Givery, Inc.

ハンズオンガイド Day1

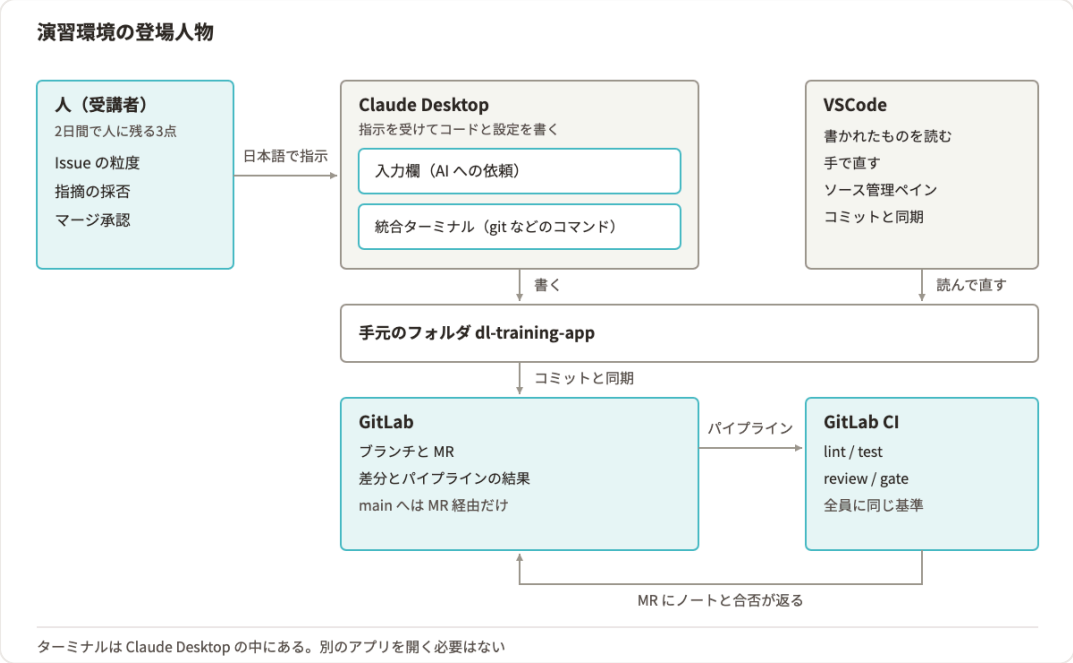
AIカスタム研修 ／ 2026年9月29日（火）

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

目次

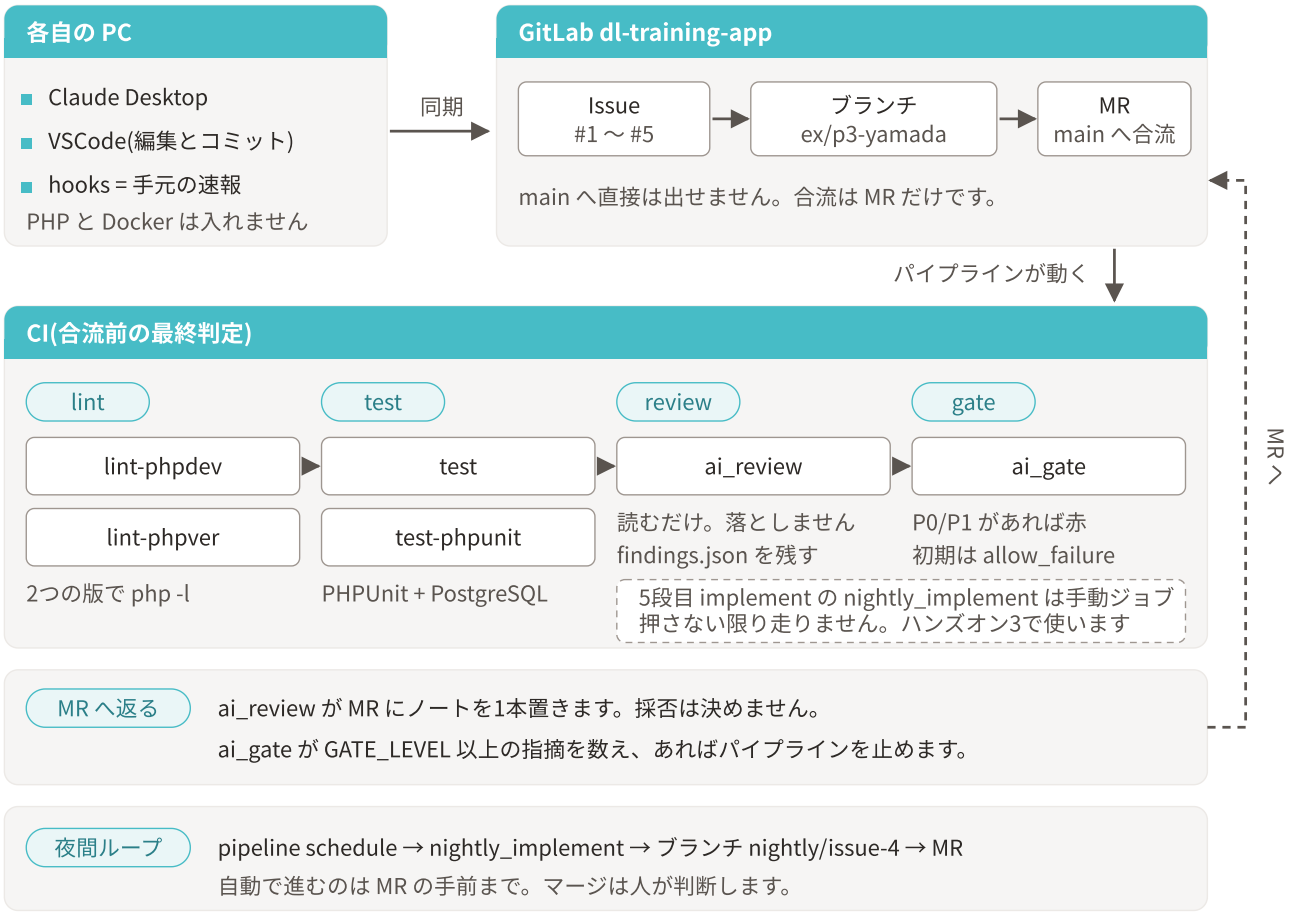
演習環境の全体像

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



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

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

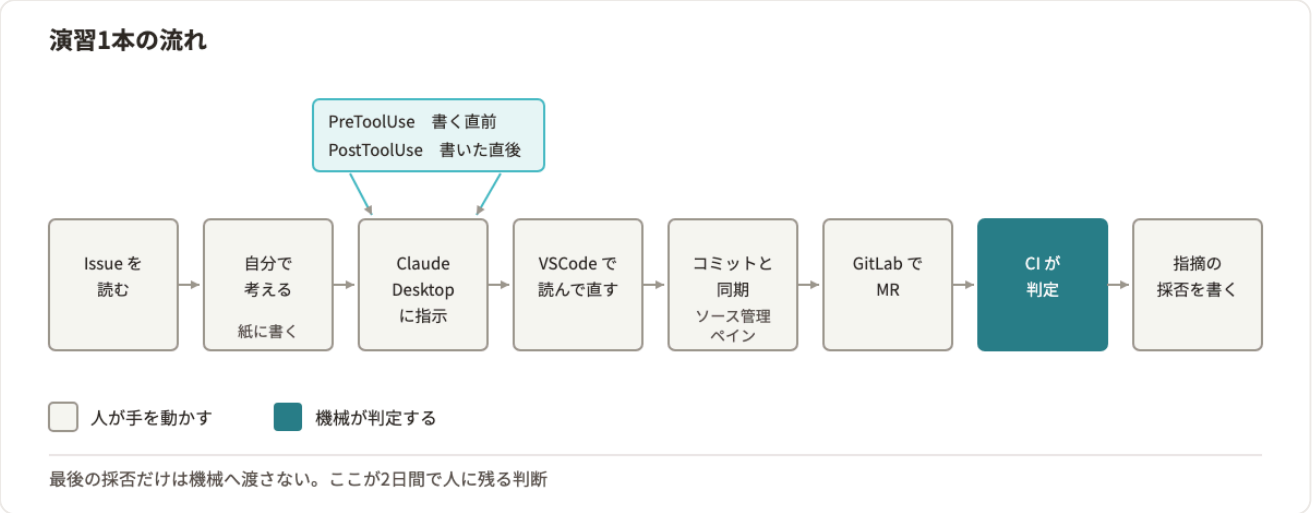


上段が手元と GitLab、中段が MR を出したときに動く CI のジョブです。中段右下の破線が5段目の implement で、手動でしか動きません。右端の点線は、CI の結果が MR へ戻ってくる経路を示します。最下段の夜間ループはハンズオン3で扱います。

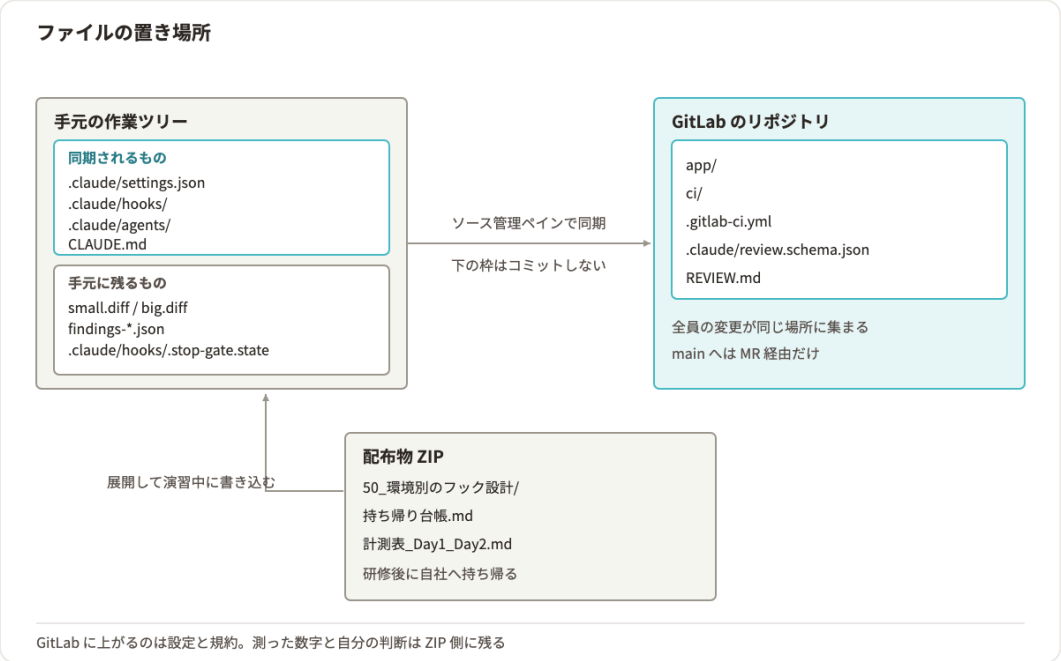
図に出てくる語のうち、3つだけ先に押さえてください。MR(Merge Request)は、自分のブランチを `main` に合流させる申請です。`ai_review` は MR の差分を AI に読ませ、指摘をコメントとして置くジョブです。`ai_gate` は、その指摘に重大なものがあればパイプラインを落とすジョブです。残りの語は末尾の用語集にあります。

この図で押さえる4点

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



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



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

手元のフックと CI に同じ検査を置く理由

同じコミットに2本のパイプラインが並ぶ仕組み

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

各演習の読み方

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

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

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

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

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

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

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

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

確認が出ないモード

Mac をお使いの方の読み替え

共通の進め方

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

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

STEP 1　Claude Desktop の基本動作

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

1. プロジェクトフォルダを選びます。新しいセッションを始めると、入力欄の上の行で環境、プロジェクトフォルダ、ブランチを選べます。プロジェクトフォルダに演習リポジトリ `dl-training-app` のフォルダを指定します。

2. パーミッションモードを **手動** にします。入力欄の下の行の左にあるモード名を押して選びます。新しいセッションを開いたときは、送る前にここを一度見てください。

3. 指示を出します。入力欄に日本語の文章を貼り付けて送ります。ファイル名は文章の中にそのまま書いて構いません。

4. 結果を見ます。書き換えたファイルの名前と差分が返信の中に並びます。実行前に確認が出たら、その1回を許可する側の選択肢を選びます。書かれた中身は VSCode で開いて自分の目で確かめてください。

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

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

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

この Step が終わった状態

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

補足 CLI の位置づけ

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

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

STEP 2 ブランチ名の形

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

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

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

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

main に戻らない理由

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

STEP 3 VSCode のソース管理ペイン

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

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

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

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

切り替えで止まったときの見方

この Step が終わった状態

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

STEP 4 GitLab の Web 画面

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

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

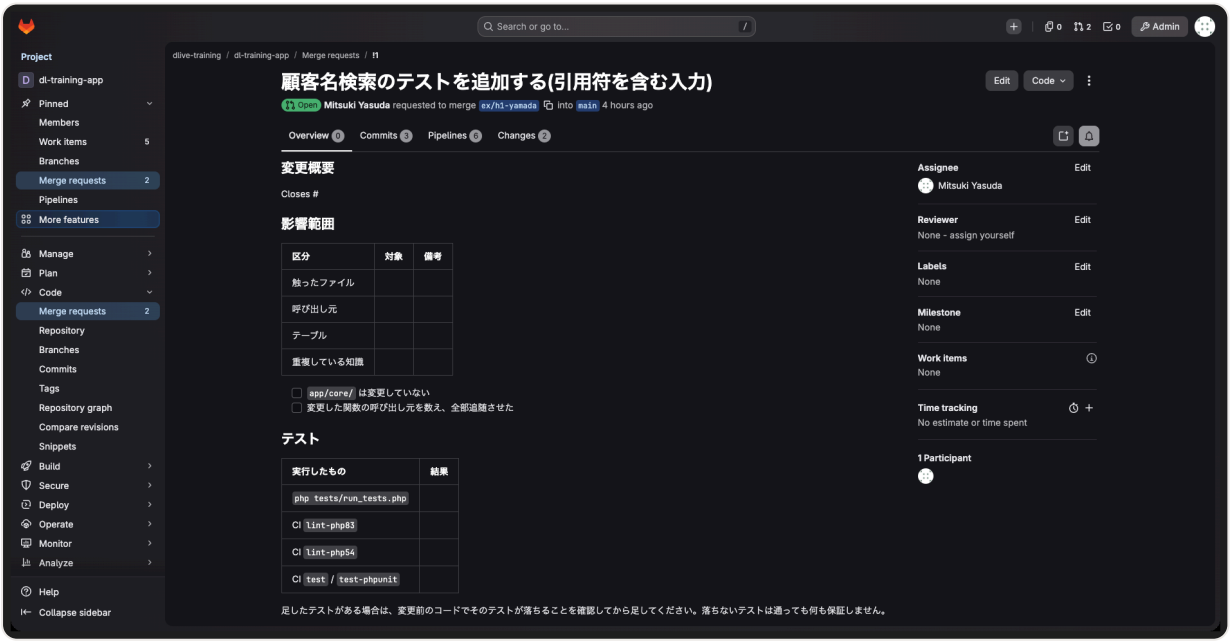
2. 合流先が `main` 、元が自分のブランチになっていることを確かめます。

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

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

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

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



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

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

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

落ちたジョブのログの読み方

マージを自分で押さない理由

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

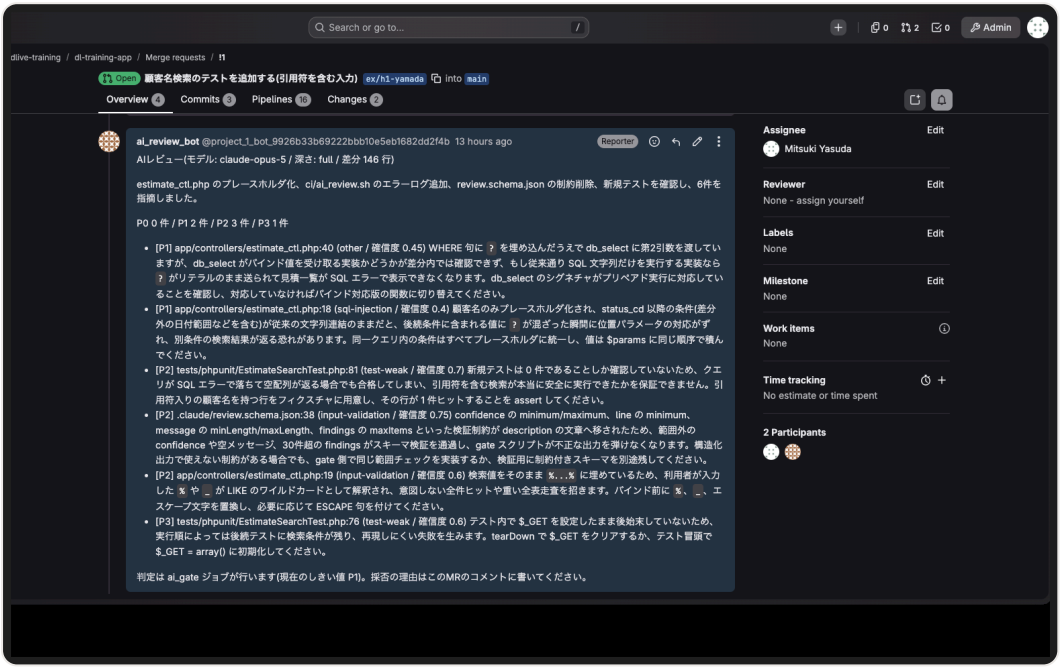
> **Tips：** `ai_gate` は初期状態で `allow_failure: true` です。落ちてもパイプライン全体は赤になりません。ハンズオン2でこの1行を外し、本当に合流を止めるゲートにします。 `test-phpunit` だけ他より2分ほど長くかかりますが、止まっているように見えても待ってください。

STEP 5　AI レビューのノートの読み方

[4min]

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

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



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

ノートの本文の例

P0 から P3 の線引き

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

補足 統合ターミナルとコマンド

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

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

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

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

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

1つのプロジェクトを全員で使う形

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

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

作業がぶつからない理由

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

他の方の手元への影響

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

他の方のブランチと MR の見え方

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

パイプラインの順番待ち

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

マージを演習中に行わない理由

ブランチ名の接頭辞で見分けられること

記録台帳と計測表

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

台帳 持ち帰り台帳.md

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

台帳の書き方の例

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

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

計測 計測表

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

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

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

計測表の書き方の例

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

この章の持ち帰り物

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

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

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

用語集

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

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

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

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

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

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

フック4イベントの位置

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

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

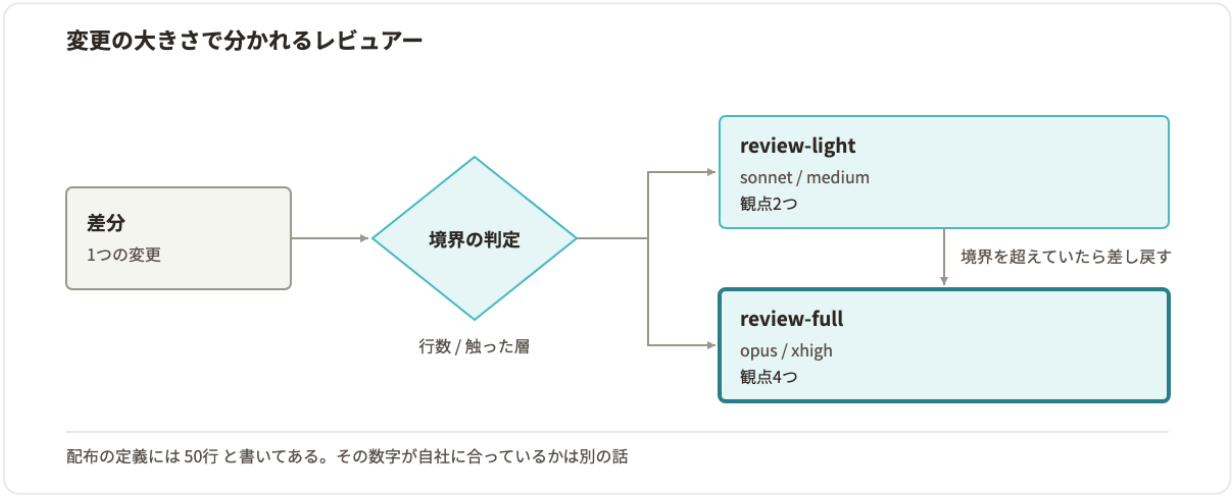
プチ演習1　規模別レビュアーの2定義

D1-05　レビュアーの切り替え境界

目的

変更の大きさでレビュアー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	`.claude/agents/review-light.md` 、`.claude/agents/review-full.md` 、`.claude/settings.json` 。差分を作るために `app/libraries/util.php` と `app/controllers/stock_ctl.php` 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、`settings.json` の3行(`model` / `effortLevel` / `maxEffortLevel`)、記録表4行
所要	[20min]



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

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行は配布時点で入っているので、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。

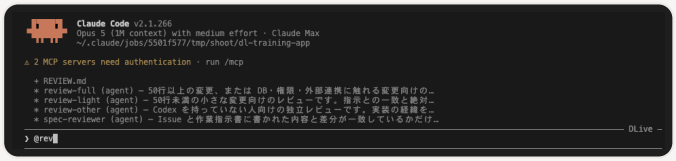
```
git diff --output=small.diff -- app/libraries/util.php
git diff --stat -- app/libraries/util.php
```

--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「レビュアー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

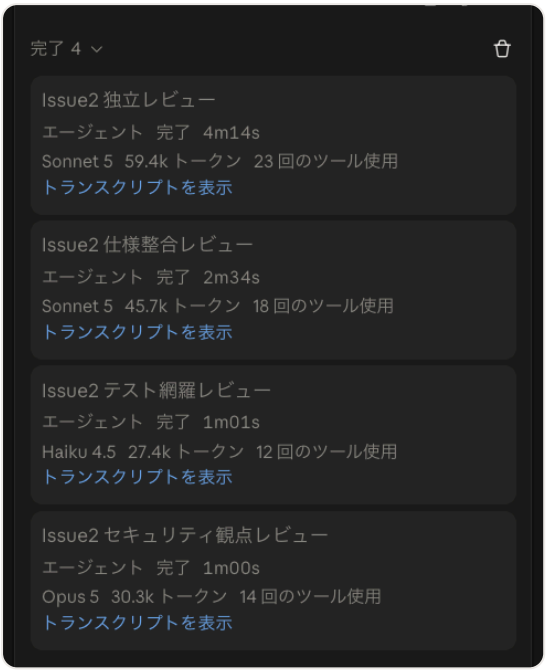
見る場所	バックグラウンドタスクのペイン(エージェント名、モデル名、トークン数)、返答末尾の `判定:` の1行
できた形	小さい差分では観点2つ分の出力と `判定:` の1行だけが返り、DB に触る依頼では `review-full の対象です` の1行が返る

- [ ] `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、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` が出ていれば、定義した frontmatter がそのまま効いています。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

レビュアーを2本に分ける理由

モデルとエフォートの対応

トークンの数え方

つまずいたときは

追加と考察、発展課題

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
`review-light.md`	`.claude/agents/` 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	`.claude/agents/` 直下。中身の差はありません
`review-full.md`	同上。既存の4観点はこちらへ寄せます	同上
`settings.json` の3行	既存の `settings.json` 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	`持ち帰り台帳.md` の1行	同じ

プチ演習2　巻き戻し制約の機械強制

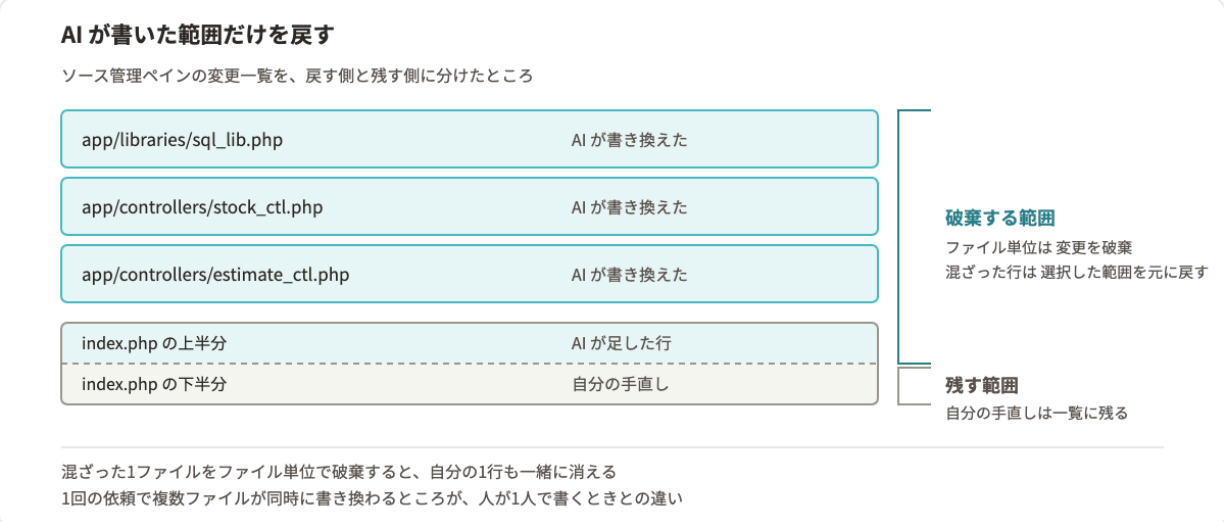
D1-07　戻せない操作の停止

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	`.claude/settings.json` の `permissions`（`ask` と `deny`）、`CLAUDE.md` の戻し方の節。実験用に `index.php`、`app/libraries/util.php`、`app/controllers/` 配下を動かします
作るもの	確認に回す `ask` 3本、戻せない操作を止める `deny` 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。



> 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。

> 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

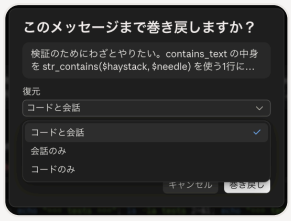
1. **[2min]**　Claude Desktop の入力欄に次を貼って送り、2つの変更を続けて実行させます。1つ目はファイル編集ツール、2つ目はシェルのコマンドです。確認はどちらも、その1回を許可する側の選択肢を選びます。

   ```
   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。`.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

- [ ] **コードで巻き戻したあと、** `index.php` は戻り `util.bak` が残ることを自分の目で見た
- [ ] **編集を受け入れる** のあと、`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

> 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

混ざる前に避ける手と、混ざった後の手

ファイル数での制限

コミット済みの変更を戻すとき

CLI で同じことをする場合

つまずいたときは

追加と考察、発展課題

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
`ask` 3本	`permissions.ask` 。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
`deny` 10本	既存の `permissions.deny` に追記します。 `PowerShell(...)` の側を落とさないでください	同じ10本。bash を正にしていても、Windows の端末で動かす人がいれば対のまま置きます
`CLAUDE.md` の戻し方の1行	`CLAUDE.md` 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

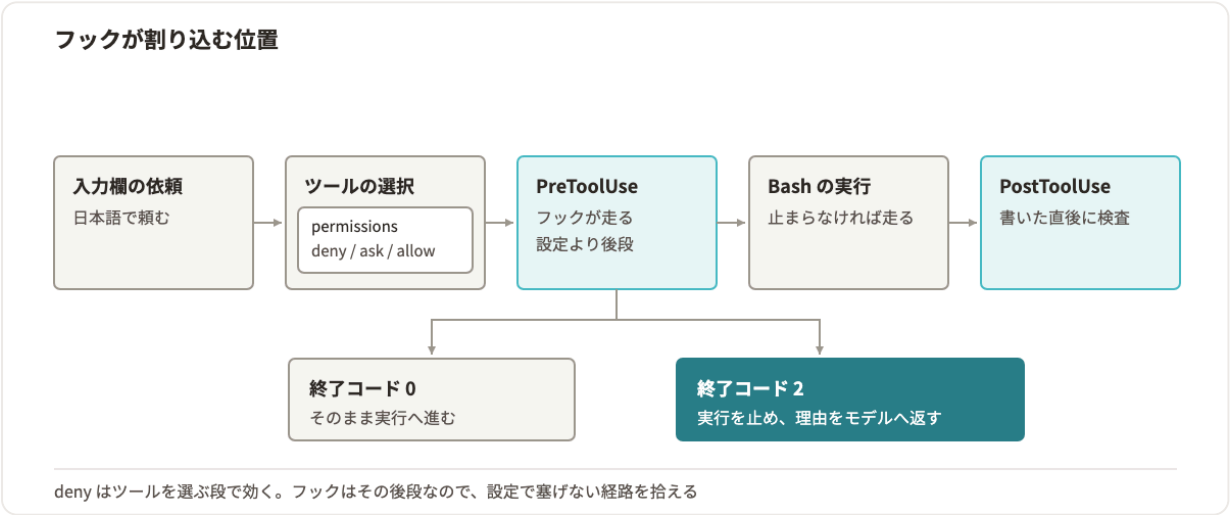
プチ演習3　秘密情報の遮断とフックの単体テスト

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



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

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", "${C
LAUDE_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。

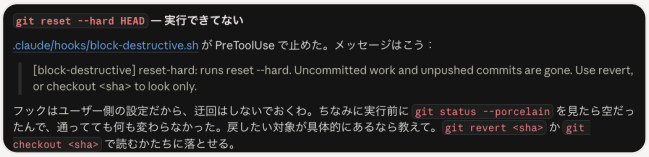
```
{
  "id": "ps-ng-my-secret",
```

```
  "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。今度は止まります。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。



> 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。

達成状態

見る場所	`$LASTEXITCODE`、ランナーの `PASS` / `FAIL` 行と末尾の件数、`settings.json` の登録内容
できた形	手で流した JSON に理由の2行と `2` が返る。ランナーが `23 cases: 23 passed, 0 failed`。Claude Desktop から `.env` 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

> 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

止める幅の決め方

PowerShell で引っかかるところ

CLI で同じことをする場合

追加と考察、発展課題

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。`.gitattributes` に `*.sh text eol=lf` 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 `/mnt/c` 配下では `chmod +x` 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	`jq` 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と、環境が増えても入口を1本足すだけで済みます。組み合わせ別の設計は、配布物の `50_環境別のフック設計/環境別のフック設計.md` にまとめてあります。

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 `deny` 3本	既存の `permissions.deny` に追記します。`Read(...)` は PowerShell の `Get-Content` には効かないので、フックと組にします	同じ3本
`block-destructive.ps1`	`hooks/` 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
`block-destructive.sh`	持ち込みません	`hooks/` 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	`hooks/tests/cases.json` 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

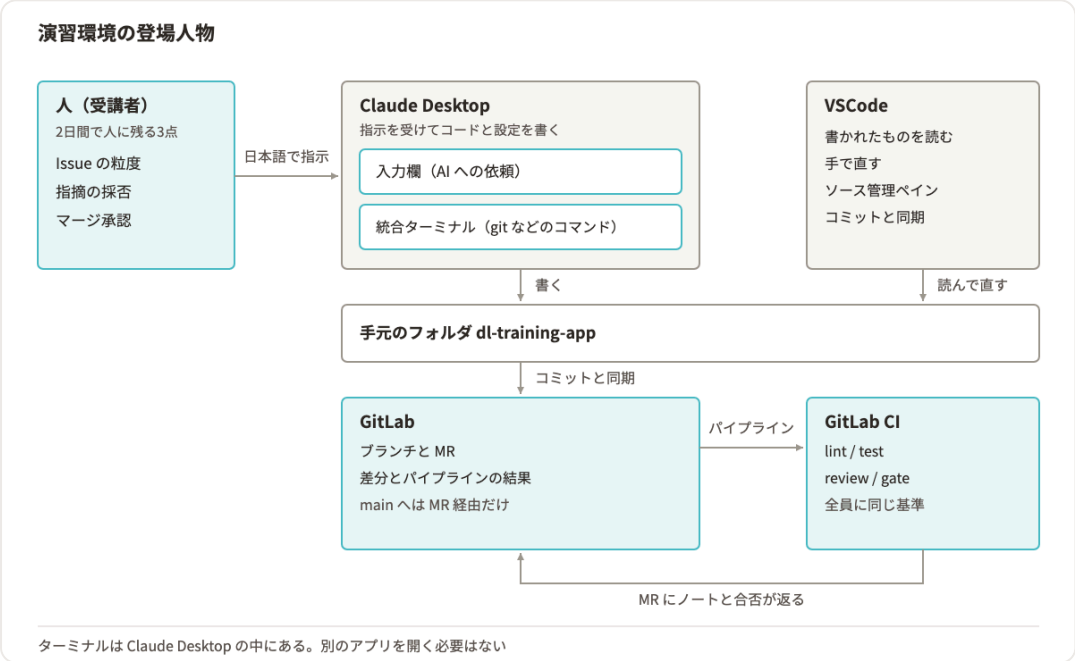
Issue 1本を調査から MR まで一周させ、同じ差分を3通りで読ませたときに指摘がどう変わるかを、自分の数字で言えるようになります。

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。



> 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 `dl-training-app` 。読むだけのものは Issue #1、`REVIEW.md` 、`.claude/agents/review-other.md`
直前の演習の成果物	プチ演習1〜3で `.claude/settings.json` に足した model、effort、ask、deny、PreToolUse フックの設定。`ex/p3-` 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/dlive-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あります。

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。

> 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	`app/controllers/estimate_ctl.php` の `index()` 、`tests/phpunit/EstimateSearchTest.php` 、MR の説明欄、`docs/shijisho/issue-1.md`
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、grep-scout に調査を委譲する。テストを先に push 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 `test-phpunit` が赤から緑に変わる履歴、`lint-phpver` 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、`lint-phpver` 以外の3ジョブが緑(`lint-phpver` 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	`main` は保護されていて直接 push できない。worktree には `.env` と `vendor/` が入らない。`date_from` / `date_to` と `sql_lib.php` は範囲外

この演習で使う語と、作るものの置き場所

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

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

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「作業を始めます」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

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

CLI から同じことをする場合

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。

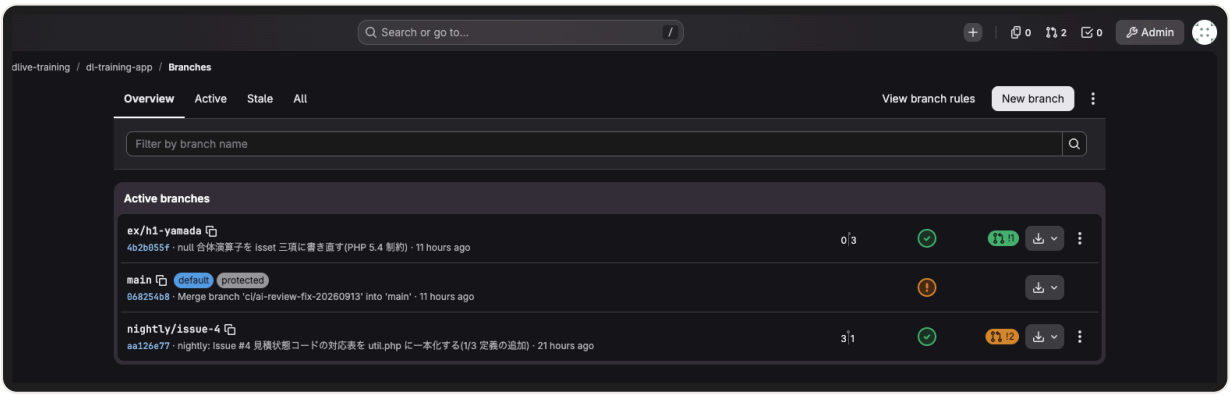
7 実装に入ります。Step 1 で書いた改修方針を、Claude Desktop に渡してください。渡すのは方針と範囲外の2つです。角括弧の中を自分の紙の内容で埋めます。

> 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

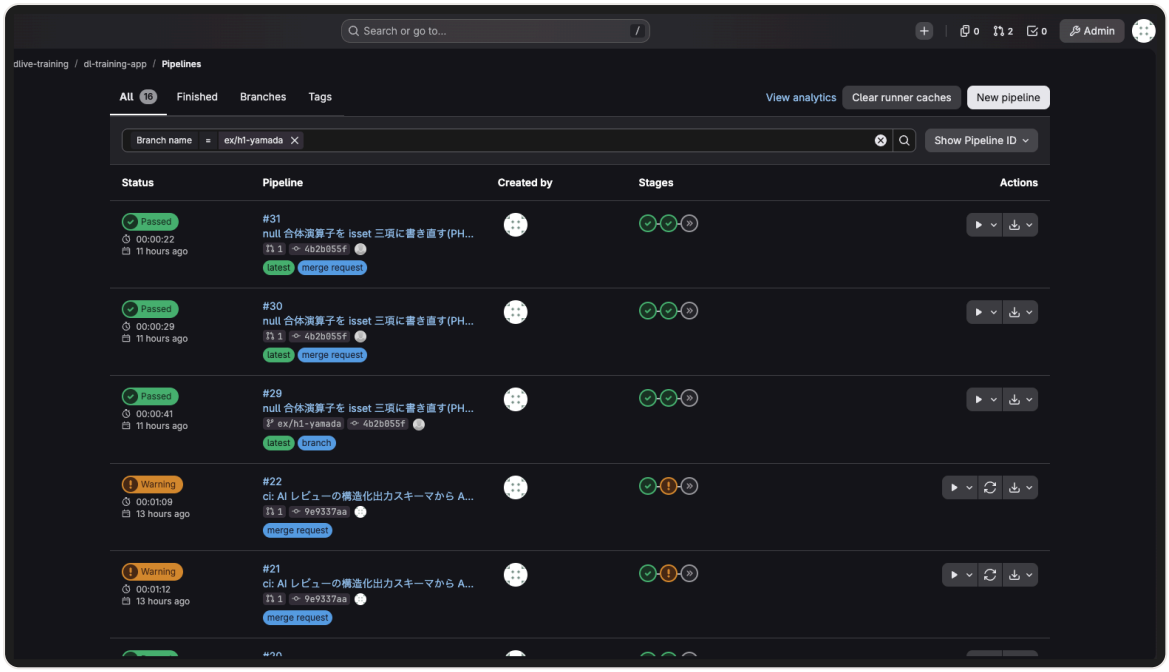
8 ソース管理ペインで `app/controllers/estimate_ctl.php` をクリックし、差分を自分で読みます。方針と違う場所に手が入っていたら、その場で Claude Desktop に戻させてください。

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

PHP 7.4 に合わせる書き方と、静的検査を通り抜けるもの

CI の出力の読み方と、赤と緑の見分け

手が止まったときの確認先

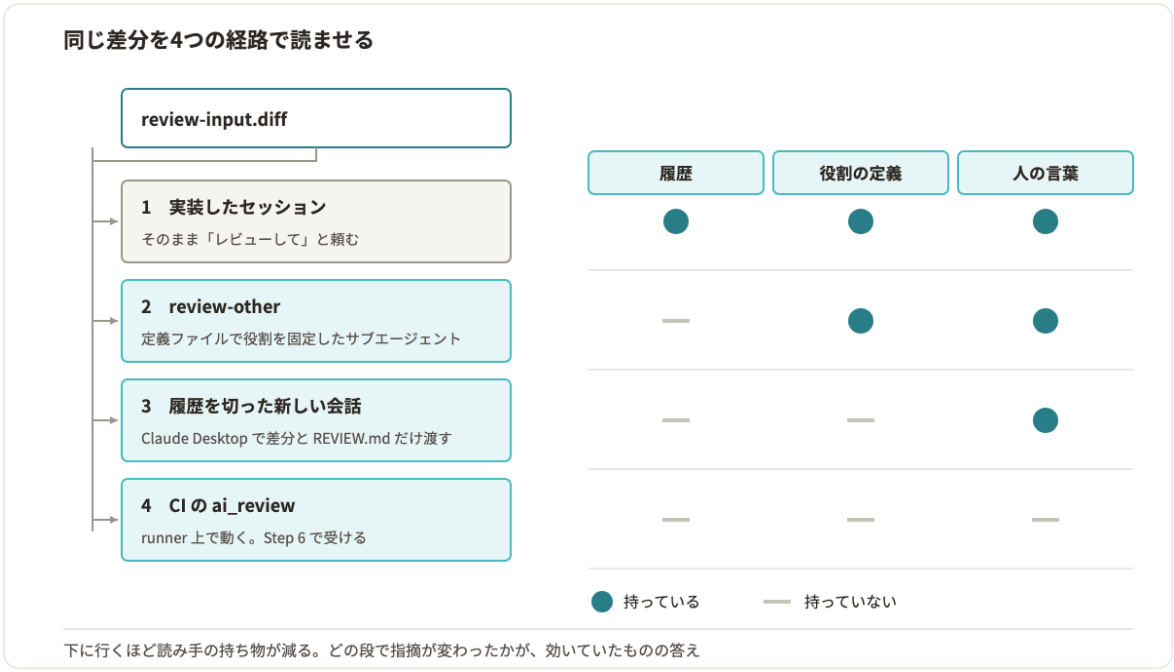
ハンズオン1　3通りのレビューと採否

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、`REVIEW.md`、役割の指定	
2 `review-other` サブエージェント	`REVIEW.md`、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 `REVIEW.md` だけ	役割と観点の定義
4 CI の `ai_review`	差分と `REVIEW.md`。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れます。



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)。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

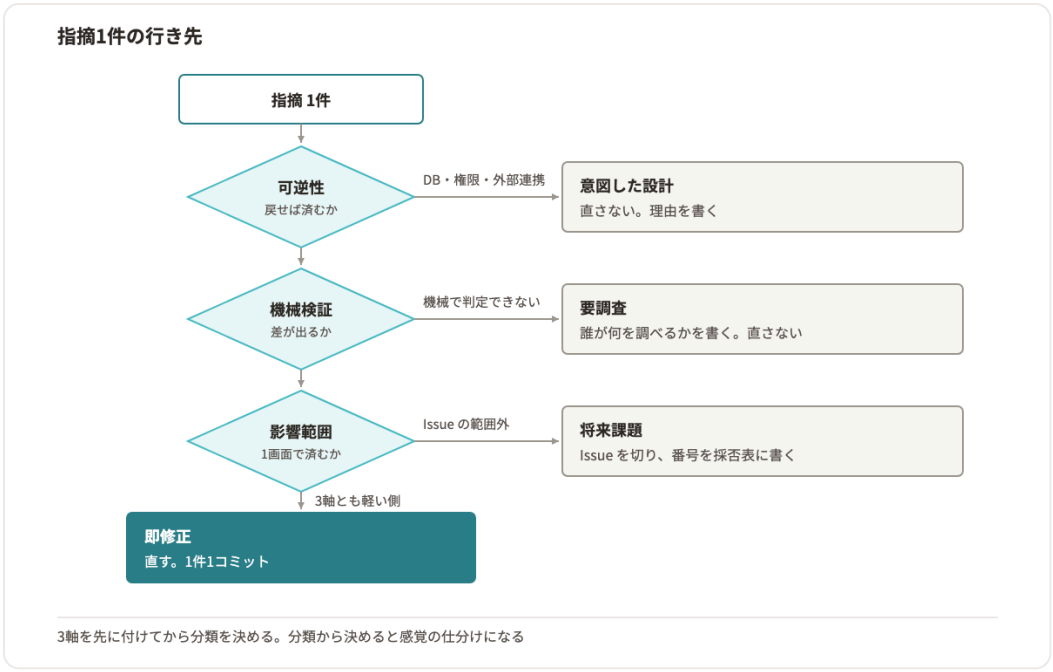
基準が空のままレビューさせる意味と、JSON で受け取る形

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。正しいかどうかが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



> 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。出口の箱の下の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。

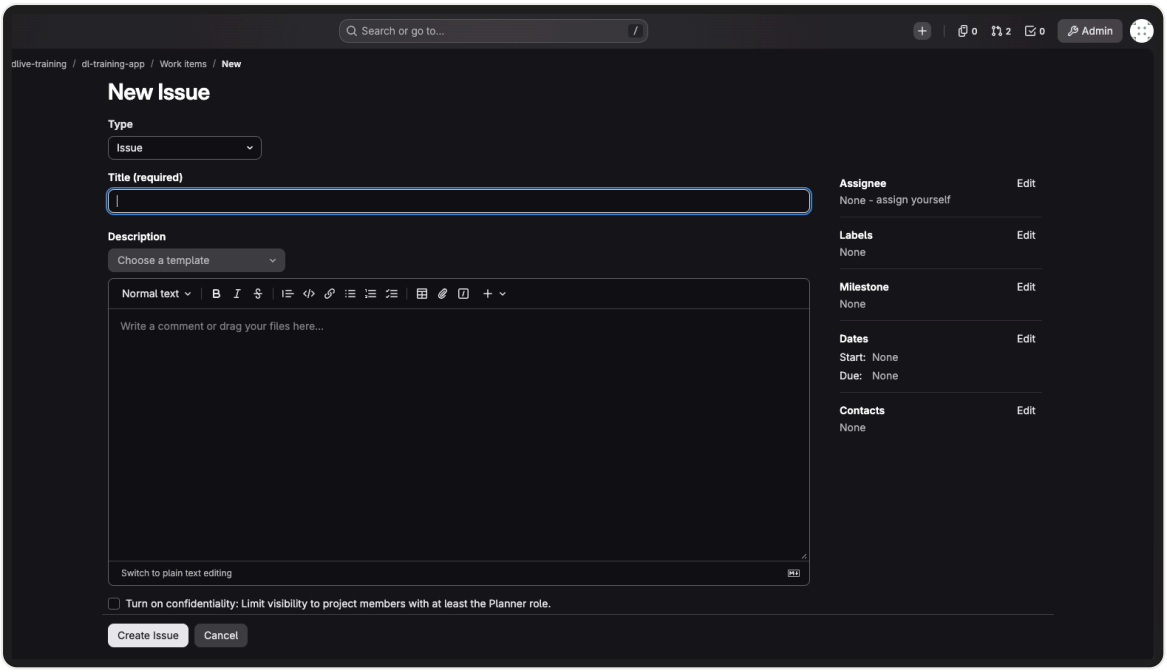
3. 3軸から分類を決めます。分類から先に決めると、感覚で仕分けることになります。

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。



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

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 `php -l` で `match` 式や `enum` のような 8.x の構文を、2段目の `php ci/undefined_functions.php` で `str_contains()` 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、`findings.json` 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`	`findings.json` に P0 か P1 があれば `exit 1`	今日は `allow_failure` が付いている。赤くても止まらない

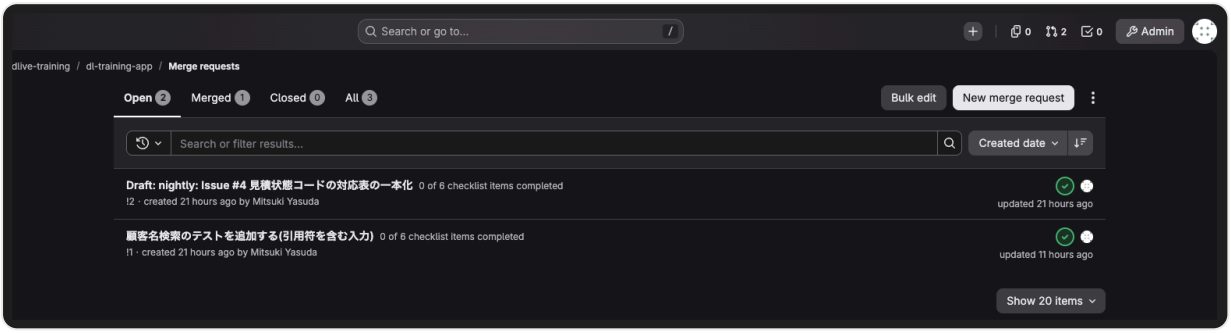
`ai_review` が動くと、MR の **Overview** タブの下のほうに「AIレビュー(モデル: … / 深さ: … / 差分 … 行)」で始まるコメントが1本付きます。見出し行の件数を表の4行目に写してください。所要時間はジョブの画面右側の **Duration** を使います。

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。

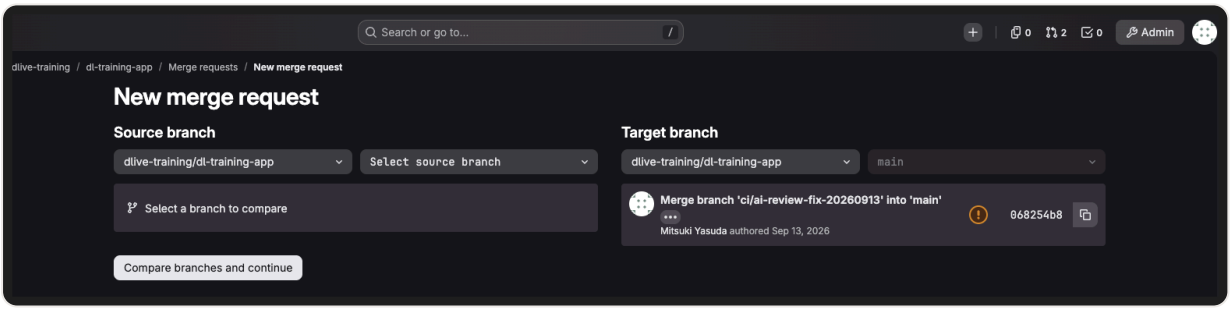
`lint-phpver` が落ちた方は、直す前に記録してください。記録先は MR の説明欄です。落ちた構文は Day2 の最初のセッションでそのまま教材になります。

記録すること	書き方

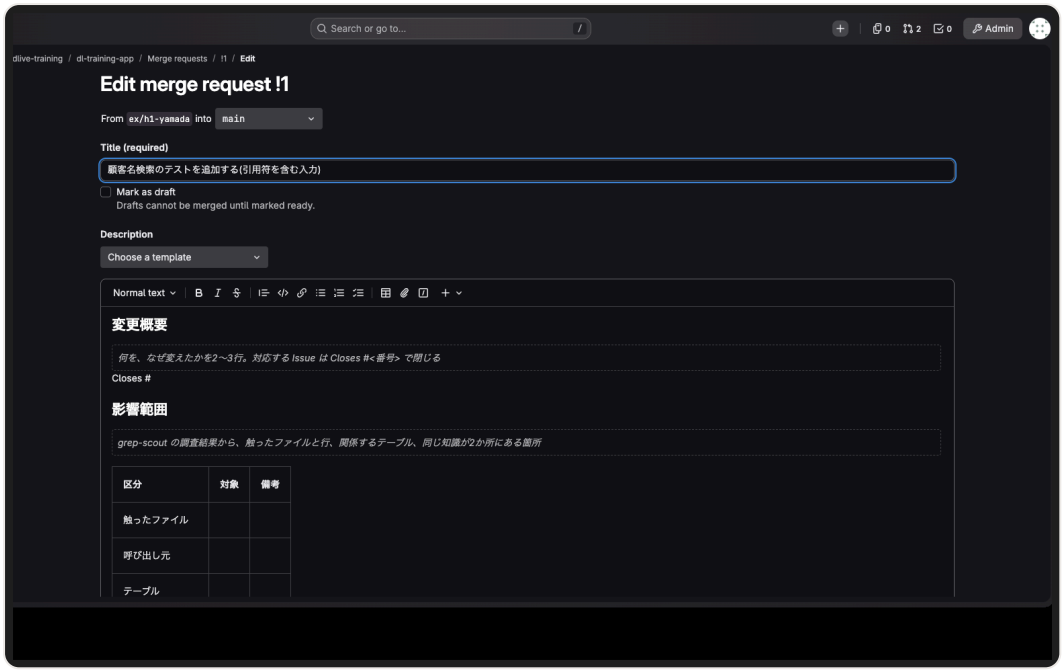
落ちた行	ファイル名と行番号
出たエラー文	ログに出たエラーの1行目を、そのまま貼る
混入した構文	`match` のように、どの構文が入ったか
指示のどこに書いてあったか	`CLAUDE.md` に書いてあったのに混入したのか、書いていなかったのか



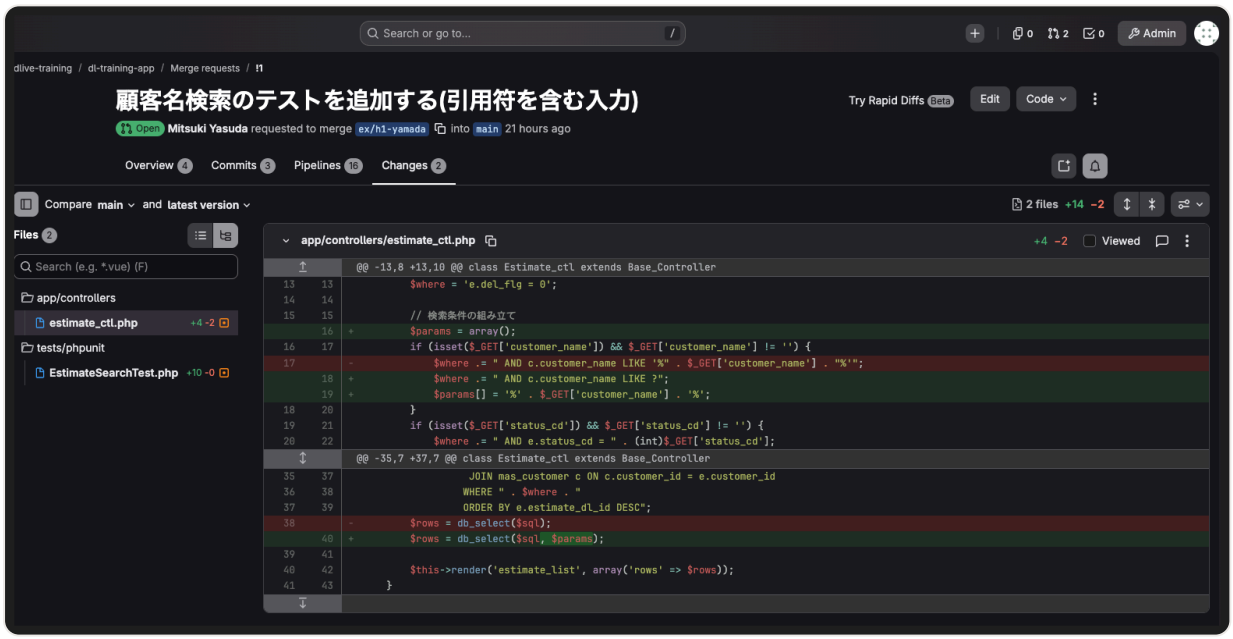
手順2で開く **Code** > **Merge requests** の一覧です。右上の **New merge request** から作ります。作成後は、**Open** タブにソースブランチが自分のもので作成者が自分の名前になっている行が並び、行の左下に `!1` のような MR の番号が出ます。この画面は講師環境のものです



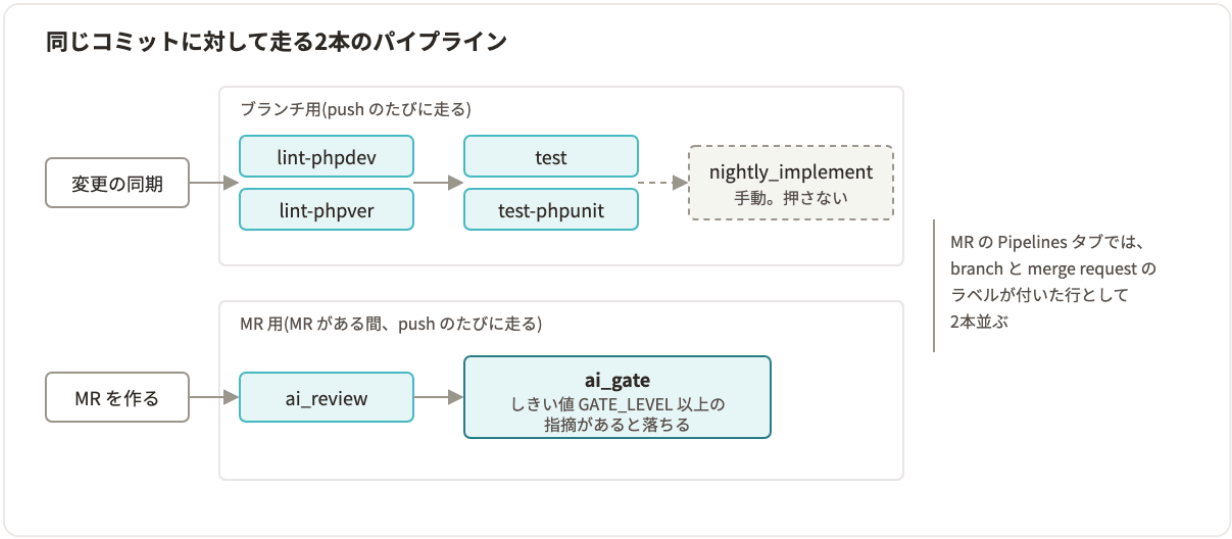
手順3の画面です。左の **Source branch** の **Select source branch** で `worktree-issue-1` を選び、右の **Target branch** が `main` になっていることを見てから、左下の **Compare branches and continue** を押します



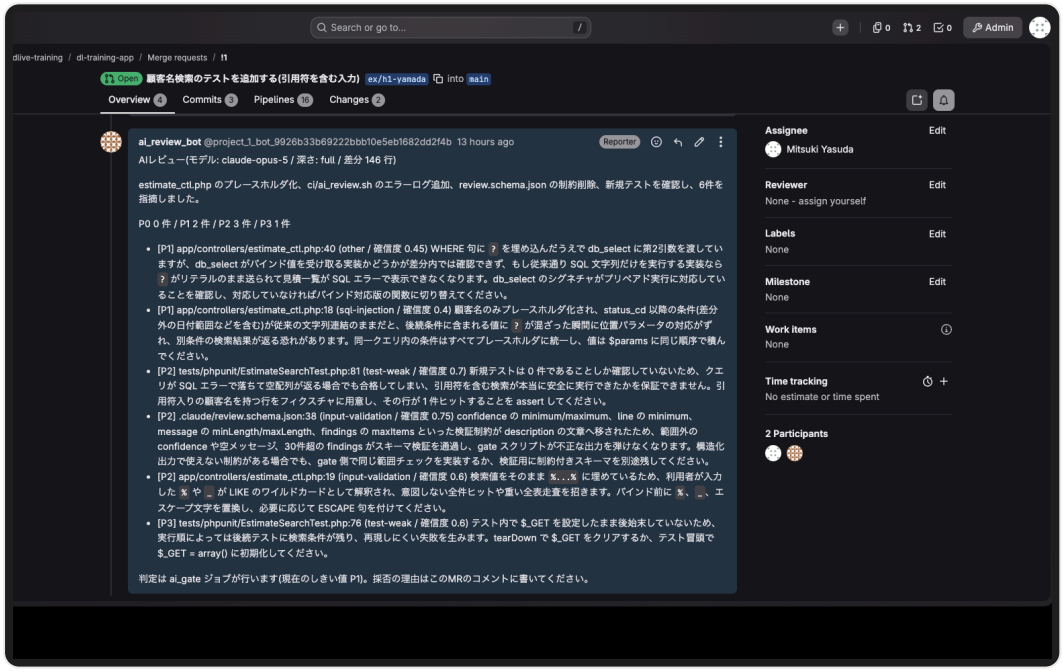
手順4から手順6で使う画面です。上の **Title (required)** を書き換えます。**Description** には「変更概要」「影響範囲」から始まる節の見出しと表が入っています。斜体の1行が各節に何を書くかの案内です



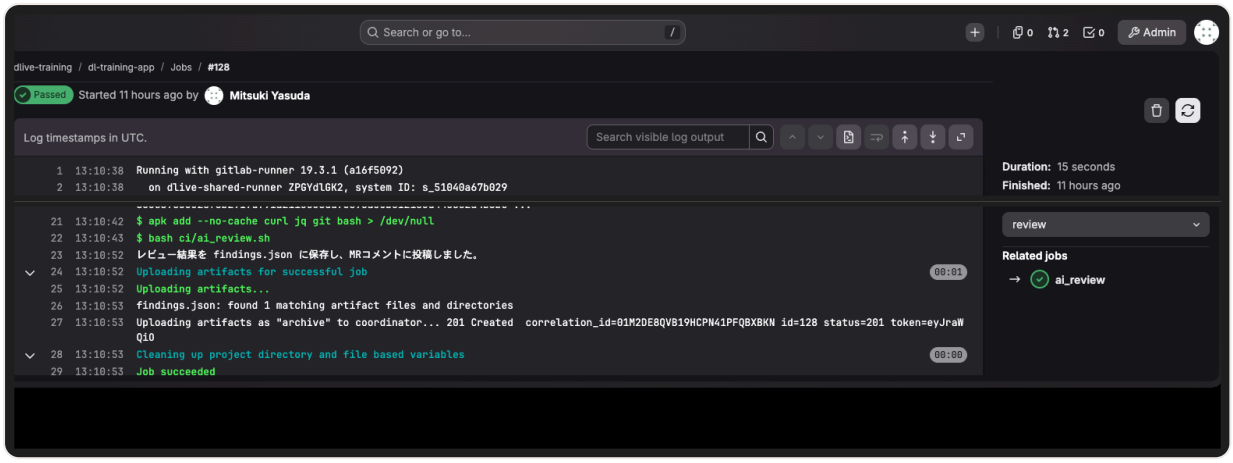
手順8で開く **Changes** タブです。左の列に変更したファイルの一覧、右上に `2 files +14 -2` のようにファイル数と足した行数、消した行数が出ます。差分の左の2列の数字が変更前と変更後の行番号で、赤い行が消した行、緑の行が足した行です



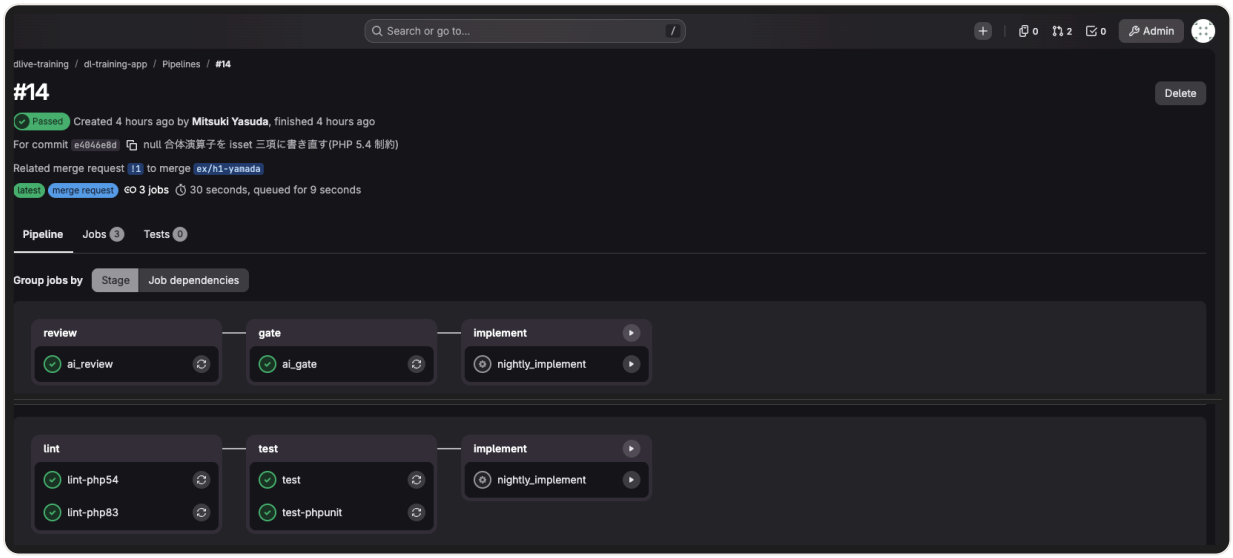
手順7で見る2本の関係です。上のブランチ用は同期のたびに走り、`lint-phpdev` と `lint-phpver` のあとに `test` と `test-phpunit` が動きます。点線の先の `nightly_implement` は手動なので押しません。下の MR 用は MR がある間の push のたびに走り、`ai_review` のあとに `ai_gate` が `GATE_LEVEL` 以上の指摘があると落ちます



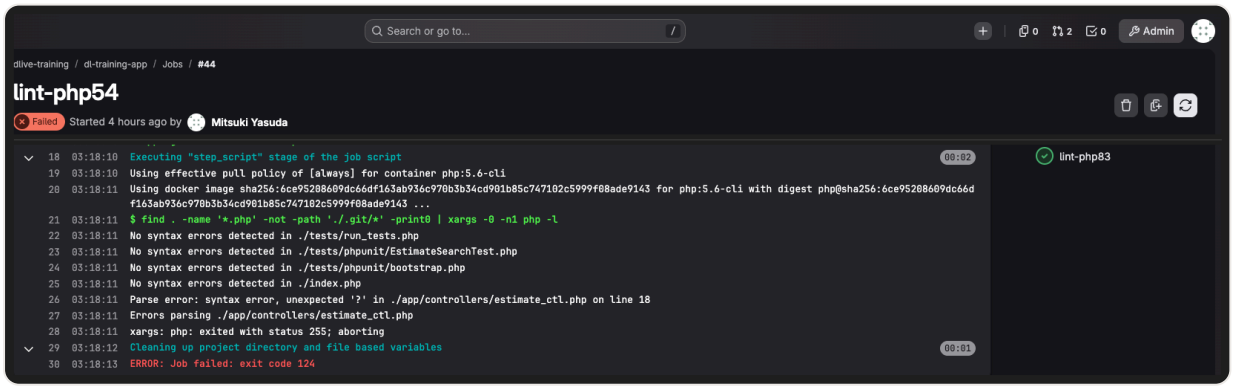
Overview タブで読む `ai_review_bot` のコメントです。1行目に「AIレビュー(モデル: … / 深さ: … / 差分 … 行)」、次に要約、その下に `P0 0 件 / P1 2 件 / P2 3 件 / P3 1 件` のような件数の行が出るので、これを表の4行目に写します。この画面は差分 146 行で `claude-opus-5` が6件指摘した例です。差分が 43 行だと sonnet に切り替わり、指摘の件数は減ります



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で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

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` だけを押すと、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

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	`.claude/agents/review-other.md`
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	`ci/ai_review.sh` と `.gitlab-ci.yml` の review ステージ

1. 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。

2. 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。

3. 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。

4. 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
`.claude/agents/review-other.md`	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、`model:` 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
`ci/ai_review.sh`	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	`.gitlab/merge_request_templates/`	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

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。
