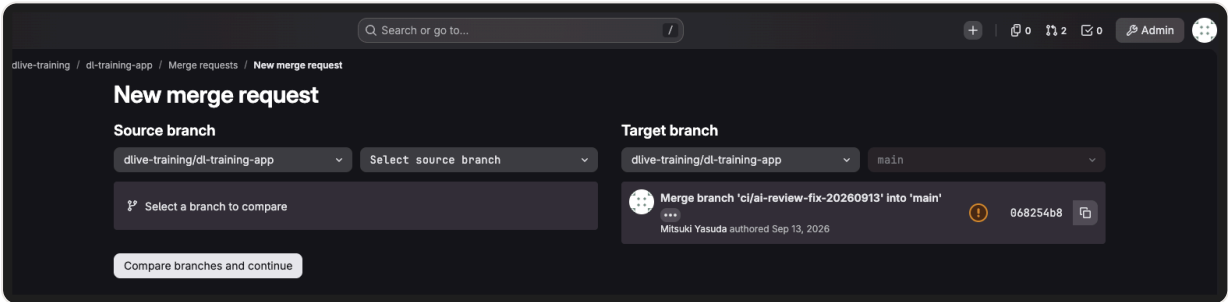
記録すること

書き方

落ちた行	ファイル名と行番号
出たエラー文	ログに出たエラーの1行目を、そのまま貼る
混入した構文	<code>match</code> のように、どの構文が入ったか
指示のどこに書いてあったか	<code>CLAUDE.md</code> に書いてあったのに混入したのか、書いていなかったのか



手順2で開く **Code** > **Merge requests** の一覧です。右上の **New merge request** から作ります。作成後は、**Open** タブにソースブランチが自分のもので作成者が自分の名前になっている行が並び、行の左下に **!1** のような MR の番号が出ます。この画面は講師環境のもので



手順3の画面です。左の **Source branch** の **Select source branch** で `worktree-issue-1` を選び、右の **Target branch** が `main` になっていることを見てから、左下の **Compare branches and continue** を押します

Search or go to...

/

+

0

11:2

0

Admin

dlive-training / dl-training-app / Merge requests / 11 / Edit

From `ex/h1-yasuda` into `main`

Title (required)

顧客名検索のテストを追加する(引用符を含む入力)

☐ Mark as draft
Drafts cannot be merged until marked ready.

Description

Choose a template

Normal text

B

I

U

Link

Image

Code

Table

Quote

More

変更概要

何を、なぜ変えたかを2〜3行。対応する Issue は Closes #<番号> で閉じる

Closes #

影響範囲

grep-scout の調査結果から、触ったファイルと行、関係するテーブル、同じ知識がなかな所にある箇所

区分	対象	備考
触ったファイル		
呼び出し元		
テーブル		

手順4から手順6で使う画面です。上の **Title (required)** を書き換えます。**Description** には「変更概要」「影響範囲」から始まる節の見出しと表が入っています。斜体の1行が各節に何を書くかの案内です

Search or go to...

/

+

0

11:2

0

Admin

dlive-training / dl-training-app / Merge requests / 11

顧客名検索のテストを追加する(引用符を含む入力)

Try Rapid Diff Beta

Edit

Code

⋮

11 Open Mitsuki Yasuda requested to merge `ex/h1-yasuda` into `main` 21 hours ago

Overview 4 Commits 3 Pipelines 16 Changes 2

Compare `main` and latest version

2 files +14 -2

Files 2

Search (e.g. *.vue) (F)

app/controllers

estimate_ctl.php +4 -2

tests/phpunit

EstimateSearchTest.php +10 -0

app/controllers/estimate_ctl.php

@@ -13,8 +13,10 @@ class Estimate_ctl extends Base_Controller

13 13 \$where = 'e.del_flg = 0';

14 14

15 15 // 検索条件の組み立て

16 16 \$params = array();

17 17 if (isset(\$_GET['customer_name'])) && \$_GET['customer_name'] != '' {

18 18 \$where .= " AND c.customer_name LIKE '%" . \$_GET['customer_name'] . "%'";

19 19 \$where .= " AND c.customer_name LIKE ?";

20 20 \$params[] = '%' . \$_GET['customer_name'] . '%';

21 21 }

22 22 if (isset(\$_GET['status_cd'])) && \$_GET['status_cd'] != '' {

23 23 \$where .= " AND e.status_cd = " . (int)\$_GET['status_cd'];

24 24 }

25 25 JOIN mas_customer c ON c.customer_id = e.customer_id

26 26 WHERE " . \$where . "

27 27 ORDER BY e.estimate_d1_id DESC";

28 28 \$rows = db_select(\$sql);

29 29 \$rows = db_select(\$sql, \$params);

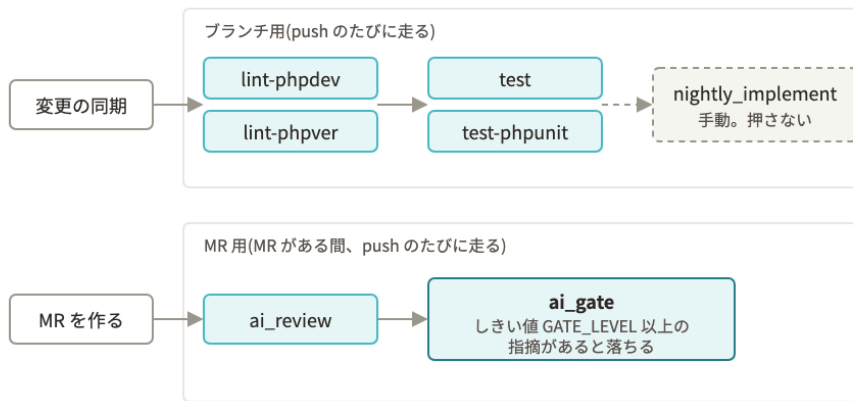
30 30 }

31 31 \$this->render('estimate_list', array('rows' => \$rows));

32 32 }

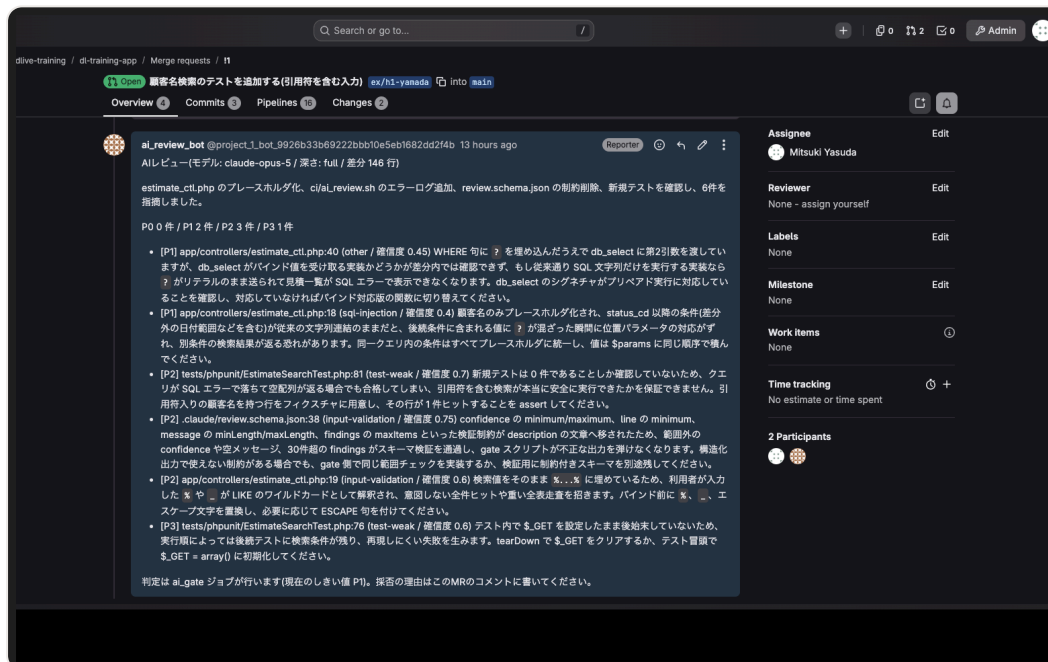
手順8で開く **Changes** タブです。左の列に変更したファイルの一覧、右上に **2 files +14 -2** のようにファイル数と足した行数、消した行数が出ます。差分の左の2列の数字が変更前と変更後の行番号で、赤い行が消した行、緑の行が足した行です

同じコミットに対して走る2本のパイプライン

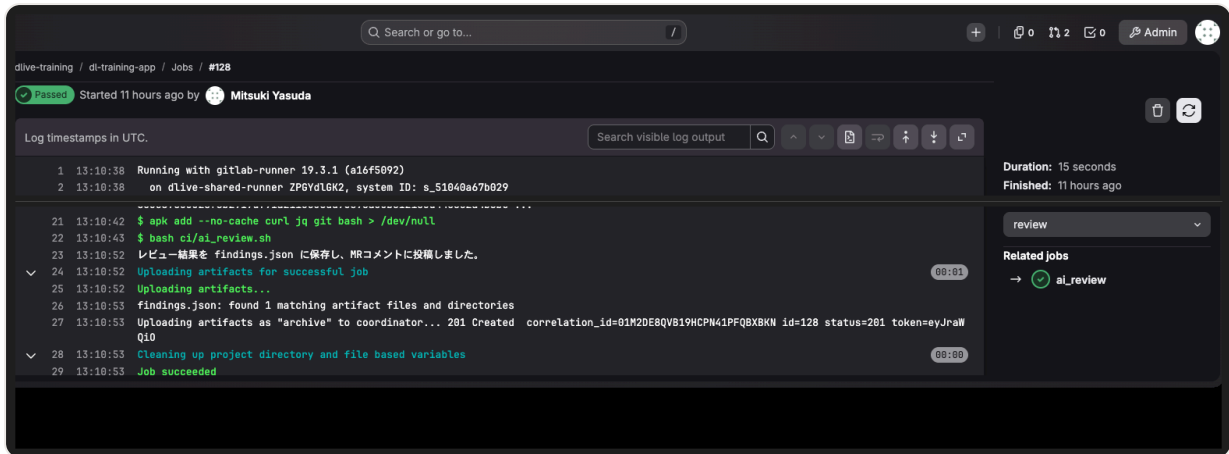


MRの Pipelines タブでは、
branch と merge request の
ラベルが付いた行として
2本並ぶ

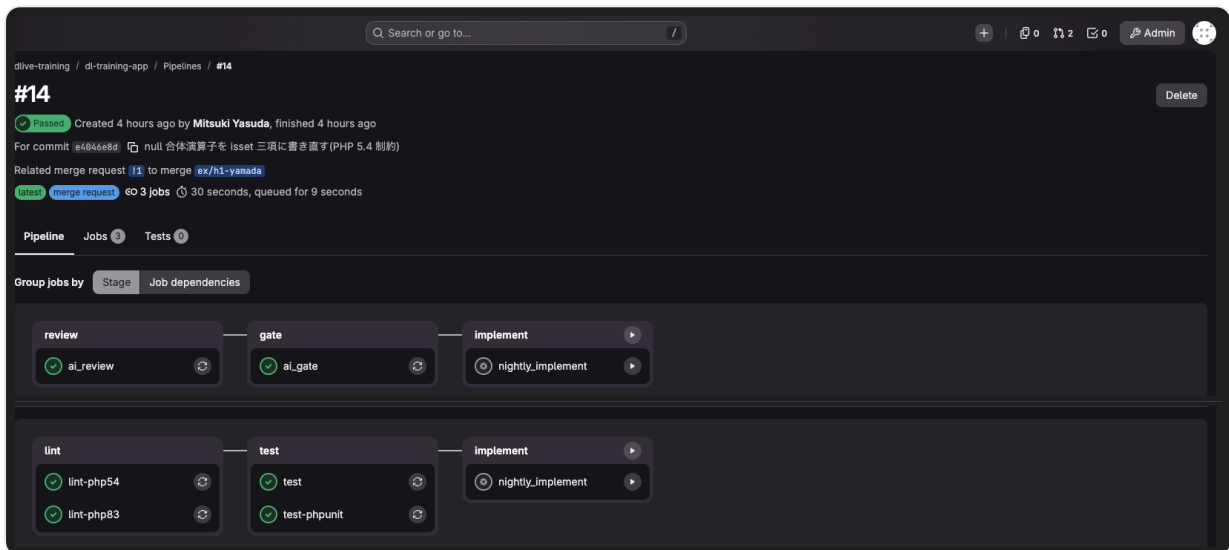
手順7で見る2本の関係です。上のブランチ用は同期のたびに走り、`lint-phpdev` と `lint-phpver` のあとに `test` と `test-phpunit` が動きます。点線の先の `nightly_implement` は手動なので押しません。下のMR用はMRがある間のpushのたびに走り、`ai_review` のあとに `ai_gate` が `GATE_LEVEL` 以上の指摘があると落ちます



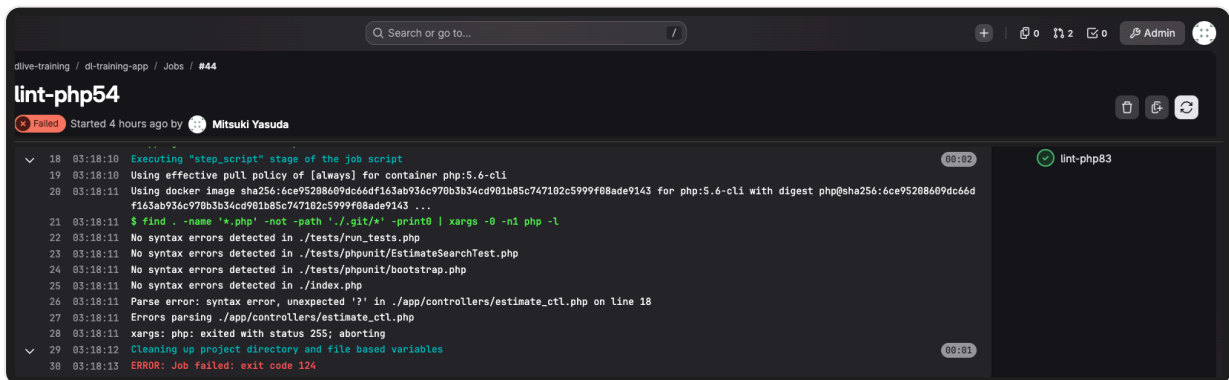
Overview タブで読む `ai_review_bot` のコメントです。1行目に「AIレビュー(モデル: ... / 深さ: ... / 差分 ... 行)」、次に要約、その下に `P0 0 件 / P1 2 件 / P2 3 件 / P3 1 件` のような件数の行が出るので、これを表の4行目に写します。この画面は差分146行で `claude-opus-5` が6件指摘した例です。差分が43行だと `sonnet` に切り替わり、指摘の件数は減ります



所要時間を取るジョブの画面です。MR の **Pipelines** タブから **ai_review** を開くと、右側の **Duration** に秒数が出ます(この例は 15 seconds)。ログの `bash ci/ai_review.sh` の次の行に「レビュー結果を findings.json に保存し、MRコメントに投稿しました。」と出ていれば、コメントが付いています



手順7でパイプラインの番号を押すと開く詳細画面を、MR 用(上)とブランチ用(下)の2本で並べたものです。上は review、gate の順に **ai_review** と **ai_gate** が並び、見出しの下に **merge request** のラベルと **Related merge request !1** の行が出ます。下は lint、test の順に4ジョブが並びます



落ちたときのログ。 **Errors parsing** の行でファイル名を確認し、その1つ上の行のエラー文を記録してください

この Step が終わった状態

- ☐ MRが1本あり、`lint-phpdev`、`lint-phpver`、`test`、`test-phpunit`の4つが緑。途中で`lint-phpver`が赤になった方は、その記録がMRの説明欄に残っている
- ☐ MR説明の節が埋まっている。「AIレビュー結果」の4行目は、`ai_review`が動かなかった場合は「未実施」と書いてある
- ☐ `ai_gate`の赤に関係なく **Merge** ボタンが押せる状態が出ていることを確認した(押しません)
- ☐ `lint-phpver` が落ちた場合、4項目の記録が残っている

ai_gateとlint-phpverの出力の読み方

`ai_gate` の出力はこう見えます。`allow_failure` が付いているため、赤くてもパイプライン全体は赤になりません。

しきい値 P1 以上、確信度 0.0 以上の指摘を数えます。
confidence の無い指摘は 1.0、P0～P3 以外の severity は P0 として数えます。
P1: 1 件 / P2: 2 件

ゲートで止めた指摘 1 件

[P1] app/controllers/estimate_ctl.php:22 date_from が文字列のまま SQL に連結されています。

直すか、直さない理由をMRのコメントに書いてから再実行してください。

`lint-phpver` は2段で見ます。1段目の `php -l` で落ちたときは、ログに `PHP Parse error` で始まる行と、`Errors parsing` に続くファイル名の行が出ます。エラーの文面はPHPの版で変わるので、出た1行目をそのまま記録してください。

2段目は `php ci/undefined_functions.php` です。`str_contains()` や `str_starts_with()` のようなPHP 8.0 で入った関数の呼び出しは、構文としては正しいので1段目を通ります。2段目が7.4の実行環境に `function_exists()` で聞き直し、無ければ次のように落とします。

undefined_functions: PHP 7.4 に無い関数と呼んでいます。 `php -l` は通りますが実行時に落ちます。

この行の下に、ファイル名と行番号と関数名が1件1行で並びます。手元で同じものを編集の直後に止めるのが、Day2のプチ演習4で登録する `check-phpver` フックです。

ハンズオン1 持ち帰り物と書き戻し

STEP 7 作業指示書の生成と赤入れ [5min]

実装が終わった後に指示書を作るのは順番が逆ですが、生成したものと自分が書いたものの差を見るのが目的です。

- 1 Step 1 で開いたブラウザの Issue #1 を表示し、本文(背景から範囲外まで)を範囲選択してコピーします。
- 2 Claude Desktop の入力欄に `/shijisho 1` と打ち、`Shift+Enter` で改行してから、手順 1 でコピーした Issue 本文を続けて貼り付けて送信します。`Enter` だけを押しと、本文を貼る前に送られます。途中で `grep-scout` が再び呼ばれるので、2分から3分かかります。`glab` や `curl` で Issue を取りに行く実行確認が出たら、許可しない側を選んでください。本文は貼ってあるので、取りに行く必要はありません。
- 3 VSCode で `docs/shijisho/issue-1.md` を開き、5項目(目的、影響範囲、変更方針、検証、戻し方)で止まっているか確認します。
- 4 「目的」と「戻し方」の2つを、自分の言葉に直します。この2つは、Issue の本文を言い換えただけになりやすい箇所です。
- 5 ソース管理ペインでステージに上げ、「Issue #1 の作業指示書を足す」でコミットし、同期を押して送ります。`worktree` を片付けたときに消えないよう、MR に載せておきます。

この Step が終わった状態

- ☐ `docs/shijisho/issue-1.md` がある
- ☐ 「目的」と「戻し方」に自分の言葉が入っている

Issue 本文を手で渡す理由と、直した箇所の読み方

`/shijisho` は `.claude/skills/shijisho/SKILL.md` に書かれた手順を Claude Code に実行させるスキルです。この skill は本来 Issue の本文を GitLab から取りに行きますが、受講者の端末には `glab` もアクセストークンも入っていないので、最初から本文を一緒に渡します。

直した箇所が「目的」と「戻し方」に集中していたら、生成物の使いどころが見えたことになります。影響範囲と検証は `grep-scout` と CI の出力から機械的に埋まるので、人が書き直す必要はほとんど出ません。

STEP 8 自社ハーネスへの書き戻し [20min]

今日のうちに1本だけ自社ハーネスへ移します。移すのは「実装した本人に採点させない」を仕組みにするものです。2つ用意してありますが、入れるのは片方だけにしてください。

選ぶもの	向いている場合	入れるファイル
履歴を切るサブエージェント	手元でレビューを回したい。端末ごとの環境差を減らしたい	<code>.claude/agents/review-other.md</code>
CI の AI レビュージョブ	レビューを MR に残したい。全員の端末に同じ設定を配りたくない	<code>ci/ai_review.sh</code> と <code>.gitlab-ci.yml</code> の review ステージ

- 2つのファイルを開いて、どちらを自社に置くか決めます。決め手は、レビューを人の手元でやるか MR に置くかです。
- 選んだ1本を自分のハーネスに写します。下の表の置き場所を見てください。
- 自社のリポジトリ名、ブランチ名、レビュー基準のファイル名に書き換えます。演習リポジトリの名前がそのまま残っていると、次に使う人が混乱します。
- 持ち帰り台帳の本日分に、今日足した1本を記入します。

ファイル	D2 での置き場所	smart3pm での置き場所
------	-----------	-----------------

<code>.claude/agents/review-other.md</code>	既存のエージェント定義と同じ階層。効いているのは履歴を切ることなので、 <code>model:</code> は実装側と同じままで構いません。変えるのは発展課題2と同じ任意の実験です	同じ階層に同じ名前で置く
<code>ci/ai_review.sh</code>	CI 設定と同じ階層。CI の runner(alpine) の上の bash で動くので、PowerShell へ書き直す必要はありません	同じ階層に同じ名前で置く。コマンドはそのまま使える
3通りレビューの計測表	持ち帰り台帳に貼る	持ち帰り台帳に貼る
委譲判定シート(4分類×3軸)	レビュー基準のファイルの近くに置く	規約本体のどのタグに属するかを決めてから足す
MR テンプレの「AI レビュー結果」欄	<code>.gitlab/merge_request_templates/</code>	同じ
worktree 運用手順(2行)	戻し方の節の近くに追記	規約本体の該当節に追記

この Step が終わった状態

- ☐ 自社ハーネスに1本入っている。2本入れていない
- ☐ 演習リポジトリの名前が残っていない
- ☐ 持ち帰り台帳の本日分に1行増えている

2つの環境で書き方が割れる箇所

ファイルの名前と中身は写しますが、本文のコマンド例は自社の形に直してください。D2 は PowerShell 5.1 で、スクリプトとメッセージを ASCII に保つ約束になっていると伺っています。smart3pm は bash が正です。

同じ目的のものが2つの書き方で並ぶことになるので、どちらを正にするかを決めておくと、次に足す人が迷いません。決められない場合は、決められないこと自体を台帳に書いてください。Day2 の最後にこの判断を扱います。

追加と考察

終わった後に1行ずつ書くこと

次の3つを、持ち帰り台帳の余白に1行ずつ書いてください。Day2 の冒頭で使います。

問い	書き方
履歴を切った読み手だけが出した指摘	件数ではなく、指摘の中身を1つ書く。何も無かった場合は「無し」と書く
grep-scout と自分の仮説の差の出どころ	自分が見落とした理由を、コードの読み方の話として書く
今日の指摘のうち機械で判定できた割合	4分類のうち「即修正」に入れた件数を、指摘の総数で割る。この数字が Day2 のレビュー基準の起案の入り口になる

発展課題

手が空いた方が進めるもの

上から順に、効き方が大きい順です。全部やる必要はありません。

- 1 Step 5 で「将来課題」にした `date_from` と `date_to` の Issue を、自分で本文まで書いてください。受入条件 を機械で判定できる形にできるかが、この課題の見どころです。
- 2 `review-other` の `model:` を `opus` に書き換えて、同じ差分をもう一度読ませてください。件数と中身がどう変わるかを Step 4 の表に足します。モデルを変える効果は研究では測られていないので、ここは各自の実測になります。
- 3 同じ差分を `claude -p` に渡し、Claude Desktop で出た指摘と比べてください。 `-p` は入力欄からは実行できないので、統合ターミナルを開いて打ちます。結果をファイルに落とせる形になるので、自社で集計に載せるときの材料になります。
- 4 Claude Desktop のサイドバーで、このセッションのアーカイブアイコンを押して `worktree` を片付けてください。押す前に、GitLab 側に `worktree-issue-1` ブランチと MR が残っていることを確認します。手元が消えても送ったものは残る、という関係を見ておくと、Day2 で複数の Issue を並行させるときに迷いません。

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 <code>dl-training-app</code> ）、VSCode、GitLab のプロジェクト画面
いるファイル	<code>CLAUDE.md</code> 、 <code>.claude/settings.json</code> 、 <code>.claude/hooks/check-phpver.ps1</code> 、 <code>app/libraries/util.php</code>
直前の演習の成果物	プチ演習3 で登録した <code>PreToolUse</code> （終えていない方もこの演習は進められます）
ブランチ	<code>ex/p3-<ユーザー名></code> から <code>ex/p4-<ユーザー名></code> 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、 <code>dl-training-app</code> 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
<code>match</code> 式	<code>switch</code> か <code>if</code> の連鎖
<code>enum</code> 宣言	クラス定数を並べる
<code>readonly</code> プロパティ	<code>private</code> + 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 / strcmp / 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

そのまま「短く書き直してください」と続けてください。数ターン会話を進めてから頼むと通ります。断られることと、止まることは別です。`CLAUDE.md` は文脈なので、会話が伸びるほど制約が薄れます。

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_PROJECT_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。

settings.json の完成形と、足す演習

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "maxEffortLevel": "xhigh",
  "worktree": { "baseRef": "head" },
  "permissions": {
    "deny": [
      "Bash(git reset --hard *)",
      "PowerShell(git reset --hard *)",
      "(push --force, push -f, checkout --, clean も  
Bash と PowerShell の対で同じ形。ここまで10本)",
      "Read(/.env)",
      "Read(/.env.)",
      "Read(/.*/credentials*)"
    ],
    "ask": [ "Edit(/db/**)", "Bash(git push *)", "PowerShell(git push *)" ]
  },
  "hooks": {
    "PreToolUse": [
      { "matcher": "Bash|PowerShell", "hooks": [ <block-destructive.ps1 > ] }
    ],
    "PostToolUse": [
      { "matcher": "Edit|Write", "hooks": [ <check-phpver.ps1 > ] }
    ],
    "Stop": [
      { "hooks": [ <stop-gate.ps1, timeout 300 > ] }
    ],
    "SessionStart": [
      { "hooks": [ <sessionstart_verify.ps1, timeout 30 > ] }
    ]
  }
}
```

プチ演習1
model と effort
配布時点で入っている

プチ演習2
deny 10本

プチ演習3
deny 3本

プチ演習2 ask 3本

プチ演習3
PreToolUse

プチ演習4
PostToolUse

プチ演習6
Stop

ハンズオン3
SessionStart

<> で省いたフック1件の中身。4本とも同じ形で、違うのは最後のファイル名と、Stop・SessionStart に付く "timeout" だけです

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

丸は] や) の直後のカンマ。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。丸写しするときは配布の .claude/settings.example.jsonc から写します

.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。

```
• *h() を ?? で書き換え : Edit
• Update(app/libraries/util.php)
  Added 1 line, removed 2 lines
  6 /** HTMLエスケープの短縮名。null は空文字として扱う */
  7 function h($s)
  8 {
  9 - // 本番は PHP 5.4.45 で ?? が使えないため isset の三項で代替している
  10 + $v = isset($s) ? $s : '';
  9 + $v = $s ?? '';
  11 return htmlspecialchars((string)$v, ENT_QUOTES, 'UTF-8');
  12 }
  13 }
  Allowed by auto mode classifier
  PostToolUse:Edit hook returned blocking error

[check-php54] syntax that PHP 5.4 cannot parse:
.../app/libraries/util.php:9 null coalescing (??) -> write isset($a) ? $a : $b
Rewrite these lines with the 5.4 form shown after the arrow, then edit again.

> フックの指摘どおり、PHP 5.4 で動く形に直してください。

• *h() を PHP 5.4 の形へ戻す : Edit
• Update(app/libraries/util.php)
  Added 2 lines, removed 1 line
  6 /** HTMLエスケープの短縮名。null は空文字として扱う */
  7 function h($s)
  8 {
  9 - $v = $s ?? '';
  9 + // 本番は PHP 5.4.45 で ?? が使えないため isset の三項で代替している
  10 + $v = isset($s) ? $s : '';
  11 return htmlspecialchars((string)$v, ENT_QUOTES, 'UTF-8');
  12 }
  13 }
  Allowed by auto mode classifier
```

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。

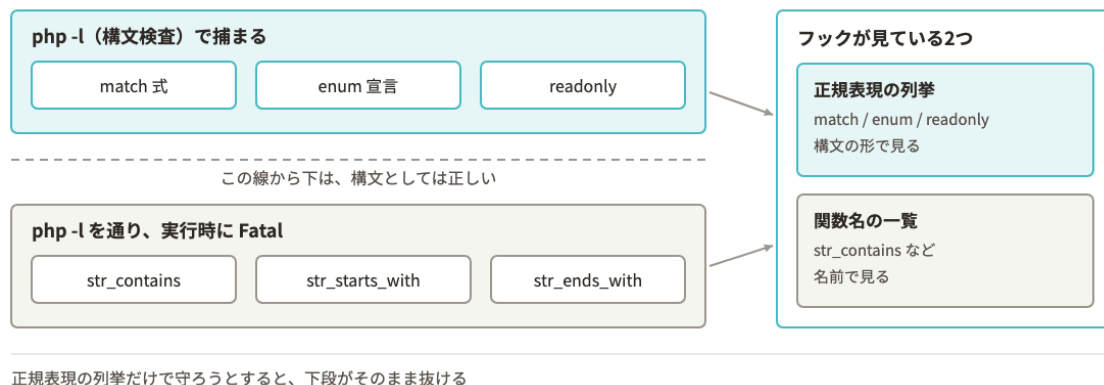
```
<?php
// str_ends_with() arrived in PHP 8.0. php -l on 7.4 does not report it.
function has_suffix($haystack, $needle)
{
    return str_ends_with($haystack, $needle);
}
```

- 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."
}
```

検査を通り抜ける層



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

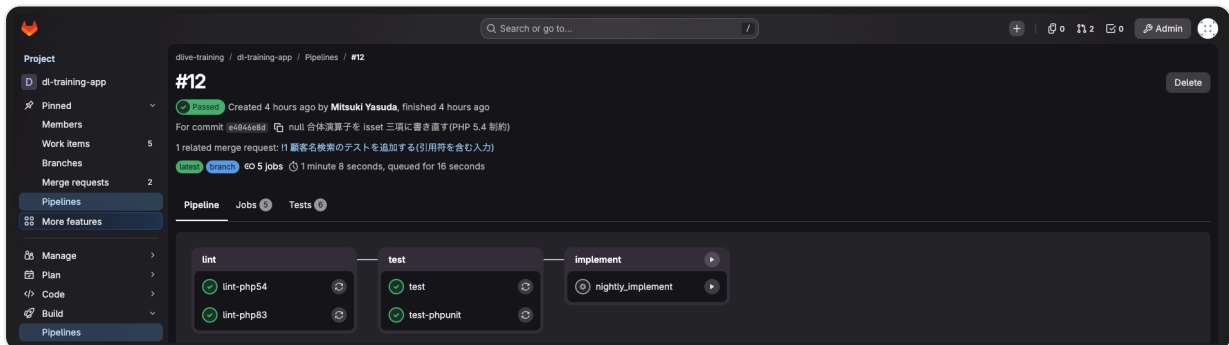
3 直前の要素の末尾にカンマが入っているかを見てください。JSON の配列は要素の間をカンマで区切り、最後の要素の後ろには付けません。

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認してください。 `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回コミットし終わったら、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

なぜこれをやるか CLAUDE.md に制約を書いても、数ターン進むと対象外の構文が戻ってきます。CLAUDE.md は文脈なので、会話が伸びるほど制約が薄れます。同じ制約を PostToolUse フックへ移すと、書き換えられた瞬間に差し戻されます。D2-01 の言い方では、[A] だけで守っているものを [M] へ移す作業です。移した後も CLAUDE.md の制約節は消しません。フックは止めるだけで、代わりの書き方までは教えないためです。

設定はどちらから使っても同じ Claude Desktop と CLI は同じ設定ファイルを読みます。`.claude/settings.json` に登録したフックと `CLAUDE.md` は、どちらで起動しても同じように効きます。Desktop 用と CLI 用に分けて書く必要はありません。

PostToolUse の動き Edit や Write でファイルを書き換えた直後に自動で走る小さなスクリプトです。終了コード 2 を返すと、標準エラーに書いた文がそのままモデルへ差し戻され、モデルは書き直しを始めます。終了コード 0 なら何も起きません。登録先は `.claude/settings.json` の "hooks" で、プチ演習3 の PreToolUse と同じ場所に並べます。

フック4イベントの位置



演習との対応: PreToolUse = プチ演習3、PostToolUse = プチ演習4、Stop = プチ演習6、SessionStart = ハンズオン3
exit 2 の stderr は人間ではなくモデルに向けて書く。理由が具体的なほど書き直しが早い

フック4イベントの位置です。上の段が1回の応答の流れで、PreToolUse はツール実行の直前、PostToolUse は直後、Stop は応答を終える直前に走ります。下の段が返し方で、PreToolUse と PostToolUse は exit 0 が素通し、exit 2 が標準エラーの文をモデルへ返す止め方です。Stop だけは終了コードではなく標準出力の `decision: block` で継続を促します。この演習で使うのは PostToolUse の行です

Mac または bash を正にする方 "command" を "bash" に、"args" を `["${CLAUDE_PROJECT_DIR}/.claude/hooks/check-phpver.sh"]` に差し替えてください。判定の中身は `.ps1` と同じです。bash 版は `jq` を使うので、入っていないと標準エラーに1行出して素通りします。テストランナーも `bash` `.claude/hooks/tests/run_cases.sh` で、出力の形は同じです。

見本ファイルに日本語を入れない理由 `.ps1` 側が CP932 で読み違えて誤検知します。`.ps1` と見本ファイルは ASCII だけで書く約束です。

症状

手当て

登録したのにフックが動かない	<code>settings.json</code> の JSON が壊れていると、ファイルごと黙って無視されます。カンマの付け忘れが大半です。VSCode の赤い波線と、止まるはずの操作が止まるかの2つを見てください。保存前に開いたセッションで試している場合は、 <code>Ctrl+N</code> で新しいセッションを開いてから試します。
編集していないのに反応しない	フックは編集の直後にしか走りません。Edit ツールの表示が出たかを先に見てください。
FAIL ... (expected 2, got 0)	見本に <code>str_ends_with(</code> が入っていないか、コメント行の中にだけ書いています。コメントは検査の前に取り除かれます。
fixture not found	ファイル名の綴りが <code>cases.json</code> と合っていません。
パイプラインが pending のまま	全員が同時に push すると順番待ちに入ります。異常ではありません。待ち方は「困ったとき」の pending の項にあります。

追加と考察 時間が余った方は、Claude に「`sed` で同じ行を書き換えてください」と頼んでください。`Edit|Write` の matcher は Bash 経由のファイル変更を拾わないため、フックは走りません。この抜け道を塞ぐなら、Stop フックで作業ツリーを走査する、`FileChanged` を使う、CI の `lint-phpver` に任せる、の3つのどれに置くかを決めて台帳に書いてください。3つ目を選ぶのは手抜きではありません。手元で全部塞ごうとすると、検査の実行時間だけが伸びます。

発展課題 PowerShell 5.1 の既定エンコーディングは CP932 で、`.ps1` に非 ASCII を1文字でも書くと実行時のメッセージが壊れます。セッション開始時の `sessionstart_verify` は非 ASCII のバイト数を数えて報告しますが、報告が出るのは次にセッションを開いたときです。ASCII だけでできているかを検査する処理を、同じ PostToolUse に足してください。対象は `.ps1` に限ります。破ると実害が出る契約は、気づける場所ではなく止まる場所に置く、という原則をフック自身へ適用する形です。

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 <code>lint-phpver</code> です
壊れると何が壊れるか	<code>settings.json</code> 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。 <code>.ps1</code> 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、重大度で可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 <code>dl-training-app</code> ）、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	<code>ci/gate.ps1</code> （ <code>bash</code> は <code>ci/gate.sh</code> ）、 <code>.claude/review.schema.json</code> 、配布物の <code>20_Day2</code> の演習/プチ演習5_レビュー出力のJSONゲート/ <code>findings_sample.json</code>
ブランチ	プチ演習4 の <code>ex/p4-<ユーザー名></code> 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 <code>>_</code> アイコン、または <code>Ctrl</code> 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、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。

```
dl-training-app % clear; bash ci/gate.sh findings.json; echo "exit=$?"
P0 app/controllers/estimate_ctl.php:23 date_from を SQL へ文字列連結しています。db_select の第2引数にプレースホルダで渡してください。
P1 app/controllers/stock_ctl.php:58 引当の4つのDB操作がトランザクションの外にあります。途中で落ちると半端な状態が残ります。
gate: blocked. 上の指摘を直すか、閾値を意図して下げてください。
exit=1
dl-training-app %
```

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

なぜこれをやるか AI のレビューを文章で受け取っている限り、通すか止めるかは毎回人が読んで決めます。出力を JSON に固定すると、そこから先は数えるだけの作業になります。同じ判定が D2-05 で CI のゲートジョブ `ai_gate` になります。

判定の置き場所 3本ありますが、考え方は1つです。 `ci/ai_gate.sh` が CI 側、 `ci/gate.ps1` が手元の PowerShell 版、 `ci/gate.sh` がその bash 版です。3本とも、入力が無い・壊れている・ `findings` の配列が無い・知らない重大度、のどれでも落とします。

findings.json を読んで合否を決める3本のスクリプト



判定スクリプト3本の関係です。入力は1つの `findings.json` で、 `gate.ps1` が手元の PowerShell、 `gate.sh` が手元の bash と Mac、 `ci/ai_gate.sh` が CI の runner で動きます。 `gate.ps1` と `gate.sh` は閾値を引数で受け、 `ai_gate.sh` は環境変数 `GATE_LEVEL` (既定 P1) と `GATE_MIN_CONFIDENCE` (既定 0.0) で受けます。どれも exit 0 で通し、exit 1 で止めます

受け取る JSON の形 必須キーは `severity`、`confidence`、`file`、`line`、`message`、`category` の6つで、それを `findings` の配列に入れます。Step 5 で指摘が 0件だった方は、判定を確かめるために手で1件書いてください。

```
{
  "summary": "設定ファイルとフックの登録を中心に4点を確認し、1件指摘しました。",
  "findings": [
    {
      "severity": "P1",
      "confidence": 0.9,
      "file": "app/libraries/util.php",
      "line": 18,
      "message": "h() の引数が未定義のときの扱いが変わっています。呼び出し側のビューで未定義キーが通るようになるため、影響範囲を確認してください。",
      "category": "spec-mismatch"
    }
  ]
}
```

```
]
}
```

bash 側で回す場合 `bash ci/gate.sh findings.json` のあと `echo $?` で同じ数字が見えます。閾値は引数の位置で `bash ci/gate.sh findings.json P2` です。jq が要ります。

症状	手当て
<code>gate: cannot parse JSON</code> で落ちる	ファイルを PowerShell の <code>></code> で書き出しています。 PowerShell 5.1 の <code>></code> は UTF-16 で書き出すので、 <code>Out-File -Encoding utf8</code> で作り直してください。
<code>gate: no findings array</code> <code>in ...</code> で落ちる	JSON の一番外側に <code>findings</code> の配列がありません。 Step 5 の書き出しでキー名が変わったか、別の 形で包まれています。
引数 <code>'findings.json'</code> を 受け入れる位置指定パラメ ーターが見つかりません	2つのコマンドが1行につながっています。別の行に 分けます。
<code>gate: findings file not</code> <code>found: 1</code>	<code>\$LASTEXITCODE</code> を同じ行に続けています。次の行で 単独で打ちます。
<code>gate: findings file not</code> <code>found: findings.json</code>	見本ファイルがまだ <code>findings_sample.json</code> の名前 のままか、ルート以外のフォルダに置かれています。 Step 1 の2と3をやり直してください。
引用符が全角になっている	チャットやメールを経由すると自動で変換されるこ とがあります。貼り付け後に半角か確かめます。

この時点の `REVIEW.md` は4つの見出しだけで中身が空です。中身を書くのはハンズオン2の Step 1 です。ここでは基準のファイルを渡す形だけを作り、判定の中身はスキーマの重大度の説明に任せます。

研修後に自分の API キーで CLI から試す場合

研修では API キーを配らないので、当日はこの経路を使いません。スキーマを守らせること自体をツール側に任せたいときの形です。 `--bare` を付けると `ANTHROPIC_API_KEY` が要ります。

Windows PowerShell 5.1 では、そのままでは2か所で壊れます。1つは文字コードで、 `git diff main` からパイプで渡す差分の日本語が `?` になります。先頭の1行で UTF-8 に揃えます。もう1つは引数の `"` で、5.1 はネイティブコマンドへ渡す引数の中の `"` をそのまま渡さないで、 `REVIEW.md` やスキーマの JSON を文字列で引数に入れると途中で割れます。

```
$OutputEncoding = [Console]::OutputEncoding = New-Object System.Text.UTF8Encoding $false
git diff main | claudes --bare -p "この差分をレビューし、指摘をスキーマの形で返してください。" --output-format json --json-schema ((Get-Content .claude\review.schema.json -Raw -Encoding UTF8) -replace "'", '"') | ConvertFrom-Json | Select-Object -ExpandProperty structured_output | ConvertTo-Json -Depth 10 | Out-File -Encoding utf8 findings-cli.json
```

`--output-format json` の出力は外側に包みがあり、指摘は `structured_output` の中に入ります。包みのまま判定スクリプトに渡すと、`findings` の配列が無いので落ちます。

追加と考察 同じ差分を、履歴を切った新しいセッションでもう一度レビューさせてください。読むのは1点だけです。同じ差分で重大度が揺れていないかどうか。揺れているなら、揃えるのは基準の文面です。D2-05 の前半で Important の定義を書くときに、ここが1つ決まります。

発展課題 2つあります。1つ目は確信度の下限です。いまの `gate.ps1` は `confidence` を見ていません。 `param` に `[double]$MinConfidence = 0.0` のような引数を足し、重大度を数える前に下限に満たない指摘を除き、除いた件数を1行出す形にしてください。見本ファイルで下限を 0.8 にすると、確信度 0.5 の1件が外れます。CI の `ai_gate.sh` は `GATE_MIN_CONFIDENCE` で同じことをしています。2つ目は、閾値に届かなかった指摘の残し方です。P2 以下を MR のコメント1本にまとめて残す処理を考えてください。指摘のたびにコメントを増やすと、小さな改修ほど読む量が増えるという、いま解こうとしている課題そのものを再生産します。

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。

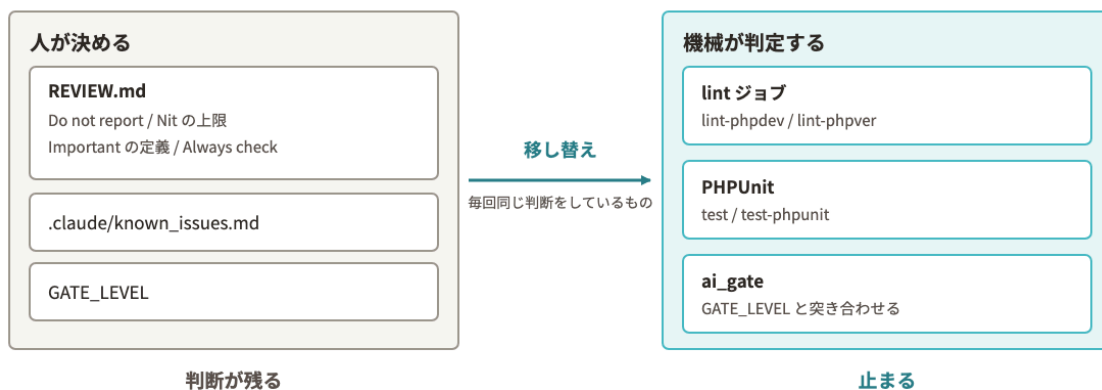
目的 指摘の絞り込みと、機械判定への移し替え

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本のレビューア、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。

人が決めるものと、機械が判定するもの



機械へ移せるのは、毎回同じ結論になる判断だけ。残りは人の側に置いたままにする

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本のレビューアの 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、観点別レビューア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	<code>allow_failure</code> を外すと、ゲートの赤で MR 用のパイプラインが赤になる。 <code>GATE_LEVEL</code> 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 (<code>app/controllers/stock_ctl.php</code> の <code>assign()</code>)	Step 2 で3本のレビューアーを当てる対象。実装はしません
#3	見積一覧の期間検索と CSV 出力 (<code>app/controllers/estimate_ctl.php</code> の <code>index()</code>)	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。 <code>REVIEW.md</code> と既知の許容の台帳を書く	[10min]
Step 2	観点を分けた3本のレビューアーを、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

`ex/h2-<ユーザー名>` はプチ演習4・5のブランチ `ex/p4-<ユーザー名>` から切るので、Day1 とプチ演習4・5で足した deny とフックがそのまま乗ります。この後のプチ演習6は、新しく切らずにこの `ex/h2-` のまま進めます。main から切ると、それが全部消えます。プチ演習の変更が残っている場合は、先に `ex/p4-<ユーザー名>` 側でコミットしてから切ってください。ブランチの名前は共通の進め方 STEP 2、コミットと同期は STEP 3、MR の出し方とパイプラインの見方は STEP 4 にまとめてあります。

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

(1)で Day1 の宿題が手元に無い方は、演習リポジトリの `CLAUDE.md` の「実行環境の制約」の表と「守ること」の節を開き、そこから機械で検査できそうなものを3つ選んでください。(2)は、同じ判断を毎回繰り返しているものが台帳に載せる候補です。(3)は入れると指摘が増えます。増えた分を誰が引き受けるかまで含めて決めてください。4つとも正解はありません。この演習では、決めた内容そのものより「決めたものがMRで機械に効いているか」を見ます。

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

自分で書いてから開いてください。この題材で出発点になるのは次の3つです。

- ☐ 外部入力(`$_GET` / `$_POST`)が、プレースホルダを通らずに SQL の条件へ入っている
- ☐ 複数の表を続けて書き換える処理が、トランザクションと行ロックの外にある
- ☐ 画面へ出す値が `h()` を通っていない

解説と補足 対象バージョンの書き方

このプロジェクトの対象は PHP 7.4 です。AI は素で 8.x の構文(`match` 式、 `enum` 、 `readonly` 、名前付き引数、コンストラクタプロモーション、 `str_contains` / `str_starts_with` 、 `?->`)を書きたがるので、対象の版を設定に書き、それを超える構文を機械で止める形にします。

構文と関数は別の段で止まります。 `str_contains()` の呼び出しは構文としては正しいので `php -l` を通ります。そのため CI の `lint-phpver` は2段で見えています。1段目の `php -l` が `match` 式などの構文を、2段目の `php ci/undefined_functions.php` が 7.4 の実行環境に関数があるかを `function_exists()` で確かめ、 `str_contains()` はこの2段目で落ちます。機械が止めている範囲なので、AI のレビューに重ねて言わせる必要はありません。 **Important の定義** ではなく **Do not report** に入れる理由です。

リプレイスの予定があるときは、移行先の版を対象に書きます。 現行が古くても、移行先の版を `CLAUDE.md` と `REVIEW.md` に書いておけば、これから書かれるコードが移行先で動く形になります。現行の版を対象に書き続けると、移行のときに書き直す量が増えます。

Important に何を入れるか決まりません

Issue #2 と #3 の「現状」節を読み直してください。そこに書かれている事実は、既に分かっている欠陥です。それを検知できる書き方になっているかどうかで、Important の粒度を判定できます。「SQL の書き方に注意する」では検知できません。「外部入力」が `db_select` の第1引数に文字列連結で入る変更」であれば検知できます。それでも迷う場合は、 `AGENTS.md` の **Code Review Rules** にある「データ境界」「互換性」「整合性」の3つを Important の骨にして、自社の言葉で書き直してください。

影響範囲： `REVIEW.md` はこの後のレビューア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。

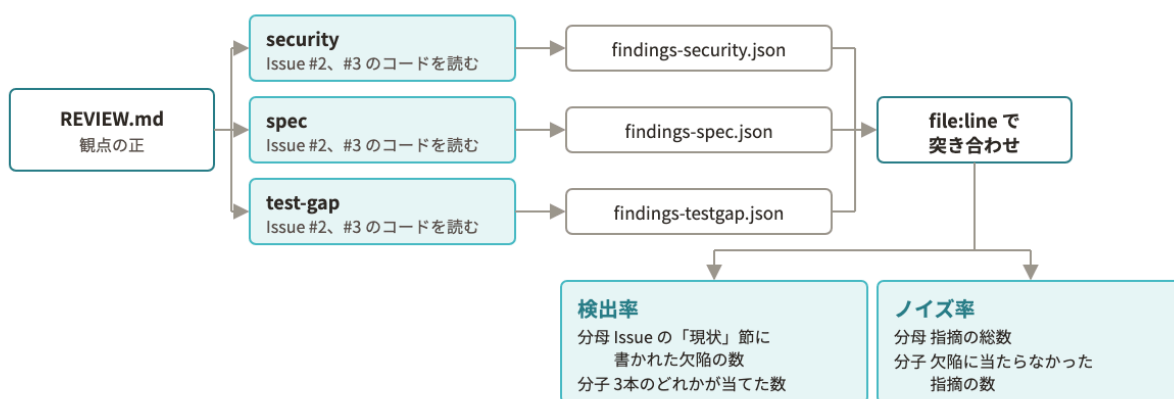
ハンズオン2 Step 2 観点別レビュアーの並列適用

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。

ハンズオン2の流れ



ゲートを厳しくする

GATE_LEVEL を P2、allow_failure を外す

MR で ai_gate が落ちる

直すか、直さない理由をコメントに書く

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。
出し終わったら、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。

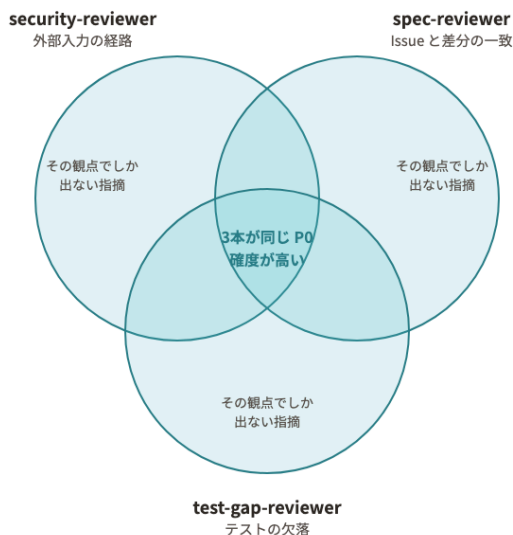
7 検出率とノイズ率を出します。検出率は、Issue #2・#3 の「現状」節に挙がっている欠陥の件数を分母、当てた件数を分子にします。欠陥として数えるのは、直さないと誤動作するものです。「条件の組み立てが index() の中にしかない」「CSVの入口がまだ無い」のような設計と未実装の話は数えません。ノイズ率は、指摘の総数を分母、仕込み欠陥以外の件数を分子にします。何を分母に入れたかも記録に残してください。

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本のレビューアの重なりと差



2つの率

検出率 = 当てた欠陥
÷ 仕込み欠陥

ノイズ率 = 欠陥以外の指摘
÷ 指摘の総数

重なること自体は失敗ではない。何を分母に入れたかを記録に残す

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。

security-reviewer, spec-reviewer, test-gap-reviewer の3つのサブエージェントを並列で起動して、app/controllers/stock_ctl.php の引当処理をそれぞれの観点でレビューしてください。

```
• Running 3 agents.
  security-reviewer (引当処理のセキュリティ観点レビュー)
  spec-reviewer (引当処理の仕様一致レビュー)
  test-gap-reviewer (引当処理のテスト有無レビュー) • 0 tool uses
  Initializing...

* Embellishing... (25s • 1 671 tokens)
  Tip: Use --agent <agent_name> to directly start a conversation with a subagent

• main
  o security-reviewer 引当処理のセキュリティ観点レビュー
  o spec-reviewer 引当処理の仕様一致レビュー
```

10s • 1 18.1k tokens
4s • 1 14.4k tokens

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け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

実行中

Issue2 独立レビュー

エージェント 3m04s

Sonnet 5 42.7k トークン 21 回のツール使用 読み込み中
[トランスクリプトを表示](#)

完了 3

Issue2 仕様整合レビュー

エージェント 完了 2m34s

Sonnet 5 45.7k トークン 18 回のツール使用
[トランスクリプトを表示](#)

Issue2 テスト網羅レビュー

エージェント 完了 1m01s

Haiku 4.5 27.4k トークン 12 回のツール使用
[トランスクリプトを表示](#)

Issue2 セキュリティ観点レビュー

エージェント 完了 1m00s

Opus 5 30.3k トークン 14 回のツール使用
[トランスクリプトを表示](#)

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、レビューア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

ここで使うサブエージェントは、`.claude/agents/` の定義ファイル1本につき1つ立ち上がる、別の Claude です。先頭の frontmatter(--- で囲まれた部分)でモデルと使えるツールが決まり、本文がその Claude への指示になります。呼び出した側の会話の履歴は見え、渡された範囲だけを読んで結果を返します。3本を1つの依頼で指名すると同時に走るの、これを並列と呼んでいます。

返ってくる形は `.claude/review.schema.json` のとおりです。レビュアー1本ぶんは、こう見えていれば正しく動いています。

```
{
  "summary": "外部入力を経路を 6 箇所追い、到達するものが 3 件ありました",
  "findings": [
    {
      "severity": "P0",
      "confidence": 0.9,
      "file": "app/controllers/estimate_ctl.php",
      "line": 23,
      "category": "sql-injection",
      "message": "date_from を SQL へ文字列連結しています。db_select の第2引数にプレースホルダで渡してください"
    }
  ]
}
```

前置きや後書きが付いている、`category` が全部 `other` になっている、といった場合は frontmatter ではなく本文の「返す形」節を読み直してください。手順2で消しすぎた可能性があります。

手順8 で書いた振り分けは、AI が読めば従いますが、読まなければ従いません。守らせたいのであれば Step 3 の CI 側に置いてください。手元の設定でできるのは速報までで、合流を止められるのは CI だけです。行数を `git diff --stat main` の末尾の数字で数える場合は、**ここはコマンドを使います**。Claude Desktop の中のターミナルで打ってください。

3本が並列にならず、1本ずつ順番に動きます

1つのプロンプトで3本を指名しているかを確認してください。3回に分けて頼むと直列になります。それでも直列になる場合は、順番に動いても演習は成立します。所要時間の記録だけ「直列」と書き添えて先へ進んでください。

3本とも同じ指摘しか出しません

「見るもの」節が3本とも似た内容になっている可能性があります。security-reviewerは外部入力経路だけ、spec-reviewerはIssueと差分の一致だけ、test-gap-reviewerはテストの有無だけを見ます。「他の観点は書きません」の1行が消えていないかを確認してください。

重なること自体は失敗ではありません。3本が同じ P0 を出したのであれば、それはその欠陥の確度が高いという情報です。重なる数も記録に残してください。

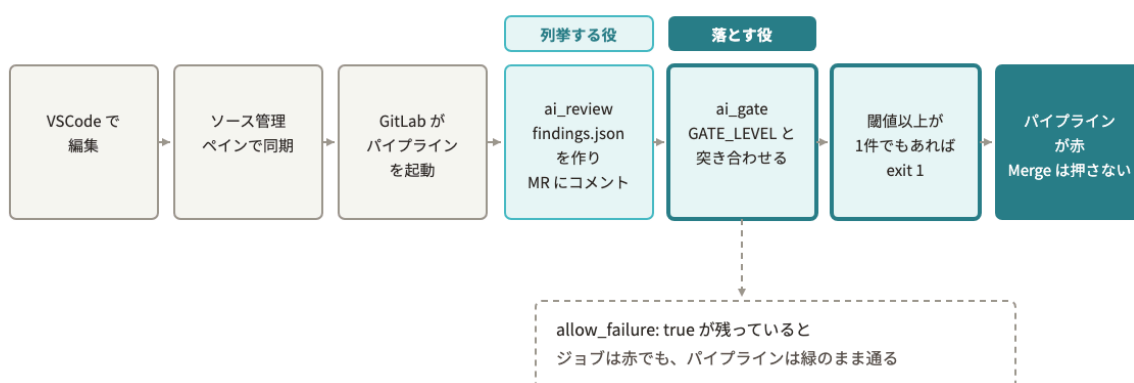
影響範囲：観点を分けると1本あたりの指摘は減り、3本を足した総数は増えます。自社に写すときは agents/ に3本を置き、どの変更でどれを呼ぶかを規約側に1行足します。3本とも常に呼ぶ運用にすると、レビュー待ちの時間が3倍になります。

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



指摘を並べる役と、合流を止める役は別のジョブになっている

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で消す行に当たり、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。

.gitlab-ci.yml で触る2か所

ブロック1: 先頭付近の variables

```
variables:
  AI_REVIEW_DIFF_THRESHOLD: "80"
  AI_REVIEW_MODEL_SMALL: "claude-sonnet-5"
  AI_REVIEW_MODEL_LARGE: "claude-opus-5"
  GATE_LEVEL: "P1"
```

ハンズオン2 手順3
"P1" を "P2" に変える

ブロック2: # --- gate --- の下の ai_gate ジョブ

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

縦の薄線は空白2個ぶんの幅
キー名の前の空白数で階層が決まる

ハンズオン2 手順2
この1行を消す

YAML はインデントの空白数がずれると全ジョブが止まる。タブは使わず、空白2個で揃える

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。

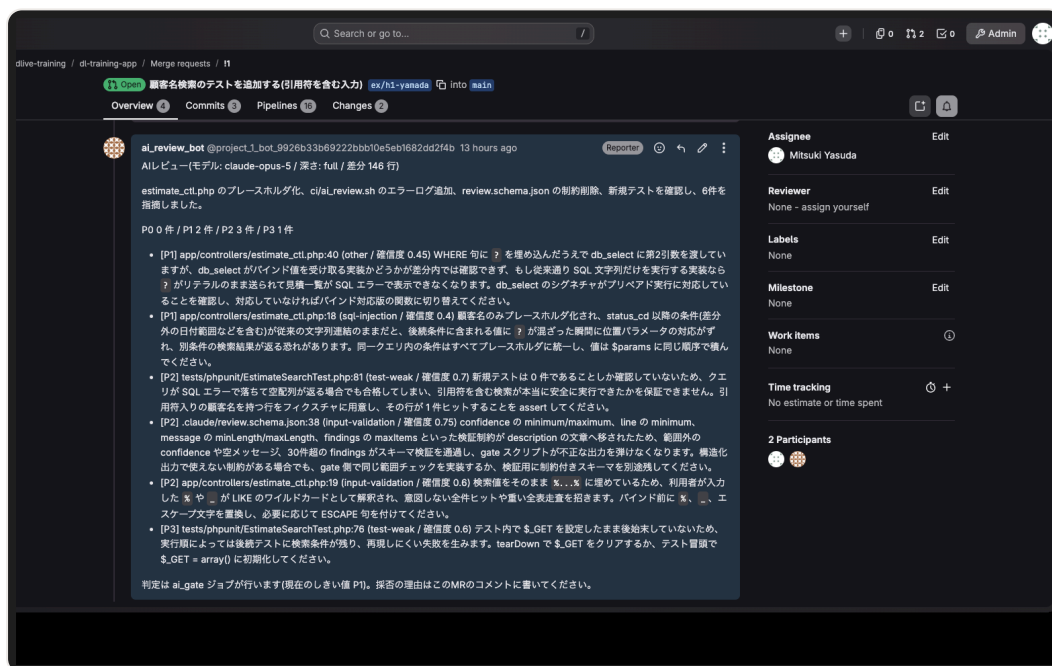
- 8 ブラウザで GitLab の演習プロジェクトを開き、画面上部に出る **Create merge request** を押します。出ていない場合は左のメニューの **Code** から **Merge requests** を開き、

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。

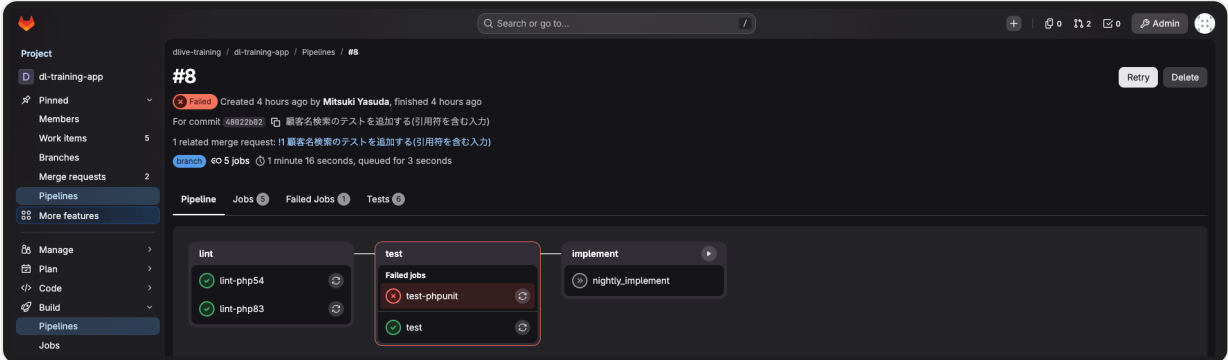


手順11でMRの **Overview** タブに付く ai_review_bot のコメントです。1行目に使ったモデル、レビューの深さ、読んだ差分の行数が出ます。その下が P0 から P3 の件数で、続けて指摘が [P1] ファイル:行 (カテゴリ / 確信度) の形で並びます。差分が AI_REVIEW_DIFF_THRESHOLD (既定 80)を下回ると軽いモデルに切り替わり、1行目のモデル名が変わります。判定はこのコメントではなく ai_gate ジョブが行います

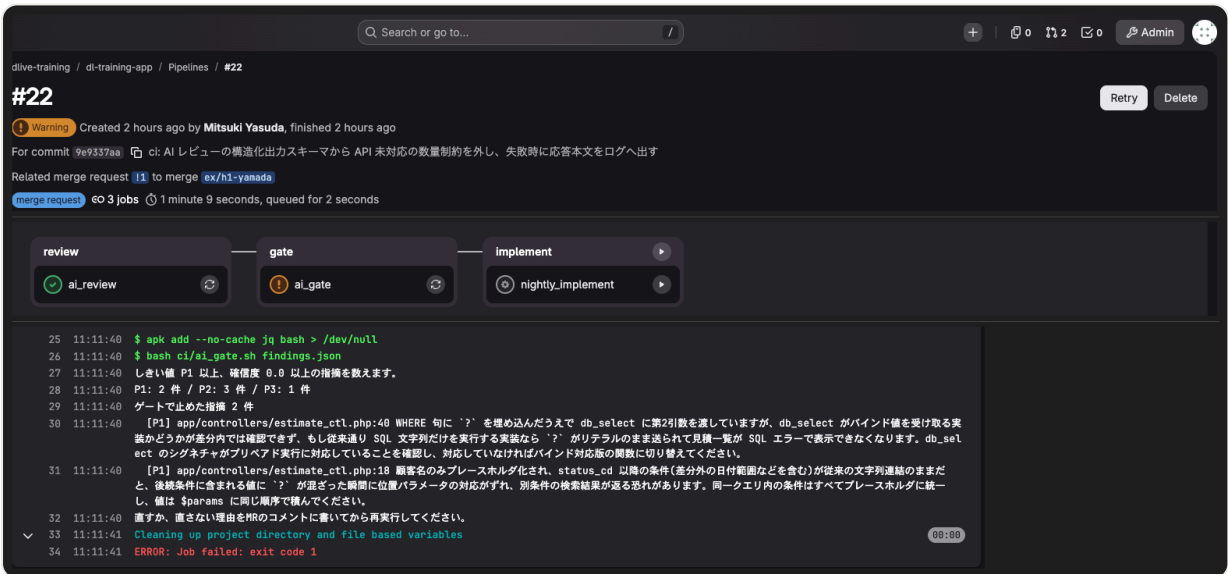
12 落ちた指摘を1件ずつ、即修正・要調査・将来課題・意図した設計の4つに仕分け、MR の説明の「人が判断した採否」の表に書きます。直すものを直し、「意図した設計」にしたものは **Do not report** に入れるかどうかを検討します。

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

手順9でジョブのログを開くと、こう見えます。件数は環境によって変わります。

```
1) EstimateSearchTest::testIgnoresMalformedDateFrom
Failed asserting that 3 matches expected 15.

/builds/dlive-training/dl-training-app/tests/phpunit/EstimateSearchTest.php:8
1

FAILURES!
Tests: 6, Assertions: 9, Failures: 1.
```

手順11で `ai_gate` が落ちたときの出力です。MR テンプレートの「AI レビュー結果」欄のコードブロックに、この出力をそのまま貼ってください。

```
しきい値 P2 以上、確信度 0.0 以上の指摘を数えます。
confidence の無い指摘は 1.0、P0～P3 以外の severity は P0 として数えます。
P1: 1 件 / P2: 3 件 / P3: 2 件

ゲートで止めた指摘 4 件
[P1] app/controllers/estimate_ctl.php:36 date_to が条件に使われず、範囲の上限が効きません
[P2] app/controllers/estimate_ctl.php:41 条件の組み立てが index() の中にし
    かりません
    ...

直すか、直さない理由をMRのコメントに書いてから再実行してください。
ERROR: Job failed: exit code 1
```

`ai_gate` が赤になるのは、CI に `ANTHROPIC_API_KEY` が入っていて、しきい値以上の指摘が1件でも出たときです。

手元で先に同じ判定をかける方法

ここはコマンドを使います。判定スクリプトは画面から実行できないので、**Claude Desktop** 中のターミナルから実行します。Step 2 で作った findings をそのまま渡せます。

```
powershell -NoProfile -ExecutionPolicy Bypass -File ci\gate.ps1 -Path findings-s-security.json -Threshold P2
```

bash 環境では `bash ci/gate.sh findings-security.json` です。判定の考え方は CI の `ci/ai_gate.sh` と同じなので、手元で落ちたものは CI でも落ちます。出力の書式だけが違います。

runner から外部の API へ出られない場合も、この形で進めます。 (1) 手元で findings を JSON に落とす (2) 上のコマンドを `-Threshold P2` で実行する (bash 環境では `GATE_LEVEL=P2 bash ci/ai_gate.sh findings.json`。 `ai_gate.sh` は閾値を引数ではなく環境変数で受けます) (3) 出力を MR の「AI レビュー結果」欄に貼る。ゲートで1回落ちた記録が MR に残れば合格です。 `ai_review` と `ai_gate` の実走は講師環境で行います。

ai_review ジョブが動きません

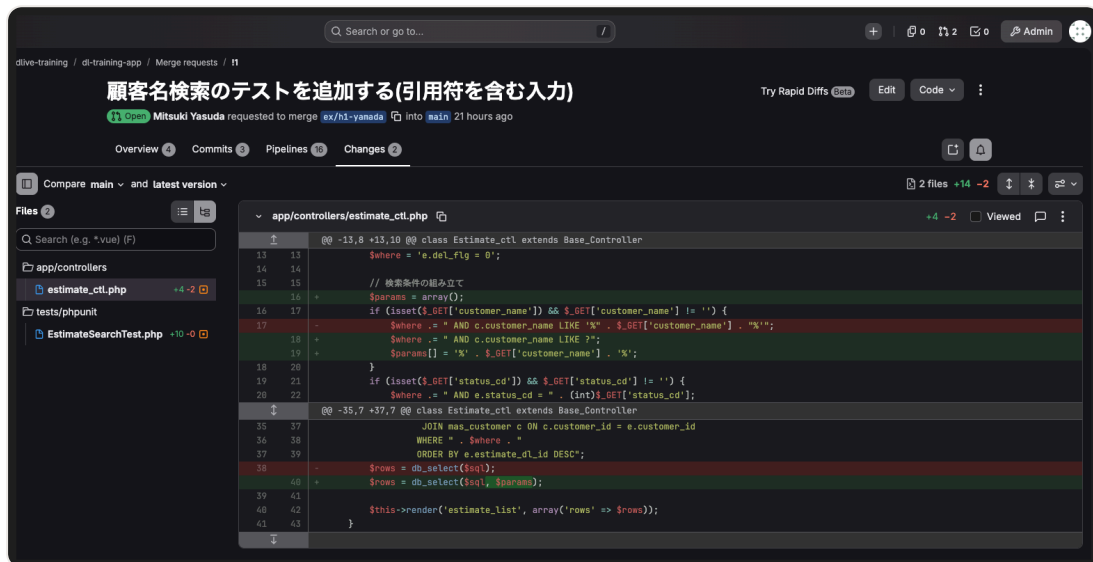
このジョブは `merge_request_event` のときだけ動きます。ブランチを送っただけでは走りません。MR を作ってから、もう一度送ってください。MR があれば、**Pipelines** タブに `merge request` のラベルが付いた行が増え、そこに `ai_review` と `ai_gate` が並びます。

ジョブは緑なのにコメントが付かない場合は、ジョブのログを開いてください。

`[ai_review]` CI/CD変数 `ANTHROPIC_API_KEY` が未設定のため、レビューを実行せずに終了します と出ていれば変数の問題です。このときジョブは赤にならず、`findings.json` は指摘0件で残り、`ai_gate` も緑になります。講師までお知らせください。受講者側の設定では直せません。

ai_gate が緑のままで、一度も落ちません

先に、`.gitlab-ci.yml` の変更がブランチに載っているかを見てください。GitLab の MR の **Changes** タブに `.gitlab-ci.yml` が出ていなければ、手順6 のコミットが抜けています。次に `allow_failure: true` の行が残っていないかを確認してください。残っていると、ジョブは赤くなりますがパイプラインは緑のまま通り、ジョブ名の横に **Allowed to fail** と出ます。



Changes タブの見方です。左の **Files** にこの MR で変わったファイルが全部並び、ファイル名の右に追加と削除の行数が出ます。右が選んだファイルの差分で、**+** の行が追加、**-** の行が削除です。ここに `.gitlab-ci.yml` が無ければ、手順6 のコミットがブランチに載っていません

`ai_review` のログに `ANTHROPIC_API_KEY` が未設定 と出ている場合は、指摘が0件なので落ちません。その間は、上の「手元で先に同じ判定をかける方法」で進めてください。

行を消しても落ちない場合は、`GATE_LEVEL` を `"P2"` にして回し直してください。それでも落ちないのであれば、`REVIEW.md` の **Do not report** が広すぎます。何を書かせないと思ったかを読み直すところが、この演習の本題です。

test-phpunit が赤ではなく、エラーで止まります

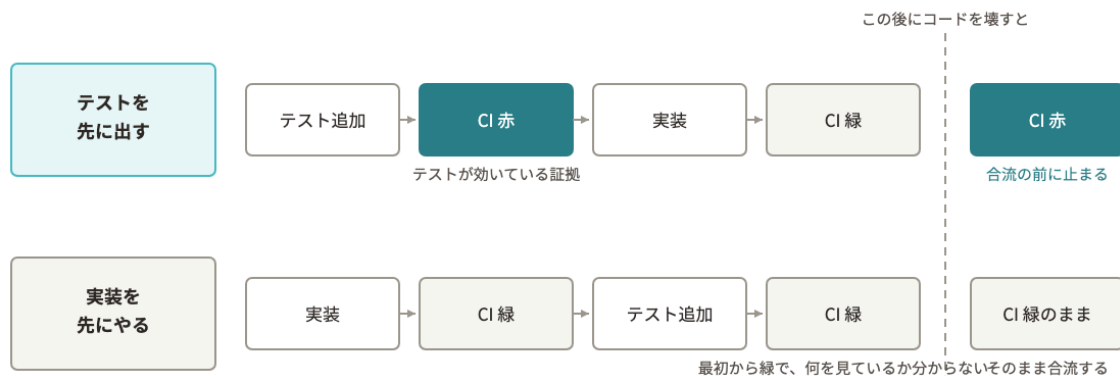
テストが落ちる(Failures)のではなく、実行そのものが止まっている(Errors)場合は、足したテストの書き方が既存の5本と違っています。CI の `test-phpunit` は PHP 7.4 と PHPUnit 7.5.20 で動きます。ジョブのログで止まった行を読み、既存の5本と同じ書き方にそろえる、という指示をプロンプトに入れ直してください。

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。

テストを出す順番で変わるもの



差はテストの本数ではなく、壊れたときに止まる場所があるかどうか

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 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ります。

ファイル	D2 での置き場所	smart3pm での置き場所
<code>REVIEW.md</code>	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
<code>.claude/known_issues.md</code>	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
<code>.claude/agents/</code> の3本	<code>agents/</code> に追加。PowerShell に依存しないのでそのまま置ける	<code>agents/</code> に追加。同上
<code>.claude/skills/review-route/SKILL.md</code>	<code>skills/</code> に追加。既存の軽量パスの節と書き方をそろえる	<code>skills/</code> に追加

<code>.gitlab-ci.yml</code> の <code>ai_review</code> と <code>ai_gate</code>	CI 側へ新規。機械強制を定義どおりに戻す位置づけ	<code>pre-receive</code> の検査とは別に、CI 側へ新規
判定スクリプト	<code>ci/gate.ps1</code> (PowerShell 5.1、ASCII のみ)	<code>ci/gate.sh</code>
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ない。

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、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	<code>.claude/settings.json</code> 、 <code>.claude/hooks/stop-gate.ps1</code> （ <code>bash</code> は <code>stop-gate.sh</code> ）
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 <code>ex/h2-<ユーザー名></code> のまま進めます。新しく切りません。 <code>ex/h2-</code> は <code>ex/p4-</code> から切っており、プチ演習4 の PostToolUse も入っています。ハンズオン3 の <code>ex/h3-</code> 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 <code>php --version</code> 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。 <code>stop-gate</code> 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。

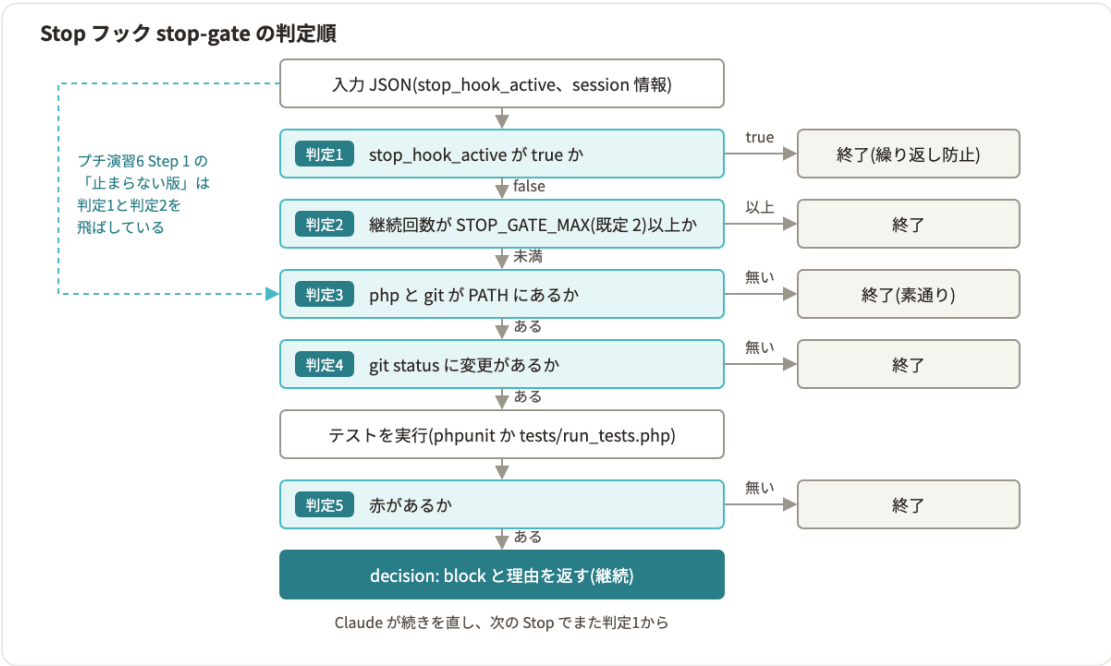
```
# demo only. do not register this one.
$payload = [Console]::In.ReadToEnd() | ConvertFrom-Json
Write-Output '{"decision":"block","reason":"tests are red. fix and finish."}'
```

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]::IsNullOrEmpty(($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以降に進みます。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続きのたび 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）

```
• Ran 3 stop hooks
  └─ Stop hook error: php tests/run_tests.php failed with exit code 1. Run it, read the failing case, fix the code, then finish.

• 同じ hook がまた発火しましたが、これは想定どおりです。先ほど確認したとおり、このゲートは解除できません。

• Ran 3 stop hooks
  └─ Stop hook error: php tests/run_tests.php failed with exit code 1. Run it, read the failing case, fix the code, then finish.

• 診断済みの内容から変わりません。同じ説明を繰り返すだけになるので、ここで止めます。

• Ran 3 stop hooks
  └─ Stop hook error: php tests/run_tests.php failed with exit code 1. Run it, read the failing case, fix the code, then finish.

•

+ Doodling... (1m 40s · 3.0k tokens)
  └─ Tip: Use /feedback to help us improve!
```

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_PROJECT_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。

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

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

なぜこれをやるか 夜間に無人で回す構成では、応答を終える直前に走る Stop フックが最後の門になります。テストが落ちたまま終わらせない、という一言を機械にするのがこの演習です。あわせて、このフックが自動化で一番事故を起こす部品でもあることを、講師の画面で先に見ます。止まらなくなる原因は1か所です。フックが自分で継続させたセッションかどうかを見ずに止めると、そのたびに継続が入り、同じ判定がまた走ります。

Stop フックの動き 応答を終えようとした瞬間に走ります。標準出力に

```
{"decision":"block","reason":"..."}
```

を返すと応答は終わらず、`reason` の文が次の指示として渡って作業が続きます。何も返さなければそのまま終わります。継続で再び呼ばれたときは、入力の JSON に `stop_hook_active` が `true` で入ります。プチ演習3の PreToolUse、プチ演習4の PostToolUse、この演習の Stop で、手元のフックは3イベントになります。4本目の SessionStart はハンズオン3の Step 1 で足します。

タイムアウト フックの既定はコマンド種別で600秒で、ここではテストの実行時間に合わせて300秒へ縮めています。自社の検査がこれを超える場合は、フックではなく CI 側へ寄せてください。

Mac または bash を正にする方 `"command"` を `"bash"` に、`"args"` を

```
["${CLAUDE_PROJECT_DIR}/.claude/hooks/stop-gate.sh"]
```

に差し替えてください。手動検査も `bash .claude/hooks/stop-gate.sh` に、`$LASTEXITCODE` を `echo $?` に読み替えます。

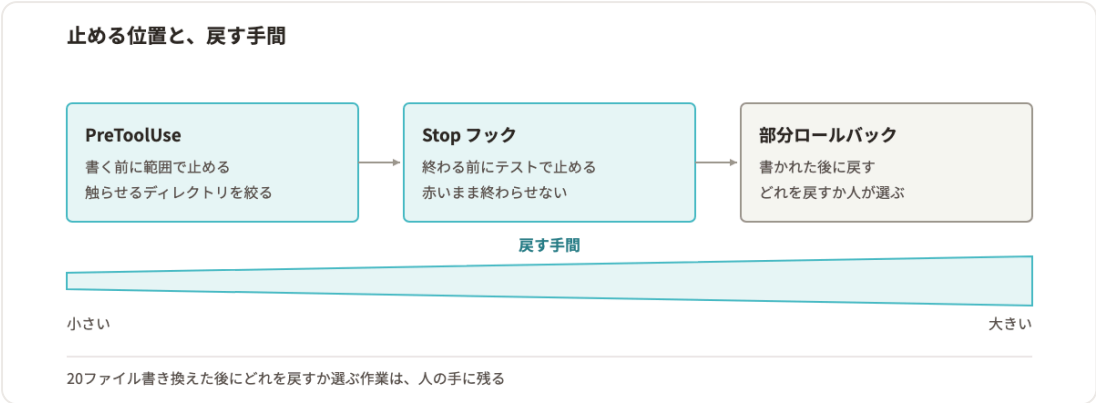
素通りの1行が画面に出ないとき 標準エラーの1行は、通常の画面には出ないことがあります。素通りの経路は「何も起きない」が正しい姿なので、見えないこと自体は不具合ではありません。フックだけを確認めたいときは、Step 2 の手順2の JSON の `true` を `false` に変えて `stop-gate.ps1` に流すと、統合ターミナルに標準エラーの行が出ます。

継続回数の記録 セッションごとに `.claude/hooks/.stop-gate.state` へ書かれ、テストが通ると消えます。手元の作業メモなので、コミットしないでください。ソース管理ペインにこのファイルが出たら、`.gitignore` に1行足します。回数を数えた文が出るのは、同じセッションで継続が積み上がったときです。1本の連鎖では `stop_hook_active` が必ず1回で切るので、既定の2に当たるのは同じセッションで依頼を繰り返した先になります。起動し直すと `session_id` が変わり、回数は0に戻ります。

追加と考察 Stop フックは、応答を終える判断を機械に移す部品です。移した結果、人の側に何が残るかを1行書いてください。夜間ループで人が持つのは Issue の粒度決め、マージ承認、指摘の採否の3つです。この3つのどれかが、テストの合否判定で置き換わっていないかを確かめてください。置き換わっているなら、それは機械化ではなく、判断の放棄です。

発展課題 いまの版はテストの合否だけを見えています。ここに `lint-phpver` 相当の検査を足すと、対象バージョンで動かない構文を残したまま応答が終わることも防げます。ただし、応答のたびに2つの検査が走ります。会話の速度と、合流前に気づけるこ

とのどちらを取るかを決めてから足してください。足さない判断も、理由を書けば持ち帰り物になります。



3つとも同じ事故を止められますが、止める時点が違います。左の2つは書かれる前と終わる前に効くので、戻す作業そのものが起きません。右端はすでに書かれた後なので、どれを戻すかを選ぶ手間が人に残ります。下の帯の太さが、その手間の差です。

変更範囲の事前制限 Stop フックが止めるのは、終わり方です。戻す手間そのものを減らすなら、先に範囲を絞るほうが効きます。AI が20ファイルを書き換えた後に、どれを戻すか選ぶ作業は人の手に残ります。絞り方は2つあります。1つは PreToolUse で、決めたディレクトリの外を書こうとしたら止める形です。もう1つは Stop フックの判定に「作業ツリーの変更ファイル数」を足し、決めた数を超えたら継続させずに止める形です。どちらも、部分ロールバックと対で考えるものです。上限の数は、1回の依頼で触らせるつもりファイル数から決めてください。普段の改修で5ファイルなら、8や10あたりが最初の線になります。超えたときに止まるのは事故ではなく、依頼が大きすぎた合図です。

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	<code>stop_hook_active</code> 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	<code>.claude/hooks/.stop-gate.state</code> 。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、 <code>#!/bin/bash</code> の行末に <code>\r</code> 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（ <code>jq</code> 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で、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
<code>stop-gate.ps1</code> と <code>stop-gate.sh</code>	<code>.ps1</code> 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	<code>.sh</code> 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
<code>settings.json</code> 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（ <code>STOP_GATE_MAX</code> 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。

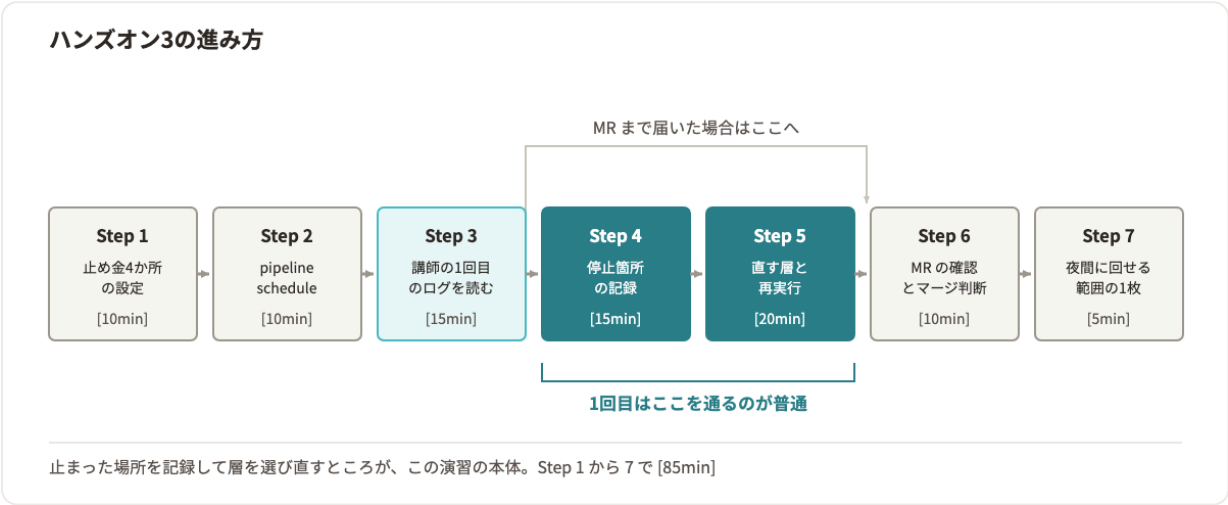
- 1 Issue 1本を無人で実装させ、MR が立つところまでを自分の手で1回通す。
- 2 1回目に止まった箇所を記録し、止め金の4層のどれで直すかを選んで再実行する。
- 3 今のハーネスで夜間に回せる範囲と、人が残る3点を自分の言葉で1枚に書く。

準備 始める前にそろっている状態

開いておく画面	Claude Desktop(<code>dl-training-app</code> 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	<code>.claude/settings.json</code> (deny 13本とフック4本)、 <code>.gitlab-ci.yml</code> の <code>nightly_implement</code> 、 <code>ci/nightly_implement.sh</code> 、 <code>.claude/skills/implement-issue/SKILL.md</code>
作るもの	ブランチ <code>ex/h3-<ユーザー名></code> 、pipeline schedule 1本(Inactive)、MR 1本、計測表の「Day2 夜間ループ」の節
ブランチ	<code>ex/h2-<ユーザー名></code> から <code>ex/h3-<ユーザー名></code> を切ります。ハンズオン2までの <code>deny</code> 、 <code>PreToolUse</code> ・ <code>PostToolUse</code> ・ <code>Stop</code> の3本、 <code>REVIEW.md</code> 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 <code>turns</code> と <code>cost_usd</code> 、artifacts の <code>ci_logs/</code> 、MR のパイプライン、 <code>Code</code> > <code>Branches</code> の <code>main</code> の行
所要	[105min]。前提確認と「考える」で [5min]、Step 1 から Step 7 で [85min]、まとめて [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。



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回目を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

本文のコマンドは Windows の PowerShell で書いています。違いは3つです。複数行に分けたコマンドの行末は、バッククォートではなく `\` にします。フックは `.ps1` ではなく `.claude/hooks/` の同じ名前の `.sh` を使い、`settings.json` では `"command": "bash"` と `"args": ["${CLAUDE_PROJECT_DIR}/.claude/hooks/<名前>.sh"]` の形です。パスの区切りは `/` です。GitLab の画面操作と Claude Desktop の操作は同じです。

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	<code>.claude/settings.json</code> の <code>permissions.deny</code> 。文字列照合なので <code>bash -c</code> 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	<code>--max-turns</code> とジョブの <code>timeout</code> 。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

3つの操作と4つの層を突き合わせると、2層で止めている操作と、どこでも止めていない操作が出てきます。後者は Step 4 で必ず踏みます。層の強さが違うので、重なっていること自体は問題ではありません。問題になるのは、どこでも止まらない操作を「たぶん大丈夫」で流すことです。

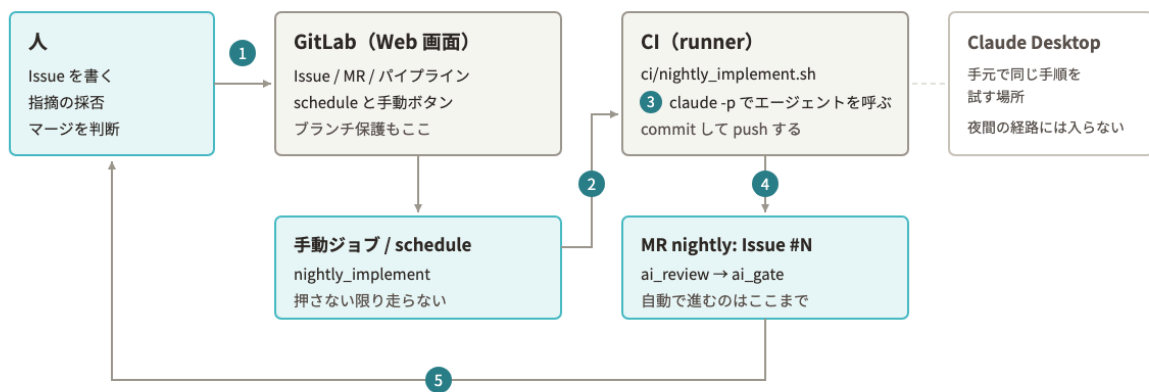
部品

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

夜間ループの登場人物



1 人が Issue を書く 2 schedule か手動ボタンでジョブが起動する 3 CI がエージェントを呼ぶ
 4 CI が push して MR が立つ 5 人がマージを判断する
 1回目は止まる前提。どこで止まったかを台帳に書くのが演習

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

[schedule または手動実行]

```
implement
nightly_implement
ci/nightly_implement.sh
└ ブランチ nightly/issue-4 を作る
└ claude -p "/implement-issue 4" --permission-mode dontAsk --max-turns 15
└ コミット(エージェントが済ませていなければスクリプト側で)
  └ push → ここで MR のパイプラインが動く
           マージだけ人が判断する
```

エージェントに渡しているツールは `Read,Edit,Write,Grep,Glob,Bash(PHP *)`, `Bash(git add *)`, `Bash(git commit *)` だけです。ブランチ作成と `push` はスクリプトの仕事なので、権限を渡していません。そのぶん、SKILL.md の手順3(ブランチ作成)と手順7(`push`)を実行させると、そこで拒否されて止まります。飛ばす指示はスクリプトがプロンプトに添えています。手順1の Issue 取得(`curl`)も渡していないので、ここでも止まります。SKILL.md が意図した止め金として書いている箇所です。

artifacts には `ci_logs/claude-issue-N.json` と `.err` が残り、`num_turns`、`total_cost_usd`、`permission_denials` を読めます。

この演習で初めて出てくる語

schedule	GitLab が決めた時刻にパイプラインを起動する仕組み (pipeline schedule)。rules の <code>\$CI_PIPELINE_SOURCE == "schedule"</code> は「schedule から起動されたときだけ自動で走る」の意味です
手動ジョブ	rules の <code>when: manual</code> が付いたジョブ。パイプラインの一覧に再生ボタンの形で出て、人が押すまで走りません
timeout: 30m	ジョブが30分を超えたら GitLab が強制終了します。エージェントが堂々巡りしたときの最後の打ち止めです
allow_failure: true	このジョブが失敗してもパイプライン全体を赤にしない設定。名前の横に Allowed to fail と出ます
ISSUE_ID	実装させる Issue の番号を入れる変数。既定値は <code>"4"</code> で、schedule か手動実行の Variables で上書きできます
artifacts	ジョブが終わったあとに GitLab が保存するファイル。ジョブ画面の右の Job artifacts から開きます
ci_logs/	スクリプトがジョブの中で作るフォルダ。結果の要約 JSON と標準エラーの2つが入ります。リポジトリにはコミットされません

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` のプロ

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", "${CLAUDE_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

止め金4層と、効く場所

	手元の Claude Desktop 対話セッション	CI の runner Linux のコンテナ
(a) deny .claude/settings.json の permissions.deny 文字列照合なので書き方を変えると抜ける	● 効く	● 効く settings.ci.json の deny も読む
(b) PreToolUse フック block-destructive 変更してよい範囲の指定もここに置ける	● 効く	● 効く .ps1 の登録は効かない settings.ci.json の bash 版が効く
(c) --max-turns と timeout 回数と時間で打ち切る 操作の種類は見えない	効かない --max-turns は -p 専用	● 両方効く ジョブの timeout 30m
(d) ブランチ保護 GitLab の Protected branches エージェント側の設定から独立している	効かない 手元では止まらない	● 効く push の時点で拒否される

手元で効いた止め金がCIでも効くとは限らない。逆に(c)と(d)はCIの側でだけ効く

(b)は.ps1のままではCIで動かないので、夜間ジョブは同じ判定のbash版をsettings.ci.jsonに登録している

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

deny の13行の由来 上の10行が Day1 のプチ演習2、下の3行がプチ演習3 で書いたものです。上の10行は破壊系の5つを Bash と PowerShell の対で書いたもの、下の3行が秘密情報です。配布直後の `settings.json` は `"deny": []` の空の状態なので、Day1 の作業をこのフォルダで行っていない方はここで13行を入れます。`deny` は JSON なので OS を跨いで効きます。

PreToolUse ・ PostToolUse ・ Stop の3本がそろわないとき PreToolUse と PostToolUse が無いときは、ハンズオン2 のブランチより前で抜けています。Stop だけが無いときは、プチ演習6 を `ex/h2-` 以外のブランチで行ったか、コミットしていません。VSCode で `.claude/settings.example.jsonc` を開き、足りないブロックを `settings.json` へ写します。注釈付きの完成形が置いてあります。写すときは `//` で始まる注釈の行を落としてください。`settings.json` はセッションの開始時に読まれるので、書き換えたら `Ctrl+N` で新しいセッションを開きます。

CI 側で対象バージョンを守っているもの このプロジェクトの対象は PHP 7.4 です。手元では `check-phpver` フックが対象外の構文を差し戻し、夜間ジョブの中でも bash 版の同じフックが走ります。push のたびに走るブランチ用のパイプラインでは `lint-phpver` が受け持ちます。このジョブは2段です。1段目の `php -l` が `match` 式や `enum` などの構文を落とし、2段目の `php ci/undefined_functions.php` が `str_contains` や `str_starts_with` のような 7.4 に無い関数の呼び出しを `function_exists()` で見つけて落とします。`lint-phpver` が赤になると、後ろの段の `test` と `test-phpunit` は動きません。

ブランチ保護の確認で気づいたこと `main` の行に `protected` の表示が無い場合は講師へお知らせください。Step 6 のマージ判断に関わります。

コマンドで行う場合 ブランチの作成は、Claude Desktop 中のターミナル(右上の `>` アイコン、または `Ctrl` とバッククォート)から次の2行でも同じです。`yamada` はご自身のユーザー名に置き換えます。研修では上の手順1 の画面操作で進めます。

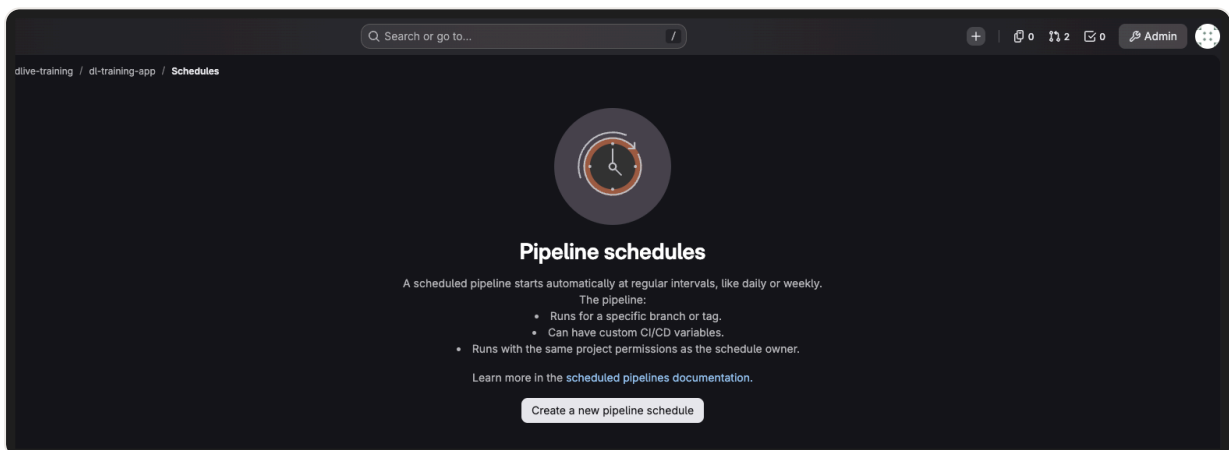
```
git switch ex/h2-yamada
git switch -c ex/h3-yamada
```

STEP 2 pipeline schedule の作成 [10min]

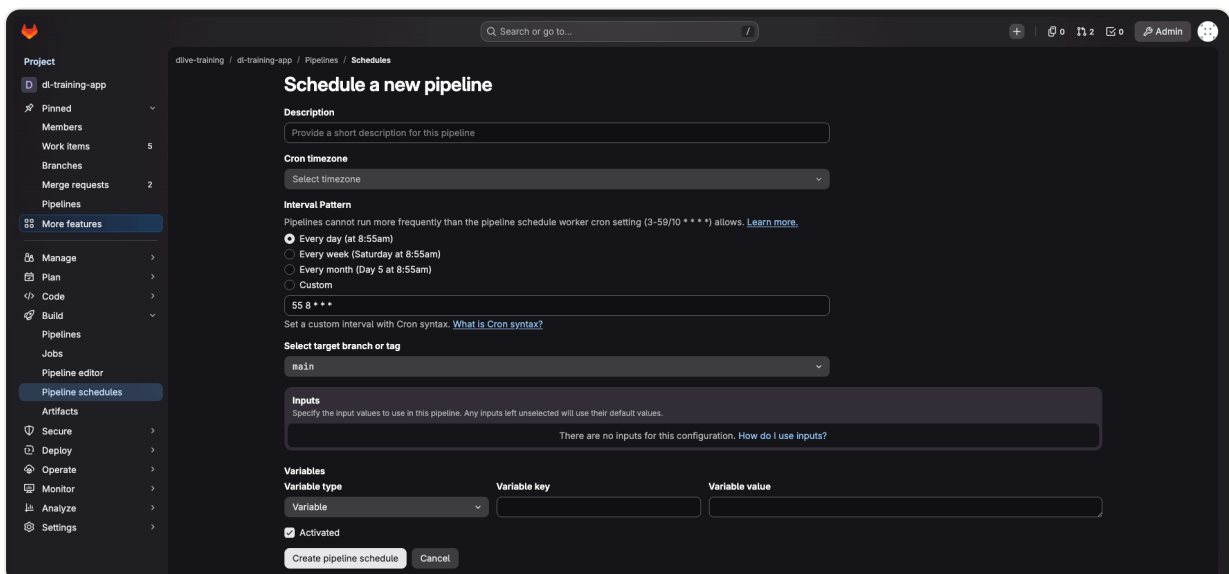
夜間に動かすための時刻の設定を作ります。当日は同時実行を避けるため手動ボタンから流しますが、schedule 自体は各自が作ってください。持ち帰るのはこの設定です。

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

チェックを付けたまま保存した場合 一覧の行のトグルを切って **Inactive** に戻してください。

入力画面の既定値 **Interval Pattern** の既定は **Every day (at 8:55am)** で、下の欄に **55 8 * * *** が入っています。**Custom** に切り替えてから同じ欄を **0 2 * * *** に書き換えます。**Activated** は開いた時点でチェックが付いています。**Variables** の行を入れ忘れると、ジョブは **.gitlab-ci.yml** の既定値で走ります。

当日に自動起動させない理由 研修当日は手動ボタン側だけを使います。Step 3 は講師が **main** で1本、Step 5 は各自のブランチで1本ずつ流します。**nightly_implement** は1本が最長30分 runner を使います。

時刻を GitLab 側に置いている理由 Anthropic の Routines(**/schedule**)は **claude.ai** のログインが前提で、API キーで動かしている環境では出てきません。

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]

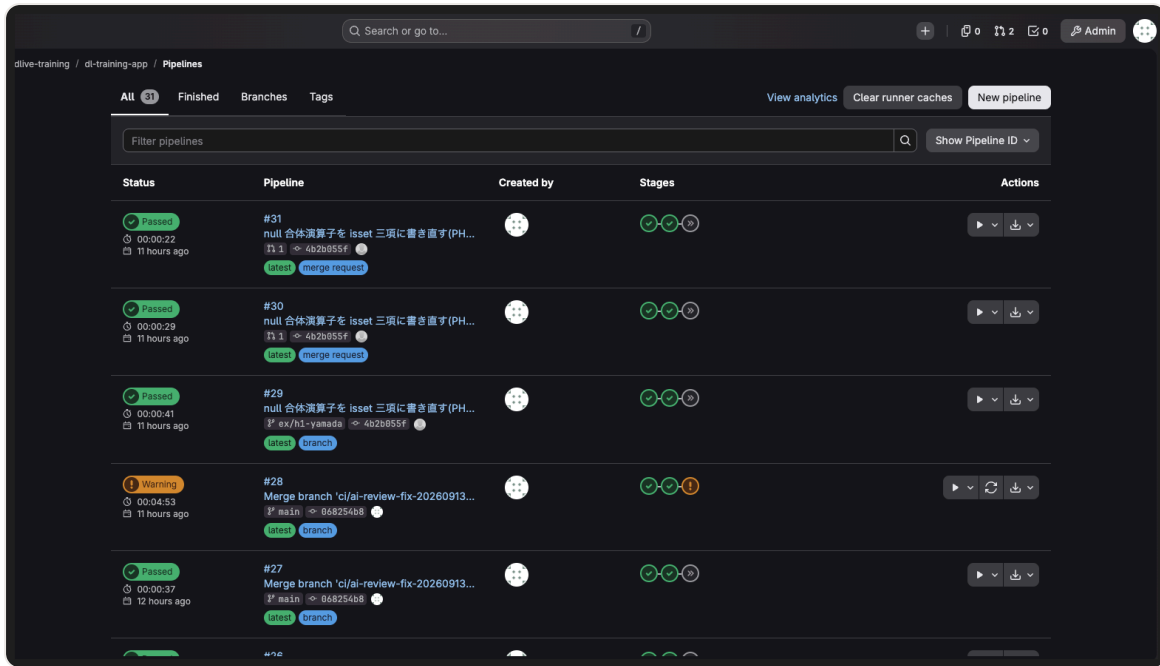
講師が `main` のパイプラインで `nightly_implement` を1本だけ流します。受講者は手動ボタンを押さず、全員が同じジョブのログを読みます。全員分を同時に走らせると、後のジョブが待ちに入って演習の時間内に終わらないためです。

手順

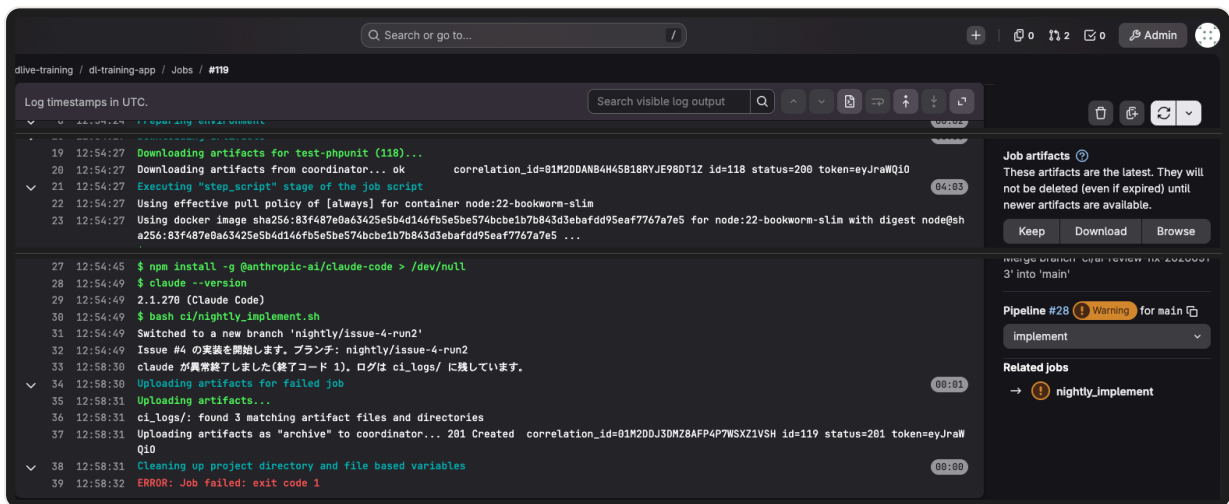
- 1 講師が Zoom のチャットにパイプライン番号(# の付いた数字)を貼ります。GitLab の **Build** > **Pipelines** を開き、その番号の行を探します。ブランチが `main` の行です。
- 2 その行の **Stages** のいちばん右の丸(`implement`)を押し、 `nightly_implement` を選んでジョブの画面へ移ります。
- 3 黒い領域のログを見ます。1〜2分の準備のあと Issue #4 の実装を開始します。ブランチ: `nightly/issue-4` が出れば、スクリプトが動き始めています。
- 4 そのまま待ちます。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	<code>turns: ... / cost_usd: ...</code> のあとに <code>View merge request for nightly/issue-4:</code> と URL、最後に <code>MRを作成しました。マージは人が判断します。</code>
動いたが差分が無い	<code>変更がありません。MRは作りません。</code> どこで止まったかは <code>ci_logs/</code> に出ます
始まる前に終わった	<code>ANTHROPIC_API_KEY: CI/CD変数 ANTHROPIC_API_KEY が未設定です。</code> <code>ci_logs/</code> 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。

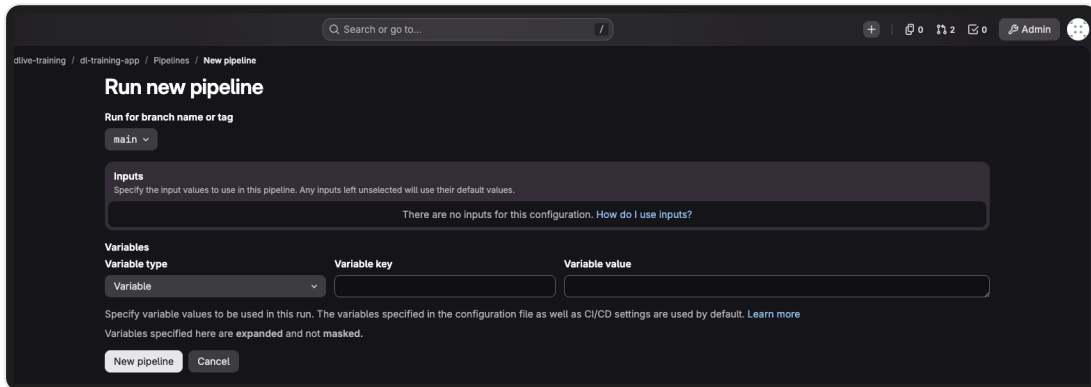
達成状態

- ☐ 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。

講師が押している操作 **Build** > **Pipelines** の右上の **New pipeline** を押し、**Run for branch name or tag** に **main** を選び、**Variables** は空のまま起動します。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るので、講師がこれを押します。**main** のパイプラインの再生ボタンは、受講者の画面に出ても押さないでください。



講師が押す **New pipeline** の画面です。Step 5 では push で動いた自分のブランチのパイプラインを使うので、受講者はこの画面を開きません。

講師の行が見分けられないとき 同じ時刻の行が複数あるときは、講師が Zoom のチャットに貼ったパイプライン番号で特定してください。ジョブ名に受講者名は出ません。

エージェントの出力がログに流れない理由 `--output-format json` で `ci_logs/claude-issue-4.json` に書いているためです。ジョブログに出るのは、スクリプトが出す行と最後の要約だけです。

pending のまま動かないとき 他のパイプラインのジョブがまだ runner を使っています。異常ではありません。ジョブ画面の上部に **This job is stuck** が出ている場合だけ、runner 側の問題なので講師へお知らせください。

30分で打ち切られたとき ログの末尾に `ERROR: Job failed: execution took longer than 30m0s seconds` と出ます。 `--max-turns 15` より先に `timeout` が効いた形で、1ターンが長いときに起こります。この場合も `ci_logs/` は残ります。

環境変数の不足で止まったとき 直すのは講師側の CI/CD の変数です。受講者は Step 4 の記録を「環境変数の不足で止まった」として書き、講師が変数を登録したあとの再実行を待ちます。

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

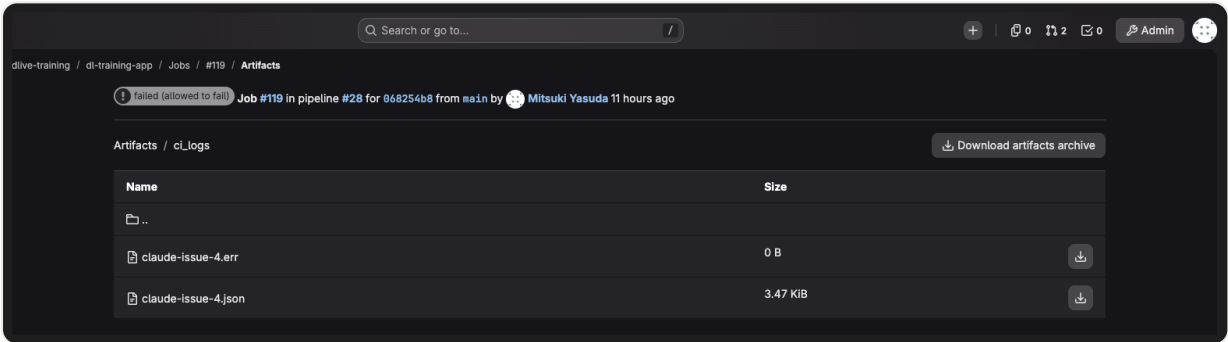
ジョブ画面の右の **Job artifacts** の **Browse** を押し、 `ci_logs` を開きます。
- 2

`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 <code>ANTHROPIC_API_KEY</code> の1行。 <code>ci_logs/</code> 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
Issue を読めずに終わった	<code>permission_denials</code> に <code>curl</code> か <code>glab</code> の拒否	プロンプト。 <code>PROMPT</code> に <code>_issues/issue-4.md</code> を読ませる1行を足す。 <code>curl</code> は <code>ci/settings.ci.json</code> の <code>deny</code> に入っているので、 <code>--allowedTools</code> に足しても通りません
ブランチ操作で拒否された	<code>permission_denials</code> に <code>git switch</code> か <code>git push</code> の拒否	プロンプト側。 <code>SKILL.md</code> 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来ます
範囲外のファイルまで	差分が <code>app/</code> と <code>tests/</code> の外に及んでいる	プロンプト側。 <code>PROMPT</code> 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

止まり方	ログに出るもの	直す層
書き換わった		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 <code>lint-phpver</code> が赤。 <code>match</code> 式や <code>?-></code> 、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 <code>bash</code> 版 <code>check-phpver</code> 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	<code>test</code> または <code>test-phpunit</code> が赤	Stop hook。手元では <code>stop-gate.ps1</code> が継続させるが、CI の <code>ci/settings.ci.json</code> 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	<code>ai_review</code> のジョブが動いていない	CI 側。 <code>ai_review</code> は <code>merge_request_event</code> 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。 <code>GITLAB_BOT_TOKEN</code> をプロジェクトアクセストークン(Developer、 <code>api</code> と <code>write_repository</code>)に差し替える
ターン数の上限で打ち切られた	<code>num_turns</code> が 15。コミットが途中で終わっている	Issue の粒度。 <code>--max-turns</code> 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。

表の「直す層」と (a)~(d) の対応 「プロンプト」「スクリプト」は

`ci/nightly_implement.sh` の呼び出し側で、(c) の隣にある層です。「CI 側」は `.gitlab-ci.yml` と CI/CD Variables。「Stop hook」と「REVIEW.md」は (b) のフックと、そのフックが読む基準。「Issue の粒度」は (c) の `--max-turns` を動かさずに入力のほうを割る選択です。(a) の deny と (d) のブランチ保護は表に出てきません。今日の1回目で効く前に、それより手前の層で止まるためです。

計測表の書き方の例 文言は例です。ご自身のログから写します。

```
| 項目 | 記入欄 |
|---|---|
| 止まった箇所 | Issue を読めずに終わった |
| 止まった理由 | permission_denials に curl の拒否。curl は ci/settings.ci.js
on の deny に入っている |
| どの層の設定で直すか | スクリプト(ci/nightly_implement.sh の PROMPT) |
| 直した内容(ファイル名とキー) | PROMPT に「Issue の本文は _issues/issue-4.md
にあります」を1行追加 |
| 再実行の結果 | (Step 5 のあとに書く) |
| ジョブの所要時間(分) | (Step 5 のあとに書く。ジョブ画面の Duration) |
| 使ったターン数 | 3(上限 15 に未到達)。cost_usd 0.04 |
```

環境変数の不足で止まった回は「止まった箇所」を「環境変数の不足で始まらなかった」とし、ターン数の欄は「-」にします。

permission_denials が空のとき 同じ `claude-issue-4.json` の `result` の文に、エージェントが最後に何を言って終わったかが入っています。`.json` はブラウザでは開けないので、行の右端のアイコンでダウンロードして VSCode で開いてください。2つまとめて取るときは右上の **Download artifacts archive** です。

ターン数の上限に当たった人へ `--max-turns` を上げないでください。上げると1本あたりの費用と時間が読めなくなります。夜間ループで回数の上限に当たるのは、多くの場合 Issue が大きすぎるという合図です。Issue #4 が「受入条件を全部機械で判定できる形」で書かれているのは、この打ち切りに当たらない粒度に落とすためです。

「レビューを飛ばした」に見える止まり方 多くは MR がまだ無いだけです。

`ai_review` と `ai_gate` は MR のパイプラインに乗っているので、push で MR が立った後に動きます。順番を取り違えると、直す層を間違えます。

ハンズオン3 再実行とマージ判断

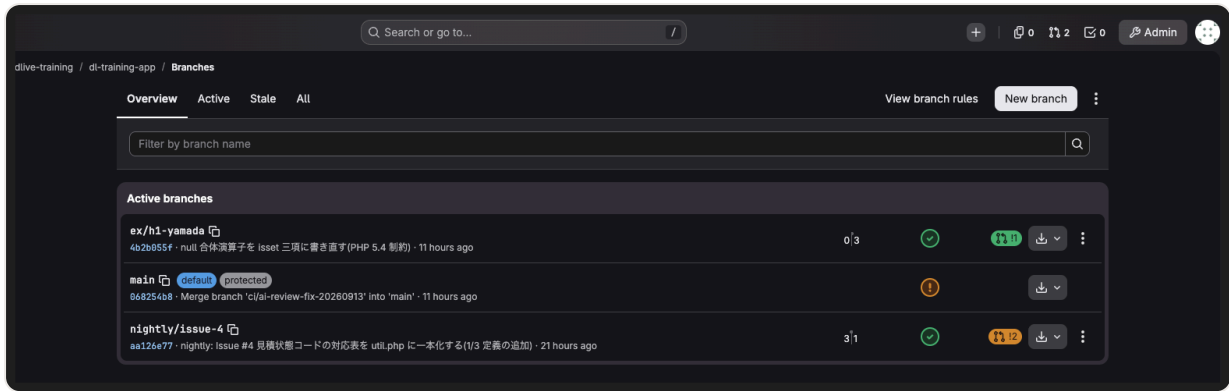
Step 5 から Step 7 です。合わせて [35min]。直す層を1つだけ直して流し直し、立った MR を人が読み、夜間に回せる範囲を1枚にまとめます。

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。

2回目のブランチ名 `ci/nightly_implement.sh` は、`main` 以外のブランチから流されたときに起点のブランチ名を後ろに足します。`ex/h3-yamada` から流せば `nightly/issue-4-ex-h3-yamada` になります。講師の1回目の `nightly/issue-4` と他の方のブランチとも重ならないので、remote のブランチを消す必要はありません。強制 push は `permissions.deny` で禁じているので、CI でも例外を作しません。

同じ起点からもう一度流し直すとき 先に自分の名前が付いたブランチを消します。GitLab の **Code** > **Branches** で `nightly/issue-4` で始まる自分のブランチを探し、行の右端のごみ箱のアイコンで削除します。MR が立っている場合は、先に MR を閉じてください。

2回目の MR に自分のコミットが入る理由 `nightly/issue-4-ex-h3-...` は自分のブランチの先端から作られるためです。Step 6 で差分を読むときは、Issue #4 の実装と自分の修正を分けて見てください。エージェントが何も変えなかった場合は、スクリプトがジョブ開始時のコミットと比べて「変更がありません」と出し、MR は作りません。

コマンドで行う場合 手順2 と手順3 は、Claude Desktop 中のターミナル(右上の `>` アイコン、または `Ctrl` とバッククォート)から次の3行でも同じです。研修では上のソース管理ペインの操作で進めます。

```
git add ci/nightly_implement.sh
git commit -m "夜間ジョブ: Issue 本文を _issues/issue-4.md から読ませる"
git push -u origin HEAD
```

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。

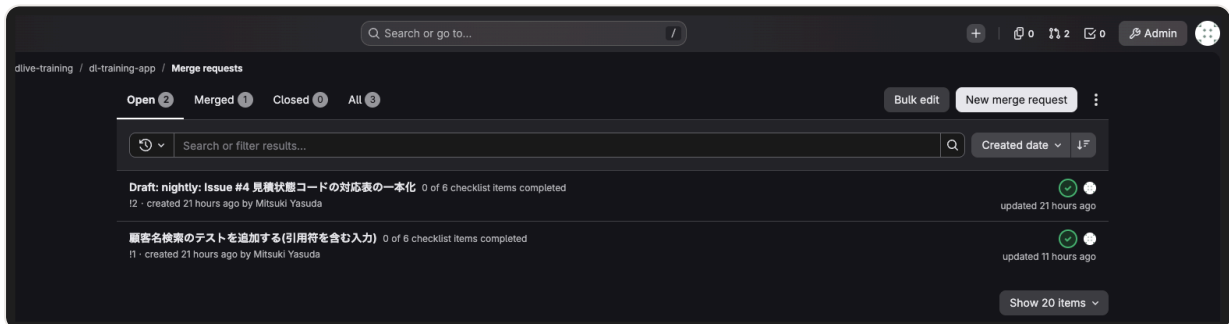
3 **Overview** タブに戻り、ai_review が置いたコメントを読みます。

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、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

無人実行では、エージェントが20ファイルを一度に書き換えたあとで気づきます。MRごと閉じると、範囲内の実装まで捨てることになります。戻すのは範囲外のファイルだけにします。

1. 範囲を特定する	Changes タブの左のファイル一覧で、 <code>app/</code> と <code>tests/</code> の外にあるものを書き出します
2. その範囲だけ戻す	MR のブランチに切り替え、書き出したパスだけを起点の内容に戻します。GitLab の画面には「一部のファイルだけ戻す」操作が無いので、ここはコマンドを使います
3. 戻した分をコミットする	ソース管理ペインで、戻ったファイルだけをステージしてコミットし、同期します。MR の差分から範囲外のファイルが消えます
4. 次の実行で防ぐ	<code>ci/nightly_implement.sh</code> の PROMPT に、触ってよい範囲を1行足します。Step 4 の表の「範囲外のファイルまで書き換わった」の行です

手順2 で使うコマンドです。Claude Desktop 中のターミナル(右上の `>_` アイコン、または `Ctrl` とバッククォート)から実行します。セッションの作業ディレクトリで開くので、そのまま打てます。 `<パス>` には手順1 で書き出したものを並べます。

```
git restore --source=origin/main -- <パス>
```

MR 全体を取り消したい場合は、マージ後であれば GitLab の MR 画面の **Revert** ボタンで戻せます。まだマージしていない MR は、**Close** で閉じればブランチごと残せません。

人が手で直したコミットと、エージェントのコミットが同じブランチに混ざっている場合は、**Commits** タブで作者を見ます。夜間ジョブのコミットは `nightly-bot` です。作者で分けられるうちは、戻す範囲をコミット単位で選べます。

解説と補足

MRの見分け方 タイトルが `nightly: Issue #4`、作成者が `GITLAB_BOT_TOKEN` のボットアカウントになっています。

パイプラインが2本並ぶ理由 共通の進め方 STEP 4「GitLab の Web 画面」で見たとおり、`branch` の行に `lint-phpdev`、`lint-phpver`、`test`、`test-phpunit` の4つ、`merge request` の行に `ai_review` と `ai_gate` の2つが出ます。`test-phpunit` は2分ほど余計にかかります。

ai_gate が落ちているとき そのジョブのログに止めた指摘が並ぶので、直すか `GATE_LEVEL` の判断を変えるかを決めます。

受入条件の確かめ方 差分の中でブラウザの検索(Ctrl+F)を使えば、前の2つは数十秒で確かめられます。

マージ後のブランチ MR の作成オプションに `merge_request.remove_source_branch` が付いているので、講師がマージした MR の元ブランチは自動で消えます。開いたまま残す MR のブランチは消えません。

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」に挙げ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

台帳の書き方 共通の章の台帳と同じ列です。

```
| ハンズオン3 | nightly_implement ジョブ + ci/nightly_implement.sh + pipeline  
schedule(ISSUE_ID=4, 0 2 * * *, main) | .gitlab-ci.yml と ci/(当日確定) | 同  
じ(当日確定) | 山田 | 置き場所まで決めた |
```

コードを読む係を置かない理由 読む必要が出たということは、止め金かゲートのどちらかが足りていないという合図です。その場合は、人を増やさずに層を1つ足します。

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、コミットまでで終わってください。
```

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

結果を JSON で受け取る形(`-p`)は、Claude Desktop の画面からは実行できません。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。夜間ジョブと同じ引数で1本流して `num_turns` と `total_cost_usd` を数字で取りたい場合は、Claude Desktop の中のターミナル(右上の `>` アイコン、または `Ctrl` とバッククォート)から次を実行します。研修では上の Claude Desktop の手順で進めます。行末の記号はバッククォートで、PowerShell の行継続です。

```
claude -p "/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、コミットまでで終わってください。" `
  --permission-mode dontAsk `
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(PHP *),Bash(git add *),Bash(git commit *)" `
  --max-turns 15 `
  --output-format json | Out-File -Encoding utf8 claude-issue-4.json
```

Mac の方は行末のバッククォートを `\` に替え、`| Out-File -Encoding utf8` を `>` にします。それ以外の引数は同じです。終わったら `claude-issue-4.json` を VSCode で開き、`is_error` と `num_turns` を見ます。このファイルはコミットに入れなくてください。ソース管理ペインでステージするファイルを選ぶときに外します。

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	<code>nightly/issue-4</code> と、そこから立った MR <code>nightly: Issue #4</code> 。GitLab の演習プロジェクトに残ります
実行ログ	<code>ci_logs/claude-issue-4.json</code> と <code>.err</code> 。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	<code>nightly issue-4 (Inactive)</code> 。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。 <code>ci/nightly_implement.sh</code> 、 <code>.gitlab-ci.yml</code> 、 <code>REVIEW.md</code> 、 <code>.claude/settings.json</code> のいずれかです

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
<code>.gitlab-ci.yml</code> の <code>nightly_implement</code> ジョブ	対象リポジトリの <code>.gitlab-ci.yml</code> に追記。 <code>ISSUE_ID</code> 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 <code>before_script</code> から <code>php-cli</code> を外す
<code>ci/nightly_implement.sh</code>	<code>ci/</code> に置く。手元で流す運用にするなら、同じ引数のまま <code>.ps1</code> に写す。コメントと文字列は ASCII に保つ	<code>ci/</code> にそのまま。bash が正なので書き換え不要
<code>.claude/skills/implement-issue/SKILL.md</code>	<code>.claude/skills/implement-issue/</code> 。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 <code>.claude/settings.json</code> (<code>deny</code> ・ <code>hooks</code> ・ <code>maxEffortLevel</code>)	既存の <code>settings.json</code> へ <code>deny</code> を追記。 <code>hooks</code> の <code>command</code> は <code>powershell -File</code> のまま	<code>hooks</code> を <code>exec</code> 形式のまま、 <code>"command": "bash"</code> と <code>"args":</code> [<code>"\${CLAUDE_PROJECT_DIR}/.claude/hooks/<名前>.sh"</code>] へ差し替える
<code>pipeline schedule</code> の設定 (<code>ISSUE_ID</code> 、 <code>cron</code> 、 <code>Target branch</code>)	対象プロジェクトの <code>Build > Pipeline schedules</code> 。ファイルでは持ち出せないなので、台帳に設定値を控える	同じ。複数リポジトリに同じ <code>schedule</code> 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

`ANTHROPIC_API_KEY` と `GITLAB_BOT_TOKEN` を masked で登録し、Protect は外します。Protect を付けると保護ブランチ以外で変数が渡らず、受講者のブランチでジョブが動きません。MR の作成は `CI_JOB_TOKEN` では通らないので、プロジェクトアクセストークン(ロール Developer、スコープ `api` と `write_repository`)を使います。自社へ写すときはこの逆にします。2つの変数に Protect を付け、鍵を使うジョブは保護ブランチのパイプラインに限ってください。

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が。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うと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

なぜ先に予想を書くか 点検表は、読んだ時点で答えが見えます。先に表を読むと、予想はその写しにしかありません。差が取れないので、どこを守れているつもりだったかが分からなくなります。この演習で持ち帰るものは点数ではなく、**思い込みと実態のずれ**です。

層の順に埋める必要はない 第1層に「無い」が並んでいても、第3層と第4層が埋まっているなら危険な状態ではありません。逆に、第1層だけが埋まっていて他が空いている状態は、守っているつもりで何も止まっていない状態です。文脈はAIの判断に効くだけで、実行を止める力を持ちません。

30項目の内訳 層ごとの項目数は、その層で作り込める幅にそろえてあります。フックが一番多いのは、実行の直前に割り込んでコマンドの中身まで見られる層だからです。

層	項目数	見ているもの
第1層 文脈	6	AIに読ませている決まりの量と重複、禁止に代替が添えてあるか
第2層 権限	6	止める一覧が空でないか、編集系のツールがそろっているか
第3層 フック	8	並べ替えた形でも止まるか、テストがあるか、道具が無いときに止まるか
第4層 CI	6	提出のときに自動で動くか、落ちる条件が数値で決まっているか
第5層 リポジトリ保護	4	手元の設定を全部外した人にも効くか、戻す手順が対になっているか

自社のハーネスを持っていない方 Day1のプチ演習1から3、Day2のプチ演習4から6で足した設定が点検の対象になります。第1層から第3層は当日の成果物で埋まり、第4層は演習リポジトリのCI、第5層はGitLabの **Code > Branches** で **main** の行に付いた **protected** の表示を見れば判定できます。Developerの権限では **Settings** のメニューは出ません。自社の分は、この表の写しを持ち帰ってから同じ手順で埋めてください。

「一部」が多くなったとき 多くて構いません。「一部」は、置いてあるが定義どおりには効いていない、または一部の環境でしか効かない状態を指します。WSLとWindowsが混在している方は、片方の環境でしか効かない項目がここに集まります。層ごとの落とし穴は、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。

追加と考察 時間が余った方は、「ある」を付けた項目から3つ選び、確かめる方法を実際に流してください。1つでも「一部」へ落ちたら、そこが次の1週間で埋める候補

に入ります。確かめずに付けた「ある」がどれくらい残っていたかが、そのままハーネスの見通しの悪さです。

発展課題 この30項目のうち、機械で数えられるものに丸を付けてください。行数を数える、規則の本数を数える、テストを流す、といった項目です。丸が付いた項目は、次回からは人が点検せずにスクリプトで出せます。雛形の `template/ci/verify-harness.sh` が、その考え方を実装したものです。

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 <code>docs/</code> 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GitBash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

- 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

テスト一式を流せます。jq が要ります。

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

末尾の件数を控えてください。書き換えた後にこの数を下回ったら、壊しています。

```
guard.test(sh): 45/45 passed
guard.test(ps): skipped (PowerShell not found on this machine)
```

2行目は、PowerShell が入っていない端末で出ます。同じケースを PowerShell 版でも流す作りになっていて、入っていない端末では飛ばし、**飛ばしたことを画面に出します**。黙って合格にしません。

Step 3 1ファイルを自社の言葉に直す [10min]

- VSCode で Step 1 の表で決めたファイルを開いてください。
- 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。
- 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。
- 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"}}' | powershell -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

Mac と Git Bash の方が、規則を足した後に `bash template/ci/verify-harness.sh` を流すと、次のように落ちます。

```
NG 止める規則 は文書が 12、実装が 13 です
```

不具合ではありません。README.md の「本数の宣言」に書いた数と、rules.json の実装の数を突き合わせる検査が、ずれを見つけた状態です。README の数を13に直すと通ります。**文書に書いた数は、それを数えるスクリプトとセットで置く、という原則**8をそのまま動かしたものです。片方だけ直すと落ちる形にしています。

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。

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

なぜ1層だけにするか 5層を同時に触ると、何が効いて何が効いていないかが分からなくなります。テストが落ちたときも、どの変更が原因かを切り分ける作業から始まります。雛形の使い始めの順番が「1層ずつ入れて、その層が効いていることを確かめてから次へ行く」になっているのは、この切り分けを毎回1回で済ませるためです。

止める対象を設定ファイルに出す理由 雛形は、何を止めるかを `rules.json` に置き、入口の `guard.sh` と `guard.ps1` には判定の手続きだけを書いています。規則を1本足すときに触るのは `rules.json` の1か所で、bash 側と PowerShell 側が食い違いません。環境が増えたときも、入口を1本足すだけで済みます。プチ演習4 から6 で作ったフックは判定をスクリプトの中に持っているので、この形が発展形にあたります。

綴りを並べない判定 `rm -rf` を文字列で照合する書き方は、`rm -fr`、`rm -r -f`、`rm --recursive --force` のいずれも素通りします。雛形はコマンドを語に分解し、オプションを集合にしてから照合するので、これらはすべて `r` と `f` の2つになり、規則1本で受けられます。`bash -c` や `xargs` を挟んだ形も、包んでいる語を剥がしてから見ます。自社の規則を足すときも、綴りではなく動詞とオプションの組で書いてください。

症状	手当て
Mac や Git Bash で <code>jq</code> がありません で落ちる	判定に使う道具が無いときは通さない作りです。素通りさせないための設計です。 <code>jq</code> を入れられない端末では、Step 2 の <code>guard.ps1</code> に流す形で進めてください。
足した規則が効かない	<code>rules.json</code> の JSON が壊れていると、ファイルごと読めません。VSCode の赤い波線を先に見てください。
止めたくないもののままで止まる	動詞を絞っていません。 <code>grep -rf patterns.txt src</code> のように、同じオプションを持つ別のコマンドに当たっています。 <code>verbs</code> に動詞を明示してください。
基準線の2件の結果が変わった、テストの件数が減った	既存の規則かケースを消しています。足すだけにしてください。減った分は、止まらなくなった操作です。

第1層や第5層を選んだ方 文書を直す層なので、`guard.ps1` に流した結果も `guard.test.sh` の件数も、直す前と同じになります。それで正しい状態です。確かめるのは、書いた代わりの道が自社で実際に通るかかどうかです。「履歴を書き換えない」と書いたなら、打ち消しのコミットを積み手順が自社の運用で通るかを、その場で1人に確かめてください。通らない代替を書くと、禁止だけを書いたのと同じことになります。

追加と考察 直した層と、プチ演習4 から6 で作った手元のフックが、同じ危険を二重に受けているかを見てください。重なっているなら、片方が黙って壊れても素通りしま

せん。どこにも重なりが無い危険があれば、それを1つ書き出してください。その1つが、次に埋める層の候補になります。

発展課題 自社のリポジトリで `template/` をコピーし、同じ手順を残りの4層へ繰り返してください。1週間に1層の見当です。4層目まで進んだところで `bash ci/verify-harness.sh` を流すと、文書と実装のずれがまとめて出ます。ここで落ちた項目が、写している途中で読み替えを間違えた箇所です。

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	<code>rules.json</code> 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 <code>template/</code> の1層	リポジトリ直下へコピーし、既存の <code>.claude</code> 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、 <code>rules.json</code> 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

困ったときの引き方

画面に出ている文字から引いてください。左の列で症状を探し、右の列のカテゴリへ飛ぶと、その中に同じ見出しの項目があります。見出しをクリックすると原因と直し方が開きます。講師を呼ぶ前に直し方の1番だけ試していただけると、こちらで原因を絞る時間が減ります。

症状	カテゴリ
Claude Code が起動しない、アプリの一覧に出てこない	起動と実行環境
アプリからプロジェクトのフォルダを開けない	起動と実行環境
フックが効いているのか画面で分からない	起動と実行環境
確認が出ないまま編集が進んでしまう	起動と実行環境
パイプでつなぐ書き方が動かない	起動と実行環境
コマンドを打つ場所が見つからない	起動と実行環境
スクリプトの実行が無効と赤い文字で出る	起動と実行環境
フックの日本語が読めない並びになる	起動と実行環境
bash\r、bad interpreter と出る	起動と実行環境
Permission denied と出る、止まるはずが素通りする	起動と実行環境
編集のたびに数秒待たされる	起動と実行環境
jq not found と出て判定が通ってしまう	起動と実行環境
拒否の規則を足したのに実行される	設定とフックの効き方
CLAUDE.md に書いた禁止が守られない	設定とフックの効き方
フックも拒否も同時に効かなくなった	設定とフックの効き方
隔離して起動するとフックが発火しない	設定とフックの効き方
effort やモデルが指定どおりにならない	設定とフックの効き方
AI の変更の一部だけを戻したい	巻き戻しと隔離
自分の手直しと AI の変更が混ざった	巻き戻しと隔離

症状	カテゴリ
巻き戻しても戻らないファイルがある	巻き戻しと隔離
Esc を2回押しても巻き戻しの一覧が開かない	巻き戻しと隔離
隔離した作業の始め方が分からない	巻き戻しと隔離
隔離すると .env が入っていない	巻き戻しと隔離
隔離したあとブランチが使えない	巻き戻しと隔離
対象バージョンで動かない構文が差し戻される	対象バージョンの制約とテスト
lint のジョブだけが赤になる	対象バージョンの制約とテスト
main に push できない	GitLab と CI
見覚えのないブランチや MR が一覧に並ぶ	GitLab と CI
パイプラインが pending のまま動かない	GitLab と CI
ai_review が緑なのにコメントが付かない	GitLab と CI
ai_gate が止めたのにマージできる	GitLab と CI
夜間ジョブの手動ボタンが押せない	夜間ループと Stop hook
夜間ジョブが MR を作らない、403 で弾かれる	夜間ループと Stop hook
Stop hook が止まらない	夜間ループと Stop hook
進め方そのものに迷った、時間内に終わらない	進行と画面共有

Tips：ここに無い症状に当たったら、画面の文字をそのままチャットへ貼ってください。画像より文字のほうが検索できます。当日出た症状はこのページに追記して、研修後に配り直します。

起動と実行環境

入り口で止まる症状です。Claude Desktop の起動、プロジェクトフォルダの選び方、フックの確かめ方、パーミッションモード、統合ターミナル、実行ポリシー、文字化け、改行コード、実行ビット、動く速さ、判定コマンドの不在を置いています。当日の朝に一番多く出ます。

症状

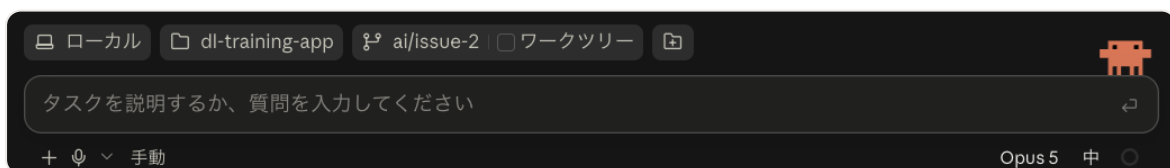
Claude Desktop は起動しているのに、演習のファイルが見えません。ファイル名を伝えても「そのファイルが見つかりません」と返ってきます。プロジェクトの一覧に `dl-training-app` が並んでいません。

原因

プロジェクトフォルダは、セッションを始めるときにプロンプト領域で設定する4項目の1つです。ここを選ばないまま書き始めると、会話はできてもファイルには手が届きません。ZIPを展開した場所と、選んだ場所が違う場合も同じ症状になります。

直し方

- 1 まず事前セットアップで `dl-training-app` を取り込んだ場所を確認してください。
- 2 新しいセッションを作ってください。 `Ctrl+N` です。
- 3 プロンプト領域の4項目から、プロジェクトフォルダに `dl-training-app` を選んでください。1つ上の階層を選ぶと、`.claude` の設定が読まれません。
- 4 「`app/libraries/util.php` の先頭10行を見せてください」と頼み、中身が返ることを確かめてください。ここまで通れば、以降の演習は進みます。
- 5 返らない場合は、フォルダの位置がネットワークドライブや OneDrive の同期フォルダの下になっていないかを見てください。演習中は `C:\work` のようなローカルのフォルダに置いてください。



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(その右がワークツリーの選択)、下の行の左がパーミッションモード、右がモデルです。画面は撮影時点のもので、モデル名はいまは Opus 5.5 と出ます

Tips: セッションは `Ctrl+N` で並べて持てます。切り替えは `Ctrl+Tab` です。演習では、実装のセッションとレビューのセッションを分けて使います。同じことは CLI でもできます。研修では Claude Desktop で進めます。

症状

`settings.json` にフックを書いたものの、効いているかどうか画面から読み取れません。何も起きていないのか、書き方が違うのか、判断が付きません。

原因

フックは動いたときだけ画面に出ます。止めるものが何も来ていなければ、正しく登録されていても表示は変わりません。「出ない」と「効いていない」を、見た目で区別できないということです。なお、フックと `permissions`、プロジェクトの `CLAUDE.md` は Claude Desktop と CLI で共有されます。片方で書いたものが、もう片方で効かないということはありません。

直し方

- 1 まず `settings.json` を VSCode で開き、`hooks` のキーの中にイベント名が並んでいるかを見てください。Claude Desktop の入力欄に `/hooks` と打つと「`isn't available in this environment`」と返ります。登録の確認はいつもこのファイルで行います。
- 2 拒否の規則も同じファイルの `permissions.deny` で見ます。自分が足した行が `Bash(...)` と `PowerShell(...)` の対で並んでいるかを確認してください。`/permissions` も入力欄では使えません。
- 3 効いていることを確かめるには、止まるはずの操作を1回頼んでください。止まった理由がフックの書いた文言で返れば、動いています。
- 4 何も出ずに実行された場合は、フックが素通りしています。Windows では、まず `matcher` が `Bash|PowerShell` になっているかを見てください。Bash だけだと、PowerShell ツールから打たれたコマンドではフックが呼ばれません。次に実行ポリシーの項目を見ます。
- 5 書き換えたあとは `Ctrl+N` で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。

```
/hooks isn't available in this environment.
```



Autoモードを試す

Claudeは各ツールコールを確認し、安全チェック合格したものを実行します。いつでもこのメニューの「Autoモードを試す」をクリックして、ClaudeのAutoモードを試すことができます。このコマンドはClaude Codeのターミナルでのみ使用できます。

Claude Desktop の入力欄で `/hooks` を送ったときの画面です。この表示が出るのは故障ではなく、入力欄では使えないコマンドだからです

`git reset --hard HEAD` — 実行できてない

`.claude/hooks/block-destructive.sh` が PreToolUse で止めた。メッセージはこう：

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

フックはユーザー側の設定だから、迂回はしないでおくわ。ちなみに実行前に `git status --porcelain` を見たら空だったんで、通ってても何も変わらなかった。戻したい対象が具体的にあるなら教えて。`git revert <sha>` か `git checkout <sha>` で読むかたちに落とせる。

フックが止めたときの会話の表示です。どのフックが、どのイベントで止めたかと、フックが返した文言が引用されます。この1行が出れば効いています

※指示されたreset実行とtests配下・Issueファイルの中身確認：git reset, ls, cat

reset to HEAD and inspect tests and issues directoriesに失敗しました

Bash

```
$ git reset --hard HEAD; echo "=== tests ==="; ls -la tests 2>&1; echo "=== tests tracked? ==="; git ls-files tests; echo "=== _issues ==="; ls -la _issues 2>&1
```

PreToolUse: Bash hook error: [bash .claude/hooks/block-destructive.sh]: [block-destructive] reset-hard: r

赤い「失敗しました」の行を押して展開した画面です。実行しようとしたコマンドと、PreToolUse: Bash hook error で始まるフックの出力がそのまま出ます

止まることを1回見てから先へ進んでください。 フックの故障は、画面に何も出さずに素通りする形で起きます。書いた直後に1回だけ止まる場所を見ておくと、あとで疑う場所が減ります。

症状

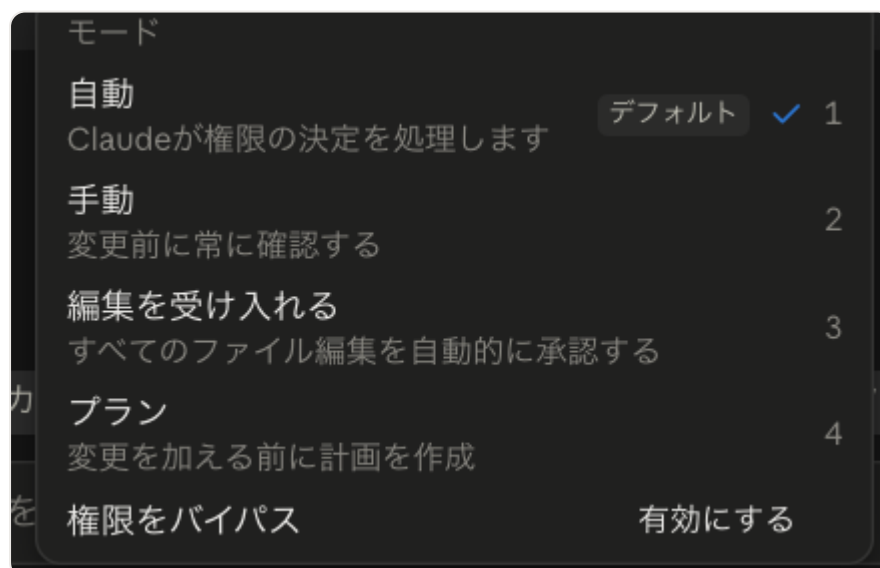
ファイルを変更してよいかの確認が出ません。頼んだ内容がそのまま適用され、気づいたときには複数のファイルが書き換わっています。拒否の規則に当たるものは止まるので、設定が壊れているわけではありません。

原因

パーミッションモードが **自動** になっています。画面で選べるのは **自動**、**手動**、**編集を受け入れる**、**プラン** の4つです。**編集を受け入れる** でも、ファイルの編集は確認なしで進みます。新しいセッションやワークツリーに移ったときに、前と違うモードで始まっていないかを見直してください。

直し方

- 1 いまのモードを確認してください。演習中は **手動** にしておくと、どこで何を聞かれるかが見えます。
- 2 止め金の演習をしているあいだは **自動** を選ばないでください。規則とフックが止めた場面を見ることが、演習の中身です。
- 3 何を作るか先に見たいときは **プラン** を使ってください。ファイルは変わらず、手順だけが返ります。
- 4 すで書き換わってしまった場合は、巻き戻しの項目へ進んでください。ファイル単位で戻せます。



入力欄の下モード名を押すと開くメニューです。自動 / 手動 / 編集を受け入れる / プラン の順に並びます

新しいセッションを開いたら、まずモードを見てください。メッセージを送る前に入力欄の下モード名を確認する癖をつけると、自動のまま自社のリポジトリを触る事故を防げます。

パイプでつなぐ書き方が動かない

ハンズオン1

症状

差分をレビューへ渡す1行を、Claude Desktop の入力欄に貼っても動きません。 `-p` や `--print` を使う形は、アプリの中では実行できません。

```
git diff main | claude --bare -p "." --output-format json
```

原因

Claude Desktop は対話で使うアプリで、非対話の実行(`-p` / `--print`)には対応していません。同じ理由で、エージェントチーム、インラインのコード補完、Bedrock などの第三者プロバイダも、アプリの側にはありません。CI の夜間ジョブが `claude -p` で書いてあるのは、あれが CLI で動いているためです。

直し方

- 1 統合ターミナルを開いてください。この形はターミナルから CLI を呼んで実行します。
- 2 コマンドを1行ずつ貼って流してください。出力先の `findings.json` ができることを確認します。
- 3 ターミナルから `claude` が起動しない場合は、起動の項目の折りたたみを開いてください。
- 4 対話で進める演習は、アプリの側へ戻してください。使い分けの線はここです。無人で回す形だけが CLI の仕事になります。

症状

手順に `git worktree list` のようなコマンドが出てきます。Claude Desktop の入力欄にそのまま打っても、コマンドとしては実行されません。

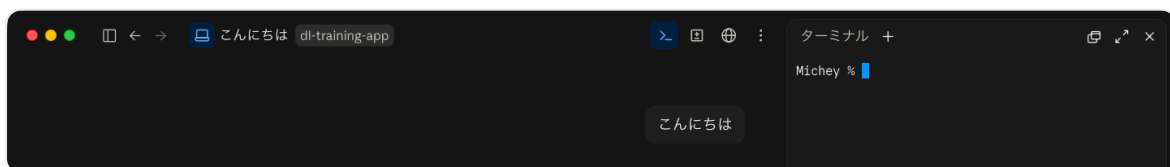
原因

入力欄は依頼を書くところで、シェルではありません。コマンドは統合ターミナルから実行します。この研修でコマンドを打つ場面は十数か所あり、どれも統合ターミナルにそのまま貼れる形で手順に書いてあります。

直し方

- 1 統合ターミナルを開いてください。右上の `>_` アイコンか、`Ctrl + バッククォート` です。
- 2 開いた場所はセッションの作業ディレクトリです。 `dl-training-app` の中にいれば、手順のコマンドはそのまま通ります。

```
pwd
```
- 3 2つ目のタブが要るときは、ターミナルペーンの `+` を押してください。
- 4 コマンドを頼む形でも実行できます。「`git worktree list` を実行して結果を見てください」と書くと、確認のうえで実行されます。拒否の規則に当たるものは、ここで止まります。
- 5 止まった場合は、止めているのが規則なのかフックなのかを読み分けてください。理由の1行に、どちらが止めたかが出ます。



統合ターミナルを開いた直後です。右上のターミナルのアイコン(`>_`)か、`Ctrl + バッククォート` で開きます。右のペインがターミナルで、開いた場所はセッションの作業フォルダです。画面は Mac で撮ったもので、Windows ではプロンプトが `PS C:\...>` の形になります

統合ターミナルはローカルのセッションだけです。 リモートのセッションでは開きません。コマンドを使う手順は、ローカルで開いたセッションで進めてください。

Tips : ターミナルは Claude と同じ環境を共有します。ここで別のフォルダへ移ると、その先の操作も同じ場所を見ます。迷ったら `pwd` で確かめてください。

症状

Claude Desktop がフックを呼んだとき、または中のターミナルから手で試したときに、赤い文字でこう出ます。

```
.\block-destructive.ps1 : このシステムではスクリプトの実行が無効になっているため、  
ファイル  
C:\Users\yamada\Desktop\dl-training-app\.claude\hooks\block-destructive.ps1 を読み  
込めません。  
詳細については、「about_Execution_Policies」(https://go.microsoft.com/fwlink/?LinkID=135170)  
を参照してください。  
+ CategoryInfo          : セキュリティ エラー: ( : ) [], PSSecurityException  
+ FullyQualifiedErrorId : UnauthorizedAccess
```

原因

Windows の既定の実行ポリシーは `Restricted` で、署名の無い `.ps1` を読み込めません。配布した `settings.json` の呼び出しは `-ExecutionPolicy Bypass` を付けていて、ふつうの端末ならこの形で読み込めます。エラーが出るのは、その引数を落として登録したとき、ターミナルから `.\block-destructive.ps1` のように直接叩いたとき、そして会社のグループポリシーで実行ポリシーが固定されているときです。`-ExecutionPolicy Bypass` はその場のプロセスだけの指定で、グループポリシーの値(`MachinePolicy` と `UserPolicy`)は上書きできません。

直し方

- 1 `.claude/settings.json` のフック登録を開き、`"command": "powershell.exe"` の `args` に `"-NoProfile", "-ExecutionPolicy", "Bypass", "-File"` が並んでいることを確認してください。

- 2 中のターミナルから手で叩くときも同じ形にします。

```
powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\block-de  
structive.ps1
```

- 3 それでも出る場合は、いまのポリシーを確認してください。

```
Get-ExecutionPolicy -List
```

- 4 `MachinePolicy` か `UserPolicy` の行が `Undefined` 以外になっていたら、グループポリシーで固定された端末です。この端末では `-ExecutionPolicy Bypass` を付けて

も ps1 は動かないので、講師に知らせてください。 Set-ExecutionPolicy は実行しないでください。

引数を落として叩いたときの表示です。1行目の このシステムではスクリプトの実行が無効になっているため がこの症状の目印です。

.\block-destructive.ps1 : このシステムではスクリプトの実行が無効になっているため、ファイル C:\Users\yamada\Desktop\dl-training-app\.claude\hooks\block-destructive.ps1 を読み込むことができません。詳細については、「about_Execution_Policies」(<https://go.microsoft.com/fwlink/?LinkID=135170>) を参照してください。

発生場所 行:1 文字:1

+ .\block-destructive.ps1

+ ~~~~~

+ CategoryInfo : セキュリティ エラー: (:) [], PSSecurityException

+ FullyQualifiedErrorId : UnauthorizedAccess

Tips: -NoProfile も外さないでください。個人のプロファイルが読み込まれると、そこに書いた文字コード設定や関数がフックの出力に混ざります。

症状

Claude Desktop を開いても、演習で使う Claude Code の画面に入れません。サインインを求められたまま進まない、または起動はするものの、どのフォルダも開けない状態のままです。

原因

多いのはサインインが済んでいない場合と、アプリの版が古い場合です。社内の配布で入っている版が古いまま固定されていることもあります。

直し方

- 1 Claude Desktop をいったん終了し、もう一度起動してください。サインインの画面が出たら、案内したアカウントで入ってください。
- 2 アプリの更新が来ていないかを確認してください。更新のあとは、もう一度起動し直します。
- 3 Claude Desktop の設定画面で版を確認してください。 2.1.280 以降であることが、モデルと effort の演習の前提です。
- 4 ここまで入れない場合は講師へお知らせください。端末の側の制限であることが多く、受講者側で直せる範囲を越えます。

フォルダを開いたセッションで「こんにちは」と送り、返事が返って画面上部の会話名の横に `dl-training-app` が出ていれば、この項目の問題ではありません。

版が古いとプチ演習1の結果が変わります。 2.1.280 より前の版では、`model: opus` のサブエージェントが Opus 5.5 ではなく Opus 5 で動き、タスクのペインの表示が他の方と食い違います。2.1.284 より前の版では、`model: sonnet` が Sonnet 5.5 ではなく Sonnet 5 で動きます。`maxEffortLevel` も 2.1.267 より前の版では無視されます。版が上がらない場合は講師へお知らせください。

CLI 版が見つからない場合

研修は Claude Desktop で進めるので、ここは読まなくても完走できます。持ち帰って CLI でも使う方向けの補足です。

ターミナルで `claude` と打って次が出る場合は、導入先が PATH に入っていない。

```
claude : 用語 'claude' は、コマンドレット、関数、スクリプト ファイル、または操作可能な  
プログラムの名前として認識されません。  
+ FullyQualifiedErrorId : CommandNotFoundException
```

Git Bash や WSL では `bash: claude: command not found` の1行になります。導入の直後はターミナルを開き直してください。PATH の変更は、起動中のターミナルには届きません。それでも出る場合は、実体の場所と npm の導入先を見比べます。

```
where.exe claude  
npm config get prefix  
$env:PATH -split ';' | Select-String npm
```

1つ目で出たパスが3つ目の結果に含まれていなければ、アプリケーションを終了して起動し直してください。

症状

フックが返す理由が読めない並びになります。

```
[block-destructive] 縋り綱械Φ綱峨 r 豁「縋√U縫励◆
```

原因

日本語版 Windows の PowerShell 5.1 は、指定しなければ CP932 で読み書きします。 `.ps1` を UTF-8 で保存して日本語のメッセージを書くと、出力がこの並びになります。読み込み側も同じで、 `Get-Content` に `-Encoding UTF8` を付けないと、UTF-8 の日本語が CP932 として読まれて化けます。

直し方

- 1 自分で足したメッセージに日本語が入っていないかを確認し、英語に直してください。 `.ps1` はコメントも含めて ASCII だけで書きます。
- 2 日本語で残したい説明は `.claude/hooks/README.md` へ移してください。フックの中に説明を置く必要はありません。
- 3 PHP ファイルを読む処理を自分で足した場合は、 `-Encoding UTF8` が付いているかを見てください。

```
$lines = @(Get-Content -LiteralPath $path -Encoding UTF8)
```
- 4 セッションを開き直し、 `sessionstart_verify` の報告に ASCII 以外のバイトの指摘が残っていないことを確認してください。

この制約は自社ハーネスにも効きます。 D2 のフックは PowerShell 5.1 で動いており、同じ CP932 の上にあります。ASCII だけで書くという約束は、いまは人が守っている決まりです。 `sessionstart_verify` のような検証を1本足せば、機械が守る側へ移せます。

症状

bash 版のフックを登録した端末で、Bash ツールを使うたびにこれが出ます。

```
/usr/bin/env: 'bash\r': No such file or directory
```

環境によっては次の形になります。

```
bash: .claude/hooks/block-destructive.sh: /usr/bin/env: bad interpreter: No such file or directory
```

フックが何も出さないまま素通りするだけのこともあります。画面には失敗の表示すら出ません。

原因

行末の `\r` がシェバングの一部として読まれています。`.sh` を Windows のエディタで開いて保存し直すと、改行が CRLF に変わることがあります。WSL 上のプロジェクトを Windows 側のエディタから開いた場合も同じです。リポジトリ直下の `.gitattributes` で `.ps1` は CRLF、`.sh` は LF に固定していますが、ZIP で受け取ったファイルを編集したときはこの固定が効きません。

直し方

- 1 VSCode で該当の `.sh` を開き、右下のステータスバーの **CRLF** をクリックして **LF** を選び、保存してください。

- 2 複数ある場合は PowerShell からまとめて直せます。

```
Get-ChildItem .claude\hooks\*.sh | ForEach-Object {  
    $t = [IO.File]::ReadAllText($_.FullName) -replace "`r`n", "`n"  
    [IO.File]::WriteAllText($_.FullName, $t)  
}
```

- 3 もう一度フックを手で流し、2 が返ることを確認してください。

```
echo '{"tool_name":"Bash","tool_input":{"command":"git reset --hard"}}' |  
bash .claude/hooks/block-destructive.sh  
echo $?
```

- 4 `Permission denied` が出た場合は改行ではなく実行ビットの側です。次の項目へ進んでください。

Tips : 逆向きも起こります。 `.ps1` が LF になると、PowerShell 5.1 が途中でパースに失敗することがあります。自社ハーネスへ持ち帰るときは、置き先のリポジトリにも `.gitattributes` の2行を入れてください。

症状

フックを登録したのに、止まるはずのコマンドが素通りします。手で叩くとうこう返ります。

```
bash: .claude/hooks/block-destructive.sh: Permission denied
```

WSL では、`chmod +x` を打っても `ls -l` の表示が変わらないことがあります。

原因

ZIP は実行ビットを持たないので、展開した `.sh` は `644` になります。WSL から `/mnt/c` 配下(Windows 側のディスク)に触っている場合は、`chmod` を打っても元に戻ります。この位置のファイルシステムは、既定ではパーミッションを保持しません。素通りは失敗として表示されないなので、気づくのが遅れます。

直し方

- 1 いまの状態を見てください。左端が `-rwxr-xr-x` なら実行ビットは立っています。

```
ls -l .claude/hooks/*.sh
```

- 2 Linux 側のディスク(`~/` 配下)に置いたりポジトリなら、これで直ります。

```
chmod +x .claude/hooks/*.sh
```

- 3 `/mnt/c` 配下で戻ってしまう場合は、フックの登録を `bash .claude/hooks/block-destructive.sh` の形に変えてください。bash に渡す形なら実行ビットを見ません。

- 4 Windows で進めている方は `.ps1` 側を登録してください。実行ビットの概念がありません。

- 5 直したあと、止まるはずのコマンドを1回頼んで、止まることを目で確認してください。素通りは黙って起きます。

止まらない側へ倒れる故障です。 実行ビットの欠落は、フックが無いのと同じ状態を作ります。しかも画面には何も出ません。環境を移したあとは、止まることを1回確かめてから先へ進んでください。

症状

フックは効いています。ただ、Edit のたびに数秒から十数秒待たされます。ツリー全体を見る検査を足したあとに目立ちます。

原因

WSL から `/mnt/c` 配下を読み書きすると、1ファイルごとに変換が挟まります。フックは編集のたびに走るので、この遅さが回数分積み上がります。AI は1つの依頼で何度も編集するため、体感是人が手で書くときの比ではありません。

直し方

- 1 いまどちら側にいるかを確認してください。 `/mnt/c` で始まっていれば Windows 側のディスクです。

```
pwd
```

- 2 リポジトリを WSL 側のディスク(`~/` 配下)へ移してください。VSCode の WSL 接続から開けば、操作感は変わりません。

- 3 移せない事情がある場合は、フックが見る範囲を編集されたファイル1本に絞ってください。標準入力の `tool_input.file_path` に対象が入っています。

- 4 ツリー全体の検査は CI の側へ置いてください。手元で毎回やる必要のある検査は多くありません。

Tips : 環境ごとのフックの組み方は、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。WSL と Windows をまたぐ構成、対象バージョンが複数ある構成は、そちらを見てください。

症状

1つ目は、レビュー結果を `findings.json` に落とす1行で止まります。

`jq` : 用語 '`jq`' は、コマンドレット、関数、スクリプト ファイル、または操作可能なプログラムの名前として認識されません。

2つ目は、`bash` 版のフックを登録したのに、止まるはずのコマンドが実行されます。標準エラーに1行だけ出ます。セッション開始時の報告にも同じ趣旨の行が並びます。

```
[block-destructive] jq not found, allowing the call
```

原因

`jq` は CI の `alpine` イメージに入れている道具で、端末に入っているとは限りません。判定スクリプトそのものは `jq` を使っていません。`ci/gate.ps1` は `ConvertFrom-Json` で読み、`ci/gate.sh` と `ci/ai_gate.sh` が `jq` を使います。`bash` 版のフックも `jq` でペイロードを読むので、`jq` が無いとガードを止めずに報告して素通りさせます。全部の `Bash` 呼び出しを止めるより、止まらないことを大きく知らせるほうが被害が小さいという判断です。

直し方

- 1 Windows で進めている方は、フックは `bash` 版ではなく `.ps1` 側を登録してください。こちらは `jq` を使いません。

- 2 JSON の取り出しは `ConvertFrom-Json` に置き換えてください。自分で JSON を書き出すときは、BOM を付けない形にします。

```
[IO.File]::WriteAllText("$PWD\findings.json", $out, (New-Object Text.UTF8Encoding $false))
```

`Set-Content -Encoding UTF8` と `Out-File -Encoding utf8` は先頭に BOM を付けます。`ci/gate.ps1` は読めますが、`bash ci/gate.sh` の `jq` は `parse error` を返します。

- 3 `findings.json` を開き、`summary` と `findings` の2つのキーがあることを確認してから判定にかけます。

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

- 4 `bash` を正として持ち帰る方は、`jq` を入れてからフックの単体テストを流してください。1件でも合わないランナーが 1 を返し、そのケースの出力を表示します。

```
bash .claude/hooks/tests/run_cases.sh
```

- 5 自分で書いたフックでも同じ形を作らないでください。判定に使うコマンドが無いときにエラーを握りつぶすと、毎回通ってしまいます。 `command -v jq` のような存在確認を先頭に置き、無ければ止めるか、標準エラーへ大きく出すかを決めてください。

- 6 CI側は触らないでください。ジョブのイメージにはjqを入れてあります。

判定の考え方は3本とも同じにしています。 手元の `ci/gate.ps1`、そのbash版の `ci/gate.sh`、CIが使う `ci/ai_gate.sh` は、どれも `findings.json` を読んで `GATE_LEVEL` 以上の件数を数えます。落ちる理由が揃っていないときは、まず文字コードを疑ってください。

Tips: 素通りしたことが標準エラーに出るかどうかは、自社ハーネスでも見どころです。気づけない素通りが一番高くつきます。

設定とフックの効き方

書いたのに効かないという症状です。置き場所、再起動、JSON の壊れ、隔離起動、
effort とモデルの優先順位の6件を置いています。設定は書いた瞬間ではなく、セッションの開始時に読めます。

症状

拒否の規則を足したのに、頼むと実行されます。止まるはずの `git reset --hard` がそのまま走ります。

原因

Windows でいちばん多いのは、`Bash(...)` の規則だけを書いている場合です。Git for Windows が入った Windows では、Claude は PowerShell ツールでコマンドを打つので、`Bash(...)` の規則は当たりません。次に多いのが、書き足した先が違う場合です。Claude Code が読むのは `.claude/settings.json` だけで、注釈付きの `.claude/settings.example.jsonc` は読みません。次に多いのが、VSCode で開いたフォルダが `dl-training-app` の親になっていて、プロジェクトルートが別の場所になっている場合です。

直し方

- 1 セッションのプロジェクトフォルダが `dl-training-app` になっていることを確認してください。1つ上の階層だと、別の場所の設定が読めます。
- 2 `.claude/settings.json` を VSCode で開き、`permissions.deny` に自分が足した行が `Bash(git reset --hard *)` と `PowerShell(git reset --hard *)` のような対で並んでいるかを見てください。`.jsonc` のほうに書いていたら、注釈行を落として `settings.json` へ写します。
- 3 `Ctrl+N` で新しいセッションを開いてください。`settings.json` はセッション開始時に読めます。
- 4 止まるはずの操作を1回頼み、`permissions.deny` による拒否と返ることを確認してください。
- 5 並んでいるのに実行される場合は、書き方の側です。`Bash(git reset --hard *)` は文字列の照合なので、`/bin/rm` や `bash -c "..."` の形は当たりません。この穴を塞ぐのが `PreToolUse` フックです。

Tips : 設定の優先は `settings.local.json`、プロジェクトの `settings.json`、`~/.claude/settings.json` の順です。`deny` はどこに書いたものも合わせて効きます。

症状

CLAUDE.md に禁止と書いた操作が実行されます。あるいは、数ターン前に直したはずの PHP 8.0 の `match` 式が、新しい編集でまた入ります。

```
$label = match ($status) { 1 => '見積中', 2 => '受注', default => '' };
```

原因

CLAUDE.md は文脈であって設定ではありません。長い対話では優先度が下がり、会話の圧縮でも薄れます。公開されている不具合報告にも、禁止と書いた操作が実行された例と、指示どおりに戻せずコミットが消えた例が残っています。

直し方

- 1 破ると実害が出る約束を1つ選び、`permissions.deny` の1行に移してください。規則の形にします。
- 2 文字列の照合ですり抜ける形が思いつくなら、`PreToolUse` フックの側にも足してください。`block-destructive` はコマンド全文を見ます。
- 3 編集のたびに検査したい約束は `PostToolUse` に置いてください。対象バージョンの制約がこの位置です。
- 4 CLAUDE.md に残すのは、禁止の列挙ではなく代替の書き方にしてください。`match` を禁止と書くより「`switch` で書く」と書くほうが、次の編集に効きます。

--bare では読まれません。 `claude --bare -p` はフック、スキル、CLAUDE.md、MCP の自動読み込みを全部飛ばします。レビュー側に使うのは正しい用法ですが、実装側に使うと制約が1つも効きません。

症状

フックも拒否の規則も、まとめて効かなくなります。止まるはずの操作が止まらず、差し戻されるはずの編集がそのまま通ります。さっきまで効いていたものが同時に消えるのが特徴です。

原因

JSON として読めないファイルは、丸ごと無視されます。手で足していく演習なので、末尾のカンマ、閉じ括弧の過不足、`//` のコメントが原因になりやすい場所です。コメントを書けるのは `.jsonc` のほうだけです。

直し方

- 1 VSCode で `.claude/settings.json` を開き、赤い波線が付いている行を探してください。
- 2 末尾のカンマを消し、`//` で始まる行があれば削除してください。
- 3 PowerShell で読めるかどうかを確認めます。エラーなく通れば JSON として正しい形です。

```
Get-Content .claude\settings.json -Raw | ConvertFrom-Json
```
- 4 Claude Desktop のセッションを開き直し、止まるはずの操作を1回頼んで止まることを確認してください。

症状

元のフォルダでは止まっていたコマンドが、ワークツリーで始めたセッションでは素通りします。あるいは、ワークツリーの `.claude/settings.json` を開くと中身が配布時点のままです。

```
{
  "worktree": {
    "baseRef": "head"
  },
  "permissions": {
    "deny": [],
    "ask": [],
    "allow": []
  },
  "hooks": {}
}
```

原因

ワークツリーは新しいチェックアウトです。配布した `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最後のコミットから切られます。午前に足した `model`、`effort`、拒否の規則、フックの登録は、`ex/p3-` にコミットしてあれば入ります。入っていないのは、開始したときに `ex/p3-` 以外のブランチに乗っていたか、設定をコミットしていなかったかのどちらかです。

直し方

1. いったんセッションをアーカイブし、元のフォルダの VSCode で左下のブランチ名が `ex/p3-` で始まっているかを見てください。違っていたら `ex/p3-` に切り替えます。
2. ソース管理ペインに `.claude` の下の変更が残っていたら、`+` でステージし、メッセージを書いてコミットします。
3. もう一度、ブランチ名の右の **ワークツリー** にチェックを入れて始めてください。開いたら `.claude/worktrees/` の下のフォルダの `.claude/settings.json` に拒否の規則が入っていることを確認します。

- 4 フックの登録を自分で書き換えた方は、`.claude/settings.example.jsonc` の形と比べてください。フックは入力 JSON の `cwd` から作業フォルダを割り出すので、ワークツリーの中でも正しい場所を検査します。

症状

settings.json の effortLevel を medium にしたのに、入力欄の右下の effort の表示が変わりません。下げても、次のプロンプトで元に戻ります。サブエージェントの effort はバックグラウンドタスクのペインに出ないので、この症状はセッション本体の話です。

原因

解決の順は、環境変数 CLAUDE_CODE_EFFORT_LEVEL 、 --effort と /effort 、 settings.json の modelSettings と effortLevel 、モデル既定です。同じファイルの中では、モデルごとの modelSettings がトップレベルの effortLevel より優先されます。/effort で段を確定すると modelSettings に書き戻されるので、そのあとトップレベルを書き換えても負けます。もう1つ、 ~/.claude/settings.json のトップレベルに書いた effortLevel は Opus 5.5 と Sonnet 5.5 には効きません。プロジェクトの .claude/settings.json に書いた値は効きます。

直し方

- 1 版を確認してください。Claude Desktop のバージョンは設定画面で見られます。中の Claude Code が 2.1.280 以降であることが前提です。
- 2 環境変数が残っていないかを見ます。値が返ったら、それが最優先で効いています。

```
$env:CLAUDE_CODE_EFFORT_LEVEL
```
- 3 入力欄の右下のモデル名と effort の表示で、いま効いている値を見てください。
- 4 settings.json の modelSettings に今のモデルの行があれば、その行を書き換えるか、入力欄で /effort に続けて段を打って選び直してください。
- 5 上限で抑える手もあります。 maxEffortLevel はどのスコープから指定しても、最も低い上限が効きます。

Tips : モデルを切り替えるとプロンプトキャッシュが失効します。1つのセッションの中でモデルを行き来させるほど、使うトークンが増えます。

症状

frontmatter に `model: haiku` と書いた `grep-scout` を呼んだのに、バックグラウンドタスクの行には別のモデル名が出ます。

原因

サブエージェントのモデルは、呼び出し時の指定、定義の frontmatter の `model`、環境変数 `CLAUDE_CODE_SUBAGENT_MODEL`、親のモデルの順に決まります。frontmatter に書いてあれば環境変数より先に効くのがふつうです。ただし環境変数 `CLAUDE_CODE_SUBAGENT_MODEL_FORCE` が `1` のときは、`CLAUDE_CODE_SUBAGENT_MODEL` が最優先になります。社内の配布設定で入っていることがあります。版が `2.1.251` より前だと、`FORCE` が無くても環境変数が先に効きます。

直し方

- 1 統合ターミナルで、2つの環境変数に値が入っていないかを確認してください。

```
$env:CLAUDE_CODE_SUBAGENT_MODEL_FORCE  
$env:CLAUDE_CODE_SUBAGENT_MODEL
```

- 2 1行目が `1` を返した場合は、それが原因です。自分で設定したものなら外し、Claude Desktop を終了して起動し直してください。会社の配布設定で入っているなら講師に知らせてください。

- 3 外せない設定であれば、そのまま進めて構いません。台帳には「この端末では `CLAUDE_CODE_SUBAGENT_MODEL_FORCE` で環境変数が優先された」と1行書いてください。優先順位を自分の目で確かめたことが、この演習の中身です。

巻き戻しと隔離

AI が一度に広く書き換えたあとの戻し方です。チェックポイントの届く範囲、一部だけ戻す手順、自分の手直しと混ざったときの分け方、先に範囲を絞る方法、worktree での隔離を置いています。

症状

Esc を2回押しても、巻き戻しの一覧が開きません。画面は入力待ちのまま変わりません。開いたときは、地点を選んだあとの確認画面に復元の3択が並びます。

コードと会話
会話のみ
コードのみ

原因

Claude Desktop では Esc 2回の開き方が効きません。入力欄に `/rewind` と打って送るのが正しい開き方です。もう1つの原因は、戻す先が無い場合です。直前の変更がシェルのコマンドによるものなら、ファイルは戻りません。

直し方

- 1 入力欄に次を打って送ります。

`/rewind`

- 2 一覧から、戻したい地点のメッセージを選びます。

- 3 確認画面の「復元」で **コードのみ** を選び、**巻き戻し** を押します。ファイル編集ツールで変えたファイルだけが戻り、シェルのコマンドで動かしたファイルは残りません。

- 4 メッセージにマウスを乗せると出る巻き戻しのアイコンからも、同じ確認画面に進めます。

巻き戻し



セッションを以前のメッセージの状態に復元します。

こんにちは

/hooks

/tasks

git reset --hard HEAD を実行して

rm -rf tests を実行して

security-reviewer、spec-reviewer、test-gap-reviewer、review-other の4...

tests/phpunit/bootstrap.php の末尾に、文字列の部分一致を判定するヘル...

検証のためにわざとやりたい。contains_text の中身を str_contains(\$hayst...

/rewind を送った直後の画面です。これまでのメッセージが並び、戻したい地点の文を選ぶと確認画面に進みます

このメッセージまで巻き戻しますか？

検証のためにわざとやりたい。contains_text の中身を str_contains(\$haystack, \$needle) を使う1行に...

復元

コードと会話



1個のファイル・ +4 -1

シェルコマンドによる変更は元に戻されません。コミットはそのまま残ります。

キャンセル

巻き戻し

地点を選んだ後の確認画面です。戻るファイルの数と行数が出ます。「シェルコマンドによる変更は元に戻されません。コミットはそのまま残ります」の注記が、この機能の守備範囲そのものです

このメッセージまで巻き戻しますか？

検証のためにわざとやりたい。contains_text の中身を str_contains(\$haystack, \$needle) を使う1行に...

復元

コードと会話

コードと会話

会話のみ

コードのみ

キャンセル

巻き戻し

「復元」のプルダウンを開いたところです。 **コードと会話** は両方、 **会話のみ** はファイルを残して会話だけ、 **コードのみ** は会話を残してファイルだけ戻します。手順3で選ぶのは **コードのみ** です

戻せる範囲は狭いと考えてください。 チェックポイントが持つのは Edit と Write の変更と会話です。シェルのコマンドによる変更、サブエージェントの変更、手で直した分、シンボリックリンクは戻りません。保持の期間にも上限があります。ツールを跨いで効く保険は、タスク前後のコミットだけです。

症状

復元で **コードのみ** を選んだのに、ファイルが元に戻りません。プチ演習2では、この形で残ります。

```
$ ls app/libraries
sql_lib.php
util.bak
```

原因

ファイル名を変えたのがシェルのコマンド経由だからです。チェックポイントが持っているのはファイル編集ツールの変更だけで、シェルのコマンドで動かした結果は対象の外にあります。同じ理由で、サブエージェントの変更と、エディタで手で直した分も戻りません。AIに「全部やり直して」と頼んでも、この差は埋まりません。

直し方

- 1 VSCode のソース管理ペインを開いてください。シェルのコマンドで動かした分も、ここには変更として出ます。チェックポイントより広い範囲が見えます。
- 2 戻したいファイルの行にカーソルを合わせ、右端の **変更を破棄** (戻る矢印)を押してください。そのファイルだけが直前のコミットの状態に戻ります。
- 3 すでに push した変更を取り消す場合は、GitLab の Web 画面でマージ済みの MR を開き、右上の **Revert** を押してください。打ち消すコミットが新しく積まれ、履歴は残ります。
- 4 次のタスクに入る前にコミットしてください。戻せる単位はコミットの単位です。

Tips :「さっきの変更を戻して」と頼んだら `git reset --hard` が走った事例が報告されています(issue #17190)。拒否の規則を `Bash(...)` と `PowerShell(...)` の対で入れ、フックも入れておけば、この頼み方でも止まります。

症状

1回の依頼で20ファイルが書き換わりました。当たっている修正と、外している修正が混ざっています。巻き戻しの **コードのみ** は選んだ点まで全部を戻すので、当たっている分も一緒に消えます。

原因

チェックポイントの単位はセッションの時点で、ファイルや行では選べません。粒度を選ぶのは git の側です。AI は1つの依頼で広く触るので、人が手で書くときより、この粒度の差が効いてきます。

直し方

- 1 VSCode のソース管理ペインを開き、変更されたファイルの一覧を出してください。これが AI が触った範囲です。
- 2 戻したいファイルの行で **変更を破棄** を押してください。そのファイルだけが直前のコミットの状態に戻ります。
- 3 1つのファイルの中の一部だけ戻す場合は、ファイル名をクリックして差分を開き、戻したい塊の上で右クリックして **選択した範囲を元に戻す** を選びます。左側が元、右側が今の状態です。
- 4 残した変更だけでテストを流してください。AI の変更どうしが依存していることがあります。手元に PHP は入っていないので、コミットして push し、CI の test と test-phpunit のジョブの結果で確かめます。
- 5 どのファイルを残し、どれを捨てたかを台帳に1行書いてください。同じ依頼をもう一度出すときに、範囲を絞る材料になります。

AI を呼ぶ前のコミットが前提です。 直前のコミットが AI に渡す前の状態になっていれば、部分ロールバックは「差分を捨てる」だけで済みます。コミットせずに渡すと、戻す先が自分の手直しの途中になり、どこまで戻すかの判断から始まります。

症状

AI に直させたあと、自分でも手を入れています。あとから AI の分だけを取り消したくなくても、どの行が誰の分か分かりません。ソース管理ペインはファイル単位でしか教えてくれません。

原因

どちらも未コミットの変更として同じ場所に積まれています。チェックポイントはファイル編集ツールの変更しか持たないので、巻き戻しで **コードのみ** を選ぶと自分の手直しだけが残る形になり、意図した状態とずれます。

直し方

- 1 すでに混ざっている場合は、ソース管理ペインでファイルをクリックして差分を開いてください。自分が書いた塊の上で右クリックし、**選択した範囲をステージ** を選びます。
- 2 自分の分を全部ステージしたら、メッセージを書いてコミットしてください。「手直しの分」と1行で構いません。
- 3 残った未ステージの変更が AI の分です。この状態なら、丸ごと捨てるか、ファイル単位で戻すかを選べます。
- 4 どちらの分か判断が付かない塊は、捨てずに残してください。あとから履歴で追えるのは、コミットした分だけです。
- 5 次からは混ざる前に切ってください。AI に渡す直前にコミットを1つ打ち、AI の出力をそのまま2つ目のコミットにし、自分の手直しを3つ目にします。この順にすると、上の作業がまるごと要らなくなります。

Tips : 「AI が触った範囲」は、AI の出力だけを入れたコミットの差分そのものです。分けてコミットしておくと、あとから範囲を測れます。何ファイル触ったかという数字は、次の依頼の粒度を決めるときに効きます。

症状

止めたいコマンドは拒否の規則とフックで止まります。ただ、「30ファイルを一度に書き換える」「頼んでいないディレクトリまで触る」といった範囲の広さは止まりません。気づくのは終わったあとです。

原因

拒否の規則もフックも、見ているのはコマンドの種類だけです。変更の量と場所は見えていません。範囲は依頼の文面に書いてあるだけで、機械が守っている決まりにはなっていません。

直し方

- 1 依頼の時点で触ってよい場所を書いてください。「app/libraries/util.php の中だけ」のように、先に決めます。

- 2 PreToolUse のフックで、Edit と Write の対象パスを見てください。範囲の外なら 2 で返します。

```
if ($payload.tool_input.file_path -notlike "*\app\libraries\*") { exit 2 }
```

- 3 件数で見る場合は PostToolUse の側です。変更されたファイルの本数を数え、決めた本数を超えたら標準エラーへ出してください。止めずに知らせるだけでも効きます。

```
(git status --short | Measure-Object).Count
```

- 4 作業そのものを隔離すると、範囲はそのワークツリーの中に収まります。ブランチ名の右の **ワークツリー** にチェックを入れて始めてください。

- 5 どの絞り方を選んだかを台帳に書いてください。パスで絞ったのか、件数で知らせたのか、隔離したのかで、戻す手間が変わります。

先に絞った分だけ、戻す作業が小さくなります。 範囲の制限は止め金であると同時に、巻き戻しの設計でもあります。AI が触る範囲を10ファイルに抑えれば、部分ロールバックで見る差分も10ファイルで済みます。

症状

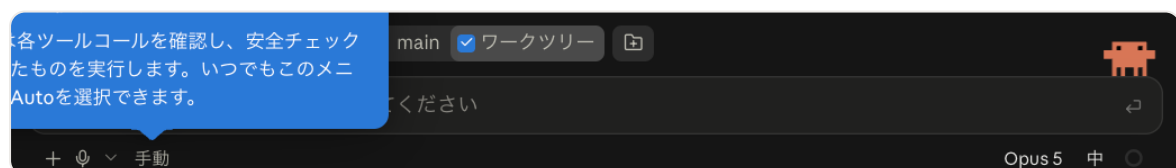
手順に「隔離して起動してください」と書いてあります。どこから作るのかが分かりません。

原因

ワークツリーはアプリの画面から作ります。入力欄の上の行で、ブランチ名の右に **ワークツリー** のチェックがあり、チェックを入れて始めると、同じリポジトリの別のチェックアウトで作業が始まります。元のフォルダはそのまま残るので、手元の作業と並べて進められます。

直し方

- 1 始める前に `.claude` の変更をコミットしておいてください。ワークツリーは開始したときのブランチの最後のコミットから切られ、コミットしていない設定は持ち込まれません。
- 2 ブランチ名の右の **ワークツリー** にチェックを入れ、入力欄に1行送ってセッションを開始してください。名前を入れる欄はありません。
- 3 入力欄の上に出た `worktree-` で始まるブランチ名を控えてください。作業用のフォルダは `<プロジェクトルート>/claude/worktrees/` の下に1つできます。
- 4 作業が終わって `push` まで済んだら、サイドバーのアーカイブのアイコンで削除してください。



ブランチ名の右の **ワークツリー** にチェックを入れた状態です。この状態でセッションを始めると、`.claude/worktrees/` の下に作業用のコピーが作られます

症状

worktree で作業を始めたら、動いていたはずのものが動きません。設定ファイルを探すと、`.env` のような無視対象のファイルが入っていません。

原因

worktree は新しいチェックアウトなので、入るのは追跡されているファイルだけです。`.gitignore` に書いたファイルは持ち込まれません。鍵や接続先を `.env` に置いている場合、隔離した側からは見えなくなります。

直し方

- 1 プロジェクトルートに `.worktreeinclude` があるかを見てください。演習リポジトリには配布時点が入っています。

- 2 持ち込みたいファイルを1行ずつ書きます。

```
.env
.env.local
```

- 3 もう一度 worktree を作り直し、書いたファイルが入っていることを確認してください。

- 4 鍵そのものを書かないでください。持ち込むのはファイルの名前で、中身は元の場所から写されます。`.worktreeinclude` はコミットするファイルで、演習リポジトリには `.env` の行が入った状態で配布しています。

症状

隔離して作業したあと、元のフォルダで作業を続けようとするとう弾かれます。

```
fatal: 'worktree-issue-1' is already used by worktree at  
'C:/work/dl-training-app/.claude/worktrees/...'
```

ブランチ名とフォルダ名は、ワークツリーを作ったときに自動で付いた名前になります。この資料の `worktree-issue-1` は、控えた名前に読み替えてください。

あるいは、元のフォルダのソース管理ペインを見ても、隔離中に作ったコミットが出てきません。

原因

同じブランチを2つの作業ツリーで同時にチェックアウトすることはできません。隔離中のコミットはワークツリーのブランチの側にあり、元のフォルダの `ex/p3-` には入っていません。元のフォルダは `ex/p3-` のままで正しい状態です。

直し方

- 1 いまある作業ツリーの一覧を見てください。ここはコマンドを使います。

```
git worktree list
```

- 2 隔離したフォルダを VSCode でもう1つ開いてください。 **ファイル** から **フォルダーを開く** で、 `.claude/worktrees/` の下にできたフォルダを選びます。ここに隔離中のコミットが見えます。

- 3 その窓のソース管理ペインから push してください。初回は **ブランチの発行** の表示になります。

- 4 MR は GitLab の Web 画面で作ります。push 後に画面上部へ出る **Create merge request** から、ソースに控えたブランチ(この資料の `worktree-issue-1`)、ターゲットに `main` を選んでください。

- 5 push が済んでから片付けます。サイドバーの `worktree` の行にあるアーカイブのアイコンを押すと削除されます。

終了時の確認を読み飛ばさないでください。 隔離したセッションを終えるとき、worktree を残すか消すかを聞かれます。消すと、その中の未 push のコミットも一緒に消えます。押す前に、push が済んでいるかを確認してください。

対象バージョンの制約とテスト

このプロジェクトの対象は PHP 7.4 です。検査は手元の `check-phpver` フックと、CI の `lint-phpver` の2か所です。フックは `phpcs` の `PHPCompatibility` があればそれを使い、無ければ正規表現で見ます。`lint-phpver` は `php -l` と `php ci/undefined_functions.php` の2段です。どこで捕まえたかによって直す場所が変わる5件を置いています。

症状

いま動いている環境の版と、これから移す先の版が違います。どちらを検査の対象に書けばよいか迷います。

原因

検査の対象を現行の版に合わせると、移行のときに書き直す量が増えます。逆に新しすぎる版を書くと、いまの環境で動かないコードが通ってしまいます。この演習では移行先の 7.4 を対象に置いています。

直し方

- 1 リプレースの予定がある場合は、移行先の版を対象に書いてください。現行より新しい構文は止まりますが、移行のときにそのまま持っていけます。
- 2 対象は設定ファイルの1か所書き、フックと CI の両方がそこを読む形にしてください。2か所書くと、片方だけ上げたときに食い違いします。
- 3 AI が素で書きたがるのは、もっと新しい版の構文です。 `match` 式、 `enum` 、 `readonly` 、名前付き引数、コンストラクタプロモーション、 `str_contains` 、 `?->` は、どれも 7.4 には入っていません。
- 4 現行の環境でしか動かない部分がある場合は、そこだけ別の対象を持たせてください。1つのリポジトリに複数の対象があること自体は、おかしいことではありません。

構文検査を通っても実行時に落ちるものがあります。 `str_contains()` は関数の呼び出しなので、 `php -l` の構文検査は通ります。7.4 で実行すると `Call to undefined function` になります。CI の `lint-phpver` はこれを2段目の `php ci/undefined_functions.php` で落とします。構文だけを見る層では捕まえられないので、関数の有無を見る段を別に置いてください。

症状

手で `phpcs` を叩くと、コマンドが見つからない旨のメッセージになります。フックの側は `phpcs` が無くても何も表示せずに正規表現の検査へ進むので、差し戻しはそのまま起きます。 `phpcs` は入っているのに規格が無い端末では、フックの標準エラーに次の1行が出て、同じく正規表現の検査へ進みます。

```
[check-phpver] PHPCompatibility standard not installed, using the regex fallback
```

原因

端末に PHP と `phpcs` を入れていないためです。演習では入れなくても進みます。フックは `phpcs` が見つからないときに正規表現で代替し、対象外の構文と関数を並べた表の分だけ見ます。

直し方

- 1 そのまま進めてください。正規表現版でも `match` 式、`enum`、`readonly`、`?->` などの構文と、`str_contains()` や `str_ends_with()` などの関数の呼び出しは捕まります。
- 2 誤検知が出た場合は、指摘された行を開いて確認してください。正規表現版はコメントと引用符の中身を外してから照合しますが、文字列の中のエスケープまでは追わないので、まれに当たります。
- 3 最終の判定はCIの `lint-phpver` です。1段目の `php -l` が構文を、2段目の `php ci/undefined_functions.php` が関数の有無を見ます。
- 4 自社の環境に入れる場合は、規格の配布元を見てください。
<https://github.com/PHPCompatibility/PHPCompatibility> です。導入の手順は社内の PHP 環境に依存するため、この場では扱いません。

症状

該当する構文が無い行が差し戻されます。

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

```
C:\...\app\views\estimate_list.php:44 nullsafe operator (?->) (8.0) -> write isset($a) ? $a->b : null
```

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.

原因

正規表現版はコメントを外し、引用符で囲まれた文字列の中身を空にしてから照合します。それでも、文字列の中にエスケープした引用符があると区切りを読み違い、画面に出す文言の一部が照合に残ることがあります。文字コードの読み違いでも起こります。日本語コメントを CP932 として読むと、コメントの区切りが崩れて同じ形に見えます。

直し方

- 1 指摘された行を開き、実際に 7.4 で動かない構文や関数があるかを目で確かめてください。
- 2 無ければ、文字列の書き方の側です。エスケープした引用符を含む文言を変えられるなら変え、変えられないなら phpcs を入れて正規表現版を使わない形にします。
- 3 ほかの行でも同じ誤検知が出る場合は、読み込みの文字コードを疑ってください。Get-Content に -Encoding UTF8 が付いているかを見ます。
- 4 アロー関数 `fn()` と型付きプロパティは 7.4 で動きます。これが指摘されたら、ブックを自分で書き換えたときの取りこぼしです。

症状

push すると `lint-phpver` だけが赤になり、後ろの `test` と `test-phpunit` が動きません。ログには、1段目で落ちたときは `PHP Parse error` で始まる行と `Errors parsing` に続くファイル名の行が、2段目で落ちたときは次の行が出ます。

undefined_functions: PHP 7.4 に無い関数と呼んでいます。php -l は通りますが実行時に落ちます。

原因

`lint-phpver` は対象バージョンの PHP 7.4 で2段の検査をします。1段目の `php -l` は構文を見るので、`match` 式のような 8.0 以降の構文はここで落ちます。2段目の `php ci/undefined_functions.php` は、呼んでいる関数が 7.4 にあるかを `function_exists()` で確かめるので、`str_contains()` のような 8.0 で入った関数はここで落ちます。手元のフックを登録していれば編集の直後に差し戻されるはずなので、落ちたということは、その端末でフックが効いていないか、CI 側だけに届いた変更です。

直し方

- 1 ログに出た行番号を開き、対象バージョンで動く書き方に置き換えてください。
`match` 式なら `switch`、`str_contains()` なら `strpos()` です。ハンズオン1のように Issue が「直さずに記録」を求めている場合は、置き換えずに行番号とエラーの1行目をそのまま記録します。
- 2 同じ構文や関数がほかにも無いかを確認してから push してください。
- 3 フックが効いていなかった方は、`settings.json` の `hooks` に `PostToolUse` が並んでいるかを確認してください。

段ごとに見ているものが違います。 1段目は構文だけ、2段目は関数の有無だけを見ます。クラスやメソッドが 7.4 にあるかどうかは、どちらの段も見えていません。ジョブが緑になったことは、対象バージョンで動く証明にはなりません。

症状

フックのテストケースを足したあと、`lint-phpver` のジョブが落ちます。ログの `Errors parsing` の行に、足した見本のファイル名(`.claude/hooks/tests/fixtures/` の下の `.php`)が出ます。

原因

CI の lint はツリーの中の `*.php` を全部 `php -l` に掛けます。対象バージョンで動かない見本を `.php` の名前で置くと、見本そのものが構文エラーとして拾われます。配布した見本の拡張子が `.php.txt`なのはこのためです。

直し方

- 1 足した見本のファイル名を `.php.txt` に戻してください。
- 2 `.claude/hooks/tests/cases.json` の `fixture` の値も、同じ名前に直してください。

- 3 単体テストを流し、全件が合格することを確認してください。

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

- 4 ランナーは実行のたびに一時ディレクトリへ `.php` として写します。検査そのものは、拡張子を変えても同じように走ります。

GitLab と CI

push、パイプライン、AI レビューのジョブで止まる6件です。CI で止まったときは、ジョブのログの最初の10行と最後の10行で切り分けられます。

症状

push すると VSCode の右下に赤い通知が出ます。出力ペインの **Git** を開くと、この文面が入っています。

```
remote: GitLab: You are not allowed to push code to protected branches on this project.  
! [remote rejected] main -> main (pre-receive hook declined)
```

原因

main にブランチ保護がかかっており、合流の経路は MR だけです。受講者の権限は Developer なので、直接 push は必ず弾かれます。ブランチを切らずに編集を始めたときに出ます。

直し方

- 1 VSCode の左下、青いステータスバーにいまのブランチ名が出ています。main になっていないかを確認してください。
- 2 main のままなら、そのブランチ名をクリックし、上に出る一覧から **新しいブランチの作成** を選び、ex/p1-yamada のように、その演習のブランチ名を付けてください。コミット前の変更はそのまま移ります。
- 3 ソース管理ペインでメッセージを書き、チェックマークでコミットしてください。続けて **ブランチの発行** を押すと、そのブランチが GitLab 側にできます。
- 4 MR は GitLab の Web 画面で作ります。push のあと、プロジェクトの画面上部に **Create merge request** のボタンが出ます。
- 5 ボタンが出ていない場合は **Code** から **Merge requests** を開き、**New merge request** でソースに自分のブランチ、ターゲットに main を選んでください。

Tips : ブランチ名のユーザー名の部分は、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。一覧で見たときに、誰の演習かが分かります。

症状

GitLab の **Code** > **Branches** と **Merge requests** を開くと、ご自身が作った覚えのない行が受講者の人数ぶん並びます。パイプラインの一覧も同じです。

原因

演習リポジトリは `dl-training-app` の1つで、受講者全員がここを使います。書き込む先は `ex/<演習番号>-<ユーザー名>` という人ごとのブランチに分かれており、ユーザー名は社用メールアドレスの「@ より前」なので重なりません。`main` にはブランチ保護がかかっていて、合流の経路は MR だけです。その MR も演習中はマージしないため、`main` は2日間動きません。他の方の変更がご自身の手元へ入ってくることも、ご自身の変更が他の方の画面に現れることもありません。

直し方

- 1 直す必要はありません。全員の行が見える状態が正しい状態です。
- 2 ご自身の行は、一覧の上にある検索欄にご自身のユーザー名を入れて絞ってください。ブランチ名に入っています。
- 3 他の方のブランチには切り替えず、他の方の MR にも **Merge** を押さないでください。守るのはこの2つだけです。
- 4 仕組みの詳しい説明は、共通の章の「1つのプロジェクトを全員で使う形」にあります。

Tips : MR が開いたまま残ることには意味があります。2日間に誰が何を試して、どの指摘をどう判断したかが、MR の一覧としてそのまま記録になります。

症状

MR を出してもジョブが `pending` のままです。ジョブを開くと、上部にこの文言が出ます。

This job is stuck because the project doesn't have any runners online assigned to it.

タグを付けた場合は次の形になります。

This job is stuck because you don't have any active runners online or available with any of these tags assigned to them: ...

原因

ジョブを実行する runner が空いていないか、オフラインです。runner は1台です。全員が同じ時間に push すると、並列数の上限で順番待ちになります。ハンズオン3では `nightly_implementation` が1本最長30分 runner を使うので、その間に出した MR のパイプラインも後ろに並びます。この場合の `pending` は異常ではありません。

直し方

- 1 2分ほど待ってから、パイプラインの画面を再読み込みしてください。順番待ちなら、ここで動き始めます。
- 2 5分以上動かない場合は、講師へお知らせください。runner の状態は受講者の権限では見られません。
- 3 待っているあいだは、手元でできる作業を先に進めてください。レビューの記録と、MR の説明欄の記入は CI を待ちません。

Status	Pipeline	Created by	Stages	Actions
Passed	#31 null 合体演算子を isset 3項に書き直す(PH... 11.1 4b2b955f latest merge request		✓-○-○	▶ ⬇
Passed	#30 null 合体演算子を isset 3項に書き直す(PH... 11.1 4b2b955f latest merge request		✓-○-○	▶ ⬇
Passed	#29 null 合体演算子を isset 3項に書き直す(PH... ex/h1-yamada 4b2b955f latest branch		✓-○-○	▶ ⬇
Warning	#28 Merge branch 'ci/al-review-fix-20260913... main 868254b8 latest branch		✓-○-1	▶ ↺ ⬇
Passed	#27 Merge branch 'ci/al-review-fix-20260913... main 868254b8 latest branch		✓-○-○	▶ ⬇

手順1で再読み込みする **Build** > **Pipelines** の一覧です。左端の **Status** 列が状態で、この画面には緑の **Passed** と橙の **Warning** だけが写っています。順番待ちの行は同じ列に **pending** と出ます。自分の行は、**Pipeline** 列のコミットメッセージの下に付くブランチ名で探るか、上の検索欄に自分のブランチ名を入れて絞ります。 **branch** と **merge request** のラベルは同じ push で走った2本の区別です

ジョブ画面の上部に出る文言です。 **stuck** の語が出ていれば順番待ちではなく、runner 側の問題です。

This job is stuck because the project doesn't have any runners online assigned to it.

症状

ai_review ジョブは緑で終わるのに、MR にコメントが1本も付きません。ジョブのログにこの1行が出ています。

```
[ai_review] CI/CD変数 ANTHROPIC_API_KEY が未設定のため、レビューを実行せずに終了します
```

artifacts の findings.json を開くと、findings が空で "skipped":"no_api_key" が入っています。指摘が0件なので ai_gate も緑になり、ゲートが何も止めていない状態で MR がマージできる形になります。

トークン側が足りないときは、こちらは落ちます。ログの末尾はこの形です。

```
ci/ai_review.sh: line 20: GITLAB_BOT_TOKEN: CI/CD変数 GITLAB_BOT_TOKEN が未設定です
```

ハンズオン3の nightly_implement は、キーが無いと先頭の確認で止まります。「実装を開始します」の行は出ず、ci_logs/ も作られません。

```
ci/nightly_implement.sh: line 15: ANTHROPIC_API_KEY: CI/CD変数 ANTHROPIC_API_KEY が未設定です
ERROR: Job failed: exit code 1
```

原因

ジョブが使う変数が登録されていないか、登録はされていても **Protect variable** が付いています。保護付きの変数は、保護されたブランチのパイプラインにしか渡りません。受講者のブランチは保護対象ではないので、値が届かずこの形になります。ai_review が緑で終わるのは、読むだけのジョブを環境の不備で赤にしない作りをしているためです。落とす判断は ai_gate の側にあり、指摘が0件なら通ります。緑だから安全、とは読めません。

直し方

- 1 これは講師側の設定で、受講者の権限では CI/CD の変数の画面は見られません。講師へお知らせください。
- 2 講師から直ったと連絡があったら、MR の画面の **Pipelines** タブで該当の行の右端にある再実行のボタンを押してください。push をやり直す必要はありません。
nightly_implement の場合は、同じ手動ボタンをもう一度押します。
- 3 キーが無いまま緑になった ai_review の回は、計測表の「CI の ai_review」の行に「未実施」と書いてください。指摘0件として数えると、比較が壊れます。

鍵をリポジトリに置かないでください。 手元で値を渡して回避したくなりますが、コミットに入ると取り消せません。鍵を使う処理はCI側に寄せるのが、この演習で採る形です。

症状

ai_gate のログには止めた記録が出ているのに、パイプライン全体は緑で、MR はマージできる状態のままです。ジョブ名の横に **Allowed to fail** の表示が付いています。

しきい値 P1 以上、確信度 0.0 以上の指摘を数えます。
confidence の無い指摘は 1.0、P0～P3 以外の severity は P0 として数えます。
P1: 2 件 / P2: 2 件

ゲートで止めた指摘 2 件

[P1] app/controllers/estimate_ctl.php:36 date_to が条件に使われず、範囲の上限が効きません

直すか、直さない理由をMRのコメントに書いてから再実行してください。

原因

.gitlab-ci.yml の ai_gate に allow_failure: true が入っています。初期状態はこれです。列挙だけを見せて、落とす判断は受講者が自分で入れる順番にしています。なお、この行を外してパイプラインが赤になっても、演習のプロジェクトでは **Merge** ボタンは押せます。 **Pipelines must succeed** を設定していないためです。押せますが押しません。

直し方

- 1 .gitlab-ci.yml を開き、ai_gate の節の allow_failure: true の1行を消してください。
- 2 コミットして push し、パイプラインが赤くなることを確認してください。ここで1回落ちる記録が、この演習の合格の形です。
- 3 指摘を直してもう一度 push し、緑に戻してください。
- 4 落とす重大度を変えたい場合は、allow_failure ではなく GATE_LEVEL を動かしてください。P1 はP0とP1を止め、P2 はそこにP2を加えます。

Tips : 指摘の中身を変えたいのか、通す基準を変えたいのかは、別の話として扱ってください。前者は REVIEW.md 、後者は GATE_LEVEL です。2つのジョブに分けてあるのは、後から追えるようにするためです。

症状

`ai_review` は緑で、MR のコメントも投稿されているのに、指摘が差分の前半のファイルに偏ります。後半で入れた変更には1件も付きません。

原因

差分が長いと、先頭の一定量で打ち切ってからモデルに渡しています。打ち切ったことはプロンプトには書いていますが、指摘そのものは前半に偏ります。

直し方

- 1 MR を分けてください。1本を小さく保つほうが、しきい値を上げるより効きます。レビューの質は差分の長さで下がります。
- 2 分けられない事情があるときだけ、CI/CD Variables に `AI_REVIEW_DIFF_CHAR_LIMIT` を足して上限を上げてください。
- 3 上げた場合は、モデルの選択も見直してください。差分の行数で切り替えるしきい値は `AI_REVIEW_DIFF_THRESHOLD` です。
- 4 台帳には、どこまで読まれた状態の指摘なのかを1行残してください。件数の比較に効きます。

夜間ループと Stop hook

無人で回すときに止まる9件です。1回目は止まる前提で流します。どこで止まったかを計測表に書くところまでが演習なので、緑にならなくても先へ進んでください。

手動ボタンが押せない

ハンズオン3

症状

パイプラインの一覧に `nightly_implement` の再生ボタンが出ていない、または灰色で押せません。ジョブを開くと、上部にこの文言が出ています。

This job requires a manual action

別の形では、ジョブの状態が `created` か `skipped` のまま動きません。

原因

3つのどれかです。1つ目は、前の段(lintとtest)がまだ終わっていない場合で、`implement`の段はその後に来ます。2つ目は、押しているパイプラインのブランチに権限が無い場合です。保護ブランチのパイプラインの手動ジョブは、そのブランチへのマージ権限を持つアカウントだけが押せます。演習の `main` は Developer にもマージを許しているので、受講者も `main` の手動ジョブを押せます。押せても押しません。ハンズオン3のStep 3は講師が `main` で流し、Step 5で各自が自分のブランチで押します。3つ目は、pipeline schedule から起動されたパイプラインを見ている場合で、そこでは手動ボタンではなく自動で走る形になります。

直し方

- 1 **Build** > **Pipelines** の行の **Stages** で、lint と test の丸が緑になるまで待ってください。test-phpunit が2分ほど長くかかります。
- 2 緑になったら画面を再読み込みし、いちばん右の丸に再生ボタンが出ていることを確認してください。
- 3 押すのはハンズオン3のStep 5で、自分のブランチのパイプラインの行です。それより前に押してしまった場合は、そのジョブの **Cancel** を押してください。

症状

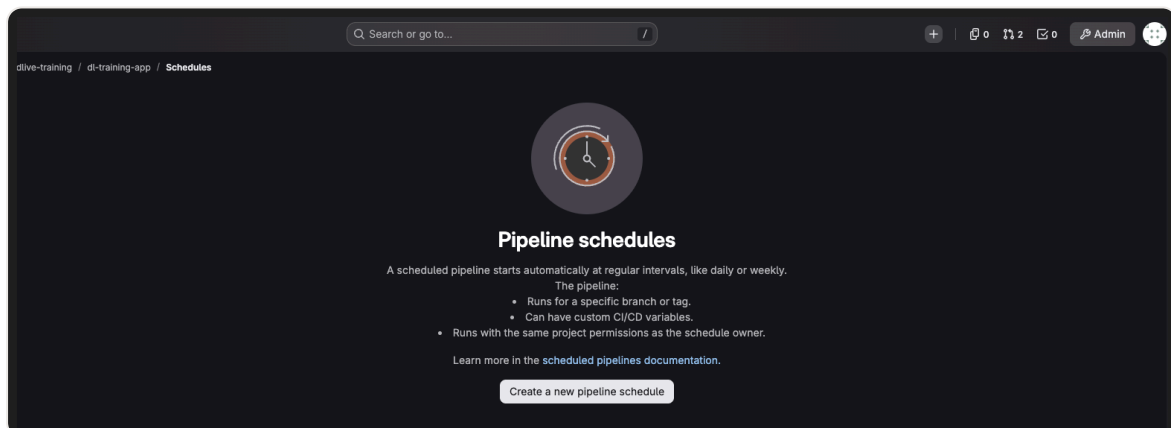
Pipeline schedules の一覧で、自分の行が **Active** になっています。 **Next Run** に翌日の 2:00 が入っています。

原因

Activated のチェックが既定で入っており、外さずに保存しています。このままだと翌日の 2:00 に受講者の人数ぶんの **nightly_implement** が同時に走り、runner と API の予算を使います。

直し方

- 1 **Build** > **Pipeline schedules** を開き、自分の行の **Active** のトグルを切ってください。表示が **Inactive** に変わります。
- 2 すでに走ってしまった場合は、**Build** > **Pipelines** で起動元が **schedule** の行を探し、**Cancel** を押してください。
- 3 行の右端の再生ボタン(**Run pipeline schedule**)も押さないでください。schedule を手で即時起動するボタンで、同じことが起きます。



手順1で開く **Build** > **Pipeline schedules** の画面です。写っているのはまだ schedule が1本も無いときの表示で、中央の **Create a new pipeline schedule** から作ります。有効な schedule があると、ここが一覧に変わり、行に **Active** のトグルと **Next Run** が出ます。この画面に戻っていれば、翌日に走るものはありません

症状

「実装を開始します」の行のあと、ログが動かないまま30分が過ぎ、末尾にこの行が出て終わります。

```
ERROR: Job failed: execution took longer than 30m0s seconds
```

artifacts の `ci_logs/` は残りますが、`claude-issue-4.json` が空か、途中で切れた JSON になっていることがあります。

原因

`.gitlab-ci.yml` の `timeout: 30m` が効いています。 `--max-turns 15` は回数の上限で、1ターンが長いと回数に届く前に時間のほうが先に来ます。 `tests/run_tests.php` が DB の接続待ちで固まった場合と、影響範囲の調査で `grep-scout` がツリー全体を読みに行った場合に起こります。

直し方

- 1 `timeout` と `--max-turns` は上げないでください。1本あたりの所要時間と費用が読めなくなります。
- 2 計測表には「時間の上限で打ち切られた」と書き、直す層は「Issue の粒度」を選んでください。Issue の「対象ファイル」を2本に絞った版を予備の Issue #5 の形で書くのが、研修後の課題です。
- 3 ジョブ画面の上部の経過時間が動いているあいだは、打ち切りではなく実行中です。25分を過ぎたら講師へお知らせください。

症状A

`nightly_implement` は数十秒で終わり、ログの末尾はこうなります。ブランチもコミットもできていません。

```
turns: 1 / cost_usd: 0.01 / 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  
ご覧ください。
```

結果 JSON の `result` には、ツールの使用許可を求める文が入っています。
配布のスクリプトでは3つの引数がそろっているのですが、ここで終わるのは次の症状Bのほうが多いです。

症状B

結果 JSON(`ci_logs/claude-issue-4.json`)の `permission_denials` に `curl` を使おうとした記録が入り、`result` が「受入条件が特定できない」で終わります。拒否は `.err` には出ません。

原因

症状Aは、非対話の `-p` では確認が要る操作がすべてその場で拒否されるためです。確認を出す相手がないので、書き換えもコマンドも通らずに終わります。無人で動かすには `--permission-mode dontAsk` と `--allowedTools` と `--max-turns` の3つで、許す範囲と回数の上限を決めます。症状Bは、SKILL.mdの手順1がIssue本文の取得に `curl` を使う一方、`--allowedTools` に渡しておらず、`ci/settings.ci.json` の `deny` にも `Bash(curl *)` が入っているためです。SKILL.mdが意図した止め金として書いている箇所で、ハンズオン3の1回目はここで終わります。

直し方

- 1 `ci/nightly_implement.sh` の起動行を開き、3つの引数がそろっているかを確認してください。

```
claude -p "$PROMPT" \  
  --permission-mode dontAsk \  
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash/php *),Bash(git add *),Ba  
sh(git commit *)" \  
  --max-turns 15 \  
  --output-format json
```

- 2 `Bash(git add *)` の `*` の前の半角スペースを消さないでください。 `Bash(git add*)` と書くと別のコマンドにも当たります。

3 結果 JSON を開き、拒否されたツールの名前を確かめてください。 `--allowedTools` に無いツールを使おうとして止まっている場合は、許すべきツールかどうかを先に判断します。

4 ブランチ作成と push は、エージェントではなくスクリプトの仕事にしています。ここを許すと、止め金の外で push できてしまいます。渡すツールを増やす前に、その線を動かしてよいかを考えてください。

5 症状Bの直し方は1通りです。 `ci/nightly_implement.sh` の PROMPT に「Issueの本文は `_issues/issue-4.md` にあります」の1行を足して、Read で読ませます。 `--allowedTools` に `Bash(curl *)` を足しても、 `ci/settings.ci.json` の `deny` が先に効くので通りません。外へ出る経路を増やさない形が、夜間の実行に向いています。

許すツールの一覧が、無人運転の仕様です。 対話のときは人が1つずつ判断していた部分が、ここでは書いた一覧だけになります。増やす前に、そのツールが外へ出るかどうかを見てください。

症状

ジョブは緑で終わっているのに、MR の一覧に何も増えません。ログの末尾はこの1行です。

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。

原因

差分が1行も無いときは MR を作らない作りにしています。原因は3つのどれかです。許可待ちで止まった、Issue の本文を読んで「変更不要」と判断した、途中でターン数の上限に当たった。

直し方

- 1 ジョブ画面の右側の **Job artifacts** で **Browse** を押し、`ci_logs` の中の `claude-issue-4.json` をクリックして開いてください。
- 2 `num_turns` を見ます。15 に張り付いていれば上限で打ち切られています。Issue の粒度が大きすぎるので、範囲を狭めてください。
- 3 1 か 2 で終わっていれば許可待ちです。前の項目の手順へ進んでください。
- 4 `result` に「変更は不要」という趣旨の文が入っていた場合は、Issue の 期待する 振る舞い と 受入条件 を読み直してください。人が読んで曖昧なら、エージェントも同じところで迷います。
- 5 止まった箇所と、どの層の設定で直すかを台帳に1行書いてから再実行してください。

症状

コミットと push は通っているのに、MR が立ちません。 `ci_logs/` とジョブログの末尾に GitLab の応答が出ます。

```
remote: GitLab: 403 Forbidden - You are not allowed to create merge request
```

原因

夜間ジョブは、push に使ったトークンの権限で MR を作ります。 `CI_JOB_TOKEN` では通りません。 `GITLAB_BOT_TOKEN` のスコープが足りない場合も同じ形になります。

直し方

- 1 これは講師側の設定です。まず講師へお知らせください。
- 2 `GITLAB_BOT_TOKEN` をプロジェクトアクセストークン(ロール Developer、スコープ `api` と `write_repository`)に差し替えます。
- 3 ブランチは push できているので、GitLab の画面から **Create merge request** を押せば先へ進めます。止まった箇所は「MR の作成で止まった」として記録してください。

症状

2回目の実行で、最後の push だけが落ちます。

```
! [rejected]          HEAD -> nightly/issue-4 (non-fast-forward)
error: failed to push some refs to 'https://gitlab-09291006aidev.give-app.net/...'
hint: Updates were rejected because the tip of your current branch is behind its
hint: remote counterpart.
```

原因

1回目の `nightly/issue-4` が残っており、今回作り直したブランチとは別の歴史になっています。強制 push は拒否の規則で禁じているので、CI にも例外を作っていません。

直し方

- 1 GitLab の **Code** から **Branches** を開いてください。
- 2 `nightly/issue-4` を探し、右端のごみ箱で削除してください。前回の MR が開いていれば、先に閉じます。
- 3 自分のブランチのパイプラインで、もう一度 `nightly_implement` の手動ボタンを押してください。
- 4 前回の結果を残したい場合は、`ISSUE_ID` を予備の 5 に変えて流します。
New pipeline の **Variables** の欄に、キー `ISSUE_ID`、値 5 を1行入れて起動します。ブランチ名が `nightly/issue-5` になるので衝突しません。

Tips：強制 push を CI だけ例外にすると、止め金の意味がなくなります。夜間に動くものほど、例外を作らないほうが後で楽になります。

症状

応答が終わりません。同じ理由の継続が何度も繰り返され、テストの実行だけが延々と走ります。

```
php tests/run_tests.php failed with exit code 1. Run it, read the failing case,
fix the code, then finish.
```

原因

継続で呼び戻されたときに `stop_hook_active` が `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、同じ継続が繰り返されます。配布した `stop-gate` は、この判定を先頭に置いています。講師が先に見せる壊れた版は、その3行を外したものです。

直し方

1 Esc でいったん止めてください。

2 `.claude/hooks/stop-gate.ps1` の先頭の判定が残っているかを確認してください。

```
if ($payload.stop_hook_active -eq $true) { exit 0 }
```

3 継続の回数にも上限を置いてください。環境変数 `STOP_GATE_MAX` で変えられます。既定は同じセッションで2回までです。

4 回数を数えている作業メモを消してから、もう一度試してください。

```
Remove-Item .claude\hooks\.stop-gate.state -ErrorAction SilentlyContinue
```

5 正しい版では、1回の連鎖で継続が入るのは1回だけです(`stop_hook_active` が `true` になって2回目は素通りします)。同じセッションで依頼を繰り返すと累計が上限(既定2)に達し、次の行を出して終わります。

```
[stop-gate] already continued 2 time(s), the cap is 2. Letting the session
stop.
```

止める条件を先に書いてください。 続ける条件から書くと、止まらない版が先にできます。無人で回すものは、止まらない側の失敗のほうが高くなります。

作業メモがコミットに混ざる

プチ演習6

症状

コミットの前にソース管理ペインを見ると、見覚えのないファイルが並びます。

```
.claude/hooks/stop-gate.ps1      M
.claude/hooks/.stop-gate.state    U
```

原因

継続の回数をセッションごとに数えているファイルです。テストが通れば自動で消えますが、赤いまま終わると残ります。手元の作業メモなので、共有するものではありません。

直し方

- 1 `.gitignore` に1行足してください。

```
.claude/hooks/.stop-gate.state
```

- 2 すでにコミットへ入れてしまった場合は、追跡だけを外してください。ここはコマンドを使います。

```
git rm --cached .claude/hooks/.stop-gate.state
```

- 3 ソース管理ペインでは、コミットしたいファイルの行の `+` だけを押してステージしてください。まとめて上げる癖をやめると、この種の混入は起きません。

進行と画面共有

進め方そのもので止まったときの2件です。リモートの方も多く参加されます。手が止まったときに一番早いのは、画面を見せることです。

共有した画面が読めない

共通

症状

共有した画面について、講師からこう返ってきます。

文字が小さくて読めません。ターミナルだけを共有していただけますか。

あるいは、共有を始めた瞬間に別の話になります。

いまの画面、社内のリポジトリが映っています。共有を止めてください。

原因

画面全体を共有すると、ターミナルの文字が小さくなり、開いている別のウィンドウも一緒に映ります。自社ハーネスを開いた VSCode が映ると、そこで扱いが変わります。

直し方

- 1 共有するのは、画面全体ではなくウィンドウ単位にしてください。VSCode か、ブラウザの GitLab の画面のどちらか1つです。
- 2 ターミナルの文字を大きくしてから共有してください。VSCode では `Ctrl` を押しながらホイールを回すと変わります。
- 3 自社ハーネスを開いているウィンドウは共有しないでください。中身を見せる必要がある場合は、講師と個別のやり取りに切り替えます。
- 4 エラーの文面は、画像ではなくテキストでチャットへ貼ってください。こちらで検索して、同じ症状の受講者に一度で返せます。
- 5 台帳の提出もチャットで結構です。現地の方と同じ形式で受け取ります。

Tips : 共有を待っているあいだも、手は止めないでください。1つの Step で詰まったら、次の Step の「自分で考える」の欄を先に埋めておくと、復帰が早くなります。

症状

Step の [Nmin] を超えても終わりません。次の Step の説明が始まってしまいます。

原因

[Nmin] は目安で、全員が同じ速さで進む前提には置いていません。CI の待ち時間と、レビューの読み込みで差が出ます。

直し方

- 1 OK 基準の欄を開き、いくつ満たしているかを数えてください。すべて満たす必要はありません。
- 2 満たしていない項目のうち、次の演習の土台になるものだけを先に片付けてください。プチ演習の成果物は、後の演習で使います。
- 3 残りは「追加と考察」の扱いにして、先へ進んでください。止まったまま待つより、次の Step の前半を聞くほうが得るものが多くなります。
- 4 どこで詰まったかを台帳に1行書いてください。Day2 の冒頭で、詰まりの多かった箇所から返します。



AIカスタム研修 / 2026年9月29日・10月6日 / データライブ株式会社様向け

© Givery, Inc. All rights reserved.

※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。