Givery, Inc.

ハンズオンガイド

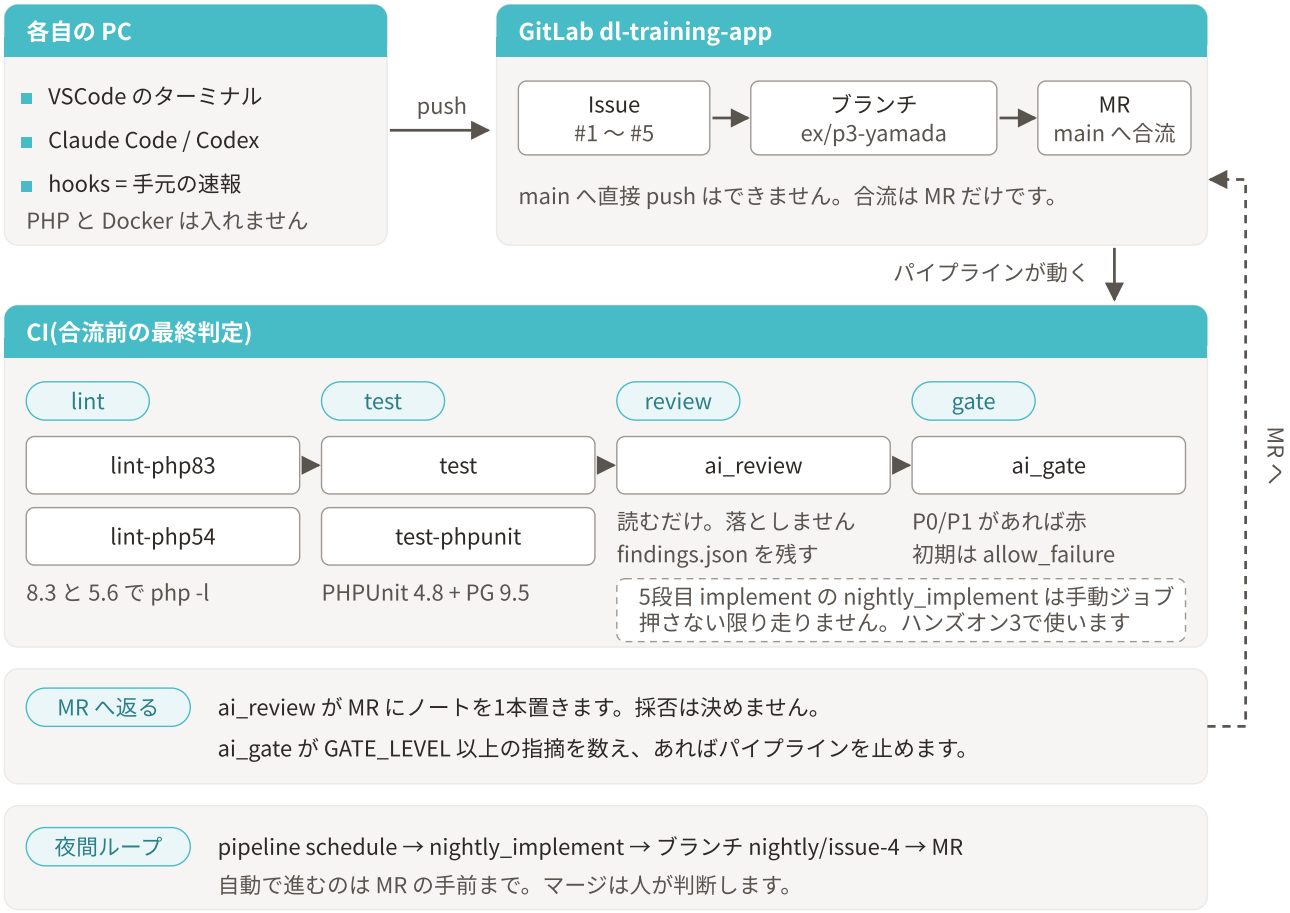
AIカスタム研修 ／ 2026年9月29日（火）・10月6日（火）

2日間の演習はプチ演習6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、このページはその進め方を1操作ずつ説明します。当日はこのページと GitLab、VSCode を並べて進めます。

目次

演習環境の全体像

2日間の演習は、手元の VSCode、GitLab の演習リポジトリ、GitLab CI の3か所を行き来します。手元のフックは書いた瞬間に出ます。CI は全員の変更に同じ基準を当てます。同じ検査を2か所に置いているのはこのためです。

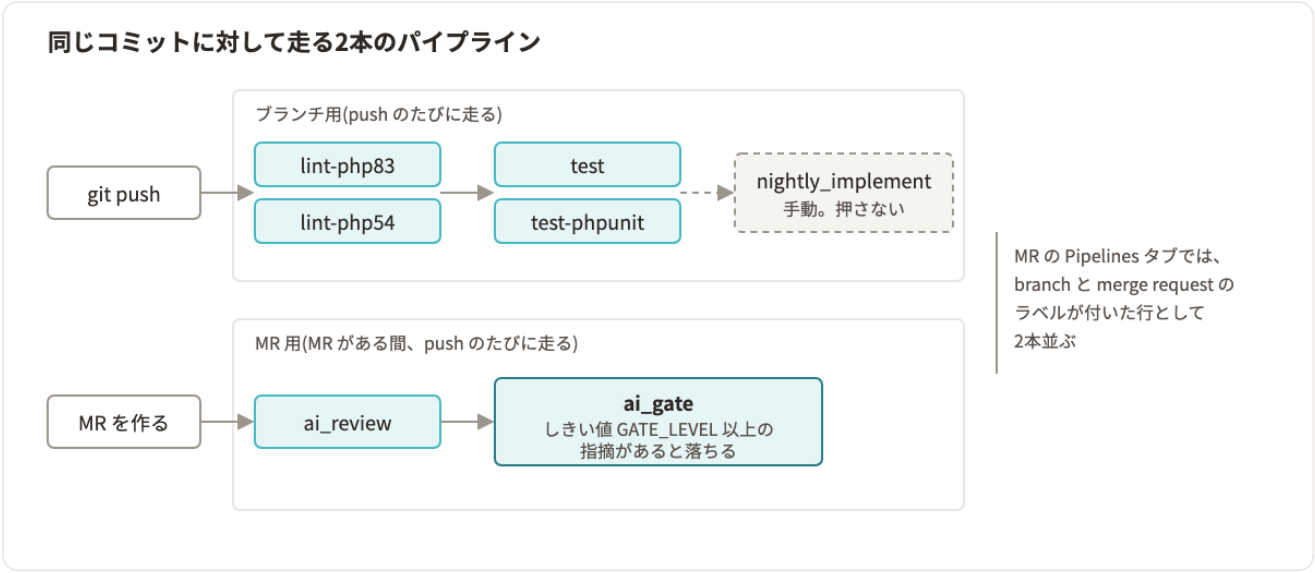


上段が手元と GitLab、中段が MR を出したときに動く CI のジョブです。中段右下の破線が5段目の implement で、手動でしか動きません。右端の点線は、CI の結果が MR へ戻ってくる経路を示します。最下段の夜間ループはハンズオン3で扱います。

> **Tips：**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。

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。push のたびに走るブランチ用(lint・test)と、MR がある間だけ走る MR 用(review・gate)の2本に分かれ、MR の Pipelines タブに並ぶ。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



2点目の補足です。同じコミットに対して、push で走るブランチ用と 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。上段右端の nightly_implement は手動なので、ハンズオン3まで押しません。

各演習の読み方

プチ演習6本とハンズオン3本は、同じ並びで書いてあります。最初の早見表で「何を触り、どこを見て、どうなれば終わりか」を先に押さえてから、操作に入ってください。

節	書いてあること
この演習で触るもの	触るファイル、操作、見る場所、達成の状態、自社での使いどころ、影響が及ぶ範囲の6行。迷ったらここに戻ります
目的	何を機械に移し、何を人に残すか
自分で考える	AI に触る前に紙に書く問い。ここを飛ばすと、返ってきたものを評価する基準が手元に残りません
操作	Step ごとの手順と、「こう見えていれば正しい」出力の例。Codex の方は各 Step の末尾の折りたたみを開きます
生成物の名前と場所	手元に残るファイルと、自社ハーネスへ写すときの置き場所
OK 基準	講師に聞かずに自分で判定できる形のチェックリスト
追加と考察 / 発展課題	時間が余った方が進めるものと、研修後に手を付ける範囲

操作の文には、どこで打つかを書いています。書き方と画面の対応は次のとおりです。VSCode のターミナルは、事前セットアップの Step 4 で開いたものです。

操作の書き方	打つ場所	見分け方
「PowerShell で」「ターミナルで」	VSCode のターミナル。Claude Code を起動していない方のタブです	行の左端が `PS C:\Users\...\dl-training-app>` (Mac は `%` か `$`)で終わっている
「Claude Code に」「次を送る」	`claude` と打って起動したターミナルのタブ。文章をそのまま貼り付けて Enter を押します	画面下に入力欄があり、左端に `>` が出ている
「`/tasks` 」「`/permissions` 」のようにスラッシュで始まるもの	Claude Code の入力欄。ターミナルに打つと「コマンドが見つかりません」になります	同上

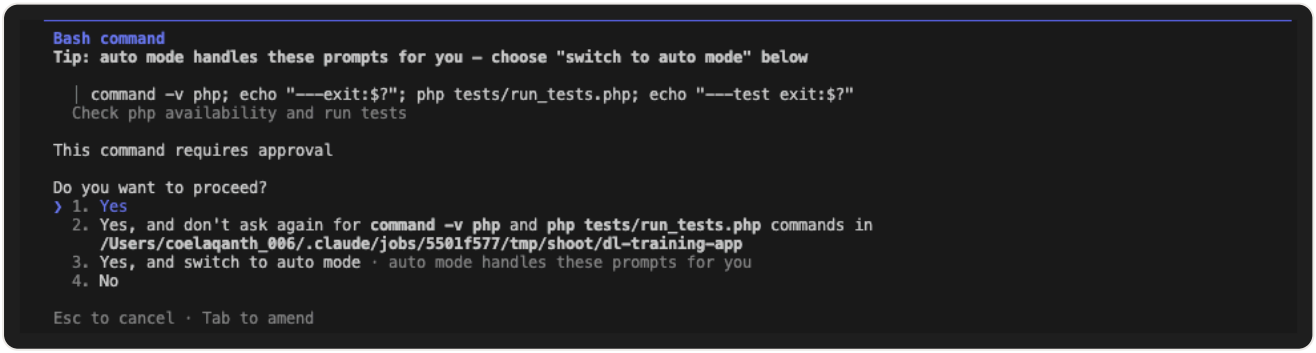
操作の書き方	打つ場所	見分け方
「VSCode で開く」	左側のエクスプローラーでファイルをクリックします。エクスプローラーが見えないときは **表示** > **エクスプローラー** です	エディタ領域にファイルが開く
「ブラウザで」	GitLab の画面。事前セットアップの Step 1 でログインした `https://gitlab-09291006aidev.give-app.net` です	アドレス欄が上の URL で始まる

> **Tips：**演習中はターミナルを2本開いておいてください。1本目で `claude` を起動し、2本目で `git` と PowerShell のコマンドを打ちます。Claude Code が動いているタブに `git status` を打つと、コマンドではなく AI への依頼として送られます。2本目はターミナル右上の `+` で開きます。

ターミナルを2本開いた状態です。右側の一覧に、コマンドを打つ側の `zsh` と、Claude Code が動いている側の `DLive` が並びます。2本目は一覧の上にある `+` で増やします。この画面は Mac の VSCode なので、Windows では `zsh` の位置に `powershell` と表示されます。

> **Mac の方：**手順の PowerShell はターミナル(zsh)に読み替えてください。 `git` と `claude` のコマンドは同じです。 `.ps1` で終わるスクリプトは同じ名前の `.sh` に、パスの `\` は `/` に置き換えます。各演習の「Codex の場合」にある bash 版のコマンドが、そのまま Mac 用の手順になります。

初回の確認ダイアログについて　Claude Code 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、入力欄の上に「実行してよいか」の選択肢が出ます。演習では **Yes** を選んでください。 **Yes, and don't ask again** を選ぶと、以後そのコマンドは確認なしで通るため、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。



Bash を実行する直前に出る確認です。選択肢は **1. Yes** 、 **2. Yes, and don't ask again for ... commands in ...** 、 **3. Yes, and switch to auto mode** 、 **4. No** の4つで、演習中は 1 を選びます。2 を選ぶと同じコマンドは以後確認なしで通り、3 を選ぶと確認そのものが出なくなるため、どちらもプチ演習2 と3 の結果が変わります。

> 入力欄で Shift+Tab を押すとモードが切り替わり、画面下に `manual mode on` または `auto mode on` と出ます。auto mode では上の確認が出ないので、演習中は `manual mode on` になっていることを確かめてください。上に出ている Transcript saving の警告行は講師環境のもので、受講者の画面には出ません。

共通の進め方

ハンズオン3本は、この6手順の上に乗ります。プチ演習6本で使うのは STEP 1 と STEP 2 だけです。手元の設定を足して、1つの変更意図ごとにコミットするところまでで終わります。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。

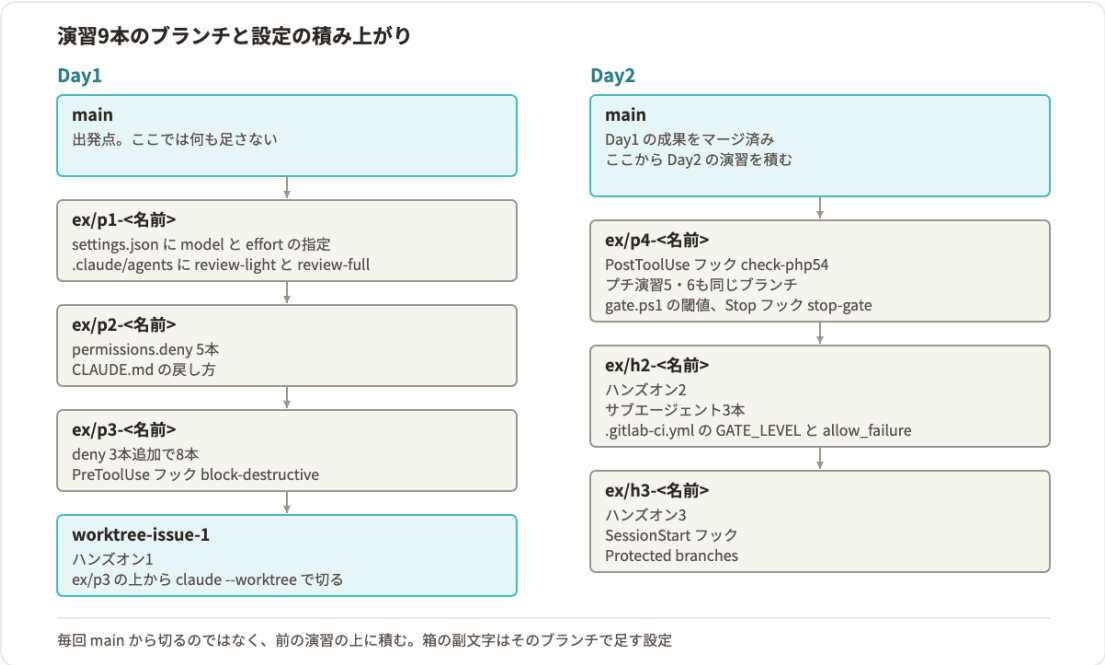
> **先に決めておくこと**　このガイドの例では GitLab のユーザー名を `yamada` としています。ご自身の演習では、社用メールアドレスの「@ より前」をそのまま使ってください。ブランチ名に使うのは同じ文字列です。

STEP 1　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 `p6` 、ハンズオンが `h2` と `h3` です。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。

ハンズオン1だけは例外です。`claude --worktree issue-1` で起動するので、ブランチは Claude Code が `worktree-issue-1` という名前で作ります。ご自身で `ex/h1-...` を切る必要はありません。

Day1 のプチ演習1〜3 も、下の手順1(main に戻る)を飛ばします。3本の演習で `.claude/settings.json` に足す設定は積み上がっていくので、main に戻ると前の演習の設定が消えます。プチ演習1 だけ main から `ex/p1-yamada` を切り、プチ演習2 と3 は、いま乗っているブランチからそのまま `git switch -c ex/p2-yamada` 、`git switch -c ex/p3-yamada` と切ってください。午後のハンズオン1 は、この `ex/p3-yamada` の上から始めます。



> 2日間のブランチの積み上がりです。Day1 は main から `ex/p1` を切ったあと、`ex/p2` と `ex/p3` はいま乗っているブランチから切り、ハンズオン1 の `worktree-issue-1` は `ex/p3` の上に乗ります。毎回 main から切るのではなく、前の演習の上に積みます。箱の副文字は、そのブランチで足す設定です。

1. 演習を始める前に main を最新にします。前の演習のブランチに乗ったまま次を始めると、2つの演習の変更が1つの MR に混ざります。

2. ブランチを切ります。 `-c` は新規作成です。

3. 今どこにいるかを確かめます。ここを飛ばすと、main のまま編集して push で弾かれます。

```
git switch main
git pull
git switch -c ex/p3-yamada
git branch --show-current
```

```
Switched to branch 'main'
Already up to date.
Switched to a new branch 'ex/p3-yamada'
ex/p3-yamada
```

こう見えていれば正しいです。最後の行が `main` のままなら、3行目が実行されていません。

`git pull` でユーザー名とパスワードを聞かれたら、事前セットアップの Step 1 で変更したパスワードを入れます。`error: Your local changes would be overwritten` と出たときは、前の演習の変更がコミットされていません。STEP 2 の形でコミットしてから `git switch` をやり直してください。

> **Codex の場合：**ブランチの操作は Codex でも同じコマンドです。ご自身の手で打ってください。 `sandbox_mode = "workspace-write"` では `.git` が読み取り専用になるため、Codex にコミットやブランチ作成を任せられません。演習では「コミットは人が打つ」で進めます。

> **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 --worktree` が作ったもの、 `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。

この Step が終わった状態

- ☐ `git branch --show-current` が `ex/` で始まる名前を返す(ハンズオン1 だけは `worktree-issue-1`)
- ☐ `git status` に、前の演習の変更が残っていない

STEP 2 コミットの粒度

[2min]
1つの変更意図につき1コミットです。次の作業へ進む前にコミットしてください。粒度を細かく保つと、戻せる単位がコミットの単位になります。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。

1. 変更したファイルだけを指定して追加します。 `git add .` は、フックが作った作業ファイルまで巻き込みます。
2. コミットメッセージは「何をどうしたか」を日本語の1行で書きます。ファイル名を並べたメッセージは、 `git log` を読むときに何も足しません。
3. 区切りごとに `git log --oneline` で並びを見ます。

```
git add app/controllers/estimate_ctl.php
git commit -m "見積一覧の顧客名検索をプレースホルダに置き換える"
git log --oneline -3
```

```
4f1c2ab 見積一覧の顧客名検索をプレースホルダに置き換える
9a3e0d7 顧客名検索のテストを追加する
1b77e5c Initial import of dl-training-app
```

こう見えていれば正しいです。1行が1つの変更意図に対応していれば、粒度は足りています。

タスクを始める前に必ずコミットしてください。 `/rewind` で戻せるのは Claude Code が Edit と Write で変えたファイルだけです。Bash で動かしたファイル、サブエージェントの変更、手で直した分は戻りません。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。

STEP 3 MR の出し方

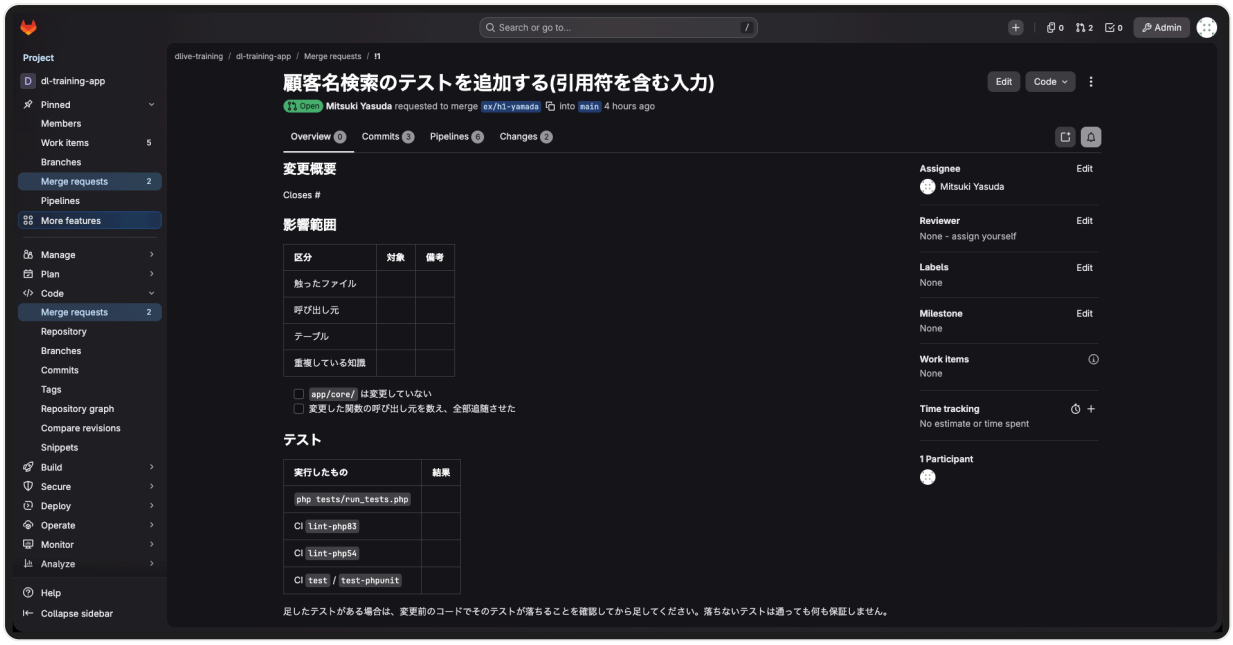
[4min]
push のオプションで MR まで作ります。ブラウザで新規作成の画面を開く必要はありません。`main` はブランチ保護がかかっており、直接 push すると `pre-receive hook declined` で弾かれます。

1. push とあわせて MR を作ります。`-o` は push オプションです。
2. 出力に表示された MR の URL を開きます。
3. 説明欄にテンプレートが入っています。**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. 2回目以降の push は `git push` だけです。オプションを付け直すと MR が二重に作られます。

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

```
remote:
remote: View merge request for ex/p3-yamada:
remote:   https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app/-/merge_requests/12
remote:
To https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app.git
 * [new branch]      HEAD -> ex/p3-yamada
```

こう見えていれば正しいです。`View merge request for` の行が出ていなければ MR は作られていないので、GitLab の画面から手で作ってください。



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。

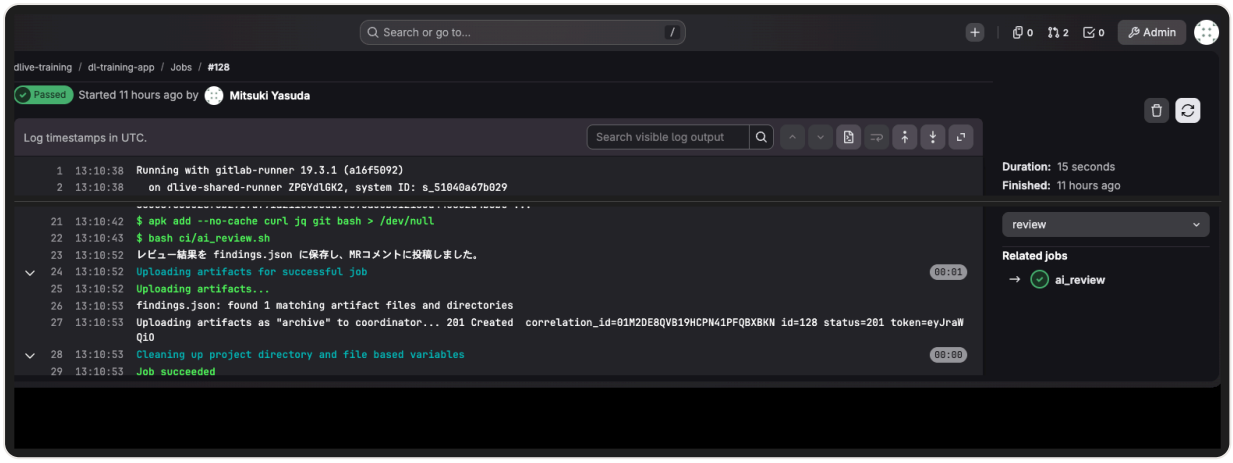
> **Tips：** `Closes #1` を説明欄に残すと、マージしたときに Issue が閉じます。演習中は閉じないので、書き換えずにそのまま残してください。

STEP 4　パイプラインの見方

[4min]
MR の **Pipelines** タブには、パイプラインが2本ずつ並びます。1本目は push のたびに走るブランチ用で、`lint` と `test` の2段4ジョブです。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。自動で走るジョブは合わせて6本です。一覧の **merge request** と **branch** のラベルで見分けます。

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、押さない限り走りません。ハンズオン3で使うので、今日は押しません。

1. MR の **Pipelines** タブを開き、いちばん上の行のステータスを見ます。
2. 赤いジョブの名前をクリックするとログが開きます。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。
3. `ai_gate` が止めたときは、ログに止めた指摘がそのまま並びます。
4. `ai_review` のジョブ画面の右側にある **Job artifacts** から `findings.json` を取得できます。

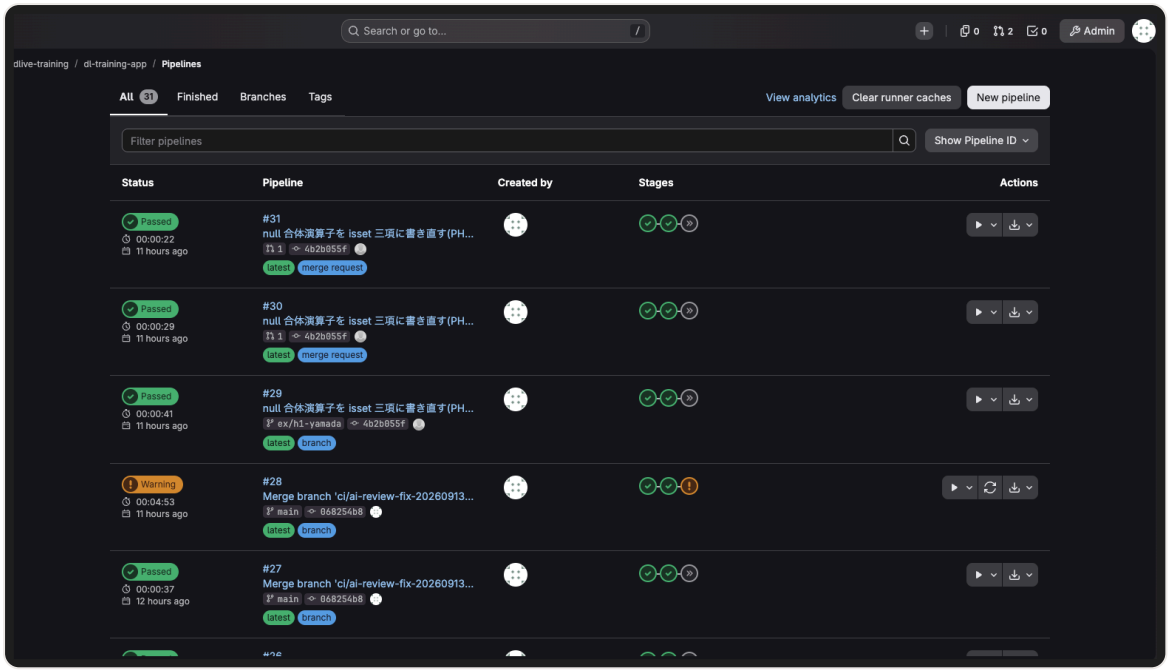


手順2 でジョブ名をクリックすると開く画面です。緑の `$` で始まる行が実行した命令で、この例では `bash ci/ai_review.sh` が走り、最後に `Job succeeded` で終わっています。右側の **Duration** が計測表の所要に写す値です。**Job artifacts** の **Browse** は同じ右側の欄にあり、この画像には写っていません。

```
しきい値 P1 以上、確信度 0.0 以上の指摘を数えます。
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。
```

これが `ai_gate` が落ちたときのログです。どの指摘で止まったかが分かれば正しく読めています。

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 push した回のブランチ用パイプラインです。



MR のタブではなく、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(`ex/h1-yamada` のような表記)で探します。左端が状態、`#` の数字がパイプライン番号、その下のラベルが **latest** と **branch** または **merge request**、右の **Stages** の丸が段ごとの結果です。

> **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。

> **Tips：** `test-phpunit` だけ他より2分ほど長くかかります。PHP 5.6 のイメージに PostgreSQL の拡張を入れてから PHPUnit 4.8.36 を取得しているためです。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

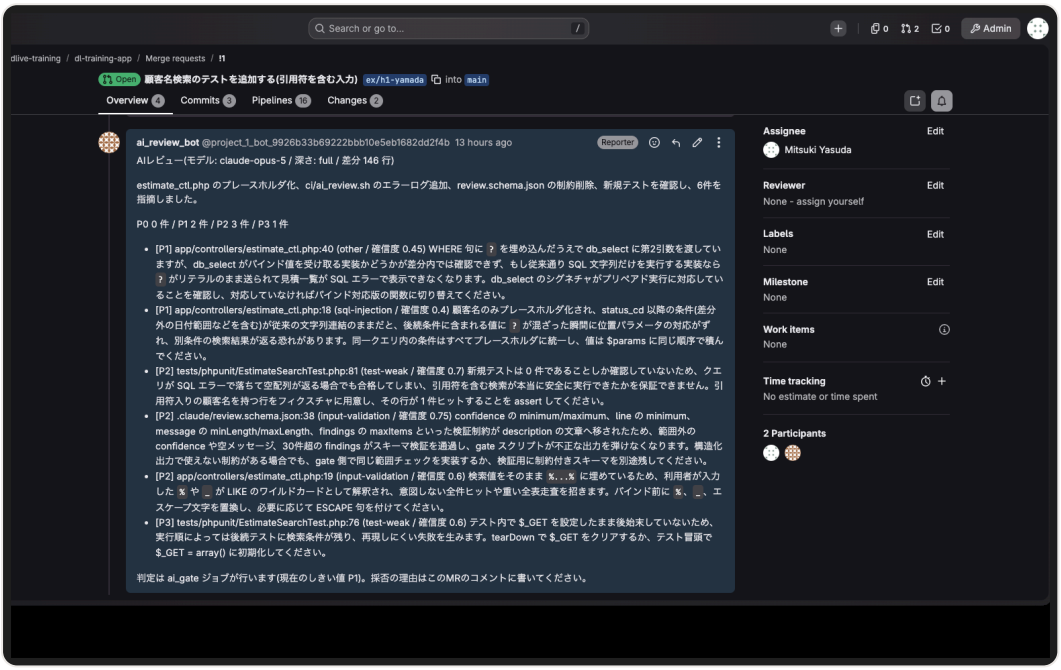
[4min]
`ai_review` は MR にコメントを1本置きます。1件の指摘は `severity`、`confidence`、`file`、`line`、`message`、`category` の6つを持ちます。ノートは列挙するだけで、直すかどうかは決めません。

```
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。

こう見えていれば正しいです。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。



> 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。この例は `ex/h1-yamada` の MR で、上のログの例とは件数も内容も違います。

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。

> **Codex の場合：**手元で同じ形の出力を得るには、PowerShell では `codex exec -s read-only --output-schema .claude/review.schema.json -o findings.json "$(Get-Content REVIEW.md -Raw) 対象: 差分は main との比較"` を使います。bash の `$(cat ...)` は PowerShell では展開されず、文字列のまま渡ります。Claude Code 側は `git diff main | claude -p --json-schema .claude/review.schema.json --output-format json "REVIEW.md の基準でレビューし、findings を返す"` です。どちらも同じ `findings` の形で返るので、`ci/gate.ps1` がそのまま読めます。

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。

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。

STEP 6 マージの扱い

[2min]
自分の MR は自分でマージしません。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。

1. CI が緑になったら、MR の説明欄の **人が判断した採否** の表を埋めます。直したものと直さなかったものの両方を残します。
2. 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。
3. **Merge** は押しません。演習の終わりに、講師が全員の MR を並べて指摘件数を数えます。

> **Tips：**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。

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。

```
| 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、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。

> **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。研修後に端末からハーネスを削除するまでが運営手順です。

計測 計測表

指摘数、重大指摘、所要時間の3つを演習ごとに取ります。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

列	取り方	取るタイミング
重大指摘	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
採用した件数	「即修正」に仕分けた件数	採否を決めた直後
見送った件数	「将来課題」「意図した設計」に仕分けた件数	同上

```
| 演習 | 読ませ方 | 指摘数 | うち P0/P1 | 所要 | 採用 | 見送り |
|---|---|---|---|---|---|---|
| ハンズオン1 | 実装したセッションでそのまま | 4 | 1 | 2分 | 1 | 3 |
| ハンズオン1 | 履歴を切った別セッション(review-other) | 6 | 2 | 3分 | 2 | 4 |
| ハンズオン1 | 別ツール(Codex または claude -p) | 3 | 2 | 6分 | 1 | 2 |
| ハンズオン1 | CI の ai_review | 6 | 1 | 1分 | 1 | 5 |
```

この形で埋まっていれば正しいです。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)
`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。

Claude Code と Codex の読み替え

演習の本文は Claude Code の書き方で進みます。Codex だけをお使いの方は、この表で置き換えてください。どちらか一方でも全9本の演習を完走できます。

目的	Claude Code	Codex
起動する	`claude`	`codex`
1回だけ非対話で走らせる	`claude -p "指示"`	`codex exec "指示"`
読むだけに閉じる	`--allowedTools "Read,Grep,Glob"`	`-s read-only`
出力の形を固定する	`--json-schema .claude/review.schema.json`	`--output-schema .claude/review.schema.json`
結果をファイルに残す	`--output-format json` の標準出力をリダイレクト	`-o findings.json`
履歴を切って独立に読ませる	`/clear` のあと `review-other`、または `claude -p`（`ANTHROPIC_API_KEY` がある端末は `claude --bare -p`）	`codex review --base main "P0/P1 のみ。3件まで。コード提案は不要"`
プロジェクトの指示を置く	`CLAUDE.md`	`AGENTS.md`
モデルを決める	`.claude/settings.json` の `model`	`~/.codex/config.toml` の `model` と `review_model`
考える深さを決める	`effortLevel`（low / medium / high / xhigh / max）	`model_reasoning_effort`（minimal / low / medium / high / xhigh）
役割ごとに設定を分ける	`.claude/agents/<名前>.md` の frontmatter	`[profiles.<名前>]` と `codex --profile <名前>`
フックを登録する	`.claude/settings.json` の `hooks`	`~/.codex/hooks.json` または `.codex/hooks.json`
巻き戻す	`/rewind`（入力欄が空の状態で Esc を2回）。Edit と Write の変更だけ	`/undo` は撤去済み。git で戻します

目的	Claude Code	Codex
いまの権限を確認する	`/permissions`	`/status` と `/permissions`

effort と model_reasoning_effort は別物です。 段の数も名前も一致しません。Claude Code は low / medium / high / xhigh / max の5段、Codex は minimal / low / medium / high / xhigh の5段で、両端が違います。同じ `high` だから同じ重さ、とは読まないでください。

> **Tips：**Codex のプロファイルの置き方は2通りが出回っています。一次情報で確認できたのは `$CODEX_HOME/<名前>.config.toml` に分けて置く形です。`config.toml` の中に `[profiles.<名前>]` を書く形は、手元の `codex --profile` が通ることを1回確かめてからお使いください(要確認)。

用語集

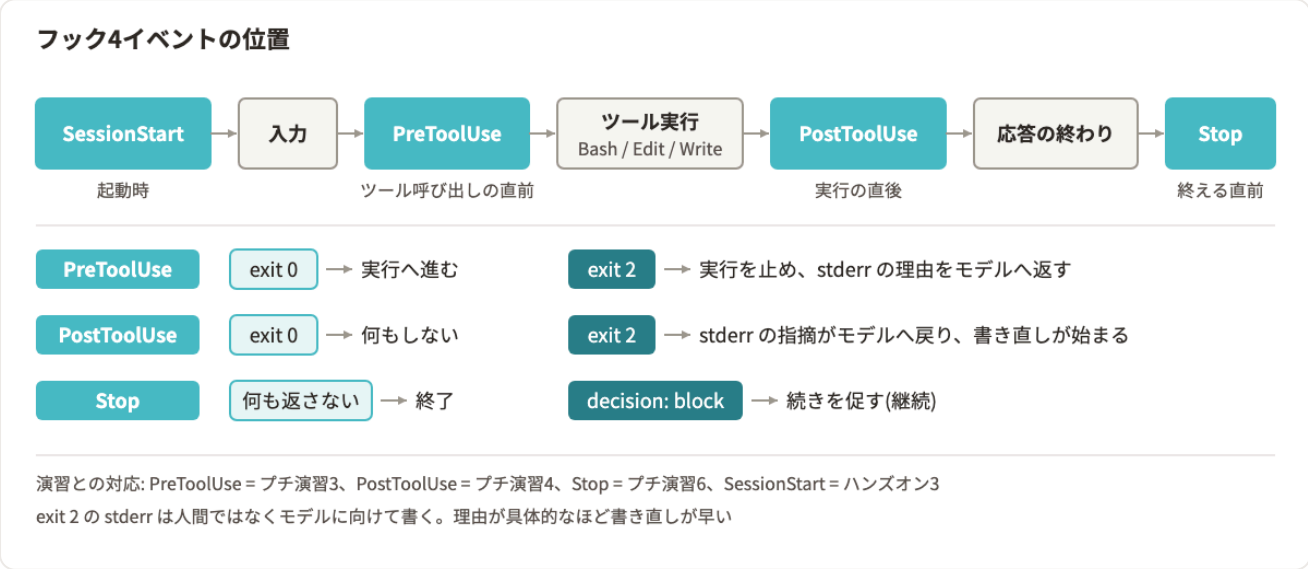
2日間の資料と演習で、説明なしに出てくる語をまとめています。手が止まったらここに戻ってください。

語	意味	出てくる場所
`CLAUDE.md`	リポジトリ直下に置く、Claude Code への指示書です。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。Codex 側の同じ役割は `AGENTS.md` です。	プチ演習2、プチ演習4
`settings.json`	`.claude/settings.json` 。Claude Code の設定ファイルで、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 `/exit` で終了して `claude` で起動し直します。	プチ演習1 から 3
フック(hook)	Claude Code が決まった場面(ツールの実行前、実行後、応答の終了時など)で自動的に呼び出すスクリプトです。 `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。	プチ演習3、プチ演習4、プチ演習6
`exit 2` / `$LASTEXITCODE`	スクリプトが終わるときに返す数字が終了コードです。フックでは `0` が「通す」、 `2` が「止める」で、 `2` のときは標準エラーに書いた文章がそのまま AI に返ります。直前に動かしたスクリプトの終了コードは、PowerShell では `$LASTEXITCODE` 、bash では `$?` と打つと表示されます。	プチ演習3、プチ演習4
サブエージェント	Claude Code が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`	Claude Code の巻き戻し機能です。入力欄が空の状態で Esc を2回押すか、 `/rewind` と打つと、過去の地点(チェックポイント)の一覧が開きます。戻せるのは Claude Code が Edit と Write で変えたファイルだけです。	プチ演習2

語	意味	出てくる場所
パイプライン / ジョブ / ランナー	GitLab CI の用語です。push 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-php54` 、 `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。default / acceptEdits / auto / dontAsk / bypassPermissions / plan の6つです。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。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
`PreToolUse`	ツールを実行する直前に走るフック。権限モードに関係なく先に走る唯一の強制点で、 `exit 2` で呼び出しを止めます。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。 `Edit` と `Write` の後に `check-php54` を走らせ、5.4 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` を返し続けると無限ループします。	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 --worktree <名前>` で `.claude/worktrees/<名前>/` に `worktree-<名前>` ブランチを作って起動します。本体の作業ツリーを触らせずに実装させたいときに使います。	ハンズオン1
`-p`	非対話実行。 `claude -p "指示"` 。既定では許可待ちのまま何もせず終わるため、 `--permission-mode` 、 `--allowedTools` 、 `--max-turns` を添えて初めて無人で動きます。	ハンズオン3

語	意味	出てくる場所
`--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。	ハンズオン1、プチ演習5
`--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。Codex 側は `--output-schema` です。	プチ演習5
`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

語	意味	出てくる場所
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
`PHPCompatibility`	phpcs の規格。`--standard=PHPCompatibility --runtime-set testVersion 5.4` で 5.4 非対応の構文を検出します。`php -l` は文法として正しければ通してしまうため、`::class` や可変長引数はこちらでないと捕まりません。	プチ演習4



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。Stop だけは終了コードではなく `decision: block` で続きを促します。

> **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

所要 [20min]

目的

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。レビュアーを2本に分け、変更の大きさで当てる側を変えます。作るのは `.claude/agents/` の定義2本と、`.claude/settings.json` の3行です。どちらも自社ハーネスにそのまま置ける形にします。

この演習で触るもの

触るファイル	`.claude/agents/review-light.md` 、`.claude/agents/review-full.md` 、`.claude/settings.json`（`model` / `effortLevel` / `maxEffortLevel` の3行)。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
操作	frontmatter を読み、境界の1文を自分の言葉に書き換え、settings に3行足す。小さい差分と大きい差分を作り、それぞれを軽い側と重い側に渡す
見る場所	`/tasks` の行(モデル名と effort)、`/usage` の累計、返答末尾の `判定:` の1行
達成の状態	小さい差分で判定の1行だけが返り、大きい差分で `review-full の対象です` が返る。記録表4行が埋まっている
自社での使いどころ	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くする
影響が及ぶ範囲	`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る。定義の `model` は環境変数 `CLAUDE_CODE_SUBAGENT_MODEL` より優先される

開始前の確認　[1min]

5項目が埋まってから Step 1 に進んでください。版と環境変数で表示が変わるので、ここが揃っていないと他の受講者と画面が食い違います。

- □ VSCode のターミナルで `claude --version` を打ち、2.1.267 以降の数字が出る（`maxEffortLevel` はこの版で入りました)
- □ VSCode で `dl-training-app`(事前セットアップの Step 3 でデスクトップに置いたフォルダ)を開き、左のエクスプローラーに `app` / `db` / `tests` / `.claude` が見え

る。`.claude` が見えないときは、エクスプローラーの空白で右クリックし、隠しファイルの表示を確かめてください

- □ `.claude` を展開すると `agents` フォルダがあり、その中に `review-light.md` と `review-full.md` がある
- □ ターミナルで `$env:CLAUDE_CODE_EFFORT_LEVEL` と打って Enter を押し、何も表示されない(Mac は `echo $CLAUDE_CODE_EFFORT_LEVEL`)。値が入っていると frontmatter の effort より強く、この演習の表示がそちらに固定されます
- □ Codex だけをお使いの方は `codex --version` で数字が出ることと、`$HOME\.codex\config.toml`(Mac は `~/.codex/config.toml`)が存在すること

確認が済んだら、この演習のブランチを切ります。ターミナルで次を打ち、最後の行に `ex/p1-` で始まる名前が出れば準備完了です。`yamada` はご自身の GitLab のユーザー名に置き換えてください。

```
git switch main
git pull
git switch -c ex/p1-yamada
git branch --show-current
```

この演習では Claude Code を、VSCode のターミナルで `claude` と打って起動します。ターミナルは2本使います。1本目に `claude`、2本目に `git` のコマンドです(共通章「各演習の読み方」の表を参照)。

自分で考える　[3min]

手を動かす前に、次の1点だけ紙か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行添えてください。

2. いま自社のレビューで「毎回出るが毎回見送っている指摘」を1つ思い出し、それを軽い側の「書かないもの」に入れるかどうかを決めます。

> **Tips：**この2行は Step 1 で定義ファイルに書き写します。先に決めておかないと、配布物の数字をそのまま持ち帰ることになります。

操作

1. **[2min]** VSCode の左のエクスプローラーから `.claude/agents/review-light.md` と `.claude/agents/review-full.md` を開きます。frontmatter(先頭の `---` で囲まれた部分)に `model` / `effort` / `tools` の3行があり、軽い側が `sonnet` / `medium`、重い側が `opus` / `xhigh` になっていることを確認します。`review-light.md` の最終行「50行を超える変更、または DB・権限・外部連携に触る変更がきたら、レビューせずに「review-full の対象です」と1行で返します。」の前半を、自分で決めた境界に書き換えてください。「review-full の対象です」の文言は変えません。Step 4 の判定に使います。`review-`

full.md の「観点2 絶対制約」の箇条書きには、自社で絶対に守らせたい制約を1行足します。2ファイルとも Ctrl+S(Mac は Cmd+S)で保存します。

2 **[3min]** .claude/settings.json を開き、{ の直後、"permissions" の前に3行を足します。保存後の形はこうなります。

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

貼ったあと、末尾の行に余分なカンマが無いこと、VSCode の波線が出ていないことを確認してください。波線が出たまま保存すると settings.json はファイルごと無視されます。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。保存したら、Claude Code を起動しているターミナルで /exit と打って終了し、同じターミナルで claude と打って起動し直してください。settings.json はセッション開始時に読まれます。起動直後に /model と打つと、現在のモデルが sonnet になっていることを確認できます。

/model で開く画面です。チェックが付いている行が現在のモデル、その下の Medium effort の行が現在のエフォートで、矢印キーで変えられます。この画面は講師環境で Opus 5 が選ばれていますが、演習では sonnet の行にチェックが付いている状態にします。確認だけなら Esc で閉じます。

3 **[4min]** 小さい差分を1つ作り、軽い側と重い側の両方に当てます。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行を足して保存してください。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落としてから渡します。2本目のターミナルで次を打ちます。

```
git diff --output=small.diff
git diff --stat
```

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。

`--stat` の行に `util.php | 1 +` と出て、エクスプローラーに `small.diff` が現れれば差分は取れています。次に Claude Code の入力欄へ、下の2行をそのまま貼り付けて送ります。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。
```

サブエージェントが動いている間に Claude Code の入力欄で `/tasks` と打ち、表示された行のモデル名と effort を控えます。終わったら `/usage` と打って累計の数字も控えます。同じ差分を `@"review-full (agent)"` にも渡し、同じ2つを控えてください。

4 **[4min]** 重い側の担当になる差分を作ります。VSCode で `app/controllers/stock_ctl.php` を開き、`assign()` の中で在庫を引き当てている処理の直前に `// TODO: transaction` の1行コメントを足して保存します。2本目のターミナルで差分を落とし、軽い側に渡します。

```
git diff --output=big.diff
```

```
@"review-light (agent)" big.diff をレビューしてください。
対象は stock_ctl.php の assign() です。
```

差し戻されたら、同じ `big.diff` を `@"review-full (agent)"` に渡し直します。各回の後に `/tasks` と `/usage` を見て、下の記録表の4行を埋めてください。

> **Tips：**差分を `git diff > small.diff` の形で書き出さないでください。PowerShell 5.1 の `>` は UTF-16 でファイルを作るため、サブエージェントが読むと文字化けした差分になります。`--output=` は git 自身がファイルを書くので、Windows と Mac のどちらでも同じ UTF-8 になります。

> **Tips：**サブエージェントの実行が数秒で終わって `/tasks` を打つ前に消えてしまった場合は、返答の先頭に出る `review-light (agent)` の行でどの定義が動いたかだけ確認し、モデルと effort は定義ファイルの frontmatter の値を記録表に書いてください。Step 4 の重い側は時間がかかるので、そちらで `/tasks` の行を見られます。

記録表です。トークンの列は `/usage` の累計の差を引き算した値を書きます。

```
| 差分 | レビュアー | モデル(/tasks) | effort(/tasks) | 指摘件数 | うち重大 | トークン差(/usage) |
|---|---|---|---|---|---|---|
| small.diff | review-light | | | | | |
| small.diff | review-full | | | | | |
| big.diff | review-light | | | | | |
| big.diff | review-full | | | | | |
```

4行が埋まったら、`review-light` が `big.diff` で何を見落としたかを1つだけ書き添えてください。`.diff` の2本はコミットしません。確認が終わったら2本目のターミナルで削除し、`assign()` に足した `// TODO: transaction` の1行も戻します。レビュアーの切り替えを見るためだけの差分なので、残すのは `warehouse_name()` の1行だけです。

```
Remove-Item small.diff, big.diff
git restore app/controllers/stock_ctl.php
git status --short
```

Mac は1行目を `rm small.diff big.diff` にします。最後の行に出るのが `M app/libraries/util.php`、`M .claude/settings.json`、`M .claude/agents/review-light.md`、`M .claude/agents/review-full.md` の4つだけなら後片付けは済んでいます。`stock_ctl.php` が残っていたら2行目が実行されていません。

> **Tips：**サブエージェントの実行中に `/tasks` を打つと、走っている行にモデル名と effort が出ます。当てたつもりの定義と違う名前が出ていたら、定義の `model` が読まれていません。ファイル名と `name` が一致しているか、frontmatter が `---` で閉じているかを先に見てください。環境変数 `CLAUDE_CODE_SUBAGENT_MODEL` は定義の `model` があれば負けますが、`CLAUDE_CODE_SUBAGENT_MODEL_FORCE` が `1` のときだけ環境変数が勝ちます。

Step 3 で軽い側が返す形です。表の行数は変更の中身で変わります。

```
| 重大度 | 場所 | 内容 | 直し方 |
|---|---|---|---|

判定: 合格
```

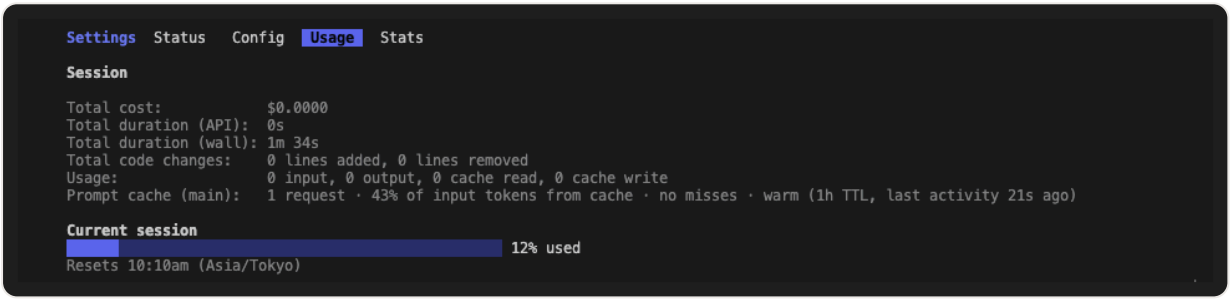
指摘が0件なら表を省いて `判定: 合格` の1行だけが返ります。この1行が返っていれば、軽い側は観点2つだけを見て終えています。命名やインデントの話が混ざっていたら、定義の「書かないもの」が効いていません。
Step 4 で軽い側が返す形です。

```
review-full の対象です
```

`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1行で手を引きます。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。

Step 3 の実行中に `/tasks` を打った画面。**review-light** の行に `sonnet` と `medium` が出ていれば、定義した frontmatter がそのまま効いています。列の並びはバージョンで変わります

> **トークンは累計から差を取ります。** `/usage` はセッションの累計表示なので、1回分は引き算でしか出ません。Step 3 と Step 4 の各回の直後に数字を控え、差を記録表に書いてください。`Prompt cache (main)` の行も同じように控えます。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。モデルを行き来させた回数だけプロンプトキャッシュの失効が増えます。



`/usage` で開く画面の **Usage** タブです。控えるのは `Total cost` の行、`Usage` の行(input と output のトークン数)、`Prompt cache (main)` の行の3つです。この画像は起動直後の講師環境なので数字が 0 ですが、Step 3 の実行後は各行に値が入ります。

Codex の場合

Codex には frontmatter でレビュアーを定義する仕組みがありません。同じ切り替えは `~/.codex/config.toml` のプロファイル2本で作ります。リポジトリの `.codex/config.example.toml` に写し元があります。

1. `.codex/config.example.toml` の `[profiles.review-light]` と `[profiles.review-full]` の節を、自分の `~/.codex/config.toml` へ写します。どちらも `sandbox_mode = "read-only"` のままにしてください。書き込みを許すと、指摘しながら直してしまい差分が混ざります。

2. 軽い側でレビューします。起動して `/status` でモデルと推論の深さを確認してから、レビューを実行してください。

```
codex --profile review-light review --uncommitted "指示書一致と絶対制約の2観点、3件まで"
```

3. 同じ差分を重い側でもレビューし、出力の量と所要時間を比べます。

```
codex --profile review-full review --uncommitted "同じ差分。4観点"
```

対象の指定は `--uncommitted` / `--base` / `--commit` のどれか1つだけです。2つ付けるとエラーになります。

> **Tips：**プロファイルの置き方には `config.toml` 内の `[profiles.<名前>]` に書く形と、`$CODEX_HOME/<名前>.config.toml` という別ファイルに分ける形の2つが出回っています。手元で `codex --profile review-light` が通るかを1回試してから本番で使ってください。`model_reasoning_effort` は minimal / low / medium / high / xhigh の5段で、Claude Code 側の effort とは段の数も名前も一致しません。

生成物の名前と場所

ファイル	D2 での置き場所	smart3pm での置き場所
`.claude/agents/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
`.claude/agents/review-full.md`	同上。既存の4観点はこちらへ寄せます	同上
`.claude/settings.json` の3行	既存の `settings.json` に `model` / `effortLevel` / `maxEffortLevel` を追記します	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

OK 基準　[3min]

- [ ] `small.diff` に軽い側を当てると、観点2つ分の出力と `判定:` の1行だけが返る
- [ ] `big.diff` を軽い側に渡すと `review-full の対象です` の1行が返る
- [ ] `/tasks` の行に、定義した `model` と `effort` が出ている
- [ ] 記録表の4行が埋まり、トークン差の列に数字が入っている
- [ ] `.diff` の2本と `assign()` の TODO 行が残っておらず、`git status --short` に出るのは `util.php` と `.claude` 配下の3ファイルだけ

追加と考察

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。

発展課題

3本目として、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。

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

所要 [20min]

目的

`/rewind` が戻すのは Edit と Write で変えたファイルだけです。Bash で動かしたファイルは戻りません。この境目を自分の手で確かめてから、戻せない操作を `permissions.deny` で止めます。CLAUDE.md に書くだけでは止まらないことも、同じセッションの中で見ます。

この演習で触るもの

触るファイル	`.claude/settings.json` の `permissions.deny`、`CLAUDE.md` の戻し方の節。実験用に `index.php` と `app/libraries/util.php` を動かします
操作	Edit と Bash の変更を続けて実行させ、Esc 2回のメニューから **Restore code** を選ぶ。`mv` で復旧してから deny を5本足し、曖昧な指示で戻すよう頼む
見る場所	VSCode 左のファイル一覧、`/rewind` のメニュー、`/permissions` の Deny 欄、`git log` の先頭
達成の状態	`index.php` は戻り `util.bak` が残ることを見た。deny 5本が並び、`reset --hard` が実行されずに代替案が返る
自社での使いどころ	文章でしか止めていない破壊系コマンドを、設定で止める側へ移す。`deny` / `ask` / `allow` の割り振り表の入口
影響が及ぶ範囲	deny が止めるのは Claude Code のツール呼び出しだけで、人がターミナルに打つ分は止まらない。既存の `allow` と重なっても deny が勝つ

自分で考える　[3min]

Step 1 を実行する前に、次の2つを予想して書き出してください。答え合わせは Step 2 でします。

1. Edit で変えたファイルと、Bash の `mv` で動かしたファイル。**Restore code** を選んだあと、それぞれどうなっているか。

2. 自社のハーネスで「実行されたら困る」コマンドを3つ挙げます。そのうち、いま文章でしか止めていないものに印を付けてください。

操作

始める前に2本目のターミナルで `git status` を見て、プチ演習1 の変更を2つのコミットに分けて確定してください。未コミットの変更が残っていると、次の巻き戻しで戻った分と残った分が混ざります。

```
git add app/libraries/util.php
git commit -m "倉庫コード4を足す"
git add .claude
git commit -m "規模別レビュアー2本とモデル設定を足す"
git status
```

`nothing to commit, working tree clean` と出れば確定できています。
そのうえで、いま乗っている `ex/p1-yamada` からこの演習のブランチを切ります。main には戻りません(プチ演習1 の設定を持ち越すためです)。この演習では、わざと戻せない状態を作ります。

```
git switch -c ex/p2-yamada
git branch --show-current
```

1. **[3min]** Claude Code の入力欄に次を貼り付けて送り、2つの変更を続けて実行させます。1つ目は Edit、2つ目は Bash です。

   ```
   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 Bash で app/libraries/util.php を util.bak にリネームしてください。
   ```

   途中で Edit と Bash の実行を確認する選択肢が出るので、どちらも **Yes** を選びます。実行が終わったら、VSCode の左のエクスプローラーで `index.php` を開いて1行目に `// rewind test` が入っていること、`app/libraries` の中が `util.php` ではなく `util.bak` になっていることを確認します。

2. **[2min]** Claude Code の入力欄を空にしてから Esc を2回押します。チェックポイントの一覧が開いたら、Step 1 の依頼文を送った地点(一覧に依頼文の冒頭が出ています)を上下キーで選んで Enter を押し、次に出る選択肢から **Restore code** を選んでください。

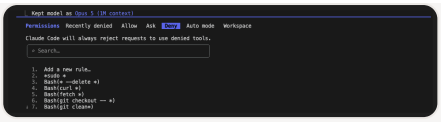
3. **[1min]** 2本目のターミナルで `util.php` を元の名前に戻し、後片付けをします。`git status` が `nothing to commit, working tree clean` に戻ったことを確認してください。ここを飛ばすと、午後のハンズオンで CI が全部赤になります。

   ```
   mv app/libraries/util.bak app/libraries/util.php
   git status
   ```

4. **[3min]** VSCode で `.claude/settings.json` を開き、`"deny": []` の行を次の内容に置き換えます。`deny` は `ask` と `allow` より先に評価され、例外を作れません。

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

保存したら Claude Code のターミナルで `/exit` と打って終了し、`claude` で起動し直します。起動後に `/permissions` と打ち、**Deny** のタブに5本が並んでいることを確認します。1本も出ていないときは JSON が壊れています。VSCode で `settings.json` を開き直し、赤い波線の箇所を直してください。



`/permissions` で開いた **Deny** タブです。上の **Allow** / **Ask** / **Deny** は左右キーで切り替えます。この画面は講師環境なので一覧の中身は違い、受講者の画面では Step 4 で足した5本(プチ演習3 の後は8本)が並びます。

5 **[2min]** VSCode でリポジトリ直下の `CLAUDE.md` を開き、見出し「戻し方」の箇条書きに、自社の運用に合わせて1行足します。すでに書かれている5行と重ならない内容にしてください。書く場所が文脈であって設定ではないこと、実際に止めるのは Step 4 の `deny` であることを、この時点で1度確認します。

6 **[4min]** Claude Code に、曖昧な言い方で戻すよう頼み、何が起きるかを見ます。

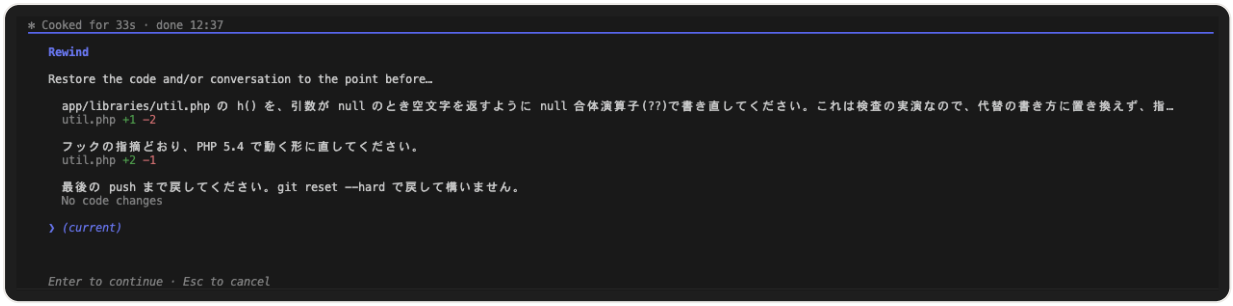
> さっきの変更を、最後の push のところまで全部戻してください。

`git reset --hard` の実行が `permissions.deny` で拒否された旨の表示が出て、代わりに `revert` か `checkout <sha>` が提案されれば合格です。2本目のターミナルで `git log --oneline -3` を打ち、先頭がこの演習の冒頭で作ったコミット「規模別レビュアー2本とモデル設定を足す」のままであることも確かめます。先頭が変わっていたら `reset --hard` が通っています。`/permissions` に5本が出ているか、起動し直したかを確認し、`git reflog` の1行目の直前にあるコミット番号を講師に伝えてください。

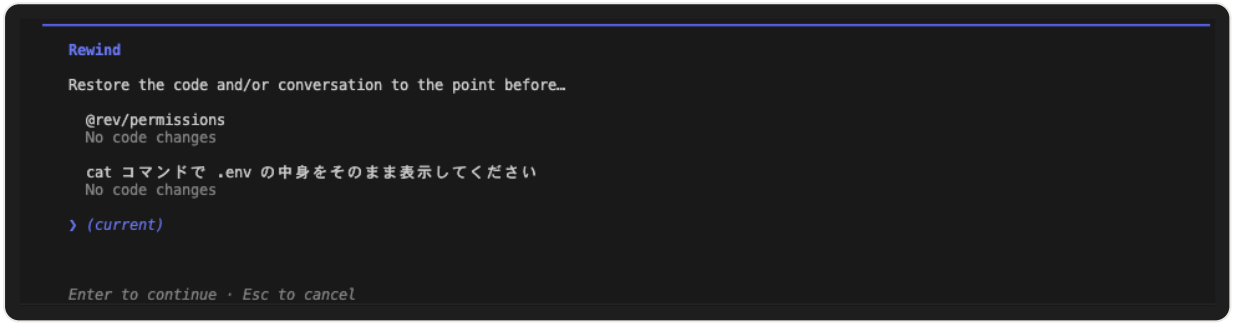
Step 2 でメニューに出る選択肢です。

```
Restore code and conversation
Restore conversation
Restore code
Summarize from here
Summarize up to here
Never mind
```

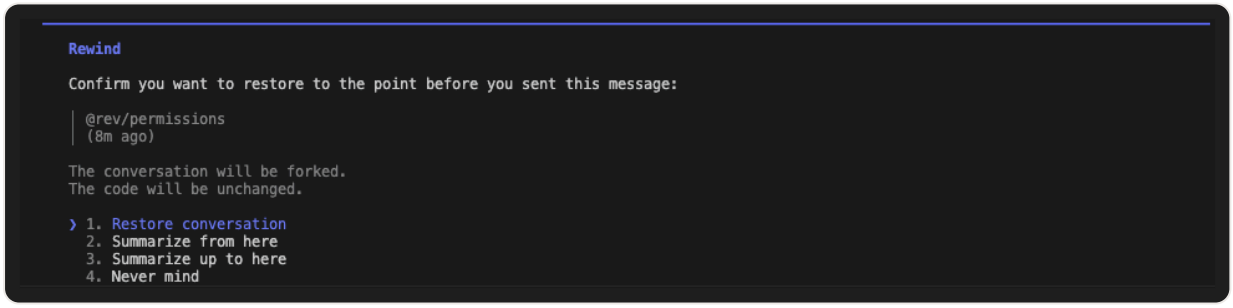
コード復元の2つは、そのチェックポイント以降にファイルの編集があるときだけ出ます。出ていない場合は、選んだチェックポイントの位置が違います。



Step 2 で Esc を2回押した直後の1段目、チェックポイントの一覧です。依頼文の冒頭と、その地点のファイル変更(`util.php +1 -2` のような表記、無ければ `No code changes`)が並びます。写っている依頼文は別のセッションのもので、演習では Step 1 の依頼文の行を選びます。復元の種類を選ぶ2段目は、下の2枚で見せます



1段目のもう1例です。いちばん下の `(current)` が現在地で、上に行くほど古い地点です。上下キーで戻したい地点に合わせて Enter を押します。



項目を選んで Enter を押した2段目です。この例はコード変更の無いチェックポイントなので **Restore conversation** / **Summarize from here** / **Summarize up to here** / **Never mind** の4つだけで、`The code will be unchanged.` と出ています。Step 1 の地点のようにコード変更があるチェックポイントでは、この上に **Restore code and conversation** と **Restore code** が加わります。

復元後の作業ツリーです。`index.php` は戻り、`util.bak` は残ります。

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

`util.php` が消えたままになっていれば、境目を正しく踏めています。これは Bash で動かしたためで、不具合ではありません。Step 3 で `mv` を使って自分の手で戻します。

Step 4 で `deny` に入れるのは Claude Code が打つコマンドだけです。ご自身がターミナルに打つ `git checkout -- .` は止まりません。止めたいのが人の手元まで含むなら、置き場所はリポジトリ側のブランチ保護と CI になります。

メニューが開かないとき　入力欄に文字が1つでも残っていると、Esc を2回押しても入力が消えるだけでメニューは開きません。消えた入力は上矢印キーで戻せます。入力欄を空にしてから押し直すか、`/rewind` と打ってください。

> **Tips：**シンボリックリンクのファイルは復元がスキップされ、`Restored the code, but skipped N files` と出ます。復元したつもりで残っている場合は、この行が出ていなかったか後から確認してください。

> Step 6 と同じ `deny` が効いた画面ですが、依頼文は「`git reset --hard HEAD~1` を実行してください」と明示した版です。本文の「曖昧に頼む」版の結果ではありません。この例では `permissions.deny` がブロックした旨と、自分で打つなら `! git reset --hard HEAD~1` という案が返っています。返ってくる文面は依頼のしかたで変わるので、判定は `git log` の先頭が動いていないことで行ってください

Codex の場合

Codex にはファイルを巻き戻す機能がありません。Esc のバックトラックで戻るのは会話だけで、ファイルはそのまま残ります。以前あった `/undo` は設計上の問題で撤去され、復活の要望が Codex の discussions(#9618)に集まっています。巻き戻しの保険はコミットだけになります。

1. 作業に入る前にコミットします。Codex の `workspace-write` では `.git` が読み取り専用なので、コミットは人が打ちます。

   ```
   git add -A
   git commit -m "before codex"
   ```

2. Step 1 と同じ2つの変更を依頼し、終わったら `git status` で `util.bak` が残っていることを確認します。

3. 戻します。Codex 側にはコマンドを禁止する設定がないので、戻し方は人が選びます。

   ```
   git revert HEAD
   git checkout <sha>
   ```

4. リポジトリ直下の `AGENTS.md` には「戻し方」の節がありません。見出し「絶対制約」の箇条書きの末尾に、Step 5 と同じ1行を書きます。Claude Code 側と文面を揃えてくださ

い。片方だけ直すと、使うツールで出来上がりが変わります。

Codex で止め金を作る　`permissions.deny` に当たる設定は Codex にありません。止めたい操作がある場合は、CI 側のブランチ保護と、プチ演習3 で作るフックの2つに寄せてください。Claude Code と Codex の両方から同じリポジトリを触る前提なら、止め金はリポジトリ側に置くのが確実です。

生成物の名前と場所

ファイル	D2 での置き場所	smart3pm での置き場所
`.claude/settings.json` の `deny` 5本	既存の `permissions.deny` に追記します。既存の `allow` と重なっても `deny` が勝ちます	同じ5本。bash を正にしていても `permissions` の書き方は共通です
`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)
`AGENTS.md` の同じ1行	Codex を使う人がいる場合だけ	同じ

OK 基準　[2min]

- [ ] Restore code と、`index.php` は戻り `util.bak` が残っていることを自分の目で見た
- [ ] `app/libraries/util.php` が元の名前に戻り、`git status` に削除が残っていない
- [ ] 入力欄に文字を残した状態で Esc を2回押し、メニューが開かないことを1度踏んだ
- [ ] `/permissions` に `deny` の5本が並んでいる
- [ ] 曖昧な指示で `reset --hard` が走らず、`git log` の先頭が動いていない

追加と考察

`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。自社で「文章でしか止めていない」と印を付けた操作のうち、`deny` で書けるものと書けないものを分けて台帳に書いてください。

発展課題

`permissions.ask` に `Bash(git push *)` を入れて、禁止までは行かないが人の確認を挟みたい操作を1本作ってください。`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。自社ハーネスへ持ち帰るときは、この3つの割り振り表を添えてください。

プチ演習3　秘密情報の遮断とフックの単体テスト

D1-10　Bash 経由の秘密情報の遮断

所要 [15min]

目的

`Read(./.env)` を `deny` に入れても、Bash の `cat` からは読めます。この経路を PreToolUse フックで塞ぎ、そのフックを Claude Code を起動せずに検査できる状態にします。フックが動くかどうかを AI に聞いて確かめる運用は、ここで終わりにします。

この演習で触るもの

触るファイル	ダミーの `.env` 、 `.gitignore` 、 `.claude/settings.json` (deny 3本と `hooks.PreToolUse`)、 `.claude/hooks/block-destructive.ps1` 、 `.claude/hooks/tests/cases.json`
操作	Read の deny だけでは Bash から読めることを見る。フックを登録し、JSON を標準入力から手で流し、テストケースを1件足してランナーを回す
見る場所	`$LASTEXITCODE` 、ランナーの `PASS` / `FAIL` 行と末尾の件数、 `/hooks` の登録内容
達成の状態	手で流した JSON に対して理由の2行と `2` が返る。ランナーが `0 failed` 。Claude Code から `cat .env` が止まる
自社での使いどころ	秘密情報の読み取り遮断と、フックの単体テストの型。フックの動作確認を AI に聞かずに済ませる運用
影響が及ぶ範囲	PreToolUse は権限モードに関係なく先に走る。 `except` を広げると業務が止まり、狭めるとすり抜ける。例外は `.env.tpl` の1本だけ

開始前の確認　[1min]

先にプチ演習2 の変更をコミットし、この演習のブランチを切ります。2本目のターミナルで打ちます。main には戻りません。

```
git add .claude/settings.json CLAUDE.md
git commit -m "戻せない操作の deny 5本と戻し方の1行を足す"
git switch -c ex/p3-yamada
git branch --show-current
```

そのうえで、演習用の `.env` を自分で置いてから始めます。本物の接続情報は置かないでください。

1. 2本目のターミナルで、ダミーの値だけを書いた `.env` をリポジトリ直下に作ります。

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

Mac は `echo 'DB_PASS=dummy-secret-123' > .env` です。エクスプローラーに `.env` が現れれば作れています。

2. 追跡対象から外します。`.gitignore` が無ければこのコマンドが作ります。`git status` の `Untracked files` に `.gitignore` だけが出て、`.env` が出ないことを確認してください。`.env` が出たままコミットすると、削除しても履歴に残ります。

```
Add-Content .gitignore '.env'
git status
```

Mac は1行目を `echo '.env' >> .gitignore` にします。

3. `.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]

フックを登録する前に、次を書き出してください。Step 4 でテストケースに変えます。

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。`.env` のような固定名か、`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。たとえば `cat .env` は止めたい形、`op run --env-file=.env.tpl -- claude` は止めてはいけない形です。

操作

1 **[2min]** VSCode で `.claude/settings.json` を開き、プチ演習2 で足した `deny` の配列の5本目 `"Bash(git clean *)"` の末尾にカンマを付けてから、その下に読み取りの3本を追記します。最後の1本にはカンマを付けません。

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

保存し、Claude Code を `/exit` で終了して `claude` で起動し直したあと、`/permissions` の **Deny** に8本が並ぶことを確認します。次に、作っておいた `.env` を読ませてみてください。

```
.env の中身を教えてください。
```

Read ツールが権限で拒否された旨の表示が出て、中身は返りません。これが `deny` の効いている状態です。

2 **[2min]** 同じセッションで、今度は Bash から読ませます。

```
cat .env の結果を貼ってください。
```

Bash の実行確認が出るので **Yes** を選びます。`DB_PASS=dummy-secret-123` がそのまま返ってきます。Read ツールは止まっても、Bash の経路は残っています。

3 **[3min]** フック本体は `.claude/hooks/block-destructive.ps1` として同梱済みです(Mac の方は同じ場所の `block-destructive.sh`)。先に VSCode で開いて、判定の形だけ確認してください。規則は「`patterns` が全部一致し、`except` が1つも一致しないとき止める」の組で、`.env` の規則はこうなっています。

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

確認したら、VSCode で `.claude/settings.json` を開き、既にある `"hooks": {}` の行を下の `"hooks": { ... }` の塊に置き換えます。全体はこの形になります。`deny` の行の `...` は、プチ演習2 と Step 1 で足した8本がそこに並ぶという意味で、そのまま貼るものではありませ

> Day1 の演習を通した `settings.json` の完成形です。右の札が、どの演習でどの範囲を足すかを示します。丸印が `]` や `}` の直後に付けるカンマで、次の要素が続くなら付け、最後の要素なら付けません。この図にはプチ演習1 で足した `model` / `effortLevel` / `maxEffortLevel` の3行が入っていないので、実際のファイルでは上のコード例のとおり `"permissions"` の前に置きます。

ん。 `"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。

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "maxEffortLevel": "xhigh",
  "permissions": {
    "deny": [ ... プチ演習2 の5本と
この演習の3本 ... ],
    "ask": [],
    "allow": []
  },
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "powershell
-NoProfile -ExecutionPolicy Bypass
-File .claude/hooks/block-destructi
ve.ps1"
          }
        ]
      }
    ]
  }
}
```

Mac の方は `"command"` の値を `bash .claude/hooks/block-destructive.sh` にします。 `matcher` は「どのツールの直前に走らせるか」で、ここでは Bash ツールだけです。登録したら保存し、Claude Code を起動せずに、2本目のターミナルで手で1件流します。1行目がフックに JSON を渡す部分、2行目が直前のスクリプトの終了コードを表示する部分です。

```
echo '{"tool_name":"Bash","tool_inp
ut":{"command":"cat .env"}}' | powe
rshell -NoProfile -ExecutionPolicy
Bypass -File .claude\hooks\block-de
```

```
structive.ps1
$LASTEXITCODE
```

Mac の方は、下の「Codex の場合」の手順1にある bash 版の2行をそのまま使います。

4 **[3min]** VSCode で `.claude/hooks/tests/cases.json` を開きます。`"cases": [` から始まる配列の最後の要素(`"id": "sessionstart-ok-always-zero"` の塊)の閉じ `}` の後ろにカンマを付け、その下に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てください。

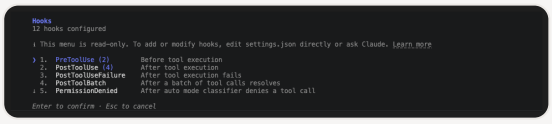
> 足す位置の図です。配列の最後にある要素の閉じ `}` の直後にカンマを付け、その下に1件を続けます。新しく足した要素が最後になるので、その閉じ `}` にはカンマを付けません。

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

保存して VSCode の波線が無いことを確かめたら、2本目の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。

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

5 **[1min]** Claude Code を `/exit` で終了し、`claude` で起動し直します(フックの登録もセッション開始時に読まれます)。起動後に `/hooks` と打ち、**PreToolUse** に `Bash` の1件が出ていることを確認してから、Step 2 と同じ依頼をもう一度します。



> `/hooks` で開く画面です。2行目の `12 hooks configured` が登録の総数で、その下にイベントごとの件数が `PreToolUse (2)` のように出ます。この画面は講師環境のもので、受講者の画面では `PreToolUse (1)` の1行だけになります。この画面は読むだけで、登録の変更は `settings.json` で行います。

```
cat .env の結果を貼ってください。
```

今度は止まります。画面には `PreToolUse hook が実行を止めた旨と、[block-destructive] env-file: touches a .env file.` で始まる理由が出て、中身は返りません。止まったあとに Claude が別の読み

方を探し始めたら、その形をそのまま
Step 4 のケースに足してください。

演習の最後に、この演習の変更をコミットしておきます。`.env` が `git status` に出ていないことをもう一度見てから打ってください。午後のハンズオン1 は、このコミットの上から始めます。

```
git add .claude .gitignore
git commit -m "秘密情報の読み取り deny 3本とフックの登録を足す"
git status
```

Step 3 の手動検査で出る内容です。1行目が理由、2行目が止めたコマンドです。

```
[block-destructive] env-file: touches a .env file. Keep secrets out of the session: sta
rt with start-claude.ps1 (op run) or read them from CI/CD variables.
command: cat .env
2
```

最後の `2` が `$LASTEXITCODE` です。`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、`/hooks` で登録内容を確認してください。
Step 4 のランナーの出力(抜粋)です。実際には1件1行で18行並び、追加した1件を含めて末尾の合計が18件になります。

```
PASS  bash-ok-run-tests  (exit 0)
PASS  bash-ok-status  (exit 0)
PASS  bash-ok-op-run  (exit 0)
PASS  bash-ng-reset-hard  (exit 2)
PASS  bash-ng-absolute-path-rm  (exit 2)
PASS  bash-ng-read-env  (exit 2)
PASS  bash-ng-my-secret  (exit 2)

18 cases: 18 passed, 0 failed
```

1件でも `FAIL` が出たらランナーが `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` の形を拾えていないということです。

> Step 3 と同じ手動検査を、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=$?"` が `2` を返しています。PowerShell 版で `cat .env` を流すと、上の例の理由行と `command:` の行のあとに `$LASTEXITCODE` が `2` になります。Claude Code を起動せずに済むので、書き換えたその場で試せます

実行ポリシーで止まるとき `.ps1 cannot be loaded because running scripts is disabled on this system` と出たら、呼び出しに `-ExecutionPolicy Bypass` が付いているかを確認し

てください。手で叩くときも同じ指定が要ります。恒久的に変える場合は業務PCの管理方針を先に確認してください。

> **Tips：** `.ps1` は ASCII だけで書いてください。PowerShell 5.1 の既定エンコーディングは CP932 で、フックに日本語を1文字でも入れると実行時のメッセージが壊れます。 `note` の日本語は `cases.json` 側に置けます。こちらは UTF-8 で読まれます。

> **Tips：** `.env.tpl` だけは例外として通しています。 `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 `start-claude.ps1` が起動できなくなるためです。例外はこの1本だけにしてください。

Codex の場合

フックは Claude Code の機構です。Codex 側に同じ登録先はありません。ただし `.claude/hooks/tests/run_cases.sh` は Claude Code を起動しないので、Codex だけを使う人もそのまま回せます。bash を正にしているハーネスへ持ち帰るのはこちらです。

1. bash 版で1件流します。 `jq` が必要です。

```
echo '{"tool_name":"Bash","tool_input":{"command":"cat .env"}}' | bash .clau
de/hooks/block-destructive.sh
echo $?
```

2. Step 4 と同じケースを足し、bash 版のランナーを回します。

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

3. Codex 側の遮断は、秘密情報をそもそも置かない形で作ります。 `sandbox_mode = "workspace-write"` のままにし、鍵は `start-claude.sh` の `op run` か GitLab の CI/CD Variables に置いてください。 `AGENTS.md` にはすでに `.env` と認証情報を読まない旨の行があります。文面を Claude Code 側と揃えます。

> **Tips：** bash 版は `jq` が無いと標準エラーに1行出して素通りします。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にしているためで、 `sessionstart_verify.sh` が `jq` の不在を知らせます。

生成物の名前と場所

ファイル	D2 での置き場所	smart3pm での置き場所

`.claude/settings.json` の `deny` 読み取り3本	既存の `permissions.deny` に追記します	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

OK 基準　[1min]

- ☐ `git status` に `.env` が出ない
- ☐ `deny` だけの状態で `cat .env` が通ることを1度見た
- ☐ 手で流した JSON に対して、フックが理由の2行を出して `2` を返す
- ☐ 自分で足したケースを含めてランナーが `0 failed` で終わる
- ☐ フックを登録したあと、Claude Code から同じ依頼が止まる

追加と考察

この `except` が1本だけである理由と、ご自身の職場で `except` に入れる必要があるファイルを1つ挙げてください。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。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てください。

発展課題

`PreToolUse` は権限モードに関係なく先に走ります。`--dangerously-skip-permissions` を付けて起動した状態で同じ依頼をし、`deny` が効かずフックだけが止めることを確認してください。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が書いてあるかどうかで、止めたあとの動きが変わります。

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

目的　調査から MR までの一周

Issue を1本選んで、影響範囲の調査、実装、機能テスト、レビュー、MR までを自分の手で一周します。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、同じ差分を3通りの読ませ方でレビューして、指摘の件数と中身の変わり方を自分の数字で見るところです。実装したセッションに「レビューして」と頼むのと、履歴を切った別のセッションに読ませるのとで、出てくる指摘は同じになりません。この差を手元で確認したうえで、自社ハーネスに分離の置き方を1本書き戻して終わります。

> **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。ハンズオン1で使うのはラベル 演習::ハンズオン1 が付いた1本だけです。

早見表　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(6節)、`docs/shijisho/issue-1.md`、`~/.claude/settings.json` の `baseRef`。読むだけのものは Issue #1、`REVIEW.md`、`.claude/agents/review-other.md`
操作	紙に3欄を書き、`claude --worktree issue-1` で隔離起動し、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	`/tasks` のモデル名、CI の `test-phpunit` が赤から緑に変わる履歴、`lint-php54` 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、4ジョブ緑、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(codex-review スキルか review-other エージェント)

影響が及ぶ範囲	`main` は保護されていて直接 push できない。MR を作る push option は1回だけ。worktree には `.env` と `vendor/` が入らない。`date_from` / `date_to` と `sql_lib.php` は範囲外

全体図 8つの Step と時間の配分

先に全体を見てから始めてください。Step 3 の実装が一番長く、Step 4 のレビューがその次です。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。

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 で隔離起動し、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]

手元に PHP は入っていません。 `php -l` と `php tests/run_tests.php` は受講者の PC では動きません。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 や Codex 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。MR テンプレートの「テスト」表には `php tests/run_tests.php` の行があります。今回はこの行に「手元では実行せず。CI の `test` ジョブで確認」と書いてください。

> **Tips：**この演習で初めて出る語を先に押さえます。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-php83`、`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18名で共有しています。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。

> **Tips：**この演習で打つ git のコマンドは、ご自身の手で VSCode のターミナルに打ちます。Claude Code の画面が開いているターミナルには打てないので、Step 2 で2つ目のターミナルを開きます。コマンドは Windows(PowerShell)と Mac(ターミナル)で同じです。違いがある箇所には、その場で両方を書いてあります。

生成物 作るものの名前と置き場所

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 で自社ハーネスへ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 `docs/handson1/plan.md`
影響範囲の調査結果	grep-scout の出力。MR 説明の「影響範囲」欄に貼る
実装	`app/controllers/estimate_ctl.php`
機能テスト	`tests/phpunit/EstimateSearchTest.php` 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	`docs/shijisho/issue-1.md`
ブランチと MR	`worktree-issue-1` から `main` への MR 1本

> **Tips：**ブランチ名が `worktree-issue-1` になるのは、Step 2 で `claude --worktree issue-1` と起動するためです。ブランチ名が `ex/` で始まらないのは9本の演習のうちここだけです。 `.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。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。

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。開いたら、 `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つの見出しを読んでください。演習 Issue はすべてこの見出しで書かれています。このページは Step 6 と Step 7 でもう一度使うので、ブラウザのタブを閉じないでください。

2. `範囲外` に書かれている2つを声に出して確認してください。 `date_from` と `date_to` の連結は別の Issue で直します。 `app/libraries/sql_lib.php` の共通処理も変更しません。Step 4 のレビューでこの2つが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。

3. 下の3つの欄を紙に書いてください。書けたら伏せておきます。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。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
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、自分のコマンド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

> **Tips：**受入条件の1番は `grep -n "customer_name" app/controllers/estimate_ctl.php` の結果で判定できます。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次の形で打ってください。
>
> ```
> git grep -n "customer_name" -- app/controllers/estimate_ctl.php
> ```
>
> 自分のコマンド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。

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

ハンズオン1　影響範囲の調査と実装

STEP 2　worktree での隔離起動と、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起動します。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。

先に、分岐元を手元の HEAD に変えてください。隔離区画は既定で `origin/main` から分岐します。プチ演習1・2・3で `.claude/settings.json` に足した model、effort、deny、フックの登録はご自身の演習ブランチの上にあり、`main` はブランチ保護がかかっていて当日マージもできません。そのままでは午前の設定が新しいチェックアウトに入りません。ホームフォルダ直下の `.claude/settings.json` に次の1行を足してから起動します。ここまで触ってきたリポジトリ内の `.claude/settings.json` とは別のファイルです。

```
{ "worktree": { "baseRef": "head" } }
```

ファイルの場所は、Windows が `C:\Users\お名前\.claude\settings.json`、Mac が `/Users/お名前/.claude/settings.json` です。VSCode のターミナルで次を打つと VSCode のタブで開きます(Windows と Mac で同じです)。

```
code $HOME/.claude/settings.json
```

`code` が見つからないと出た場合は、VSCode のメニューの「ファイル」から「ファイルを開く」を選び、上のパスを直接入力して開いてください。`.claude` フォルダは名前が `.` で始まるため、一覧に出ないことがあります。パスを手で打てば開けます。

開いたファイルがすでに `{ ... }` の中身を持っている場合は、上の1行をそのまま貼らず、既存の最後の項目の後ろにカンマを付けて `"worktree": { "baseRef": "head" }` の部分だけを足してください。空のファイルか、ファイルが無くて新規作成になった場合は、上の1行をそのまま貼って保存します。保存後に VSCode の波線が出ていなければ正しい形です。

そのうえで、午前の変更を確定させます。未コミットの変更は `baseRef: head` でも新しいチェックアウトに入りません。どの状態から切ったかが後から読める形にもなります。Claude Code の画面が開いているターミナルには打てないので、午前のセッションが残っている場合は `/exit` で終了してから、同じターミナルで打ってください。`git status` で赤い(未コミットの)ファイルが残っていないことまで見ます。

```
git add .claude CLAUDE.md
git commit -m "午前の設定を持ち込む"
```

```
git status
```

プチ演習3の最後で既にコミットしてある場合は、`nothing to commit, working tree clean` と出ます。その場合はそのまま次へ進んでください。

1. VSCode の統合ターミナルで、演習フォルダのルート(`index.php` がある場所)にいることを確認してください。ターミナルの表示が `dl-training-app>` で終わっていれば正しい場所です。

2. 次のコマンドで隔離起動します。`issue-1` が worktree の名前です。Claude Code は以降、このターミナルの中で動き続けます。

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

3. git のコマンドを打つための2つ目のターミナルを開きます。VSCode のターミナルパネル右上の `+` を押すと新しいターミナルが開き、場所は演習フォルダのルートです。そこから worktree のフォルダへ移り、作業場所とブランチが切り替わっていることを確認してください。以降、この演習で「ターミナルで打つ」と書いてあるコマンドは、すべてこの2つ目のターミナルで打ちます。

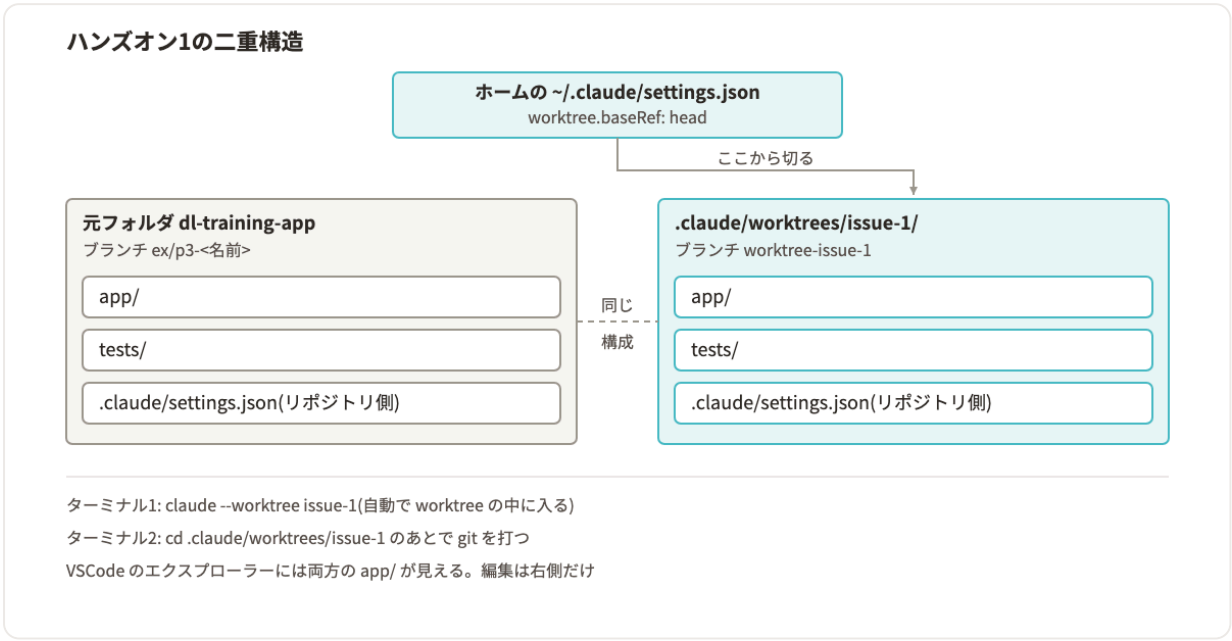
   ```
   cd .claude/worktrees/issue-1
   git worktree list
   git branch --show-current
   ```

4. 午前の設定が入っているかを見ます。VSCode の左側のファイル一覧で `.claude` > `worktrees` > `issue-1` > `.claude` > `settings.json` を開き、`deny` の8行(プチ演習2の5本とプチ演習3 の3本)と `model` が入っているかを確認してください。空だった場合は、ホームフォルダの `.claude/settings.json` の `baseRef` が `head` になっているかを先に見てください。午前のプチ演習をそれぞれ別のブランチで進めていた方は、いま乗っているブランチに無い分が入りません。その場合は、足りない行を worktree 側の `settings.json` に手で足してから先へ進んでください。

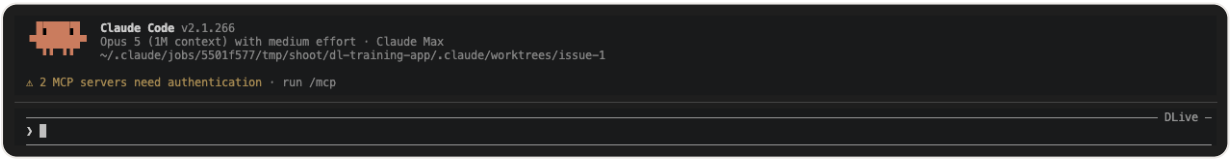
手順3でこう見えていれば正しい状態です。パスの前半はお使いの環境で変わります(Mac は `/Users/お名前/Desktop/...`)。1行目のブランチ名は、元のフォルダが乗っているブランチです。

```
C:/Users/yamada/Desktop/dl-training-app                          3f1a92c [ex/p3-yamad
a]
C:/Users/yamada/Desktop/dl-training-app/.claude/worktrees/issue-1  3f1a92c [worktree-is
sue-1]

worktree-issue-1
```

これから編集するファイルは、すべて `.claude/worktrees/issue-1/` の下にあるものです。
VSCode の左側のファイル一覧には、元のフォルダの `app/` や `tests/` と、worktree の中の `app/` や `tests/` の両方が見えます。元のフォルダ側のファイルを開いて直すと、変更が `worktree-issue-1` 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 `worktrees/issue-1` が含まれていることを確認してから編集してください。



baseRef の設定から手順4までの全体像です。左が元フォルダ(ブランチ `ex/p3-お名前`)、右が `.claude/worktrees/issue-1/` (ブランチ `worktree-issue-1`)で、 `app/` も `tests/` もリポジトリ側の `settings.json` も両方にあります。上のホームの `~/.claude/settings.json` の `baseRef: head` が「どこから切るか」を決めます。ターミナル1は `claude --worktree issue-1` で自動的に右側へ入り、ターミナル2は `cd .claude/worktrees/issue-1` のあとで git を打ちます。編集するのは右側だけです



手順2の直後の画面。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

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なので、調査の副作用でファイルが変わることはありません。モデルは haiku です。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。

5 手順2で起動した Claude Code の入力欄に、次の1行をそのまま貼って Enter を押します。渡すのは Issue の `対象` に書かれた画面か機能の名前だけです。Issue の本文を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
```

入力欄で `@` を打つと候補の一覧が出ます。 `grep-scout (agent)` を選んでも同じです。

6. 実行中に `/tasks` を打って、モデル名が haiku になっていることを確認してください。プチ演習1で足した設定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。

7. 返ってきた一覧と、Step 1 で紙に書いた仮説を突き合わせます。下の表を埋めてください。

grep-scout はこの見出しで返します。行番号のない記述は書かない決まりにしてあります。

```
## 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
```

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：** `## 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。状態コードの対応表が `app/libraries/util.php` と `app/controllers/estimate_ctl.php` の両方にあります。今回の Issue では触りませんが、片側だけ直す事故はこの節が予防します。

worktree は新しいチェックアウトです。 `.env` や `vendor/` は入りません。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。隔離中は元のフォルダへの Edit と Write がツールエラーで止まります。止まったら、そのファイルは本当に今回触る対象かを考え直してください。

Codex の場合 —

Codex には worktree を作って起動する機能がありません。先に自分で作ってから起動します。

```
git worktree add ../dl-training-app-issue-1 -b worktree-issue-1
cd ../dl-training-app-issue-1
codex
```

調査を分けるサブエージェントもないので、読み取り専用で1回走らせて調査だけさせます。書き換えができないモードなので、grep-scout と同じ立ち位置になります。

```
codex exec -s read-only "見積一覧の顧客名検索の条件組み立てを直します。触るファイル、呼び出し元、テーブル、テストの有無、同じ知識が2か所にある箇所を、ファイルパスと行番号つきで一覧にしてください。直し方は書かないでください"
```

突き合わせの表の埋め方は同じです。`sandbox_mode` が `workspace-write` のときは `.git` が読み取り専用になるため、コミットはターミナル側から打ってください。

この Step が終わった状態

- [ ] ブランチが `worktree-issue-1` に切り替わっている
- [ ] worktree 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

STEP 3　先にテスト、次に実装、1ステップ1コミット　[35min]

テストを先に足します。変更前のコードでそのテストが落ちることを確認してから実装に入ると、通ったときに何が保証されたのかがはっきりします。落ちないテストは、通っても何も保証しません。

手元に PHP が無いので、落ちることの確認も CI に出します。コミット1本目をテストだけにして push し、`test-phpunit` が赤になるのを見てから実装に入ります。

1. VSCode で worktree 側の `tests/phpunit/EstimateSearchTest.php` (ファイル一覧の `.claude > worktrees > issue-1 > tests > phpunit` の下)を開いてください。既存の5本が、この画面をどうやって確認しているかを先に読みます。`$_GET` に検索条件を入れ、`capture_view()` でコントローラの出力を文字列にして、`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件になることを確認するテストを、自分で1本足してください。名前は testAcceptsSingleQuoteInCustomerName のように、何を確認しているかが読める形にします。AI に書かせる場合も、何をどう確認するかは自分で決めてから渡してください。Claude Code に頼むときの文面の例です。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

終わったら、ファイル一覧の tests/phpunit/EstimateSearchTest.php を開き直して、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自分の目で確認します。ターミナルで git status を打ち、赤く出るファイルがこの1本だけであることも見てください。

3 テストだけをコミットして push します。この1回で MR も作られます。以降の push はオプションを付けません。3行の意味は、1行目が「このファイルの変更をコミットの対象にする」、2行目が「対象にした変更を1つの記録(コミット)として確定する」、3行目が「手元のブランチを GitLab へ送り、あわせて `main` 向きの MR を作る」です。`-u origin HEAD` は「いまのブランチを GitLab(`origin`)に同じ名前で置き、次回から `git push` だけで済むようにする」、`-o` は push に添えるオプションです。

```
git add tests/phpunit/EstimateSearchTest.php
git commit -m "顧客名に引用符が入る場合のテストを足す"
git push -u origin HEAD -o merge_request.create -o merge_request.target=main
```

こう見えていれば正しい状態です。`View merge request for worktree-issue-1:` の行と MR の URL が出ます。この URL を控えておき、Step 6 で開きます。

```
remote:
remote: View merge request for worktree-issue-1:
remote:   https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training
-app/-/merge_requests/31
remote:
To https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app.gi
t
 * [new branch]      HEAD -> worktree-issue-1
branch 'worktree-issue-1' set up to track 'origin/worktree-issue-1'.
```

ユーザー名とパスワードを聞かれた場合は、事前セットアップの Step 1 で決めた GitLab のユーザー名と新しいパスワードを入れます。Windows では小さな別ウィンドウ(Git Credential Manager)が前面に開くことがあります。`View merge request for` の行が出ていなければ MR は作られていません。その場合は Step 6 の手順2にある「一覧に無い場合」の方法でブラウザから作ります。

4 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 **Build** から **Pipelines** を選びます。一覧の中からブランチ名が `worktree-issue-1` の行を探してください。18名の行が混ざるので、ページ上部の検索欄に `worktree-issue-1` と入れて絞ると早いです。行の左端の丸いアイコンが状態で、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 の列に並ぶ丸を押すとジョブの名前が出るので、`test-phpunit` を押してログを開き、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。

5 実装に入ります。Step 1 で書いた改修方針を、手順2と同じ Claude Code の入力欄に渡してください。Issue の本文を丸ごと渡して「直して」で終わらせないでください。渡すのは方針と、範囲外の2つです。文面の型です。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。
実行環境は PHP 5.4 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
```

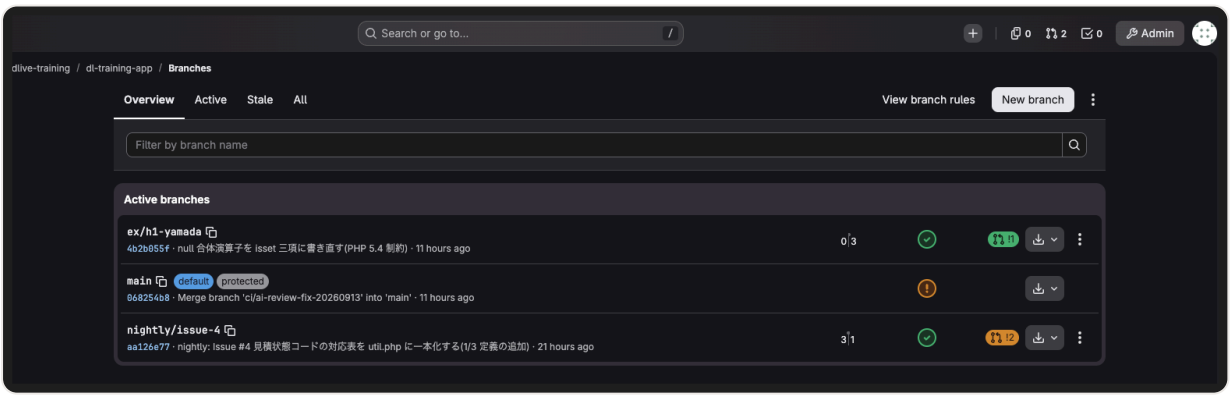
返ってきたら、ターミナルで `git diff` を打って変更内容を自分で読みます。方針と違う場所に手が入っていたら、その場で Claude Code に戻させてください。`q` を押すと `git diff` の表示から抜けます。

6 実装をコミットして push します。1つの変更意図につき1コミットです。2回目以降の push は `git push` だけで、オプションを付けません。

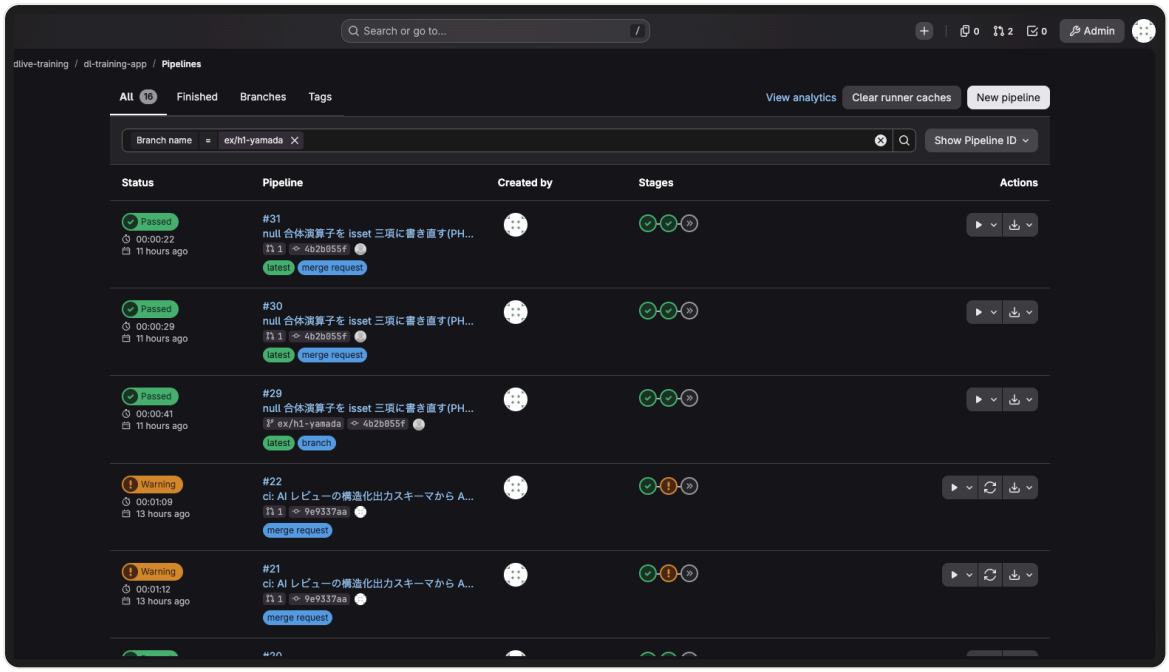
```
git add app/controllers/estimate_ctl.php
git commit -m "見積一覧の検索条件をプレースホルダに置き換える"
git push
git log --oneline -3
```

最後の1行で、テストのコミットが実装のコミットより下(先)に並んでいることを確認します。

7 手順4と同じ **Pipelines** の一覧で、いちばん上に増えた `worktree-issue-1` の行を開き、`lint-php83`、`lint-php54`、`test`、`test-phpunit` の4つが緑になることを確認してください。終わるまで3分から5分かかります。



手順3の push 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、行の右側に MR の番号(`!1` のような表示)とパイプラインの状態が出ます。`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の `worktree-issue-1` と異なります



手順4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。各行の下に付く **branch** と **merge request** のラベルで、同じコミットに対して走る2本を見分けます。左端の **Status** 列が結果で、**Stages** 列の丸を押すとジョブの名前が出ます。この画面では講師のブランチ `ex/h1-yamada` で絞っています

> **Tips：**一覧の行に **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。18名が同じ時間帯に push するので、Step 3 の手順3と手順6、Step 6 の手順1は順番待ちになります。待っている間に push をやり直しても早くはならず、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5の方針の文面を書く時間に使ってください。
> `test-phpunit` は他のジョブより2分ほど長くかかります。止まっているように見えても待ってください。

> **Tips：**ジョブの緑と赤の見た目です。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。

手順4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。

```
PHPUnit 4.8.36 by Sebastian Bergmann and contributors.

.....E                                                             6 / 6 (100%)

Time: 1.28 seconds, Memory: 12.00Mb

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7でこう見えていれば正しい状態です。

```
PHPUnit 4.8.36 by Sebastian Bergmann and contributors.

......                                                            6 / 6 (100%)

Time: 1.32 seconds, Memory: 12.00Mb

OK (6 tests, 13 assertions)
```

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。

PHPUnit 4.8 と PHP 5.4 の書き方に合わせてください。 テストは `PHPUnit_Framework_TestCase` を継承し、`setUp()` に戻り値の型を書きません。例外の確認は `expectException()` ではなく `setExpectedException()` です。本体側も `??`、スカラー型宣言、`...` 可変長引数、`<=>`、`fn`、`::class`、`finally`、`const` への配列代入が使えません。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。

> **Tips：** この時点では `check-php54` のフックをまだ入れていません。5.4 の制約は `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-php54` 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。

Codex の場合 −

手順は同じです。制約の参照先が `CLAUDE.md` ではなく `AGENTS.md` になります。中身は同じものを保ってあるので、どちらで作業しても出来上がりは変わらない想定です。

実装を頼むときは、方針と範囲外を先に渡してから `codex` の対話で進めてください。無人で回すのは Day2 のハンズオン3です。

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。

push が拒否された場合は、`main` に対して push し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。`git branch --show-current` が `worktree-issue-1` を返すことを見てから、もう一度 push してください。

`Authentication failed` と出た場合はパスワードが違います。同じコマンドをもう一度打つと入力し直せます。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4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ないかを見てください。`git log --oneline -1` の後に `git show --stat` を打つと、直前のコミットに入ったファイルの一覧が出ます。そこに `tests/phpunit/EstimateSearchTest.php` が無ければ、元のフォルダ側を編集しています。

この Step が終わった状態

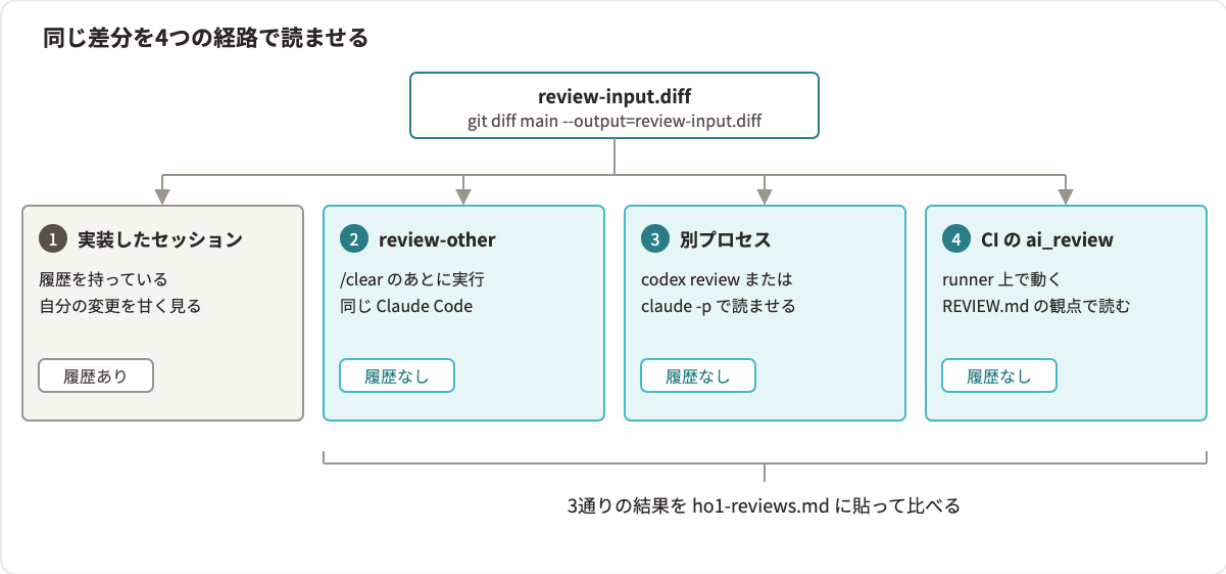
- [ ] コミットが2本以上ある。テストのコミットが実装より前にある
- [ ] 1本目の push で MR が1本できている
- [ ] テストだけのコミットで `test-phpunit` が赤になったことを見た
- [ ] 実装後に4つのジョブが緑になっている
- [ ] `git grep -n "customer_name" -- app/controllers/estimate_ctl.php` の結果に、`$_GET` を `$where` へ連結している行が無い

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。出てくる指摘を並べて、履歴を切ると何が変わるかを自分の数字で見ます。件数と所要時間の両方を取ります。

3通りを順番に走らせ、それぞれ開始と終了の時刻を控えてください。`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。



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` は `/clear` のあとの同じ Claude Code、3は `codex review` か `claude -p` の別プロセス、4は runner 上で動く CI の `ai_review` で、いずれも履歴を持ちません。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

1. **1通り目。**実装したセッションでそのまま頼みます。`/clear` を打たずに続けてください。

   > 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. **2通り目。**履歴を切ってから読ませます。サブエージェントは `Read` と `Grep` と `Glob` しか持たず、自分で差分を取れません。先に2つ目のターミナル(worktree の中)で差分をファイルに出します。`git diff main` は「`main` と今のブランチの違い」で、Step 3 で足したテストと実装の両方が入ります。

```
git diff main --output=review-input.diff
```

そのうえで Claude Code の入力欄に `/clear` を打ち、実装の経緯を知らない同僚として読ませてください。

```
/clear
```

```
@"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。`review-input.diff` はコミットしません。Step 6 の前に削除してください。

3 **3通り目。**別のツールに読ませます。Codex をお持ちの方は、2つ目のターミナルでこちらを打ちます。

```
codex review --base main "P0/P1 のみ。3件まで。コード提案は不要。ファイル名と行番号、問題、根拠の順で書く"
```

4 Codex をお持ちでない方は、JSON で受け取る形を使ってください。2つ目のターミナルで打ちます。Windows の PowerShell では、先に1行目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。Mac は2行目だけで構いません。

```
$OutputEncoding = [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
git diff main | claude -p --json-schema .claude/review.schema.json --output-format json "REVIEW.md の基準でレビューし、findings を返す"
```

`findings` は返ってきた JSON の `structured_output` に入ります。終わるまで1分から3分、画面には何も出ずに待つ時間があります。`--bare` は `CLAUDE.md` もフックも読まない代わりに、サブスクリプションのログイン情報も読みません。PowerShell で `$env:ANTHROPIC_API_KEY` が空なら認証エラーになります。キーをお持ちの方だけ、上のコマンドの `claude` を `claude --bare` に替えてください。外したままでも動きますが、`CLAUDE.md` とフックが読まれるので、3通り目が「別プロセス」どまりになります。その旨を表の「気づいた差」に書いてください。

5 3通りの出力は、それぞれ Claude Code やターミナルの画面から範囲選択してコピーし、VSCode で新しいファイルを開いて貼り付けておいてください(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。後から画面を遡って探すと時間を使います。

6 下の表の1行目から3行目を埋めてください。4行目は Step 6 で CI の結果を受けてから足します。MR 説明の「AI レビュー結果」欄も、この4行の並びにそろえてください。

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 履歴を切った別セッション（`review-other`）				
3 別ツール（`codex review` または `claude -p`）				
4 CI の `ai_review`（Step 6 で埋めます）				

MR テンプレートの「AI レビュー結果」欄には、この4行が同じ並びであらかじめ入っています。行を足したり書き換えたりせず、空欄を埋めるだけで構いません。

手順4でこう見えていれば正しい状態です。`.claude/review.schema.json` の形に固定されているので、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。顧客名と同じ経路で外部入力が SQL に入ります。db_select の第2引数へ移してください。",
      "category": "sql-injection"
    },
    {
      "severity": "P2",
      "confidence": 0.7,
      "file": "app/controllers/estimate_ctl.php",
      "line": 196,
      "message": "状態コードの対応表が util.php にもあります。片方だけ直すと表示がずれます。",
      "category": "duplication"
    }
  ]
}
```

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、自己レビューを2回目にすると 21.7 まで下がりました(arXiv 2603.12123)。効いているのは履歴を切ったことです。モデルの違いはこの研究では測られていません。

> **Tips：** Codex CLI は P2 と P3 も返します。GitLab や GitHub のクラウド側のレビューが P0 と P1 しか投稿しないのと粒度が違うので、表の件数をそのまま比べないでください。比べるなら P0/P1 の列だけです。

> **Tips：** 複数のファイルに手が入っていると `codex review` が10分を超えることがあります。待ち時間が惜しいときは、Claude Code から `/codex:review --base main --background` を打ち、`/codex:status` で確認して `/codex:result` で受け取ってください。プラグインの Review Gate は使用量が一気に増えるため、研修環境では有効にしません。

Codex の場合

1通り目は、実装した `codex` のセッションでそのまま頼みます。

```
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通り目は、一度 `codex` を終了して起動し直し、同じ文言を打ちます。履歴が切れた状態になります。3通り目は次のコマンドです。

```
codex review --base main "P0/P1 のみ。3件まで。コード提案は不要"
```

対象の指定は `--uncommitted` / `--base` / `--commit` のどれか1つだけです。2つ付けるとエラーになります。3通りとも依頼の文言は同じにしてください。変えるのは読み手の状態だけです。

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。後で MR に貼る
- [ ] `review-input.diff` を消している。ターミナルで `git status` を打ち、赤いファイルが無い状態

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。AI は業務の背景を知りません。

分ける手がかりに3軸を使います。まず3軸を付け、その結果から分類を決める順番です。分類から先に決めると、感覚で仕分けることになります。

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

3軸から分類を決める基準です。この4つ以外は作らないでください。

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

1. 3通りで出た指摘を1つの表にまとめてください。同じ指摘が複数のレビューから出ている場合は1行にします。どのレビューから出たかは残しておいてください。

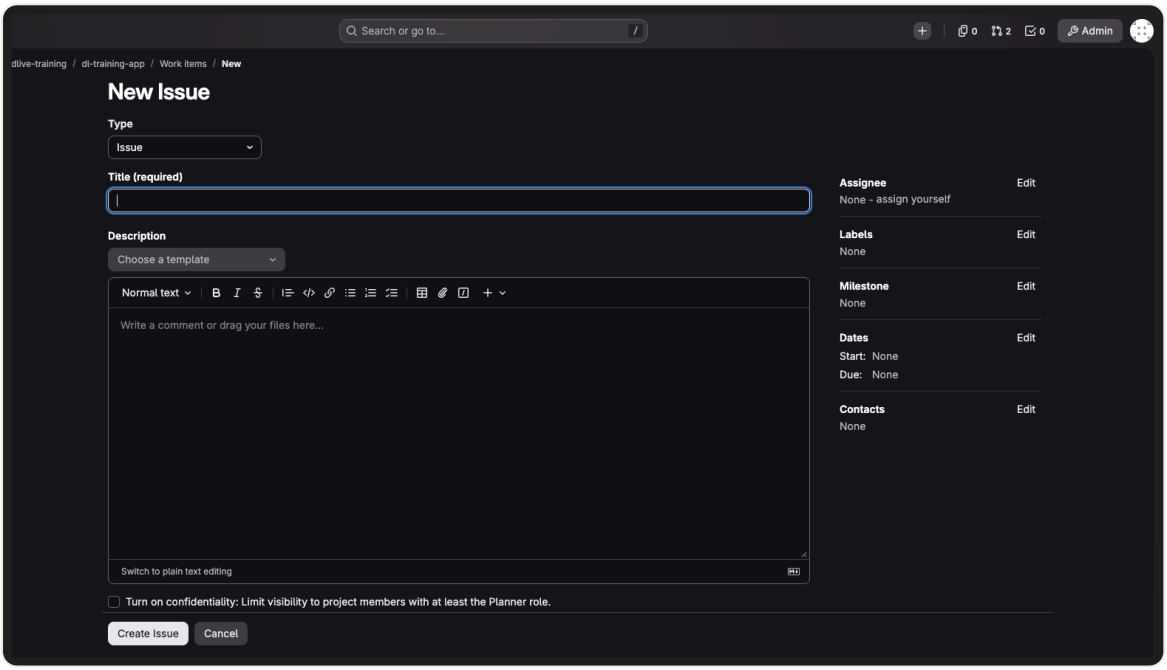
2. 各行に3軸を付けます。付けられない行は、その場でファイルを開いて確かめてください。

3. 3軸から分類を決めます。`date_from` と `date_to` の指摘は Issue の範囲外なので将来課題です。GitLab で Issue を1本切り、番号を控えてください。切り方は、ブラウザで **Plan** > **Issues** を開き、右上の **New issue** を押します。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、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つを書いてから **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。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。

4 「即修正」に入れたものだけを直します。1件1コミットです。直したら同じレビュー(その指摘を出した読ませ方)をもう一度走らせ、その指摘が消えたことだけを確認してください。新しく出てきた P2 以下は追いません。コミットは Step 3 と同じ形で、2つ目のターミナルから打ちます。ここでは push しません。push は Step 6 の手順1でまとめて行います。

```
git add app/controllers/estimate_ctl.php
git commit -m "レビュー指摘: [直した内容を1行で]"
```



手順3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。この画面では空のまま撮っています

> **Tips：**「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。

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

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]

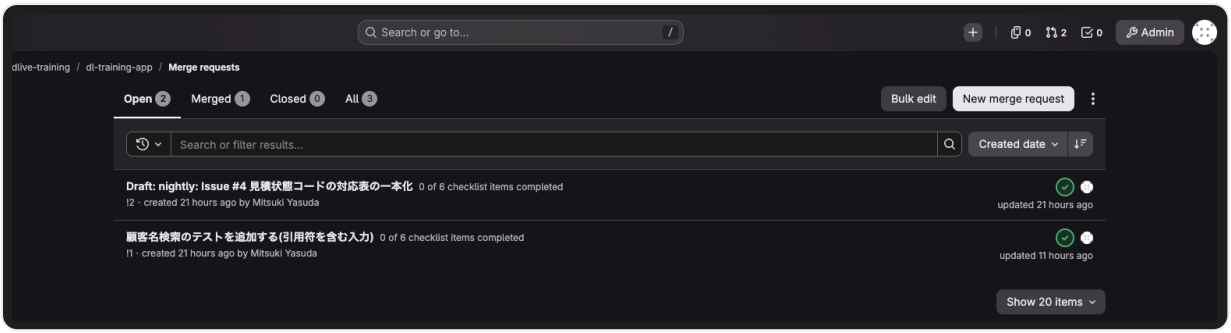
MR を出すと、push のたびに動いていた lint と test の4ジョブに加えて、MR 用のパイプラインで AI レビューの2ジョブが動きます。パイプラインは2本に分かれますが、どちらも MR の **Pipelines** タブから開けます。合流前の最終判定をここで受けます。

1. Step 5 の修正を2つ目のターミナルから `git push` します。オプションは付けません。付け直すと MR が二重に作られます。即修正が0件で新しいコミットが無い場合は push せず、Step 3 の1本目の push で作った MR をそのまま開いてください。push したかどうかに迷ったら `git status` を打ち、`Your branch is up to date with 'origin/worktree-issue-1'` と出れば送り終わっています。

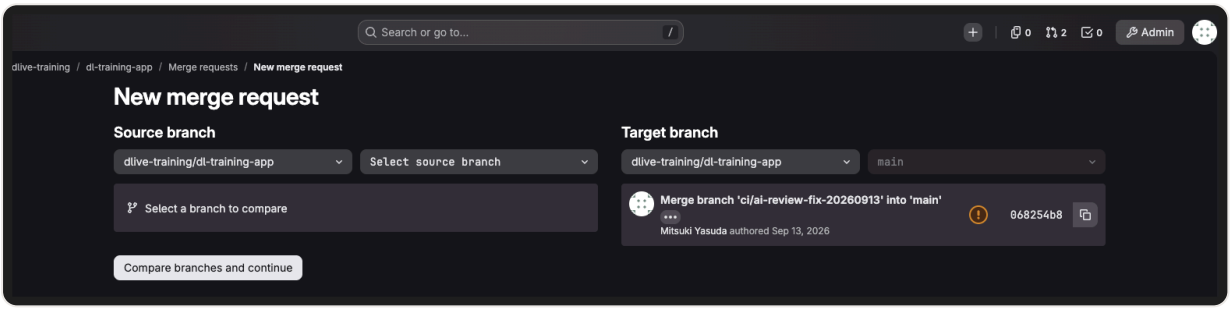
2. Step 3 で控えた MR の URL をブラウザで開いてください。手元に残っていない場合は、GitLab の左メニューの **Code** > **Merge requests** の一覧から、ソースブランチが `worktree-issue-1` でご自身の名前が付いた MR を開きます。一覧に無い場合は右上の **New merge request** で、Source branch を `worktree-issue-1`、Target branch を `main` にして **Compare branches and continue** を押し、次の画面で **Create merge request** を押してください。MR のタイトルは push option で作った場合ブランチ名から付くので、右上の **Edit** を押して「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。

3. 同じ **Edit** 画面の Description にテンプレートの6節が入っています。空のままなら、Description の上にある **Choose a template** から `Default` を選ぶと入ります。6節を埋めます。「変更概要」に何をなぜ変えたかを2行と `Closes #1` (演習中はマージしないので閉じません)、「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、「Codex レビュー結果」に3通り目の出力(お持ちでない方は `Codex なし`)、「人が判断した採否」に Step 5 の仕分けです。書き終えたら画面下の **Save changes** を押します。**Preview** タブで表の罫線が崩れていないかを見てから保存すると、やり直しが減ります。

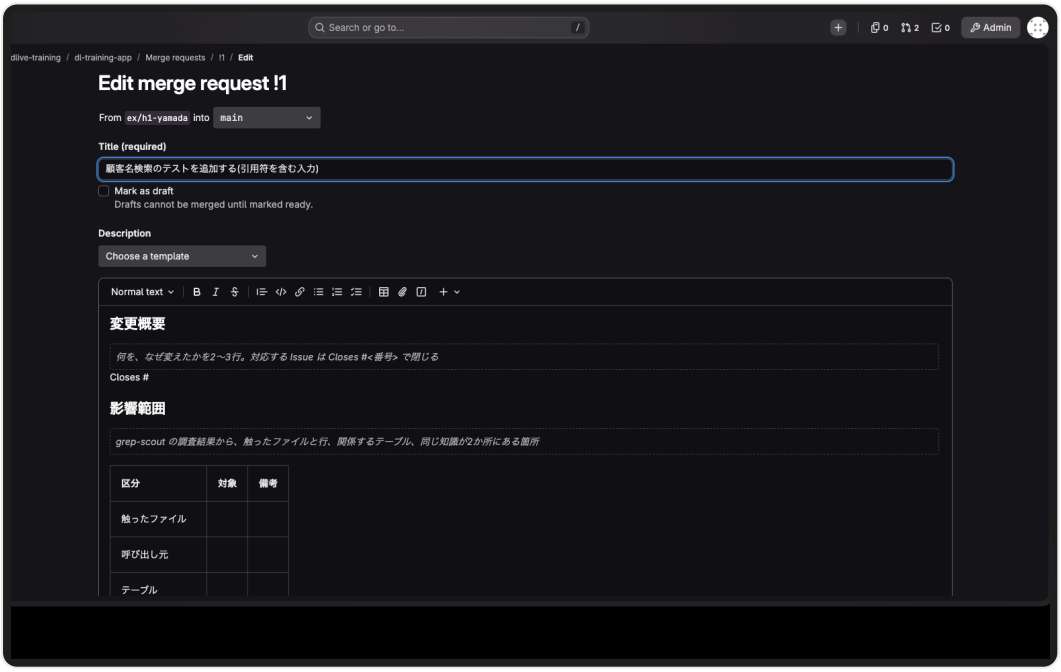
4. MR のページ上部の **Pipelines** タブを開きます。行が2本ずつ並び、`merge request` のラベルが付いた行が MR 用(`ai_review` と `ai_gate`)、`branch` のラベルが付いた行がブランチ用(`lint-php83` 、 `lint-php54` 、 `test` 、 `test-phpunit`)です。いちばん上の2本が最新で、それより下は過去の push の分です。最新の2本で、合わせて6つのジョブの結果を確認してください。`nightly_implement` は手動ジョブなので数えません。押さないでください。



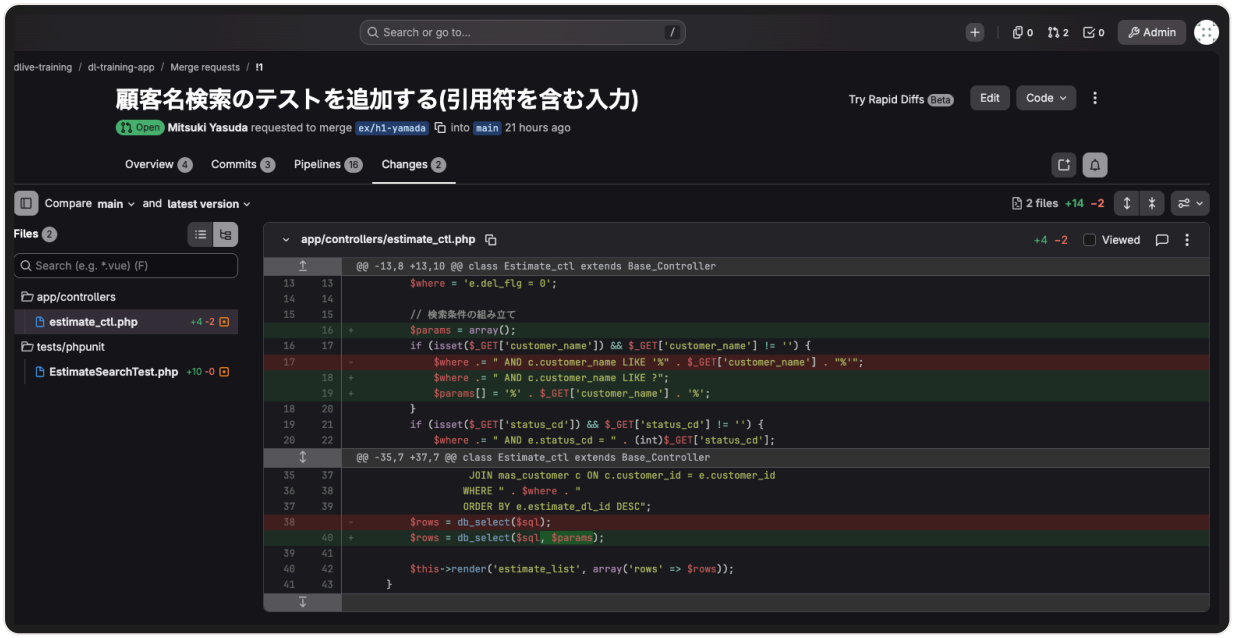
手順2で開く **Code** > **Merge requests** の一覧です。**Open** タブに、ソースブランチが自分のもので作成者が自分の名前になっている行を探します。行の左下に `!1` のような MR の番号が出ます。見当たらない場合は右上の **New merge request** を押します。この画面は講師環境のもので、並ぶ MR は皆さんの一覧と異なります



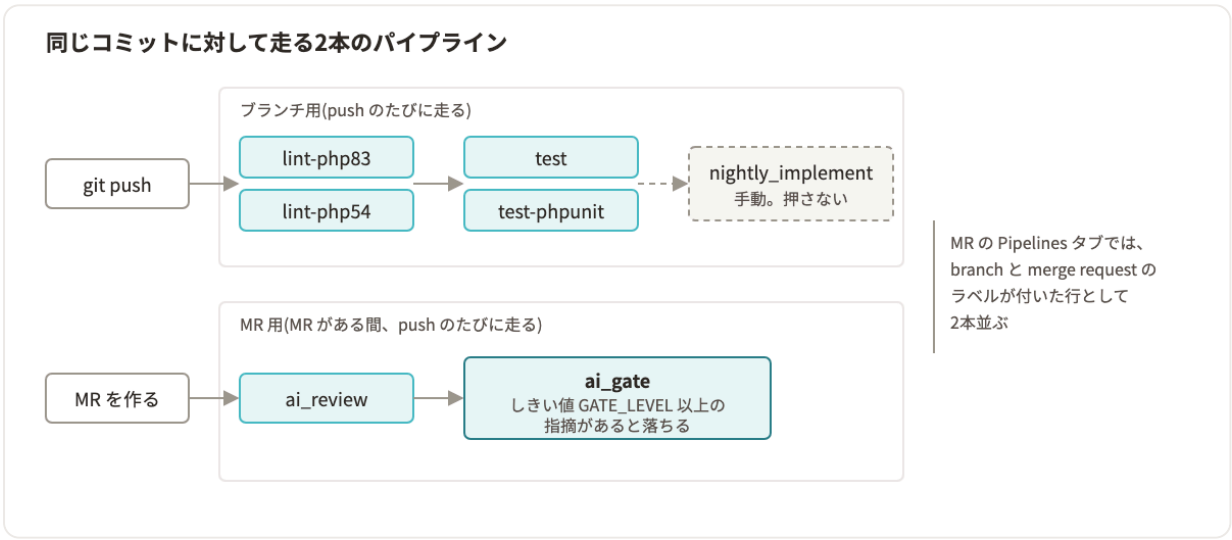
一覧に無い場合に **New merge request** を押すと開く画面です。左の **Source branch** の **Select source branch** で自分のブランチを選び、右の **Target branch** が `main` になっていることを見てから、左下の **Compare branches and continue** を押します。次の画面で **Create merge request** を押すと MR ができます



手順2から3で使う **Edit** の画面です。上の **Title (required)** を書き換え、**Description** の上にある **Choose a template** から `Default` を選ぶと、本文に「変更概要」「影響範囲」から始まる6節の見出しと表が入ります。斜体の1行が各節に何を書くかの案内です。この画面をいちばん下まで下りると **Save changes** があります



MR の **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。Step 4 で読ませた `review-input.diff` と同じ内容がここに出ているかを見てください

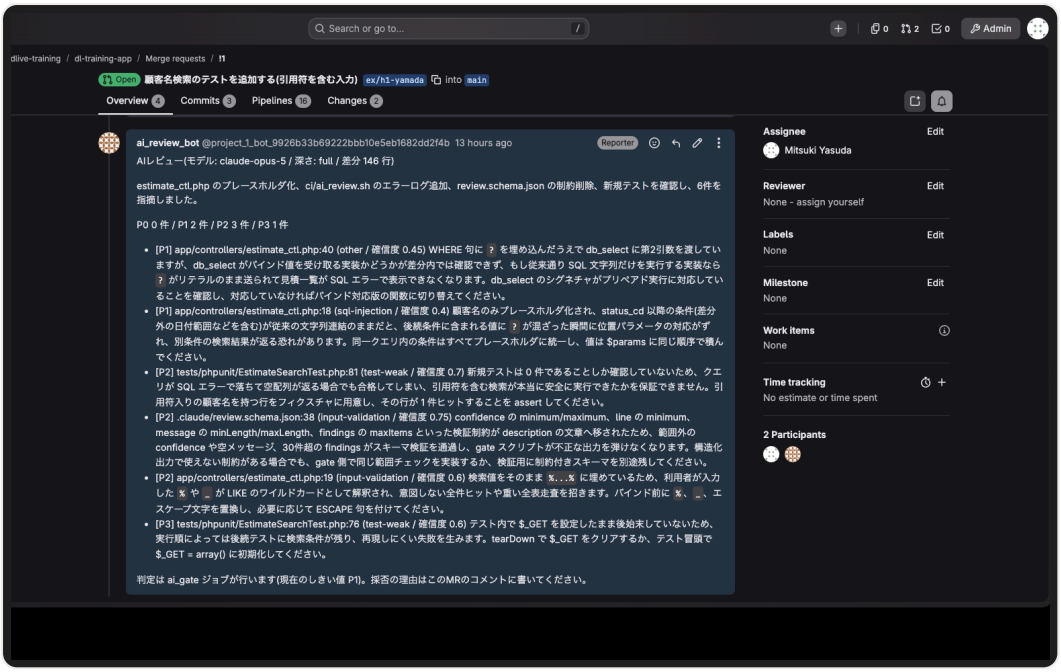


手順4で見る2本の関係です。上のブランチ用は `git push` のたびに走り、`lint-php83` と `lint-php54` のあとに `test` と `test-phpunit` が動きます。点線の先の `nightly_implement` は手動なので押しません。下の MR 用は MR がある間の push のたびに走り、`ai_review` のあとに `ai_gate` が `GATE_LEVEL` 以上の指摘があると落ちます。MR の **Pipelines** タブでは **branch** と **merge request** のラベルが付いた行として2本並びます

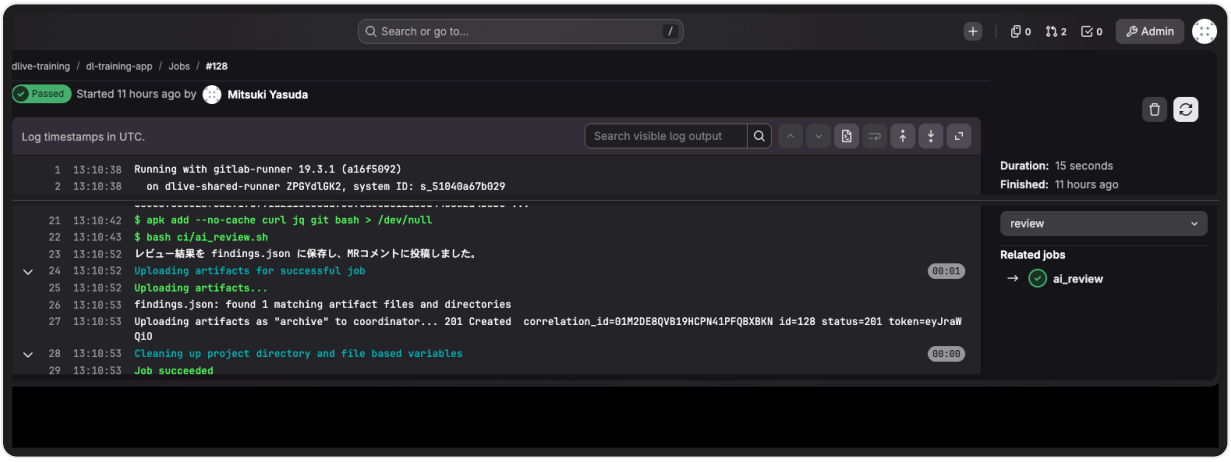
ジョブ	見ているもの	落ちたときにすること
`lint-php83`	現行環境(PHP 8.3)で構文が通るか	エラー行を直す
`lint-php54`	実環境に近い PHP 5.6 で `php -l` が通るか。落ちるのは PHP 7 以降	落ちた構文を記録してから、`CLAUDE.md` の表の書き方に直す

	に入った構文だけで、`::class` や可変長引数は通り抜ける	
`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`	PHPUnit 4.8 の機能テスト。実環境と同じ PHP 5.6 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です。CI 変数 `ANTHROPIC_API_KEY` が入っていないときは落ちずに緑で終わり、ログに「`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`」と出ます。この場合 MR にコメントは付かず、`ai_gate` も「指摘なし」で緑になります。どちらの場合も Step 4 の3通りの結果を MR に貼り、表の4行目には「未実施」と書きます
`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

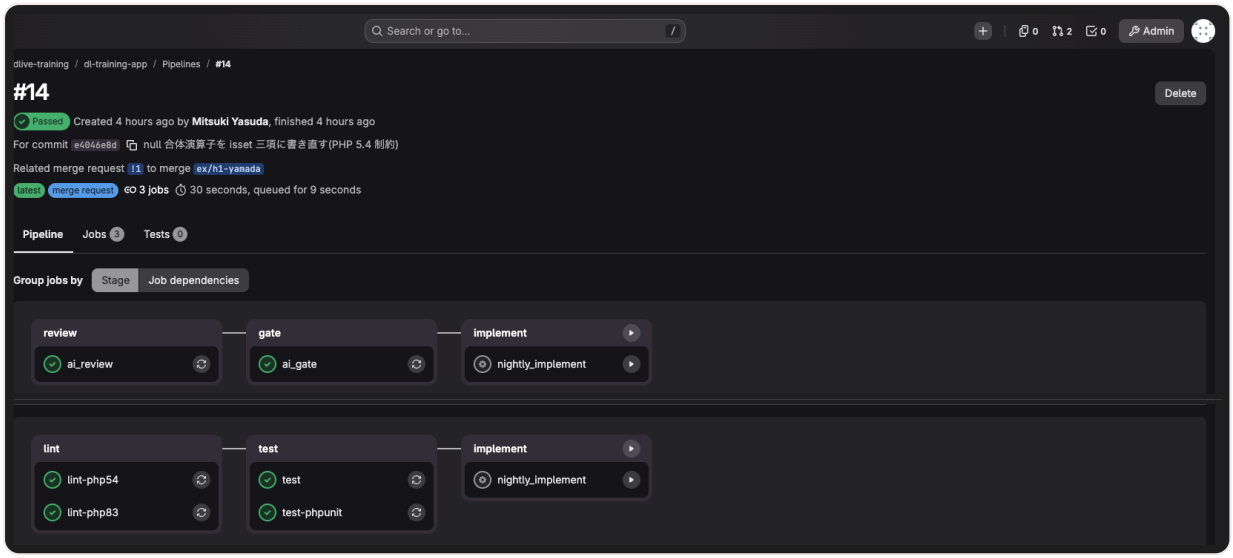
`ai_review` が動いた場合は、MR の **Overview** タブの下のほうに「AIレビュー(モデル: … / 深さ: … / 差分 … 行)」で始まるコメントが1本付きます。見出し行に P0 から P3 の件数が出るので、これを表の4行目に写します。所要時間はジョブの画面右側にある Duration を使ってください。コメントが付いていなければ、`ai_review` のログの1行目を見て、未設定の文言が出ているかどうかを確かめます。



手順4のあとに **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。箇条書きの各行はファイル名と行番号、分類と確信度、指摘の順です。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。その下の `Uploading artifacts` の行で `findings.json` が保存され、最終行が `Job succeeded` で緑です



> 手順4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。どちらもいちばん右に手動の `nightly_implement` が付いていますが、押しません。この例では `ai_gate` も緑です

`ai_gate` の出力はこう見えます。`allow_failure` が付いているため、赤くてもマージは止まりません。

```
しきい値 P1 以上、確信度 0.0 以上の指摘を数えます。
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。
```

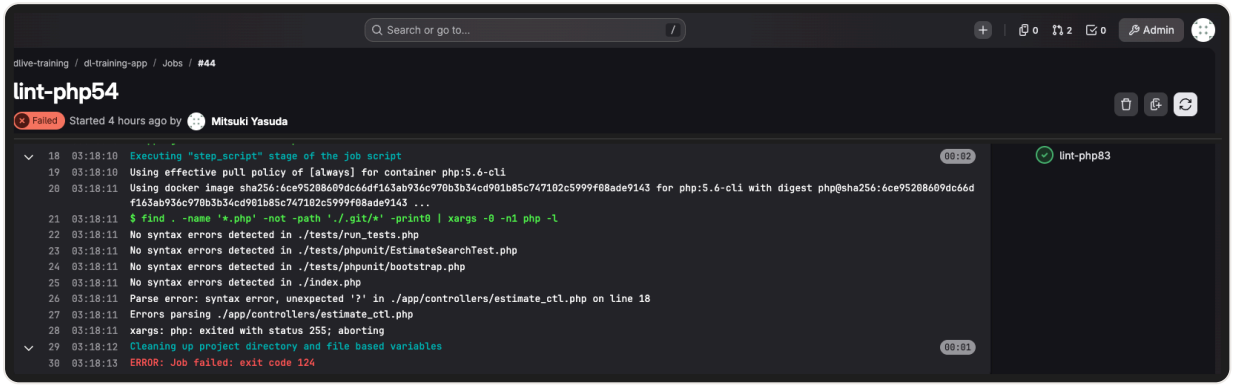
ここで一度立ち止まってください。`ai_gate` は `allow_failure: true` なので、赤くても **Merge** ボタンはブロックされません。ゲートが赤いのにマージできる状態は、ゲートではありません。列挙しているだけです。この1行を外して本当のゲートにするのが Day2 のハンズオン2です。今日は「止まらないことを見る」までにします。ボタンは押しません。

`lint-php54` が落ちた方は記録してください。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。出力はこう見えます。

```
PHP Parse error:  syntax error, unexpected '?' in /builds/dlive-training/dl-training-app/app/controllers/estimate_ctl.php on line 18
Errors parsing /builds/dlive-training/dl-training-app/app/controllers/estimate_ctl.php
```

記録すること	書き方
落ちた行	ファイル名と行番号。ログの1行目をそのまま貼る

出たエラー文	`syntax error, unexpected` の後ろまで、そのまま
混入した構文	`??` 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



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

> **Tips：** `lint-php54` は PHP 5.6 のイメージで `php -l` を回しています。落ちるのは PHP 7 以降に入った構文だけです。 `...` 可変長引数、 `::class` 、 `finally` 、ジェネレータ、 `const` への配列代入は 5.5 から 5.6 で入ったので、このジョブを通り抜けます。これらを捕まえるのは `phpcs` の側で、Day2 でフックに組み込みます。

Codex の場合

MR の作り方と CI の読み方は同じです。「Codex レビュー結果」欄に、Step 4 の3通り目の出力を貼ってください。

Codex をお持ちでない方は、この欄に `Codex なし` と書き、 `review-other` の結果を「AI レビュー結果」表の2行目に入れてください。欄を空のままにしないでください。

この Step が終わった状態

- [ ] MR が1本あり、 `lint-php83` 、 `lint-php54` 、 `test` 、 `test-phpunit` の4つが緑
- [ ] MR 説明の6節が埋まっている。「Codex レビュー結果」を空のままにしていない。「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-php54` が落ちた場合、4項目の記録が残っている

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。今回の改修は3軸すべてが軽い側なので、軽量パスになります。

1. `/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(定型の依頼文)です。この skill は Issue の本文を GitLab から取りに行きます。業務PCに `glab` もアクセストークンも入っていないので、先に 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てください。そのうえで Claude Code の入力欄に次を打ちます。

   ```
   /shijisho 1
   ```

   取得に失敗した旨が返ったら、続けて「Issue 本文は次のとおりです」と書き、コピーした本文をそのまま貼り付けて Enter を押します。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。

2. worktree の中に `docs/shijisho/issue-1.md` ができます(VSCode のファイル一覧の `.claude` > `worktrees` > `issue-1` > `docs` の下)。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てください。

3. 「目的」と「戻し方」の2つを、自分の言葉に直してください。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。直したら、2つ目のターミナルからコミットして push します。worktree を片付けたときに消えないよう、MR に載せておきます。

   ```
   git add docs/shijisho/issue-1.md
   git commit -m "Issue #1 の作業指示書を足す"
   git push
   ```

Codex の場合

Codex には skill がありません。同じ5項目を1回の非対話実行で作ります。

```
codex exec -s read-only "Issue #1 の作業指示書を 目的・影響範囲・変更方針・検証・戻し方 の5項目で書く。対象は main との差分の範囲だけ。出力は docs/shijisho/issue-1.md に貼る前提の Markdown"
```

出力を `docs/shijisho/issue-1.md` として保存します。手順2と手順3は同じです。

> **Tips：**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。

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

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。運用する人が1本を選べる状態にしておくほうが、2本置いて使われないより先に進みます。

選ぶもの	向いている場合	入れるファイル
Codex を呼ぶスキル	Codex を持っていて、呼び方を標準化したい	`.claude/skills/codex-review/SKILL.md`
別コンテキストのサブエージェント	Codex を持っていない、または端末ごとの環境差を減らしたい	`.claude/agents/review-other.md`

1. 2つのファイルを開いて、どちらを自社に置くか決めてください。決め手は Codex の有無です。
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. 写した後で、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てください。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本を記入してください。

ファイル	D2 での置き場所	smart3pm での置き場所

`.claude/skills/codex-review/SKILL.md`	既存のスキル群と同じ階層。本文のコマンドは行継続(`` ` ``)と `(Get-Content REVIEW.md -Raw)` の2か所を PowerShell の書法へ直す	同じ階層に同じ名前で置く。コマンドはそのまま使える
`.claude/agents/review-other.md`	既存のエージェント定義と同じ階層。`model:` は実装側と変える	同じ階層に同じ名前で置く
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「Codex 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。`baseRef` をどちらにするかも1行で書く	規約本体の該当節に追記

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。

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

OK 基準 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-php83` / `lint-php54` / `test` / `test-phpunit` の4つが緑になっている

- [ ] `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(Pipelines の一覧で、赤い × の付いた `worktree-issue-1` の行が1本残っている)
- [ ] 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 CI 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

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
履歴を切ったセッションだけが出した指摘は何か	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 と自分の仮説の差はどこから出たか	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、機械で判定できたものは何割か	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4行目として足します。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. `codex review` を `--background` で回し、手を止めずに次の作業を進めた場合の所要時間を測ってください。待つ形と比べて、自社の運用でどちらを標準にするかを決める材料になります。

4. worktree を `remove` で片付け、ブランチが一緒に消えることを確認してください。`keep` を選んだ場合にどこに残るかも見ておくと、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 5.4 制約の機械検査

D2-02 ／ 所要 [20min]

Day2　**?? を書かせようとしても isset 三項に戻る状態**

目的

CLAUDE.md に PHP 5.4 の制約を書いても、数ターン進むと `??` 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。この演習では同じ制約を PostToolUse フックへ移し、書き換えられた瞬間に差し戻される状態を作ります。CLAUDE.md だけの時との差を、自分の画面で1回見てください。
PostToolUse は、Claude Code が 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。スクリプトが終了コード 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 で登録した PreToolUse(実行の直前に走る側)と同じ場所に並べます。

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

> **Tips：**D2-01 の言い方に合わせると、 [A] だけで守っているものを [M] へ移す作業にあたります。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。

この演習で触るもの

触るファイル	`CLAUDE.md` の制約表(Codex は `AGENTS.md`)、 `.claude/settings.json` の `PostToolUse` 、 `.claude/hooks/check-php54.ps1` 、 `app/libraries/util.php` の `h()` 、 `hooks/tests/fixtures/ng_spaceship.php.txt` 、 `cases.json`
操作	フック無しで `??` を書かせて通ることを見る。PostToolUse に登録して同じ依頼を出し、差し戻しと書き直しを見る。検査自体にテストを1件足して MR を出す
見る場所	差分の `??` 、フックの標準エラー4行、ランナーの19件、MR の `lint-php54`
達成の状態	差し戻しの後、人が指示を足さずに `isset` 三項へ書き直された。ランナー19件が通る
自社での使いどころ	実環境で落ちる構文を、CLAUDE.md の文章からフックの機械検査へ移す。自社の「本番で落ちる書き方」3つが対象
影響が及ぶ範囲	matcher `Edit|Write` は Bash 経由の書き換えを拾わない。最終判定は CI の `lint-php54` 。 `.ps1` は ASCII のみ

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。持ち込んだ自社の CLAUDE.md がある方は、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。「`??` は使わない」ではなく「`isset($a) ? $a : $b` と書く」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、フックではなく人が見る側に残ります。その理由も1行書きます。

> **Tips：**丸が付かないものが1つも無いときは、粒度が粗い可能性があります。「性能に配慮する」のような書き方は検査できません。検査できる形まで割ってください。

操作

Step 0　代替の書き方を1行足す　[2min]

`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。禁止の列挙では

足しません。表に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。
Codex を使う方は `AGENTS.md` の同じ表にも同じ1行を入れます。

Step 1　フックが無い状態で頼む　[3min]

1. VSCode で `dl-training-app` を開き、ターミナル(PowerShell)で main を最新にしてから作業用のブランチを作ってください。ブランチ名は共通の進め方 STEP 1 の形で、`yamada` の部分をご自身のユーザー名に置き換えます。

   ```
   git switch main
   git pull
   git switch -c ex/p4-yamada
   git branch --show-current
   ```

2. 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。プチ演習3 を終えた方は `"PreToolUse"` が1つ入っている状態、プチ演習3 を終えていない方は `"hooks": {}` のままです。どちらでもこの演習は進められます。

3. 同じターミナルで `claude` を起動し、次をそのまま送ってください。

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

4. 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。差分に `??` が入り、警告もエラーも出ずに編集が終わります。`h()` は `util.php` の冒頭にある3行の関数で、書き換わる行はこの1行です。

```
- return htmlspecialchars((string)$s, ENT_QUOTES, 'UTF-8');
+ return htmlspecialchars($s ?? '', ENT_QUOTES, 'UTF-8');
```

VSCode の左側の「ソース管理」を開くか、ターミナルで `git diff app/libraries/util.php` を打つと、この差分が見えます。

> **Tips：**1回目で断られた方は、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。

Step 2　PostToolUse に登録する　[3min]

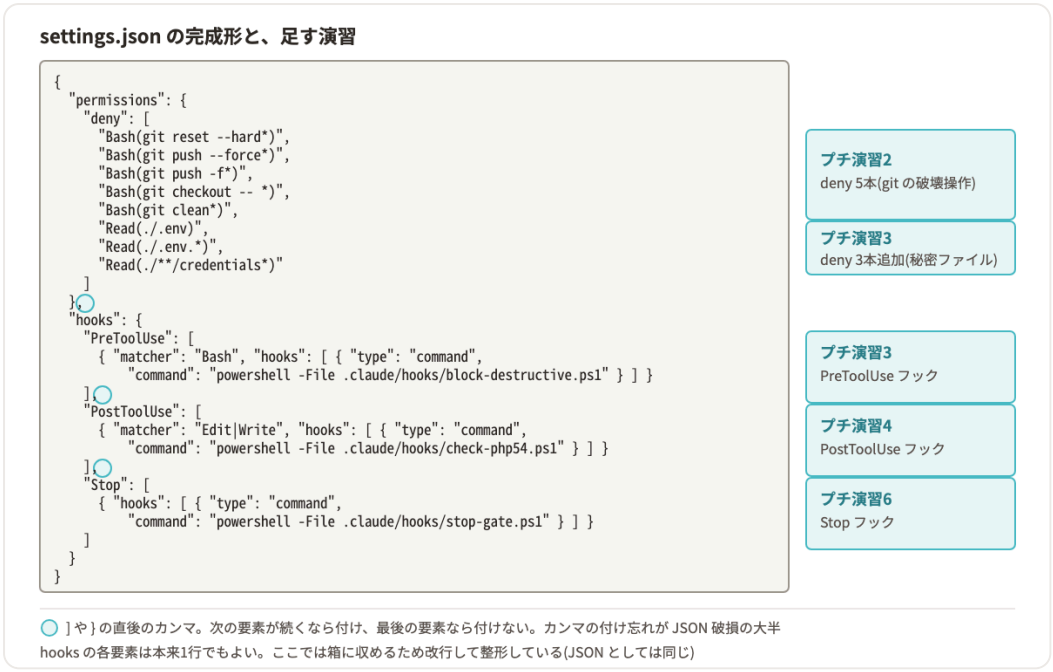
1. Claude Code を終了し(入力欄で `/exit`)、VSCode で `.claude/settings.json` の `"hooks"` に次のブロックを足してください。

```
"PostToolUse": [
  {
    "matcher": "Edit|Write",
    "hooks": [
      {
        "type": "command",
        "command": "powershell -NoProfile -ExecutionPolicy Bypass -File .claude/hooks/c
heck-php54.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。保存後の `"hooks"` の形はこうなります(中身は省略)。

```
"hooks": {
  "PreToolUse": [ ... ],
  "PostToolUse": [ ... ]
}
```

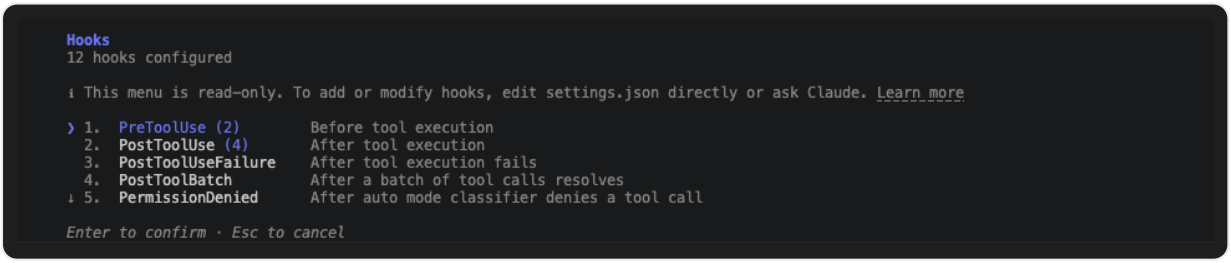
`"hooks": {}` のままだった方は、`{}` の中へ上のブロックをそのまま貼ります。カンマは要りません。`"matcher": "Edit|Write"` は「Edit と Write の2つのツールの後だけ走らせる」という意味です。



`.claude/settings.json` の完成形と、各演習で足す範囲です。`permissions.deny` の上5本がプチ演習2、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 です。丸で囲んだ3か所が `]` 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。今回足すのは PostToolUse のブロックなので、その直前の `],` のカンマを忘れないでください。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

> **Mac または bash を正にする方：** `"command"` の値だけを `bash .claude/hooks/check-php54.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。入れ方は「困ったとき」の jq の項にあります。

2. 保存したら `claude` を起動し直し、`/hooks` を打って PostToolUse に1本入っていることを確認してください。`/hooks` は読むだけの画面です。ここで編集はしません。Esc で閉じます。



Step 2 の手順2 で開く `/hooks` の画面です。イベント名の右の括弧が、そのイベントに登録されている本数です。先頭の `12 hooks configured` は全イベントの合計で、上の行の「This menu is read-only」のとおり、ここでは編集できません。画面は講師環境で撮ったもので、件数はご自身の環境と違います。この演習を終えた直後は PreToolUse が (1)、PostToolUse が (1) です

登録したのに動かないとき `settings.json` 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 で開いたときに赤い波線が出ていないこと、`/hooks` の一覧に PostToolUse が出ていることの2つを見てください。波線が無いのに `/hooks` に出ない場合は、`settings.json` を保存する前に起動したセッションが残っていま

す。`/exit` で終了してから起動し直してください。それでも動かないときは `claude --debug` で起動すると、フックの発火ログが出ます。

Step 3　同じ依頼をもう一度出す　[3min]

1. 先に Step 1 の変更を戻します。ターミナルで `git restore app/libraries/util.php` を打ち、`git diff` が何も出さないことを確認してください。戻さないと `??` が既に入っているため、Claude が編集せずに「対応済みです」と答えることがあります。

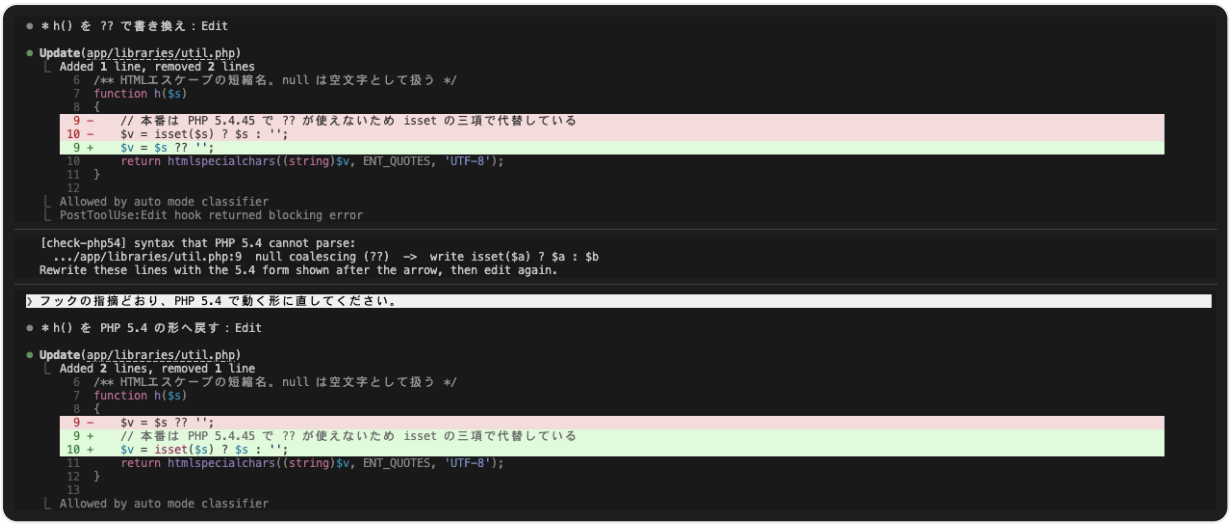
2. Step 1 と同じ文をもう一度送ってください。文面は変えません。

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。Claude Code の画面では、Edit ツールの結果の下にフックが終了コード 2 で止めた旨と、次の4行が出ます。2行目の行番号は `??` が入った行です。

```
[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.
Short array syntax [] is fine on 5.4 and is not reported here.
```

4. そのまま待つと、モデルが `isset($a) ? $a : $b` 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 `git diff app/libraries/util.php` を打ち、差分に `??` が残っていないことを見ます。

> **Tips：**フックは編集の直後にしか走りません。既に `??` が入っているファイルを開いただけでは何も起きないので、「登録したのに反応しない」と感じたら、まず編集が本当に行われたか(Edit ツールの表示が出たか)を見てください。



> Step 3 の画面です。上半分が `??` 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-php54]` の3行）、下半分が `isset` 三項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「フックの指摘どおり、PHP 5.4 で動く形に直してください」と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

Codex の場合

Codex も同じイベント名のフックを `~/.codex/hooks.json` または `.codex/hooks.json` に持ちます。呼ぶスクリプトは bash 版に差し替えてください。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "bash .claude/hooks/check-php54.sh" }
        ]
      }
    ]
  }
}
```

あわせて `AGENTS.md` の「PHP 5.4 制約」の表を、CLAUDE.md と同じ文面に保ってください。片方だけ直すと、ツールによって出来上がりが変わります。

> **Codex 側の注意**　リポジトリ配下の `.codex/` に置いた設定が対話セッションで発火しない既知の不具合が報告されています。発火しない場合は `~/.codex/hooks.json` に置いて確認してください。キーの並びは Claude 側と同形ですが、細かな書式は当日の講師端末で確認します（要確認）。

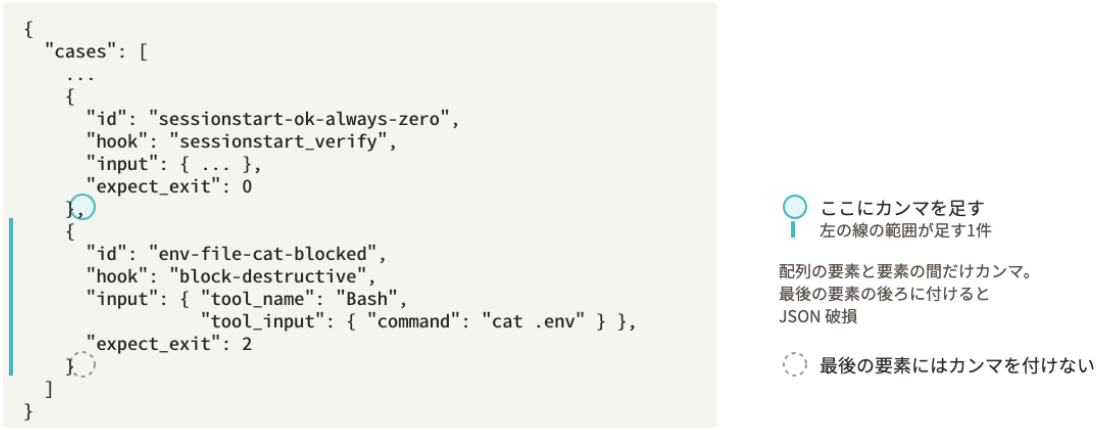
Step 4　検査そのものにテストを足す　[3min]

1. VSCode で `.claude/hooks/tests/fixtures/` に、宇宙船演算子 `<=>` を1行だけ含む見本を `ng_spaceship.php.txt` という名前で作ってください。隣にある `ng_null_coalesce.php.txt` を開くと形が分かります。1行目が `<?php` 、2行目以降に ASCII だけで数行、そのうち1行に `<=>` を使います。日本語は入れません(`.ps1` 側が CP932 で読み違えて誤検知する原因になります)。拡張子が `.php.txt` なのは、CI の lint ジョブがツリー内の `*.php` を全部 `php -l` に掛けるためです。

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

```
{
  "id": "php54-ng-spaceship",
  "hook": "check-php54",
  "fixture": "ng_spaceship.php.txt",
  "input": { "tool_name": "Edit", "tool_input": { "file_path": "__FIXTURE__" } },
  "expect_exit": 2,
  "note": "spaceship operator は 7.0 から。5.4 では構文エラーになる"
}
```

cases.json の配列末尾に1件足す位置



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. `cases.json` を足した直後に、追加した `}` の手前の要素の末尾へ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ます。最後の要素の後ろにはカンマを付けません。

4. Claude Code を起動せずに、VSCode のターミナルでテストを回してください。

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

こう見えていれば正しい状態です(抜粋。実際は全ケースが1行ずつ並びます)。雛形の17件に、プチ演習3 で足した1件と今回の1件が乗るので、末尾の合計が19件になります。プチ演習3 のケースを足していない方は18件です。

```
PASS  php54-ok-5-4-style  (exit 0)
PASS  php54-ng-null-coalesce  (exit 2)
PASS  php54-ng-scalar-type  (exit 2)
PASS  php54-ng-spaceship  (exit 2)
PASS  php54-ok-not-php  (exit 0)

19 cases: 19 passed, 0 failed
```

失敗したときは、その行が `FAIL php54-ng-spaceship (expected 2, got 0)` の形になり、末尾が `18 passed, 1 failed` になります。`got 0` は、見本に `<=>` が入っていないか、コメント行の中にだけ書いてしまった(コメントは検査の前に取り除かれます)ことを示します。`fixture not found` と出たら、ファイル名の綴りが `cases.json` と合っていません。

Mac、bash を正にする方、Codex の場合　同じケースを bash 版のランナーで回します。`bash .claude/hooks/tests/run_cases.sh` です。判定の中身は同じで、書き方だけが違います。出力の形も同じです。

Step 5　MR を出す　[3min]

1. 先に `git status` で変更されたファイルを確かめてください。`app/libraries/util.php` は Step 3 で `isset` 三項に戻った形のはずです。`??` のまま残っていたら、コミットに入れずに `git restore app/libraries/util.php` で戻します。

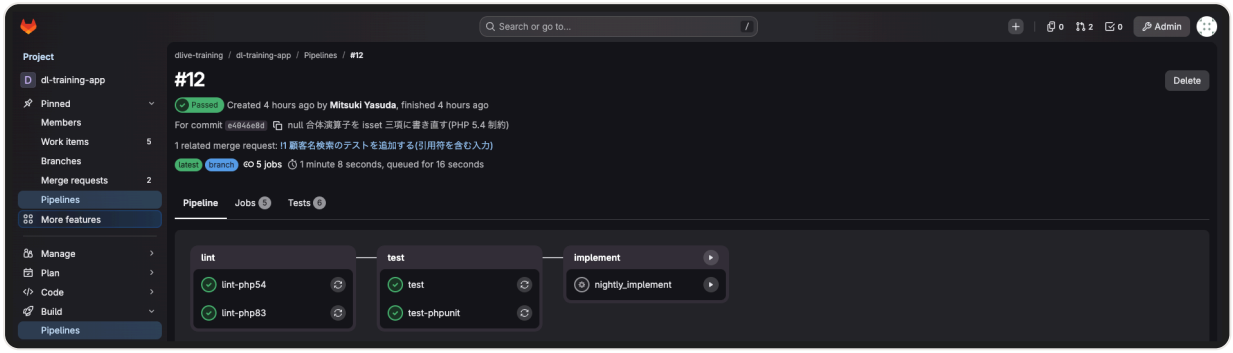
2. 1ステップ1コミットで、3回に分けてコミットします。Step 0 の制約節、フックの登録、テストの追加の順です。ファイル名を指定して `git add` し、`git add .` は使いません。

   ```
   git add CLAUDE.md
   git commit -m "PHP 5.4 制約の表に代替の書き方を1行足す"
   git add .claude/settings.json
   git commit -m "PostToolUse に check-php54 を登録する"
   git add .claude/hooks/tests/cases.json .claude/hooks/tests/fixtures/ng_spacesh
   ip.php.txt
   git commit -m "check-php54 に宇宙船演算子のテストケースを足す"
   ```

3. push と同時に MR を作ります。共通の進め方 STEP 3 と同じコマンドです。出力に `View merge request for` の行と URL が出たら、ブラウザで開いてください。

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

4 ブラウザで MR の **Pipelines** タブを開き、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。`lint-php54` はブランチ用のパイプライン(**branch** のラベルの行)に入っています。結果はプチ演習5 の途中で見に戻ってください。18名が同時に push すると、ジョブは `pending` のまま順番待ちに入ります。これは異常ではありません。「困ったとき」の pending の項に待ち方があります。



Step 5 で見る画面。**lint-php54** の緑は、5.5 以降の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

生成物の名前と場所

この演習で手元に残るのは、ファイル3つと MR 1本です。自社ハーネスへ持ち帰るときの置き場所を、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-php54.ps1` と `check-php54.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（と Codex を使う方は AGENTS.md の同じ表）	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

OK 基準

この演習が終わった状態

- [ ] フックを登録する前は `??` 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 `isset` 三項へ書き直された

- [ ] ケースを1件足した状態で、テストランナーが19件すべて通る
- [ ] MR の `lint-php54` の結果を、プチ演習5 の途中で見に戻って確認した

追加と考察

時間が余った方は、次を試してから台帳に1行書いてください。

1. 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。

2. この抜け道を塞ぐなら、どの層に置くかを決めてください。選択肢は Stop フックで作業ツリーを走査する、`FileChanged` を使う、CI の `lint-php54` に任せる、の3つです。どれを選んだかと理由を台帳に書きます。

> **Tips：**3つ目を選ぶのは手抜きではありません。手元の速報と合流前の最終判定は役割が違います。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。編集した瞬間に差し戻す形へ移してください。ASCII だけでできているかを検査する処理を、同じ PostToolUse に足します。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。

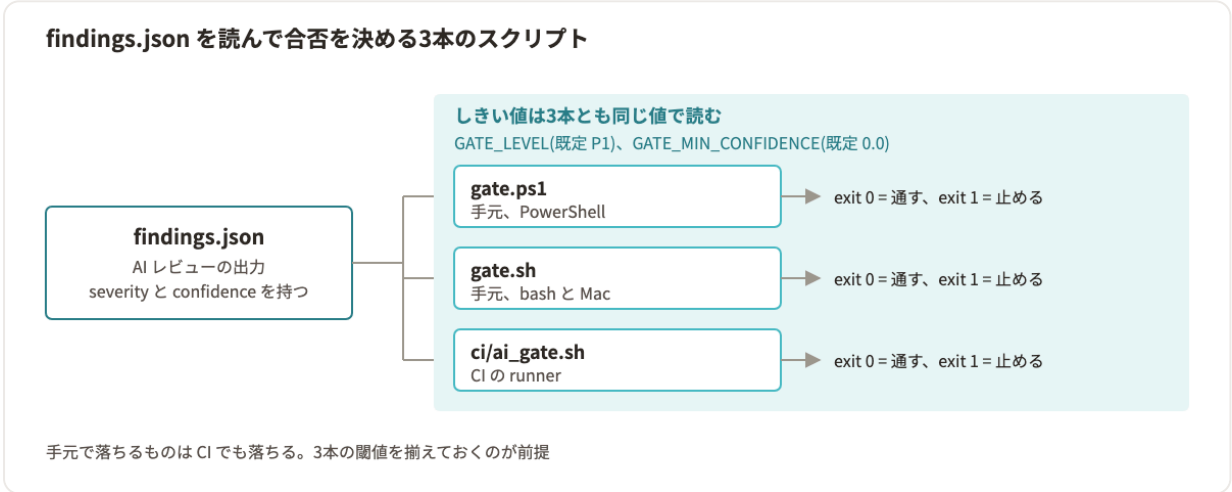
プチ演習5　レビュー出力の JSON ゲート

D2-04 ／ 所要 [15min]

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

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。この演習では、レビュー結果を `findings.json` として受け取り、重大度で合否を返すスクリプトを完成させます。作ったものは D2-05 でそのまま CI のゲートジョブになります。
判定の置き場所は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` がこの演習で完成させる手元の版、`ci/gate.sh` がその bash 版です。手元で落ちたものが CI でも落ちる状態にしておきます。



> 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。3本とも `GATE_LEVEL` (既定 P1)と `GATE_MIN_CONFIDENCE` (既定 0.0)を同じ値で読み、exit 0 で通し、exit 1 で止めます。この演習で書き足すのは真ん中の2本のうち、ご自身の環境に合う1本です

この演習で触るもの

触るファイル	`findings.json` (自分で作る)、`ci/gate.ps1` (bash は `ci/gate.sh`)、`.claude/review.schema.json` 、`REVIEW.md`
操作	差分を JSON でレビューさせ、判定スクリプトを既定の閾値で回し、閾値を P2 に下げ、確信度の下限を引数として実装する
見る場所	`gate:` で始まる出力行、`$LASTEXITCODE` 、`blocking` の件数、`excluded by confidence` の行

達成の状態	閾値 P1 と P2 で `blocking` が変わることを見た。下限を上げると数から外れる指摘が1件ある
自社での使いどころ	ハンズオン2の CI ゲート `ci/ai_gate.sh` と同じ判定を手元で先に回す。MR を出す前の自己チェック
影響が及ぶ範囲	手元と CI で判定を割らない。未知の `severity` は落とす側に寄せたまま。PowerShell 5.1 の `>` は UTF-16 で書き出すので `Out-File -Encoding utf8` を使う

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` の定義では、P0 が動かないか壊す、P1 が実環境で落ちるか外部入力が素通し、P2 が指示との不一致と波及漏れ、P3 がそれ以外です。

2. 確信度の低い指摘を数に入れるかを決めてください。スキーマでは、経路の一部が読めていない指摘は 0.5 以下を付けることになっています。

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。数が出せない方は「止まってよい上限」を件数で書きます。

操作

Step 1　レビューを JSON で受け取る　[4min]

レビュー対象は、プチ演習4 で作ったブランチ(`ex/p4-...`)の差分です。そのブランチに乗ったまま進めます。プチ演習4 を終えていない方は、Day1 の `worktree-issue-1` ブランチの差分でも構いません。どちらを使ったかを台帳に書いてください。

JSON を作る経路は3つあります。先に自分の端末がどれかを決めてください。

端末の状態	使う手順
環境変数 `ANTHROPIC_API_KEY` がある(`start-claude.ps1` で起動している)	下の手順2 をそのまま
Claude Code にサブスクでログインしている(鍵は無い)	下の手順2 のコマンドから `--bare` を外す。CLAUDE.md とフックも読まれますが、この演習で見たいのは JSON の形なので支障ありません
Codex を使う	この Step の末尾「Codex の場合」

鍵も Codex も無い	この Step の末尾「鍵も Codex も無い場合」の見本ファイル

1. VSCode のターミナル(PowerShell)で、リポジトリのルート(`index.php` がある場所)にいることを確認してください。 `Get-Location` で今の場所が出ます。Claude Code の対話画面は使いません。1回だけ非対話で走らせる `-p` の形です。

2. 差分を標準入力から渡し、スキーマに沿った JSON を受け取ります。判定の基準は `REVIEW.md` をそのまま添えて渡します。1行で打ちます。終わるまで1分前後かかり、その間は何も表示されません。

```
git diff main | claude --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。" --append-system-prompt "$(Get-Content REVIEW.md -Raw)" --output-format json --json-schema .claude\review.schema.json | jq ".structured_output" | Out-File -Encoding utf8 findings.json
```

3. VSCode で `findings.json` を開き、 `summary` と `findings` の2つのキーがあることを確認してください。

> **Tips：**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。

> **`jq` が無い端末：** `jq ".structured_output"` の部分を PowerShell だけで置き換えられます。 `| ConvertFrom-Json | Select-Object -ExpandProperty structured_output | ConvertTo-Json -Depth 10 | Out-File -Encoding utf8 findings.json` とします。中身は同じです。

書き出しに `>` を使わないでください　PowerShell 5.1 の `>` は UTF-16 で書き出します。 `Out-File -Encoding utf8` を使ってください。 `>` で作った `findings.json` は、 `gate.ps1` が `gate: cannot parse JSON` で落とします。Step 2 の1行目で止まる原因の大半がこれです。

こう見えていれば正しい状態です。件数と中身は人によって違います。

```
{
  "summary": "設定ファイルとフックの登録を中心に4点を確認し、2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"
    }
  ]
}
```

`--bare` を使う理由と、その代償　`--bare` 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。レビュー側は履歴も文脈も切りたいので、ここでは飛ばす側が正です。ただし OAuth とキーチェーンも読まないため、環境変数 `ANTHROPIC_API_KEY` が必要です。サブスクでログインしている方は認証エラーになります。その場合は上の表のとおり `--bare` を外すか、Codex 側の手順へ進んでください。

鍵も Codex も無い場合

配布フォルダの `findings_sample.json` を `findings.json` という名前でリポジトリのルートへコピーし、Step 2 へ進んでください。中身は P0 1件・P1 1件・P2 2件で、うち1件の `confidence` が 0.5 です。Step 3 の閾値の切り替えと Step 4 の確信度の足切りは、この見本でも同じように確認できます。自分の差分でレビューを受けるところだけが後回しになります。コピーは PowerShell で次のとおりです(配布フォルダの場所はご自身の展開先に読み替えてください)。

```
Copy-Item C:\work\haifu\findings_sample.json .\findings.json
```

> **Tips：** `--json-schema` にファイルパスを渡す形が通らない場合は、中身を変数に入れて渡します。`$schema = Get-Content .claude\review.schema.json -Raw` としてから、コマンドの `--json-schema .claude\review.schema.json` を `--json-schema $schema` に置き換えてください。どちらの受け付け方が通るかは当日の講師端末で確定させます（要確認）。

Codex の場合

同じスキーマをそのまま使えます。出力先の指定まで1行で済みます。

```
codex exec -s read-only --output-schema .claude\review.schema.json -o findings.json "$(Get-Content REVIEW.md -Raw) git diff main の変更をレビューし、指摘をスキーマの形で返してください。"
```

Mac のターミナルでは `$(Get-Content REVIEW.md -Raw)` を `$(cat REVIEW.md)` に、パスの `\` を `/` に読み替えてください。

Codex は P2 と P3 も素直に出します。件数が Claude 側より多くなるのは、粒度の差であって品質の差ではありません。次の Step で閾値を動かすときに、この差が効いてきます。

Step 2　既定の閾値で回す　[3min]

1. 同じターミナルで、判定スクリプトを引数なしで実行してください。既定は `findings.json` と閾値 P1 です。

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

2. 続けて終了コードを見てください。`$LASTEXITCODE` は、直前に実行したプログラムの終了コードを PowerShell が入れておく変数です。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。

```
$LASTEXITCODE
```

Mac または bash を正にする方は、`bash ci/gate.sh` のあと `echo $?` で同じ数字が見えます。
P0 か P1 があるときは、こう見えます。終了コードは 1 です。

```
gate: threshold=P1 total=2 blocking=1
  P1 app/libraries/util.php:18 h() の引数が未定義のときの扱いが変わっています。
gate: blocked. fix the findings above or lower the threshold on purpose.
```

P2 以下しか無いときは、こう見えます。終了コードは 0 です。

```
gate: threshold=P1 total=2 blocking=0
gate: passed.
```

Step 2 の出力例です。Mac で `bash ci/gate.sh findings.json` のあとに `echo "exit=$?"` を続けて撮ったもので、P0 と P1 の指摘が1行ずつ並び、`gate: blocked.` の行で止まって終了コードが 1 になっています。PowerShell 版の `gate.ps1` は上のコードブロックのとおり `gate: threshold=P1 total=… blocking=…` の1行が先に出て、そのあとに指摘の行が続き、`$LASTEXITCODE` が 1 になります。書式は違いますが、止まる条件と終了コードは同じです

> **Tips：**指摘が0件だった方は、判定だけを確かめるために手で1件書いてください。スキーマの必須キーは `severity` 、 `confidence` 、 `file` 、 `line` 、 `message` 、 `category` の6つです。1件書いてみると、必須キーの意味が早く入ります。

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` の件数が P1 のときより増えることを確認し、増えた分を1つずつ読んでください。

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。

bash 側で回す方は、引数の位置で同じことができます。

```
bash ci/gate.sh findings.json P2
```

`Out-File -Encoding utf8` は先頭に BOM を付けます。bash 側の `jq` はこの3バイトで `parse error` を返すので、渡す前に落としてください。

```
$json = (Get-Content findings.json -Raw).TrimStart([char]0xFEFF)
[IO.File]::WriteAllText("$PWD\findings.json", $json, (New-Object Text.UTF8Encoding $fal
se))
```

Step 4　確信度の下限を足して完成させる　[3min]

手元の版には、確信度で足切りする機能がまだありません。CI 側の `ci/ai_gate.sh` は `GATE_MIN_CONFIDENCE` で同じことをしています。手元と CI で判定が食い違わないよう、ここを埋めます。

1. `ci/gate.ps1` の `param` に、確信度の下限を受け取る引数を1つ足してください。既定値は CI 側と合わせて 0.0 にします。

2. 重大度を数えている箇所の手前で、下限に満たない指摘を除いてください。除いた件数が出力に残る形にします。落ちた理由と、数に入らなかった理由を後から追えるようにするためです。

3. `findings.json` を開き、`confidence` が 0.8 未満の指摘が1件も無い方は、1件の値を `0.5` に書き換えてください。足切りが効いたかどうかは、数から外れる指摘が1件あって初めて分かります。

4. 下限を 0.8 にして回し、確信度 0.5 の指摘が数から外れることを確認してください。引数名は手順1 で自分が付けた名前です(例: `-MinConfidence 0.8`)。除いた件数の行が出て、`total` と `blocking` がその分だけ減ります。

   ```
   gate: excluded by confidence=1
   ```

<details>
<summary>PowerShell の書き方に迷ったとき</summary>

触るのは3か所です。`param( ... )` の中に引数を1行足す(数値を受けるときは `[double]$名前 = 0.0` の形)、`foreach ($f in $all)` の中で重大度を見る前に `$f.confidence` と下限を比べて除外する(小なりは `-lt`)、除いた件数を `Write-Output` で1行出す、の3つです。日本語はコメントにも書きません。`.ps1` は ASCII だけで書く約束で
</details>

す。bash 版を正にする方は `ci/gate.sh` の3番目の引数として同じものを足し、`jq` の `select` に確信度の条件を加えます。

未知の値の扱いを変えないでください　いまの版は、知らない `severity` が来たら落とす側に寄せています。確信度の足切りを足すときも、この向きは変えません。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。

生成物の名前と場所

残るのは、判定スクリプトと、その入力である JSON です。

ファイル	D2 での置き場所	smart3pm での置き場所
`gate.ps1`（確信度の下限を足した版）	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

OK 基準

この演習が終わった状態

- [ ] スキーマどおりの `findings.json` が手元にある
- [ ] 閾値 P1 と P2 で `blocking` の件数が変わることを、自分の画面で見た
- [ ] 確信度の下限を足した版が動き、下限を上げると数から外れる指摘がある
- [ ] 止める線を自分の言葉で説明できる

追加と考察

同じ差分に対して、Claude 側と Codex 側の両方で `findings.json` を作れた方は、件数と重大度の付き方を並べてください。片方だけしか無い方は、隣の方の画面を見せてもらってください。読むのは1点だけです。同じ差分で、同じ指摘に別の重大度が付いていないかどうか。付いていた場合、その差はモデルが `review.schema.json` の重大度の説明をどう読んだかの差です。基準を渡しても揃わないのであれば、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

ゲートを通った指摘の行方を決める課題です。いまの作りでは、閾値に届かなかった指摘は数えられただけで消えます。P2 以下を MR のコメントとして残す処理を足してください。残す先は MR のコメント1本にまとめます。指摘のたびにコメントを増やすと、小さな改修ほど読む量が増えるという、いま解こうとしている課題そのものを再生産します。

プチ演習6　止まる Stop hook

D2-07 ／ 所要 [15min]

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

目的

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。人が Ctrl+C を押すまで止まりません。
Stop フックは、Claude Code が応答を終えようとした瞬間に走るスクリプトです。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、そこで4本そろっていることを確認します。

この演習で触るもの

触るファイル	`.claude/settings.json` の `Stop` 、`.claude/hooks/stop-gate.ps1` (bash は `stop-gate.sh`)、`.claude/hooks/.stop-gate.state` 、環境変数 `STOP_GATE_MAX`
操作	止まらない版を講師の画面で見る。正しい版を登録し、`stop_hook_active` の門を単体で確かめ、継続の経路を1回踏み、上限 0 で継続が入らないことを見る
見る場所	`Stop hook error` の文、末尾の `continuation 1 of 2` 、標準エラーの `[stop-gate]` 行
達成の状態	継続が1回だけ入って止まる。上限を動かすと継続が入らずに応答が終わる。通す条件3つを説明できる
自社での使いどころ	夜間の無人実行で、テストが赤いまま応答を終わらせない最後の門。ハンズオン3の止め金の1つ
影響が及ぶ範囲	応答のたびに検査が走る(timeout 300秒)。`stop_hook_active` を見ないと人が Ctrl+C を押すまで止まらない。`.stop-gate.state` はコミットしない

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。

操作

Step 1　止まらない版を見る　[3min]

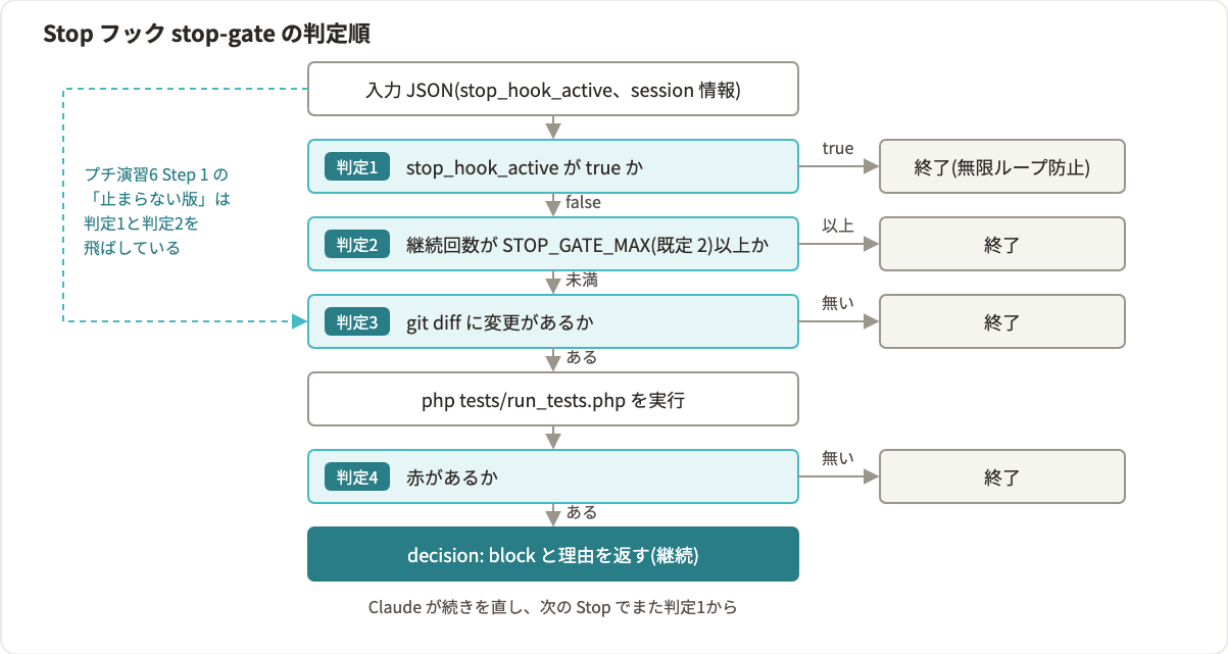
講師が2つの版を並べて回します。受講者は手を動かさずに画面を見てください。止まらない版は各自の端末では回しません。

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3つ置いています。この3つのどれかに当たった時点で、何も返さずに終わります。

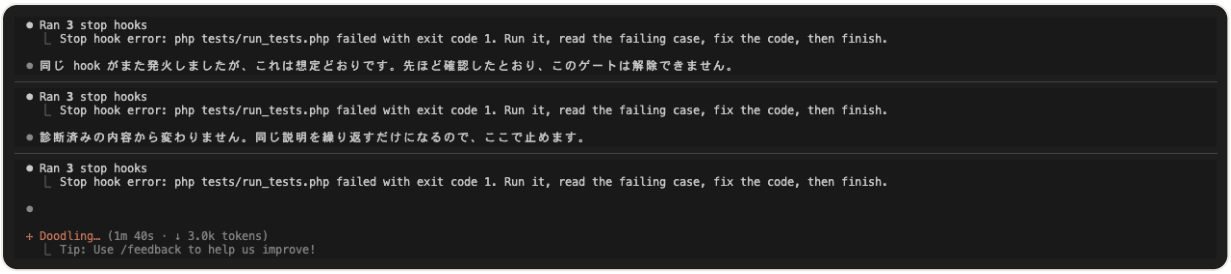
```
# Guard 1: already continued by this hook.
if ($payload.stop_hook_active -eq $true) { exit 0 }

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

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



> 配布の `stop-gate` の判定順です。上から、判定1 で `stop_hook_active` が true なら終了、判定2 で継続回数が `STOP_GATE_MAX` (既定 2)以上なら終了、判定3 で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4 で赤があるときだけ `decision: block` を返します。左の点線が示すとおり、Step 1 の止まらない版は判定1 と判定2 を持たず、入力を読んだらすぐ判定3 以降へ進みます。継続のたびに同じ判定へ戻る理由がここにあります

止まらない版との差は、この3つを持っているかどうかです。読む順に並べると次のようになります。

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、結果も同じになる	Guard1 が先に当たる。`stop_hook_active` が true なので、何も返さずに終わる
画面の見え方	同じ理由の文が数秒おきに増え続ける。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 Ctrl+C を押す以外にない	放っておいても止まる
使用量	1ターンぶんの入力が継続のたびに積み上がる	1本の継続連鎖は Guard1 が必ず1回で切る。`STOP_GATE_MAX` は同じセッションで積み上がる継続回数の上限（既定 2）



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

> **研修環境では有効にしないもの**　Claude の応答のたびに Codex のレビューを走らせるゲートが、公式のプラグインに用意されています。監視下でのみ使うようにという警告が README に明記されており、外すと使用量を一気に使います。この演習では扱いません。

Step 2　正しい版を登録する　[3min]

1. Claude Code を終了し(`/exit`)、VSCode で `.claude/settings.json` の `"hooks"` に Stop のブロックを足してください。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 には `matcher` がありません。応答の終わりは1種類しか無いためです。足したあとの全体の形とカンマの位置は、プチ演習4 Step 2 の `settings.json` の完成形の図で確認してください。

```
"Stop": [
  {
    "hooks": [
      {
        "type": "command",
        "command": "powershell -NoProfile -ExecutionPolicy Bypass -File .claude/hooks/s
top-gate.ps1",
        "timeout": 300
      }
    ]
  }
]
```

> **Mac または bash を正にする方：** `"command"` の値を `bash .claude/hooks/stop-gate.sh` に差し替えてください。以降の手動検査も `powershell ... -File .claude\hooks\stop-gate.ps1` を `bash .claude/hooks/stop-gate.sh` に、`$LASTEXITCODE` を `echo $?` に読み替えます。

2. Claude Code を起動せずに、VSCode のターミナルで門の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` を起動し、`/hooks` で Stop に1本入っていることを確認してください。出ていなければ `settings.json` のカンマを見直します(プチ演習4 Step 2 の「登録したのに動かないとき」と同じ切り分けです)。

> **Tips：**フックの既定のタイムアウトは、コマンド種別で600秒です。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。

Step 3　継続の経路を見る　[4min]

作業ツリーに変更がある状態で応答を終わらせると、2つ目と3つ目の門に進みます。

1. ターミナルで `php --version` を実行して、手元に PHP があるかを確認してください。研修用 PC には PHP を入れていないので、多くの方は「認識されていません」の表示になります。どちらでも演習は成立し、この後に見える画面が変わるだけです。

2. `claude` を起動し、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。編集の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。編集が終わって Claude が応答を返し終えた瞬間が、Stop フックの走る場所です。

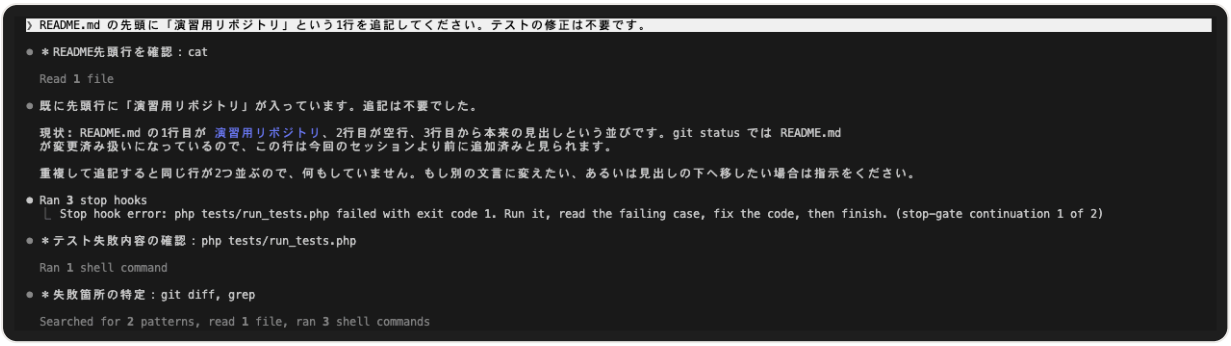
PHP がある端末では、テストが DB に繋がらずに落ち、フックが標準出力へ次を返します。この `reason` の文がそのまま画面に出て、セッションが終わらずに続きます。表示の体裁は版によって変わります。`exit code` の数字は環境で変わります。

```
{"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、標準エラーに1行出て、そのまま応答が終わります。素通りしたことが分かる形です。

```
[stop-gate] php not found on PATH, letting the session stop
```

> **Tips：**フックが素通りしたときの標準エラーの1行は、通常の画面には出ないことがあります。見えない場合は `/exit` で終了してから `claude --debug` で起動し直し、同じ操作をしてください。フックの起動と出力がログとして並びます。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。



> 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

> **Tips：**継続の回数はセッションごとに `.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。`git status` にこのファイルが出たら、`.gitignore` に `.claude/hooks/.stop-gate.state` を1行足します。ファイル名を指定して `git add` する習慣があれば、足さなくても混ざりません。

Codex の場合

Codex も Stop のイベントを持ち、`{"decision":"block","reason":"..."}` を返すと継続のプロンプトになります。登録先は `~/.codex/hooks.json` で、呼ぶスクリプトは `stop-gate.sh` に差し替えてください。

> **Codex 側で無限ループを避ける**　継続中かどうかを表す入力キーが Claude 側と同じ名前かは、当日の講師端末で確認します（要確認）。確認が取れるまでは、次の Step の継続回数の上限を必ず効かせた状態で登録してください。判定キーが読めなくても、上限があれば止まります。

Step 4　上限の効き方を確かめる　[3min]

1. Claude Code を `/exit` で終了し、同じターミナルで環境変数を入れてから `claude` を起動し直してください。フックは Claude Code を起動したターミナルの環境変数を引き継ぐので、別のターミナルで入れても効きません。0 は継続そのものを止める値です。

   ```
   $env:STOP_GATE_MAX = "0"
   claude
   ```

   Mac のターミナルでは `export STOP_GATE_MAX=0` のあと `claude` です。

2. 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。継続の回数が 0 のままでも上限に当たるので、標準エラーに次が出て応答が終わります。PHP が無い端末でも、この行は出ます。上限の判定はテストを走らせる手前にあるためです。見えないときは Step 3 の Tips と同じく `claude --debug` で起動します。

```
[stop-gate] already continued 0 time(s), the cap is 0. Letting the session stop.
```

確認が済んだら環境変数を消して、既定の 2 に戻します。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。PowerShell は `Remove-Item Env:STOP_GATE_MAX`、Mac は `unset STOP_GATE_MAX` です。ターミナルを閉じても消えます。
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、`.stop-gate.state` の回数は 0 に戻ります。

3 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。

生成物の名前と場所

ファイル	D2 での置き場所	smart3pm での置き場所
`stop-gate.ps1` と `stop-gate.sh`	`.ps1` を正にして hooks 配下へ置く。実行するテストの行を、自社のテストコマンドへ差し替える	`.sh` を正にして hooks 配下へ置く。既存の受入テストと同じ並びにテストを1本添える
`settings.json` の Stop ブロック	タイムアウトを自社の検査時間に合わせて書き換えてから写す	同じ検査を bash 側の設定へ写す。2言語で別々の判定を持たない
継続の上限（`STOP_GATE_MAX` の値）	既定値をスクリプト内で決め、台帳に理由を1行残す	同じ値にする。片方だけ緩いと、緩いほうが実質の上限になる

OK 基準

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
- [ ] 継続が入る経路か、素通りする経路のどちらかを、自分の画面で見た
- [ ] 上限の値を動かすと、継続が入らずに応答が終わることを確認した
- [ ] `STOP_GATE_MAX` の環境変数を消し、`.claude/settings.json` の Stop ブロックを1コミットで残した(ブランチはプチ演習4 の `ex/p4-...` のままで構いません)

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

判定の対象を広げる課題です。いまの版はテストの合否だけを見ています。ここに `lint-php54` 相当の検査を足すと、5.4 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つの検査が走ることになります。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。

ハンズオン2　レビュー観点の切り分けと CI ゲート化

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、ご自身の数字で見ていただきました。ここではその先を扱います。出てきた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。
この演習を終えると、レビュー基準の実物(`REVIEW.md`)、観点を分けた3本のレビュアー、そして P0/P1 で合流を止める CI ゲートの3つが、同じ1本の MR の上でつながった状態になります。最後に残るのは「この指摘を人が見る必要があるか」という判断だけです。

> **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。

早見表　この演習で触るもの

触るファイル	`REVIEW.md` 、 `.claude/known_issues.md` 、 `.claude/agents/` の3本、 `.claude/skills/review-route/SKILL.md` 、 `.gitlab-ci.yml` (`ai_gate` の `allow_failure` と `GATE_LEVEL`)、 `tests/phpunit/EstimateSearchTest.php` 、 `app/controllers/estimate_ctl.php`
操作	レビュー基準を人が先に書き、観点別レビュアー3本を今の main に並列で当てて検出率を出す。ゲートを有効にし、テストを先に push し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	`/tasks` の3行、findings の `file:line` の重なり、 `test-phpunit` の赤、 `ai_gate` のログ、 `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、最後のパイプラインで lint・テスト・ゲートが緑。検出率とノイズ率が計測表に入っている
自社での使いどころ	自社のレビュー基準の初版と、機械で判定できる指摘を CI へ移す型。既知の許容の台帳

影響が及ぶ範囲	`allow_failure` を外すと合流が本当に止まる。`GATE_LEVEL` は仮に P2 にしてから P1 に戻す。**Do not report** が広すぎるとゲートが空振りする

題材 Issue #2 と Issue #3

2本の Issue を使い分けます。#2 は実装せず、レビュアーを当てる対象として使います。#3 を実装して MR まで通します。

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

> **Tips：**Issue #3 の受入条件4(`EstimateCsvTest.php` の3本)は、この120分には含めていません。発展課題に回しています。時間が余った方から進めてください。

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	[10min]
Step 1	レビュー基準を人が先に決める。`REVIEW.md` と既知の許容の台帳を書く	[15min]
Step 2	観点を分けた3本のレビュアーを、今のコードに並列で当てる	[45min]
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	[40min]
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	[10min]
合計		[120min]

成果物の提出先は GitLab の Merge Request です。ブランチ名は共通の進め方 STEP 1 の形で `ex/h2-<ユーザー名>` に揃えます。Step 1 の手順1 でこのブランチを切り、Step 2 と Step 3 の成果物も同じブランチに載せて1本の MR で出してください。プチ演習4 から 6 のブランチ(`ex/p4-...`)とは別のブランチです。
使うツールは3つです。ファイルの編集は VSCode、git のコマンドは VSCode のターミナル(PowerShell)、サブエージェントの起動とテストの生成は同じターミナルで起動した Claude Code、MR とパイプラインの確認はブラウザの GitLab です。各手順の先頭に、どれを使うかを書いています。

自分で考える [10min] **人が先に決めること**

Step 1 に入る前に、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. Day1 の宿題で「機械判定できる」に分類した観点を3つ挙げます。CI の `lint-php83` 、`lint-php54` 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。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. 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。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. 「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。入れると指摘は増えます。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. 合流を止める重大度の下限を決めます。P0 だけで止めるのか、P1 まで止めるのか。Step 3 の `GATE_LEVEL` がこの値になります。

> **Tips：**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 [15min]　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。`ci/ai_review.sh` と `.claude/skills/codex-review/SKILL.md` の両方がこのファイルを読みます。

1. VSCode のターミナルで、main を最新にしてから作業ブランチを切ります。この演習の成果物は、Step 3 まで全部このブランチに載せます。 `<ユーザー名>` はご自身のものに置き換えます。

   ```
   git switch main
   git pull
   git switch -c ex/h2-<ユーザー名>
   git branch --show-current
   ```

   プチ演習6 までの変更が `git status` に残っている場合は、先に `ex/p4-...` 側でコミットしてから main に戻ってください。

2. VSCode で `REVIEW.md` を開きます。埋める順番はファイルの冒頭に書いてあるとおり、**Do not report** 、 **Nit の上限** 、 **Important の定義** 、 **Always check** です。先に「書かせないもの」を決めると、Important の輪郭がはっきりします。

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、ここに入れてください。

4. **Nit の上限** に件数を数字で書きます。まとめ方も書いてください。「5件までを1つのコメントに」のように、AI が守れる形にします。

5. **Important の定義** を3件から5件に絞ります。この題材であれば、外部入力が条件に素通りすること、複数の表を続けて書き換える処理がトランザクションの外にあること、PHP 5.4.45 で落ちる構文が入ることの3つが出発点です。

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。参照の書き方は次の手順で決めます。

9 手順1 で切ったブランチのまま、コミットします。

```
git add REVIEW.md .claude/known_issues.md
git commit -m "handson2: レビュー基準の初版と既知の許容の台帳"
```

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、`ci/ai_review.sh` の `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、その理由を MR の説明に1行書いてください。

コミットが済んだら、次の形になっていれば正しく進んでいます。

```
$ git diff --stat HEAD~1
 .claude/known_issues.md | 11 ++++++++++
 REVIEW.md               | 16 ++++++++++++++---
 2 files changed, 24 insertions(+), 3 deletions(-)
```

> **Codex の場合：** `AGENTS.md` の **Code Review Rules** 節を、`REVIEW.md` と同じ内容に揃えてください。演習リポジトリの `AGENTS.md` には書き方の見本が入っています。「結果で書く」「機械的なチェックは書かない」の2つは、そのまま自社版でも使えます。`AGENTS.md` の冒頭にあるとおり、片方だけ直すとツールによって出来上がりが変わります。

この Step が終わった状態

- [ ] 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 に何を入れるか決まりません

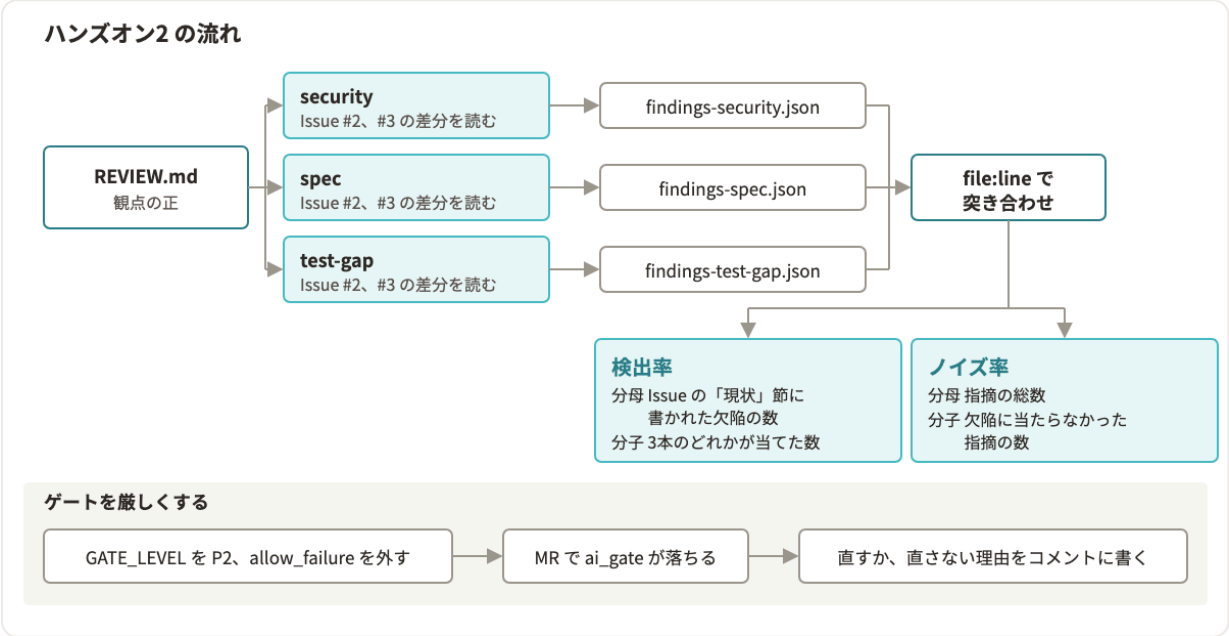
Issue #2 と #3 の「現状」節を読み直してください。そこに書かれている事実は、既に分かっている欠陥です。それを検知できる書き方になっているかどうかで、Important の粒度を判定できます。「SQL の書き方に注意する」では検知できません。「外部入力が `db_select` の第1引数に文字列連結で入る変更」であれば検知できます。

それでも迷う場合は、いま書いてある `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。

ハンズオン2 Step 2　観点別レビュアーの並列適用

STEP 2 [45min]　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。結果が分かっている改修で指摘の当たりを測る、という考え方をこの形で代用します。
ここで使うサブエージェントは、 `.claude/agents/` に置いた定義ファイル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ず、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で、これを並列と呼んでいます。



> ハンズオン2 の流れです。Step 1 で書いた `REVIEW.md` を観点の正として、Step 2 で security、spec、test-gap の3本が同じ差分を並列で読み、 `findings-security.json` 、 `findings-spec.json` 、 `findings-testgap.json` の3本が残ります。それを `file:line` で突き合わせ、検出率(分母は Issue の「現状」節の欠陥の数、分子は3本のどれかが当てた数)とノイズ率(分母は指摘の総数、分子は欠陥に当たらなかった数)を出します。下段が Step 3 で、 `GATE_LEVEL` を P2 にして `allow_failure` を外し、MR で `ai_gate` を1回落とし、直すか直さない理由を書くところまでです

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。 `test-gap-reviewer.md` だけ `effort` がありません。理由はファイルの末尾に書いてあります。

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てください。ここを直さないと、基準を書いた意味がレビュアー側に届きません。

3. 手順2 の変更を保存してから、ターミナルで `claude` を起動し、3本を並列で当てます。次のとおり渡してください。

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

4. 3本が動いている間に `/tasks` を打ち、3本が同時に動いていることと、3本のモデル名が違うことを確認します。終わるまで数分かかります。

5. 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てください。この3ファイルは Step 3 の Codex の手順でも使います。コミットには入れません。

6. 検出率とノイズ率の2つを出します。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。分母は先に自分で数えて表に書いてください。「現状」の箇条書きのうち、今のコードの欠陥を述べているものが対象で、「CSV を出す入口はまだ無い」のような未実装の項目は入れません。何を分母に入れたかも記録に残します。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。配布の `計測表_Day1_Day2.md` と同じ定義です。

7. 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。`CLAUDE.md` の「レビューの頼み方」節と同じ内容を、手順の形に直して書いてください。ファイルの形は隣の `.claude/skills/codex-review/SKILL.md` と同じで、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。
---
```

行数は `git diff --stat main` の末尾に出る数字で数えます。

手順4の画面。3つのエージェント名が同時に走り、下の一覧でトークン量が別々に増えていれば並列になっています。モデル名は `/tasks` の一覧で確認できます

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": "外部入力の経路を 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。

> **Codex の場合：**サブエージェントに当たる仕組みがないため、観点ごとに3回まわします。出力先のファイル名を分ければ、あとの突き合わせは同じです。
>
> ```
> codex exec -s read-only --output-schema .claude/review.schema.json -o findings-security.json \
>   "$(cat REVIEW.md) 外部入力が SQL と画面出力に届く経路だけを見てください。他の観点は書かないでください。対象: app/controllers/stock_ctl.php と app/controllers/estimate_ctl.php"
> ```
>
> 同じ形で `-o findings-spec.json`（指示と差分の一致）、`-o findings-testgap.json`（テストの欠落）を回します。重さの切り替えは `~/.codex/config.toml` の `[profiles.review-light]` と `[profiles.review-full]` で、`codex --profile review-full` のように指定します。プロファイルの書き方は2通りが出回っているため、`codex --profile review-light` が通ることを1回だけ手元で確かめてから本番で使ってください(要確認)。

skill は誘導であって保証ではありません　手順7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。

この Step が終わった状態

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

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 は外部入力の経路だけ、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。重なりの数も記録に残してください。

ハンズオン2 Step 3　CI ゲートの有効化とテスト生成

STEP 3 [40min]　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 push して、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。

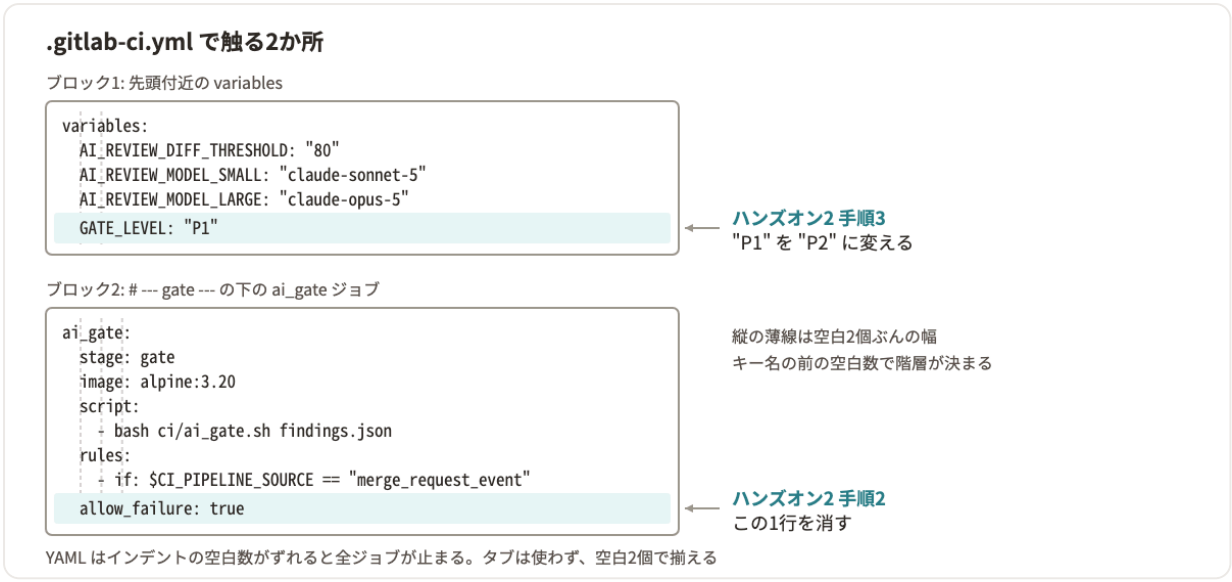
runner から外部の API へ出られない場合　手元で `findings.json` を作って、同じ判定を回します。(1) プチ演習5 Step 1 と同じ形でレビューを JSON に落とす (2) `powershell -NoProfile -ExecutionPolicy Bypass -File ci\gate.ps1 -Threshold P2` を実行する（bash 環境では `GATE_LEVEL=P2 bash ci/ai_gate.sh findings.json`。`ai_gate.sh` は閾値を引数ではなく環境変数で受けます） (3) 出力を MR の「AI レビュー結果」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。手順8 以降の `ai_review` と `ai_gate` の実走は講師環境で行います。

1. Step 1 で切ったブランチのまま進めます。

```
git branch --show-current
```

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。これでゲートがパイプラインを落とせるようになります。他の行のインデント(先頭の空白)は変えません。YAML は空白の数で構造が決まります。

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す。手順11で `"P1"` に戻します。ファイル内に `GATE_LEVEL` は1か所しかありません。



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` で、いちばん下の `GATE_LEVEL: "P1"` を `"P2"` に変えます。下のブロックが `# --- gate ---` の下の `ai_gate` ジョブで、末尾の `allow_failure: true` の1行を消します。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 AI に渡す前に、この検索が条件として使われていないとき一覧に何件出るかを紙に書きます。 `db/seed.sql` の見積は15件です。

5 Issue #3 の受入条件3を読み、テストを1本足します。足す先は `tests/phpunit/EstimateSearchTest.php` です。

```
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(PHPUnit 4.8 / PHP 5.6、
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 ターミナルで、Step 2 の成果物と、手順2 と 3 で直した `.gitlab-ci.yml` を、別々のコミットにします。 `.gitlab-ci.yml` をコミットに入れ忘れると、push しても CI 側は `allow_failure: true` のままで、ゲートは一度も落ちません。

```
git add .claude/agents .claude/skills/review-route
git commit -m "handson2: 観点別レビュアーと振り分けの skill"
git add .gitlab-ci.yml
git commit -m "handson2: ai_gate の allow_failure を外し GATE_LEVEL を仮に P2 にする"
```

`findings-*.json` は手元の記録なので `git add` しません。

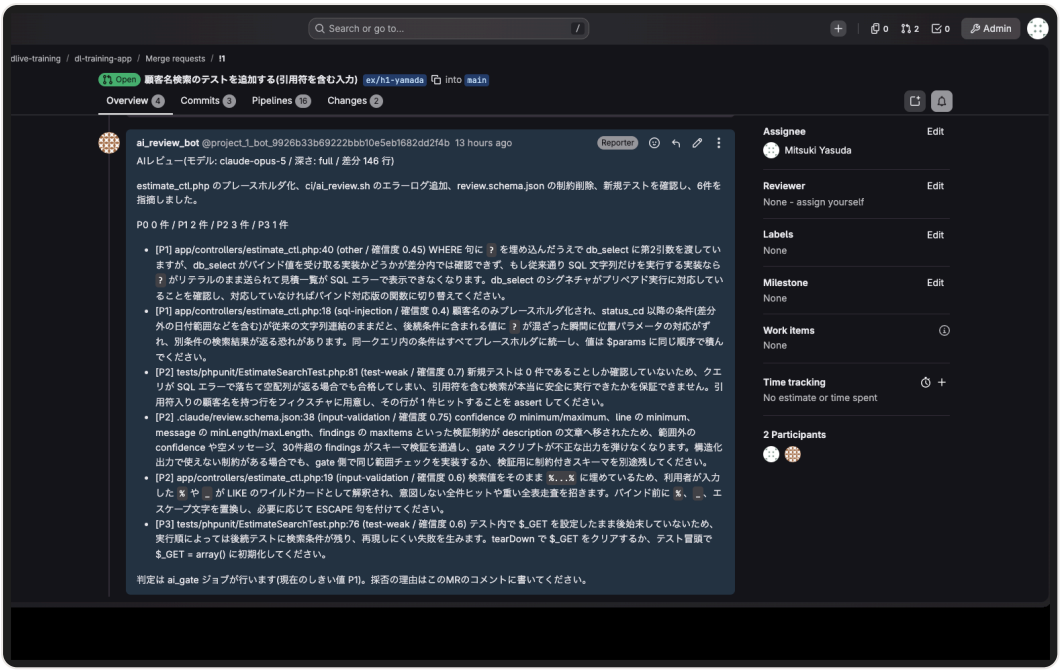
7. テストだけを別のコミットにして push し、MR を作ります。push が終わると `View merge request for` の行に MR の URL が出るので、ブラウザで開きます。

```
git add tests/phpunit/EstimateSearchTest.php
git commit -m "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"
git push -u origin HEAD -o merge_request.create -o merge_request.target=main
```

8. ブラウザで MR の **Pipelines** タブを開きます。**branch** のラベルが付いた行(lint と test)で `test-phpunit` が赤になっていることを確認します。`test-phpunit` は他のジョブより2分ほど長くかかります。赤いジョブ名をクリックしてログを開き、落ちた出力を MR の説明の「テスト」欄に貼ってください。

9. 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。日付をプレースホルダで渡すこと、`YYYY-MM-DD` の形に合わない値を条件に使わないこと、条件の組み立てを `index()` の外の private メソッド1つにまとめることの3点です。`csv_text()` から同じメソッドを呼ぶところは、発展課題1で受入条件4と一緒に足します。ここでも、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 AI に渡してください。Claude Code への依頼文は自分で書きます。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。PHP 5.4 の制約は CLAUDE.md とプチ演習4 のフックが見ています。

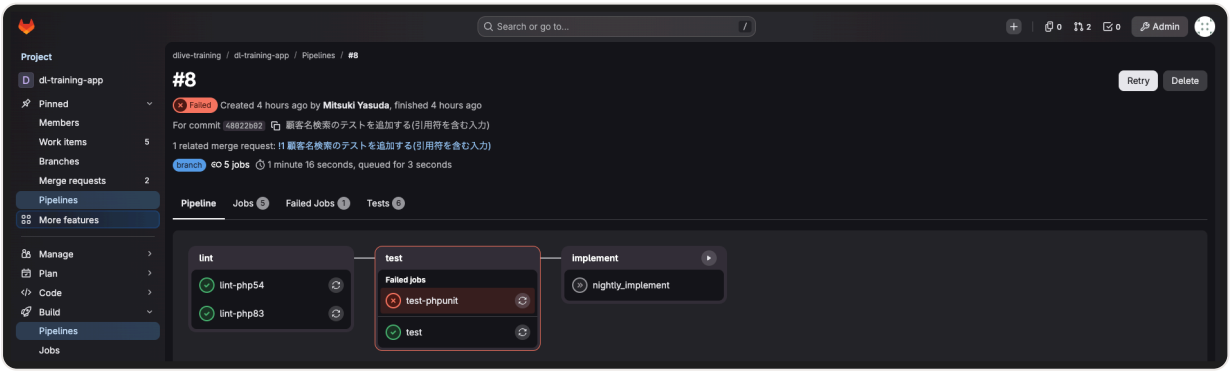
10. 実装をコミットして push します(2回目以降の push は `git push` だけです)。**branch** の行で `test-phpunit` が緑になり、**merge request** の行で `ai_review` が MR にコメントを1本置き、`ai_gate` が赤になります。`ai_gate` が赤になるのは、CI に `ANTHROPIC_API_KEY` が入っていて、P2 以上の指摘が1件でも出たときです。緑のままなら下の「ai_gate が緑のままで、一度も落ちません」を開いてください。



手順10 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1件ずつ `[P1] ファイル:行（カテゴリ / 確信度）` の形で並びます。画面は差分 146 行の MR に `claude-opus-5` が6件出した例です。差分が `AI_REVIEW_DIFF_THRESHOLD` (既定 80)を下回る、例えば 43 行の MR では `claude-sonnet-5` に切り替わり、1行目のモデル名がそのように変わります。末尾の1文にあるとおり、判定はこのコメントではなく `ai_gate` ジョブが行います

11 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つに仕分けます。仕分けは 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 `REVIEW.md` の **Do not report** に入れるかどうかを検討します。終わったら `.gitlab-ci.yml` の `GATE_LEVEL` を `"P1"` に戻し、修正と合わせてコミットして push します。

12 lint(`lint-php83` と `lint-php54`)、テスト(`test` と `test-phpunit`)、ゲート(`ai_gate`)の3つが緑になったことを確認します。



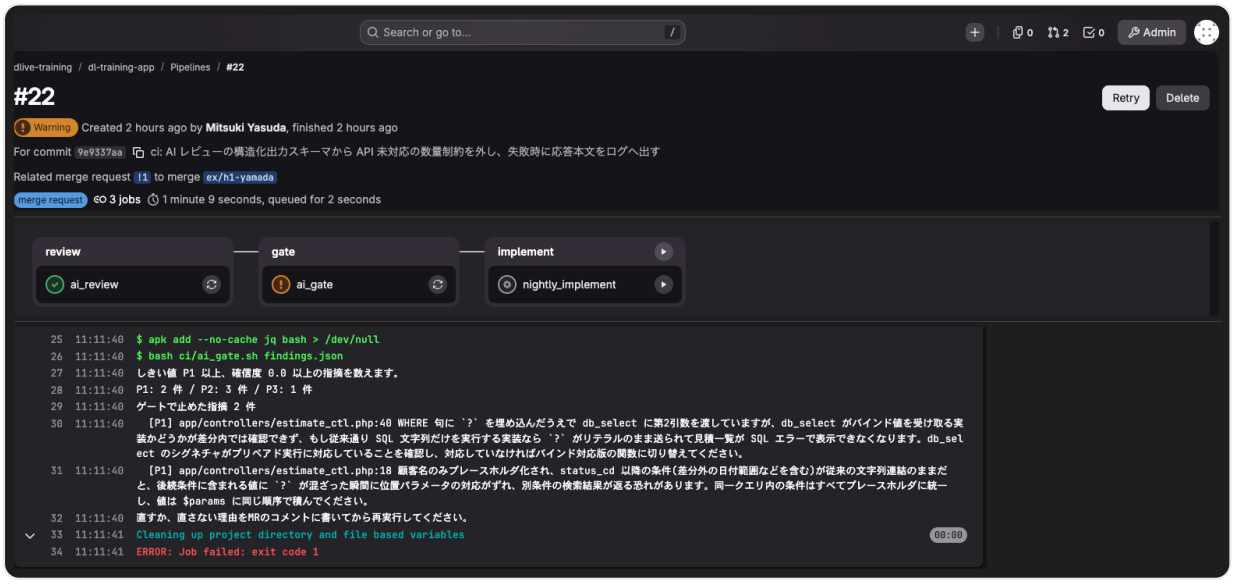
手順8で開くブランチ用のパイプライン。実装前なので `test-phpunit` だけが赤になります。この赤が、足したテストが今のコードを見ていた証拠です

手順8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:81

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

手順10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。

```
$ bash ci/ai_gate.sh findings.json
しきい値 P2 以上、確信度 0.0 以上の指摘を数えます。
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() の中にしかありません
  ...

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
```



手順10で開く MR 用のパイプラインと、その下は `ai_gate` のログです。`ai_review` は緑のまま、`ai_gate` がしきい値 P1 以上の指摘2件で止まっています。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で外したあとは赤の × になり、Merge ボタンが押せなくなります

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。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。

> **Codex の場合：** CI 側のジョブはそのままです(`ci/ai_review.sh` は Anthropic の API を直接叩いています)。手元で同じ判定を先にかけたい場合は、Step 2 で作った findings を使って次を実行してください。
>
> ```
> powershell -NoProfile -ExecutionPolicy Bypass -File ci/gate.ps1 -Path findings-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。

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
- [ ] 最後のパイプラインで lint・テスト・ゲートの3つが緑になっている

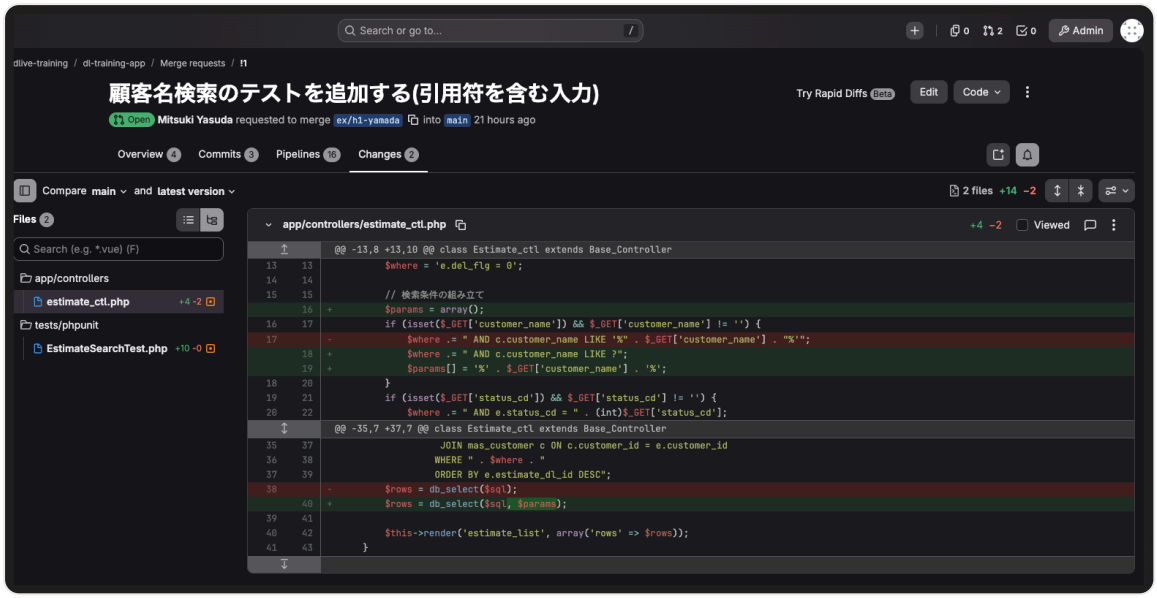
ai_review ジョブが動きません

このジョブは `merge_request_event` のときだけ動きます。ブランチを push しただけでは走りません。MR を作ってから、もう一度 push してください。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(`+4 -2` のように)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。画面は別の MR(2ファイル、`+14 -2`)で撮ったもので、ご自身の MR では `.gitlab-ci.yml` と `tests/phpunit/EstimateSearchTest.php` が左の一覧に見えているはずです

`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。上の「ai_review ジョブが動きません」と同じく講師の設定です。その間は、この Step の冒頭にある「runner から外部の API へ出られない場合」の手順で、手元の `gate.ps1` を使って進めてください。
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。何を書かせないと決め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、PHP 5.6 で動かない書き方が入っています。PHPUnit 4.8 には `assertStringContainsString` がありません。`assertContains` を使ってください。`$this->expectException()` も 4.8 にはありません。
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すと直ります。

STEP 3 補足 [10min] 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。

1. 手順9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. push してパイプラインを見ます。 `lint-php83` 、 `lint-php54` 、 `test` は緑のままです。既存の5本のテストも緑です。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。

```
git revert HEAD
git push
```

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。push 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

この4つがそろっていれば完了です。ご自身で判定できます。

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
- □ 最後のパイプラインで lint・テスト・ゲートの3つが緑になっている
- □ 足したテストが、実装前は赤だったことがジョブの履歴から追える
- □ 配布の `計測表_Day1_Day2.md` の「Day2 検出率」に、検出率とノイズ率が入っている

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

生成物　ファイルの名前と置き場所

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

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

`.gitlab-ci.yml` の `ai_review` と `ai_gate`	CI 側へ新規。機械強制を定義どおりに戻す位置づけ	pre-receive の検査とは別に、CI 側へ新規
判定スクリプト	`ci/gate.ps1` (PowerShell 5.1、ASCII のみ)	`ci/gate.sh`
追加した PHPUnit テスト	テストの置き場所の規約に合わせる(要確認)	同左(要確認)
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。列の意味は次のとおりです。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)					
CI の `ai_review`					

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

1. Day1 の3通りレビューで数えた指摘件数と比べて、増えたか減ったか。減ったのであれば、それは `REVIEW.md` のどの行が効いた結果か。

2. 3本のうち、既知の欠陥を1件も拾えなかったレビュアーがいたか。いた場合、観点の書き方と当てた対象のどちらに原因があるか。

3. ゲートで止まった指摘のうち、直さない判断をしたものの割合。この割合が高いのであれば、`GATE_LEVEL` か **Important の定義** のどちらかが現場に合っていません。

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

発展課題　CSV 出力とレビュアーの4本目

早く終わった方から、上から順に進めてください。全部やる必要はありません。

1. Issue #3 の受入条件4(`tests/phpunit/EstimateCsvTest.php` の3本)を満たすところまで実装します。`csv_text()` を public メソッドに分ける形は「期待する振る舞い」の5 に書いてあります。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。トランザクションの外にある4つの DB 操作が、ここで初めて `ai_gate` の対象に入ります。`ai_review` は MR の差分しか読まないので、`stock_ctl.php` の指摘が出るのはこの MR からです。こちらは DB を複数回書き換えるため、Step 1 で決めた **Important の定義** がそのまま効きます。

3. `.gitlab-ci.yml` の `GATE_MIN_CONFIDENCE` を `0.6` にして回し直し、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。小さな改修に最重量のレビューを当てない、という切り替えの実体がここにあります。

5. レビュアーの4本目を作ります。PHP 5.4.45 で落ちる構文だけを見る1本です。プチ演習4で書いた `check-php54` 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。

> **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。

ハンズオン3　夜間ループの最小構成

Day2 の最後は、ここまでに書いた設定を1本の自動実行につなぎます。GitLab のパイプラインからエージェントを起動し、ブランチ作成・実装・5.4 の検査・テスト・レビュー・MR 作成までを人の操作なしで通します。マージだけを人に残します。この資料では、この「人が寝ている時間に GitLab がエージェントを起動し、朝に MR だけが残っている」流れを夜間ループと呼びます。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になっているので、無人で流す1本目に向いています。
自社で同じ形を組むと、日中の人の時間を「Issue を書く」「MR を読んでマージを押す」の2つに寄せられます。実装と検査とレビューは夜のうちに終わっていて、朝に見るのは結果だけです。今日の演習は、その形が今のハーネスでどこまで成り立つかを、実際に1本流して確かめます。
この演習で新しく書くコードはほとんどありません。作るのは止め金の配置と、止まったときにどの層を直すかの判断です。夜間ループは、止まる場所が決まっているかどうかで結果が変わります。

> **CI が外へ出られない場合**　GitLab の runner から `api.anthropic.com` へ到達できるかは、貴社側の通信制限の解除手続きの進み方によります。未解除のままだった場合は、Step 3 と Step 5 の CI 実走を講師環境で行い、受講者は同じ `-p` の呼び方を手元の PowerShell で回します。進め方は「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。

> **Tips：**D2 では、機械強制のタグが「エージェントの完了報告に載ること」で担保されていると伺っています。夜間ループは、その報告を CI の実行記録で置き換える足し方になります。新しい仕組みを1つ増やす作業ではなく、すでにある約束を機械が実行した証拠に結び直す作業です。

早見表 この演習で触るもの

触るファイル	`.claude/settings.json` (deny 8本とフック4本)、`.gitlab-ci.yml` の `nightly_implement` 、`ci/nightly_implement.sh` 、`.claude/skills/implement-issue/SKILL.md` 、GitLab の Pipeline schedule とブランチ保護の設定
操作	止め金4か所を確認し、schedule を Inactive のまま作り、手動ジョブを講師の合図で流す。1回目に止まった箇所を記録し、直す層を1つ選んで再実行し、MR を確認する
見る場所	ジョブログ末尾の `turns` と `cost_usd` 、`ci_logs/claude-issue-4.err` 、MR `nightly/issue-4` のパイプライン、**Protected branches** の設定
達成の状態	MR が人の操作なしで立つ。止まった箇所と直した層が記録にある。夜間に回せる範囲と人が残る3点が1枚に書けている
自社での使いどころ	自社の夜間ループの最小構成。schedule の設定と止め金の配置をそのまま持ち帰る
影響が及ぶ範囲	CI の中(Linux)ではフックが動かず、効くのは deny と CI ジョブだけ。schedule を有効にすると全員分が同時に走り、runner と API の予算を使う。マージだけは人が押す

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「配布された設定を写す作業」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。「これが起きたら朝までに気づけない」と思うものから挙げてください。

2. その3つを止めるのは、次の4層のどれかを指定します。(a) `.claude/settings.json` の `permissions.deny` (b) PreToolUse フック (c) `--max-turns` とジョブの `timeout` (d) GitLab のブランチ保護

3. 翌朝に自分が最初に見る画面を1つ決め、そこで何を見て「マージしてよい」と判断するかを3つ書きます。指摘の件数、テストの色、差分の行数など、実際に目で見るものだけを挙げてください。

> **Tips：**(a)〜(d) は強度が違います。deny は文字列照合なので `bash -c` のような書き方ですり抜けます。フックは権限モードに関係なく先に走ります。 `--max-turns` は操作の種類を見ずに回数だけで打ち切ります。ブランチ保護だけは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。同じ操作を2層で止めている箇所と、どこでも止めていない箇所が出てくるはずです。

書き出せた状態

- [ ] 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

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。

1. `.gitlab-ci.yml` の `nightly_implement` ジョブ。 `schedule` から自動で、それ以外のパイプラインでは手動ボタンとして出ます。 `timeout: 30m` と `allow_failure: true` が付いています。

2. `ci/nightly_implement.sh` 。ブランチ `nightly/issue-4` を作り、エージェントを呼び、コミットを確かめ、push で MR を作るところまでを1本にしたスクリプトです。

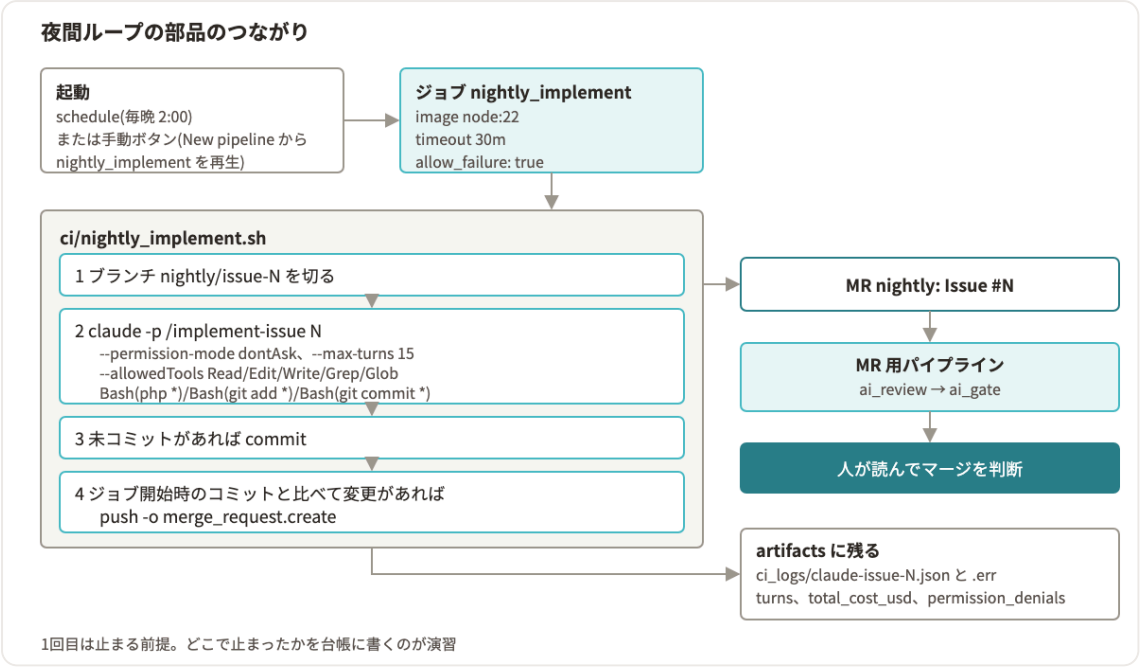
3. `.claude/skills/implement-issue/SKILL.md` 。Issue を読む・方針を3行で書く・影響範囲を調べる・実装する・テストを回す、の手順書です。対話でも無人でも同じ手順を踏ませます。

4. `.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です。

ジョブの並びは次のとおりです。夜間ジョブが push した瞬間に、MR 側のパイプライン(lint・test・ai_review・ai_gate)が動き出します。

```
[ 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 -o merge_request.create  →  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
```



上の並びを図にしたものです。左上の起動(schedule か手動ボタン)がジョブ `nightly_implement` を呼び、その中で `ci/nightly_implement.sh` が4段を順に進めます。ブランチを切る、`claude -p /implement-issue N` を回す、未コミットがあれば commit、ジョブ開始時と比べて変更があれば `push -o merge_request.create` の4つです。push の先で MR が立ち、MR 用パイプライン(`ai_review` から `ai_gate`)が動き、最後に人が読んでマージを判断します。下の枝が artifacts で、`ci_logs/claude-issue-N.json` と `.err` に turns、`total_cost_usd`、`permission_denials` が残ります。Step 4 で見るのはこの枝です

> **Tips：**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。

この演習で初めて出てくる語です。`.gitlab-ci.yml` の `nightly_implement` の節に、そのままの綴りで出てきます。

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 `$CI_PIPELINE_SOURCE == "schedule"` は「schedule から起動されたときだけ自動で走る」の意味です。Step 2 で作り、当日は使いません
手動ジョブ	`rules` の `when: manual` 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。schedule 以外のパイプラインでは `nightly_implement` がこの形になります

`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。ジョブの名前の横に **Allowed to fail** と出て、アイコンは赤ではなくオレンジの警告になります。1回目は止まる前提で流すので付けてあります
`ISSUE_ID`	実装させる Issue の番号を入れる変数。`.gitlab-ci.yml` の既定値は `"4"` で、schedule の Variables か手動実行の Variables で上書きできます。スクリプトはこの値からブランチ名 `nightly/issue-4` とログ名 `claude-issue-4.json` を組み立てます
`artifacts`	ジョブが終わったあとに GitLab が保存するファイル。このジョブでは `ci_logs/` の中身(エージェントの結果 JSON と標準エラー)が1週間残ります。ジョブ画面の右側の **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。`claude-issue-4.json`(結果の要約。`num_turns`、`total_cost_usd`、`is_error`、`result`)と `claude-issue-4.err`(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　操作の手順

Step 1 から Step 7 まで、合計 [120min] です。Step 3 は講師の合図で3名ずつ流します。自分の番が来るまでの待ち時間は Step 4 の記録欄の下書きに使ってください。
使う道具は3つです。ファイルの確認と編集は VSCode(`dl-training-app` を開いた状態)、コマンドは VSCode の統合ターミナル(Windows は PowerShell、Mac は zsh か bash)、GitLab の操作はブラウザです。各 Step の手順に、どれを使うかを書いてあります。

> **Mac の方：**本文のコマンドは Windows の PowerShell で書いています。違いは3つだけです。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` の `command` は `bash .claude/hooks/<名前>.sh` の形です。パスの区切りは `\` ではなく `/` です。git のコマンドと GitLab の操作は同じです。

STEP 1　止め金4か所の設定 [15min]

「考える」で書き出した3つが、実際にどこで止まるかを確かめます。4か所を上から順に見ます。
先に、この演習のブランチを切ります。VSCode の統合ターミナルで、共通の STEP 1 と同じ形です。ハンズオン2のブランチに乗ったままだと、2つの演習の変更が混ざります。`<ユーザー名>` は社用メールアドレスの「@ より前」です。

```
git switch main
git pull
git switch -c ex/h3-<ユーザー名>
git branch --show-current
```

最後の行に `ex/h3-` で始まる名前が返れば、この上で進めます。Step 1 で `.claude/settings.json` を書き足した場合も、Step 5 で直す層も、全部このブランチにコミットします。

1. VSCode で `.claude/settings.json` を開き、`permissions.deny` に次の8行が入っていることを確認してください。無ければ足します。上の5行が Day1 のプチ演習2で、下の3行がプチ演習3で書いたものです。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8行を入れます。足したら `git add .claude/settings.json` と `git commit -m "夜間ループの止め金: deny 8行"` でコミットしてください。

   ```
   "Bash(git reset --hard *)",
   "Bash(git push --force *)",
   "Bash(git push -f *)",
   ```

```
  "Bash(git checkout -- *)",
  "Bash(git clean *)",
  "Read(./.env)",
  "Read(./.env.*)",
  "Read(./**/credentials*)"
```

上の5本が破壊系、下の3本が秘密情報です。 `deny` は JSON なので OS を跨いで効きます。CI runner(Linux)で効く止め金は、この `deny` だけです。

2. 同じターミナルで `claude` と打って Claude Code を起動し、入力欄に `/hooks` と打って Enter を押します。登録済みのフックがイベント名ごとに並ぶ画面が出ます。PreToolUse(`block-destructive`)、PostToolUse(`check-php54`)、Stop(`stop-gate` 。プチ演習6 で登録)、SessionStart(`sessionstart_verify`)の4本があることを確認してください。4つ出ていなければ、Esc で画面を閉じ、VSCode で `.claude/settings.example.jsonc` を開いて `hooks` のキーを `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 Claude Code を `/exit` で終了して起動し直し、もう一度 `/hooks` で4本を確認します。

3. VSCode で `.gitlab-ci.yml` を開き、いちばん下の `nightly_implement:` の節を見ます。 `timeout: 30m` と、 `variables` の `ISSUE_ID: "4"` があることを確認します。ここは書き換えません。

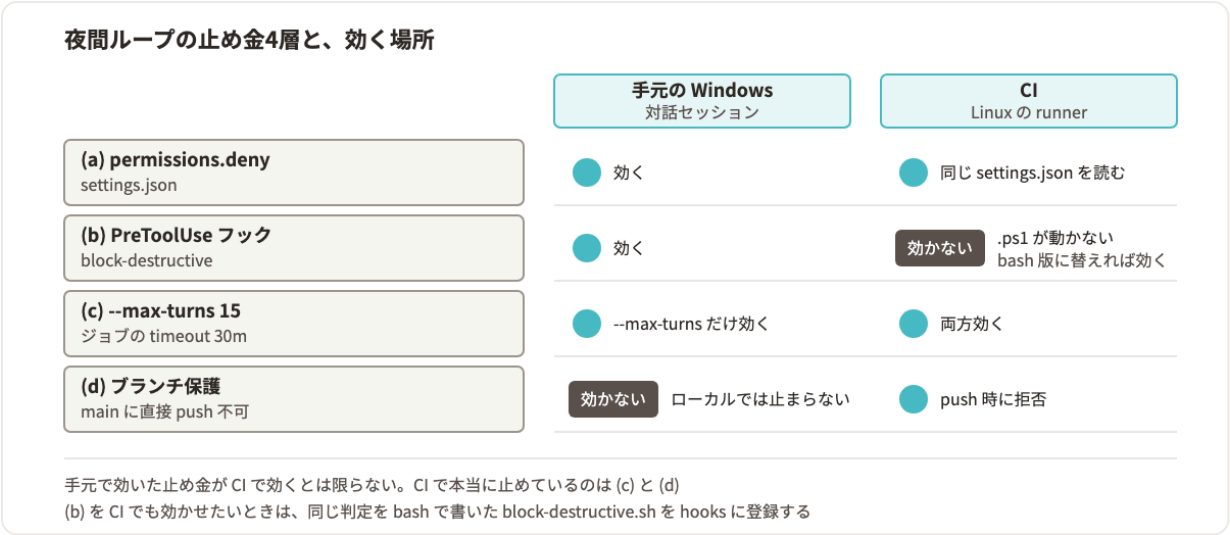
4. VSCode で `ci/nightly_implement.sh` を開き、 `claude -p "$PROMPT" \` で始まる5行を見ます。 `--max-turns 15` が付いていることを確認します。値は事前に計測した1本あたりの所要時間と費用から決めているので、上げないでください。

5. 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 **Settings** > **Repository** を選びます。ページの中ほどにある **Protected branches** の節の **Expand** を押して広げます。表に `main` の行があり、 **Allowed to push and merge** が **No one** 、 **Allowed to merge** に **Developers** が含まれていることを確認します。直接 push できないことが4層目の止め金です。見るだけで、変更はしません。 **Allowed to merge** に **Developers** 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。

4か所が出そろったところで、「考える」で書いた3つの操作がどこで止まるかを突き合わせてください。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。

> **Tips：**この設定のままだと、CI の中ではフックが動きません。 `nightly_implement` のイメージは Linux (`node:22-bookworm-slim`) で、 `powershell` が入っていないためです。5.4 の混入を CI 側で止めているのは、フックではなく MR パイプラインの `lint-php54` です。ただし `lint-php54` は `php:5.6` の `php -l` なので、 `::class` ・可変長引数・ `finally` ・ジェネレータ・ `const` への配列代入は通り抜けます。ここを塞ぐ `phpcs` (PHPCompatibility の testVersion 5.4)のジョブは、まだ CI にありません。足すのはハンズオン2の発展課題5です。手

> 元と CI で効く層が違うことを、この時点で頭に入れておいてください。Step 4 の記録で使います。



止め金4層と、効く場所の対応です。左の (a)〜(d) が「考える」の手順2 の4層で、右の2列が手元の対話セッションと CI の runner です。(a) の `deny` は両方で効き、(b) のフックは手元では効きますが CI では `.ps1` が動かないので効きません(bash 版に替えれば効きます)。(c) は手元で効くのは `--max-turns` だけで、CI では `timeout` も一緒に効きます。(d) のブランチ保護はローカルでは何も止めず、push の時点で拒否します。手元で効いた止め金が CI でも効くとは限らないことを、この表で確かめてください

> **Codex の場合：**止め金の置き場所が変わります。 `~/.codex/config.toml` の `sandbox_mode = "workspace-write"` が書き込みの範囲を決め、 `approval_policy` が確認の取り方を決めます。無人実行では確認に答える人がいないので `never` にします。 `[sandbox_workspace_write]` の `network_access = false` と `writable_roots = []` はそのままにしてください。 `.git` を渡していないため、コミットはスクリプト側の仕事になります。 `--max-turns` に当たる回数の打ち止めは一次情報で確認できていないため、Codex 側の打ち止めはジョブの `timeout: 30m` だけと考えて進めます(要確認)。

この Step が終わった状態

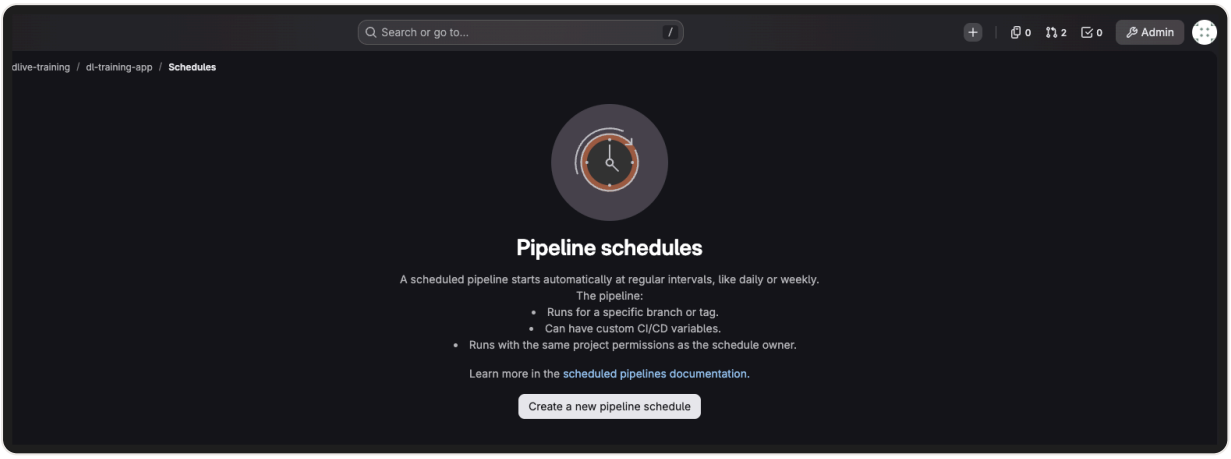
- [ ] deny 8行、フック4本、timeout、--max-turns、ブランチ保護の5つを自分の目で確認した
- [ ] 「考える」で書いた3つの操作が、どの層で止まるかを紙の上で対応づけた
- [ ] どこでも止まらない操作が1つ以上見つかっている

STEP 2 pipeline schedule の作成 [15min]

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。

1. GitLab で演習プロジェクトを開き、左のメニューから **Build** > **Pipeline schedules** を選びます。

2. **New schedule** を押します。



> 手順1 で開く **Pipeline schedules** の画面です。まだ1本も無いときはこの表示になり、押すのは中央の **Create a new pipeline schedule** です。手順2 の **New schedule** と同じ入力画面へ進みます。保存後は一覧の形に変わり、各行に **Active** のトグルと **Next Run** の列が出ます。手順7 で **Activated** を外して保存した schedule は、この一覧に **Inactive** で並びます

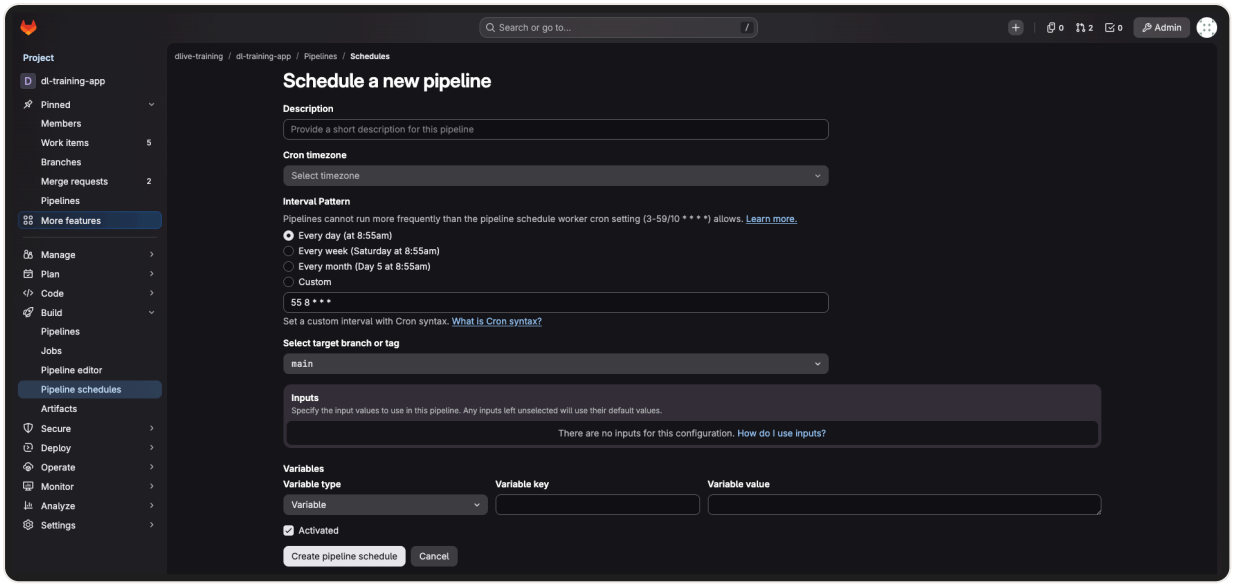
3. **Description** に `nightly issue-4` と入力します。

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。ここで渡した値が `.gitlab-ci.yml` の `variables` を上書きします。

7. **Activated** のチェックは外したまま、フォームの下の保存ボタン（**Create pipeline schedule**）を押します。一覧に `nightly issue-4` の行が増え、**Inactive** と表示されていれば作成できています。チェックを付けたまま保存してしまった場合は、一覧の行のトグルを切って **Inactive** に戻してください。時刻で動かすのは、自社のリポジトリに写してからです。18名ぶんが同じ時刻に走ると、runner の並列数と API の予算を一晩で使い切ります。



Step 2 の入力画面を、開いた直後の未入力の状態で撮ったものです。上から順に、**Description** が手順3、**Cron timezone** が手順5 の Tokyo、**Interval Pattern** が手順4 で、既定は **Every day (at 8:55am)** が選ばれ下の欄に `55 8 * * *` が入っているので、**Custom** に切り替えてから同じ欄を `0 2 * * *` に書き換えます。**Select target branch or tag** が手順5 の `main` 、**Variables** の Key と Value が手順6 の `ISSUE_ID` と `4` です。いちばん下の **Activated** は開いた時点でチェックが付いているので、手順7 のとおり外してから **Create pipeline schedule** を押します。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

> **Tips：**研修当日はこの schedule から自動起動させません。18名ぶんが同じ時刻に走ると、runner の並列数と API の予算を同時に使い切ります。当日は手動ボタン側だけを使い、講師が3名ずつ順番に流します。

> **Tips：**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。時刻を持たせる場所を GitLab 側に置いているのはこのためです。

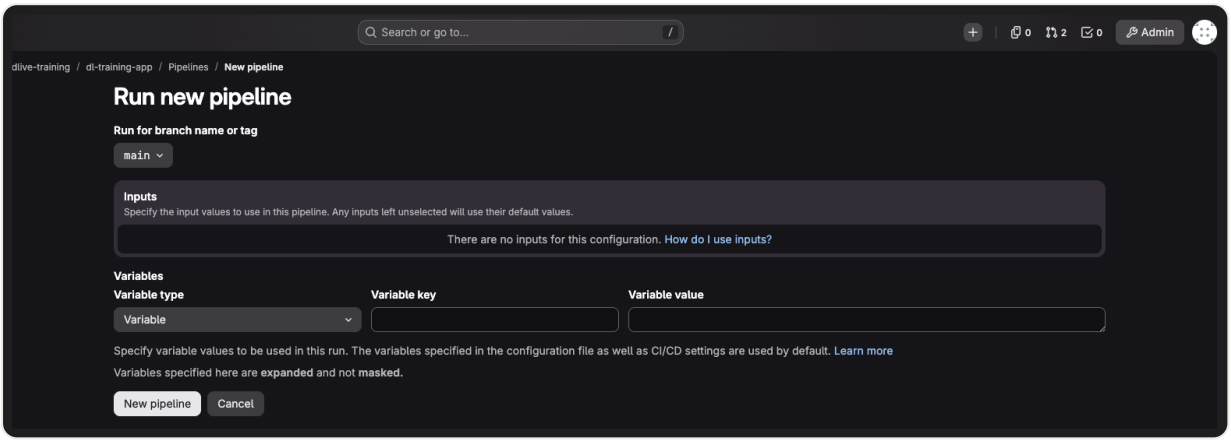
この Step が終わった状態

- [ ] 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

STEP 3　手動ジョブの実行 [20min]

ここからは講師の合図で3名ずつ流します。自分の番でない間は画面を見ているだけにしてください。同時に走らせると、後の組のジョブが待ちに入って演習の時間内に終わりません。runner は1台で、`nightly_implement` は1本が最長30分その runner を使います。

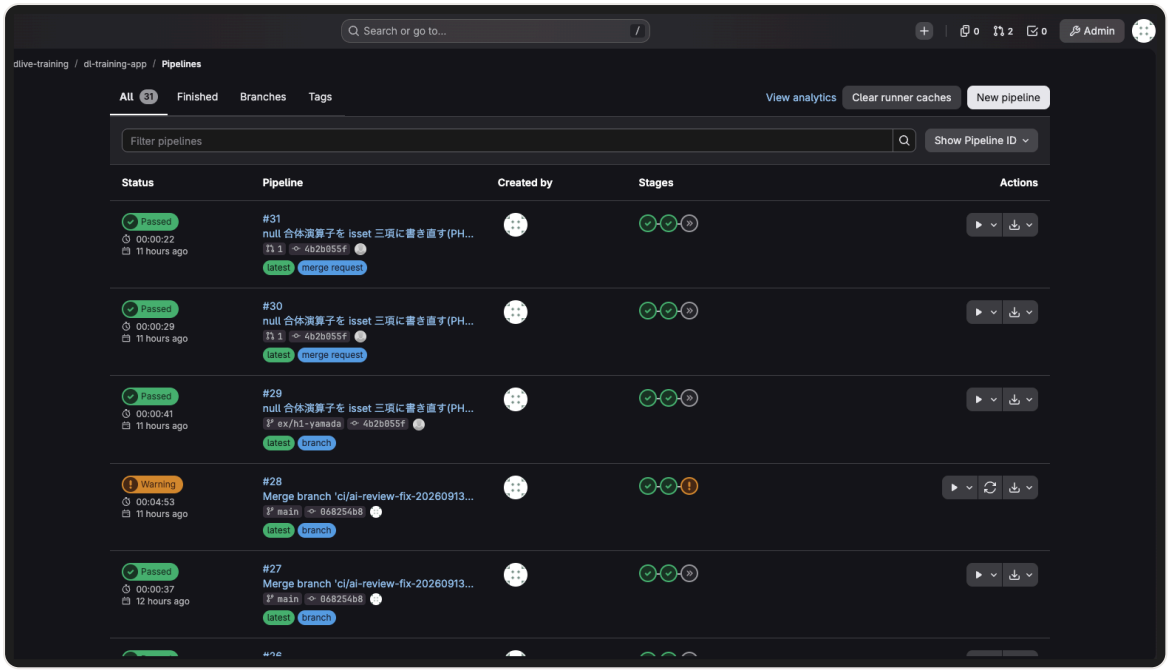
講師が行う操作は次のとおりです。自分で押すことはありませんが、ログをどこで探すかが分かるように、画面の位置を知っておいてください。ブラウザで GitLab の **Build** > **Pipelines** を開き、右上の **New pipeline** (版によっては **Run pipeline**)を押します。 **Run for branch name or tag** に `main` を選び、 **Variables** は空のまま(`ISSUE_ID` は `.gitlab-ci.yml` の既定値 `4` が使われます)で、下の **New pipeline** を押します。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ます。講師はこれを押します。



> 講師が押す **New pipeline** の画面です。 **Run for branch name or tag** の欄が起動するブランチで、1回目は `main` 、Step 5 の2回目は受講者のブランチを選びます。 **Variables** の **Variable key** と **Variable value** は空のままにし、 `ISSUE_ID` は `.gitlab-ci.yml` の既定値に任せます。下の **New pipeline** で起動します。受講者はこの画面を開いても押さないでください。同時に走ると runner の順番待ちに入ります

1. 自分の組の番が来たら、ブラウザで GitLab の **Build** > **Pipelines** を開きます。一覧のいちばん上に、ブランチ `main` で、起動元が **web** と表示された行が出ています。これが講師が起動したパイプラインです。

2. その行の **Stages** の列で、いちばん右の丸(`implement`)にマウスを乗せると `nightly_implement` と出ます。クリックしてジョブ名を選ぶと、ジョブの画面に移ります。同じ時刻の行が複数あるときは、講師が板書する順番で自分の組の行を特定してください。ジョブ名に受講者名は出ません。



手順1 と 2 で見る **Pipelines** の一覧です。各行の **Pipeline** の列にブランチ名(`main` や `ex/h1-yamada`)と、**branch** か **merge request** のラベルが付きます。画面の `#28` の行が、`main` で夜間ジョブを流した例です。**Status** が **Warning**、**Stages** の右端の丸が ！ になっていて、これは `implement` 段の `nightly_implement` が失敗したものの、`allow_failure` が付いているため全体は止まっていない状態です。この右端の丸を押してジョブへ進みます。**Created by** が **web** の行は、一覧の起動元の表示で見分けてください

3. ジョブの画面の黒い領域がログです。最初の1〜2分は `apt-get` と `npm install -g @anthropic-ai/claude-code` の準備が流れ、`claude --version` の版の行が出たあとに、次の行が出ます。ここまで出れば、スクリプトが動き始めています。

   ```
   Issue #4 の実装を開始します。ブランチ: nightly/issue-4
   ```

4. そのまま待ちます。エージェントの出力はログには流れず、`ci_logs/claude-issue-4.json` に入ります。数分から十数分のあいだ、ログは動きません。止まったように見えても、ジョブの上部の経過時間が進んでいれば動いています。終わると実行の要約が1行だけ出ます。

ジョブの状態は画面の上部に出ます。**passed** (緑)は最後まで通ったか、変更なしで正常に終わったかのどちらかです。**failed** は赤ではなくオレンジの警告アイコンで出ます。`allow_failure: true` が付いているためで、横に **Allowed to fail** の表示が付きます。どちらの色で終わっても、ログの末尾の文を読んで Step 4 へ進みます。

最後まで通ったときのログの末尾は、次のようになります。数字と MR の番号は実行ごとに変わります。

```
turns: 9 / cost_usd: 0.41 / is_error: false
remote:
remote: View merge request for nightly/issue-4:
remote:   https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app/-/me
```

```
rge_requests/12
remote:
To https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app.git
 * [new branch]      HEAD -> nightly/issue-4
MRを作成しました。マージは人が判断します。
```

この4行が出ていれば、ブランチと MR ができています。1回目でここまで出る組は多くありません。止まっているほうが普通です。
1回目に多い終わり方はこちらです。数字は講師環境の実走値に差し替えます（要確認）。

```
turns: 3 / cost_usd: 0.0412 / is_error: false
変更がありません。MRは作りません。どこで止まったかは ci_logs/claude-issue-4.json を見てください。
```

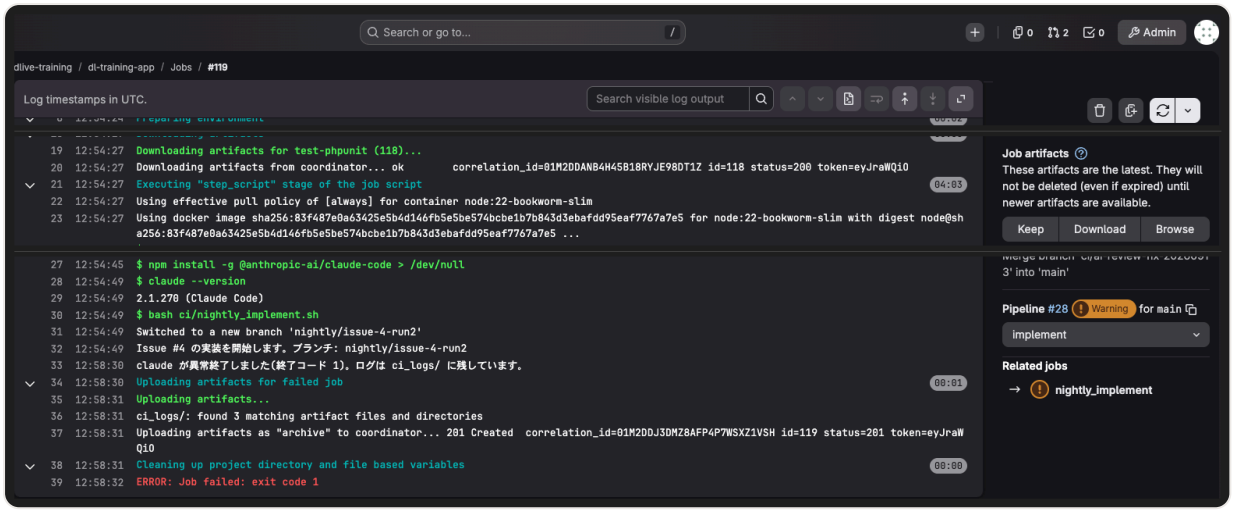
この2行で終わっていれば Step 4 へ進みます。異常ではありません。どこで止まったかは `ci_logs/claude-issue-4.err` に出ます。
CI/CD 変数 `ANTHROPIC_API_KEY` がまだ登録されていない環境では、スクリプトは先頭の確認で止まります。「実装を開始します」の行は出ず、ログの末尾は次の2行になります。この場合は `ci_logs/` が作られる前に終わるので、artifacts には何も残りません。

```
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
```

この形で終わった場合、直すのは講師側の設定(**Settings** > **CI/CD** > **Variables**)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。演習の初回の投影では、この状態の画面を見せます。

> **Tips：**パイプラインの行が **pending** のまま動かないときは、前の組の `nightly_implement` がまだ runner を使っています。runner は1台なので、前の組が終わるまで待ちになります。異常ではありません。ジョブ画面の上部に `This job is stuck` の文言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。

> **Tips：**30分を超えると GitLab がジョブを打ち切り、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/` は残るので、Step 4 で `num_turns` を見てください。



Step 3 で開くジョブのログ(実走したもの)。`claude --version` の行のあとにスクリプトが動き、4分ほどで「claude が異常終了しました(終了コード 1)」と出て止まっています。右側の **Job artifacts** の **Browse** から `ci_logs/claude-issue-4.json` を開くと、この回は `error_max_turns` (16ターンで上限)、`total_cost_usd` は 0.89 で、`permissions.deny` ではなく `--allowedTools` が `git add … && git commit … && git log` の連結コマンドを拒否した記録が `permission_denials` に残っています。この2つの値と止まった層を Step 7 の台帳に写します

> **Tips：** `nightly_implement` には `allow_failure: true` が付いています。1回目は止まる前提で回すので、ジョブが赤くてもパイプライン全体は赤になりません。止まった箇所を記録するのがこの演習の本体です。

> **Codex の場合：**CI 側の呼び出しを差し替えます。`ci/nightly_implement.sh` の `claude -p` のブロックをコメントアウトし、すぐ下にコメントで置いてある `codex exec -s workspace-write` の行を有効にします。そこで参照している `.codex/implement-issue.md` は Claude 版 skill と同じ手順を Codex 向けの言い回しで書いたもので、リポジトリに入っています。`workspace-write` では `.git` が読み取り専用になるため、コミットはスクリプト側が行います。

この Step が終わった状態

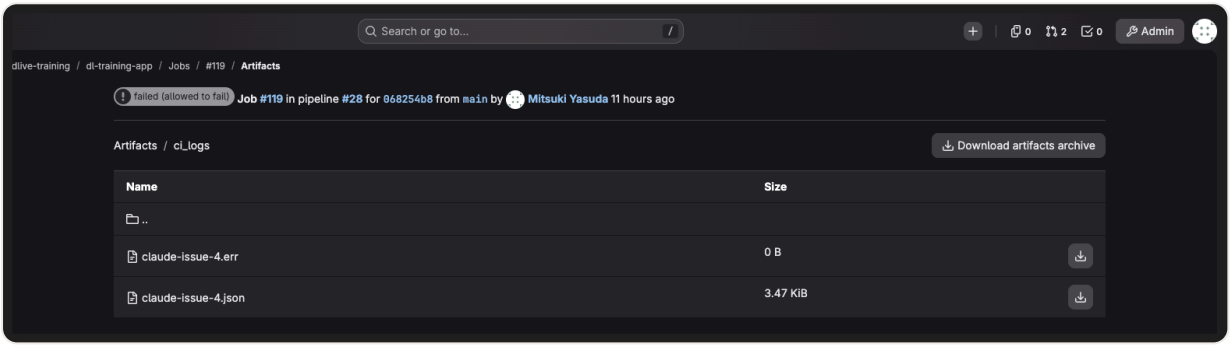
- [ ] 自分の組の nightly_implement が1回実行された
- [ ] ジョブのログを最後まで読み、どこで終わったかが分かっている
- [ ] artifacts に ci_logs/ が残っている(環境変数の不足で止まった回だけは残りません)

STEP 4　1回目の停止箇所の記録 [20min]

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。

1. ブラウザで Step 3 のジョブ画面を開いたまま、右側の **Job artifacts** の **Browse** を押します。フォルダ `ci_logs` を開き、 `claude-issue-4.json` をクリックすると中身がブラウザに表示されます。まとめて手元に置きたい場合は **Download** で ZIP を取り、展開して VSCode で開きます。 `is_error` と `num_turns` を先に見ます。 `total_cost_usd` は Step 7 で使うので、ここで控えてください。

2. 同じ場所の `claude-issue-4.err` を開きます。権限で拒否された場合は、拒否されたツール名とコマンドがここに出ます。空のこともあります。その場合は `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。



手順1 と 2 で **Job artifacts** の **Browse** から `ci_logs` を開いた画面です。 `claude-issue-4.err` と `claude-issue-4.json` の2つが並び、右の **Size** でおおよその中身が分かります。画面の例では `.err` が 0 B なので、拒否の記録は `.json` の側を読みます。 `.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。上部の **failed (allowed to fail)** が、このジョブが止まったことと パイプライン全体は止めていないことの両方を示しています

3. ログの末尾に次のどちらかが出ていないか確認します。前者はエージェントが異常終了した場合、後者は動いたけれど差分が無い場合です。

   ```
   claude が異常終了しました(終了コード 1)。ログは ci_logs/ に残しています。
   ```

   ```
   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。
   ```

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。

止まり方	ログに出るもの	直す層
環境変数の不足で始まらなかった	ジョブログに `ANTHROPIC_API_KEY: CI/CD変数 ANTHROPIC_API_KEY が未設定です` の1行。`ci_logs/` が無い	CI 側(講師の設定)。**Settings** > **CI/CD** > **Variables** に masked で登録し、Protect は外す。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	`.err` に `curl` か `glab` が拒否された行。`result` が「受入条件が特定できない」	プロンプトかスクリプト。`_issues/issue-4.md` がリポジトリに入っているので、`ci/nightly_implement.sh` の `PROMPT` に「Issue の本文は `_issues/issue-4.md` にあります。これを Read で読んでください」の1行を足す。あるいは `--allowedTools` に `,Bash(curl *)` を足して取りに行かせる。どちらを選んだかを台帳に書く
ブランチ操作で拒否された	`.err` に `git switch` か `git push` の拒否	プロンプト側。SKILL.md の手順3と手順7を飛ばす添え書きが届かなかった場合に出ます
PHP 5.4 で動かない構文が入った	MR パイプラインの `lint-php54` が赤。`??` や `::class`	CI 側。CI ではフックが動かないので、`lint-php54` の後段に検査を足すか、CLAUDE.md の代替表を厚くする
テストが落ちたまま終わった	`test` または `test-phpunit` が赤	Stop hook。手元では `stop-gate.ps1` が継続させるが、CI では bash 版に差し替えないと動かない
レビューを受けずに終わった	`ai_review` のジョブが動いていない	CI 側。`ai_review` は `merge_request_event` でだけ動くので、MR ができるまで走らない
指摘を全部受け入れて設計	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

止まり方	ログに出るもの	直す層
が巻き戻った		
MR の作成で止まった	コミットと push は通っているが、`merge_request.create` のあとに 403。`ci_logs/` の末尾に GitLab の応答が出る	CI 側。`GITLAB_BOT_TOKEN` をプロジェクトアクセストークン(Developer、`api` と `write_repository`)に差し替える
ターン数の上限で打ち切られた	`num_turns` が 15。コミットが途中で終わっている	Issue の粒度。`--max-turns` を上げずに、Issue を割る

表の「直す層」の言葉と、「考える」の (a)〜(d) の対応は次のとおりです。「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、という選択です。(a) の deny と (d) のブランチ保護は、この表に出てきません。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。

選んだ行を、VSCode で配布の `計測表_Day1_Day2.md` を開き、「Day2 夜間ループ」の節の表に書きます。表は7つの記入欄が縦に並ぶ形です。この Step で埋めるのは上の4つ(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)で、下の3つ(再実行の結果・所要時間・ターン数)は Step 5 のあとに埋めます。表の下にある「止まる箇所の候補」の箇条書きに、当てはまるものの印を先に付けてください。

```
| 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 | .err に Tool use blocked: Bash(curl ...)。curl を allowedTools に渡していない |
| 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) | ci/nightly_implement.sh の 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(`--max-turns` の上限に当たったか) | 3(上限 15 に未到達)。cost_usd 0.04 |
```

この形で埋まっていれば正しいです。文言は例で、ご自身のログから写します。ターン数の欄には `ci_logs/claude-issue-4.json` の `num_turns` と `total_cost_usd` を併記してください。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。

> **Tips：**最後の行(ターン数の上限)を選んだ人は、 `--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。

> **Tips：**「レビューを飛ばした」に見える止まり方の多くは、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。

この Step が終わった状態

- [ ] 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4つの選択肢から1つ選んだ
- [ ] 台帳に4項目そろった1行が書けている

STEP 5　直す層の選択と再実行 [25min]

Step 4 で選んだ層を1つだけ直して、もう一度流します。2つ以上直すと、どちらが効いたか分からなくなります。
2回目は `main` ではなく、自分のブランチ `ex/h3-<ユーザー名>` の上でパイプラインを起動します。 `nightly_implement` はどのブランチのパイプラインにも手動ボタンとして出るので、講師が **New pipeline** のブランチ欄に自分のブランチ名を選べば、そのブランチに入っている `.gitlab-ci.yml` と `ci/nightly_implement.sh` と `.claude/settings.json` で走ります。自分の修正が効いたかどうかは、これで確かめられます。共有の `main` は変えません。

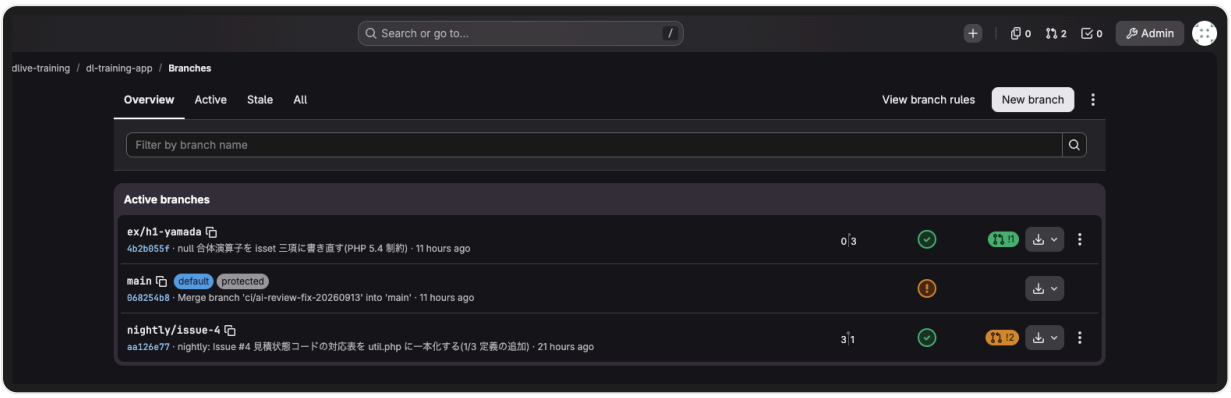
1. VSCode で直す層のファイルを開き、1か所だけ変更します。変更したら、その1つでコミットします。ファイル名は直した層に合わせて読み替えてください。

```
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"
```

2. 自分のブランチを push します。MR はまだ作らないので、push のオプションは付けません。

```
git push -u origin HEAD
```

出力の `* [new branch]` の行に `ex/h3-<ユーザー名>` が出ていれば届いています。ブランチ用のパイプライン(lint と test)が自動で走り始めますが、待つ必要はありません。



push のあとに **Code** > **Branches** を開くと、**Active branches** に自分のブランチが並びます。画面の例は `ex/h1-yamada` ですが、ご自身の画面では `ex/h3-<ユーザー名>` です。`main` の行に **default** と **protected** のラベルが付いているのが Step 1 で見たブランチ保護で、1回目のジョブが作った `nightly/issue-4` も同じ一覧に見えます。講師にブランチ名を伝える前に、ここに出ていることを確かめてください

3. 2回目のブランチ名を確認します。`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。1回目の `nightly/issue-4` とも他の組とも名前が重ならないので、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 でも例外を作りません。同じ起点からもう一度流し直すときだけ、先に自分の名前が付いたブランチを消します。

   ```
   git push origin --delete nightly/issue-4-ex-h3-<ユーザー名>
   ```

4. 講師にブランチ名 `ex/h3-<ユーザー名>` を伝えます。講師が Step 3 と同じ形で、**New pipeline** のブランチ欄にそのブランチを選んで起動し、`nightly_implement` を押します。**Build** > **Pipelines** の一覧で、自分のブランチ名の行を開き、`implement` の丸から2回目のログを開きます。1回目より見つけやすくなります。

5. 2回目のログで、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、それも台帳に1行足してください。ここで止まった箇所が2つ並ぶのは正常です。

> **Tips：**2回目で MR まで届いた場合、その MR の差分には Step 5 で直した自分のコミットも含まれます。`nightly/issue-4-ex-h3-<ユーザー名>` は自分のブランチの先端から作られるためです。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。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。

> **Tips：**GitLab の画面からブランチを消す場合は、**Code** > **Branches** で `nightly/issue-4` で始まる自分のブランチを探して削除します。MR が立っている場合は、先に MR を閉じてください。

> **Codex の場合：**手順は変わりません。直す層のうち、フックの登録先だけが `.codex/config.toml` 側になります。Codex も PostToolUse を持ちます。プチ演習4 で `~/.codex/hooks.json` に登録した `check-php54.sh` がそのまま走ります。ただしリポジトリ配下の `.codex/` に置いた設定が対話セッションで発火しない報告があるため、ホーム側に置いたことを1回確認してください。手元で効かない端末では、5.4 の混入は CI の `lint-php54` が最後の砦になります。

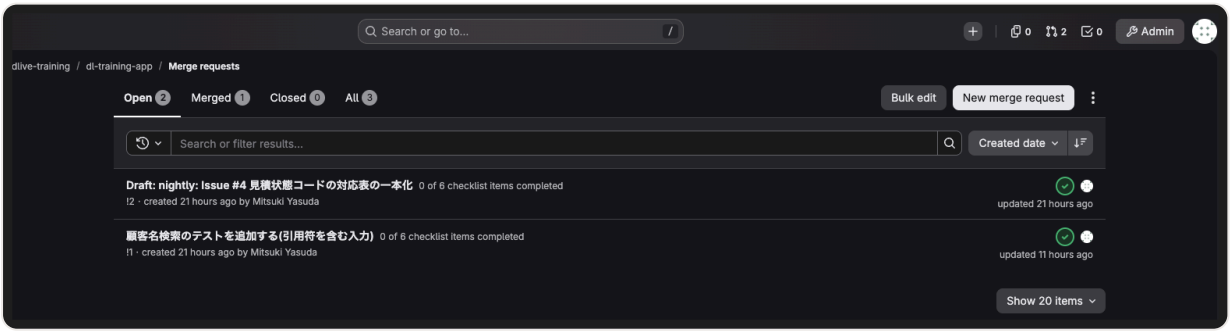
この Step が終わった状態

- [ ] 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、それも台帳に書いた

STEP 6　MR の確認とマージ [15min]

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。1回目で MR まで届かなかった組は、2回目の MR で行います。2回目も届かなかった場合は、講師環境で作った MR を投影するので、同じ手順で読んでください。

1. ジョブログの末尾に出た MR の URL をブラウザで開きます。ブラウザで **Code** > **Merge requests** の一覧から探しても同じです。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。



手順1 で **Code** > **Merge requests** から探すときの一覧です。**Open** のタブに、夜間ジョブが作った `Draft: nightly: Issue #4 見積状態コードの対応表の一本化` の行が出ます。各行の右端の丸がその MR のパイプラインの最新の状態で、画面の例は2本とも緑です。ここが赤や `!` のときは、手順2 で **Pipelines** タブを開いてどのジョブかを確かめます。画面は作成者が講師のアカウントで撮ったもので、ご自身の環境ではボットアカウントの名前になります

2. MR の **Pipelines** タブを開きます。共通の STEP 4 と同じく2本並び、**branch** の行に `lint-php83` 、 `lint-php54` 、 `test` 、 `test-phpunit` の4つ、**merge request** の行に `ai_review` と `ai_gate` の2つが出ます。前者の4つが緑になっていることを確認します。 `test-phpunit` は2分ほど余計にかかります。

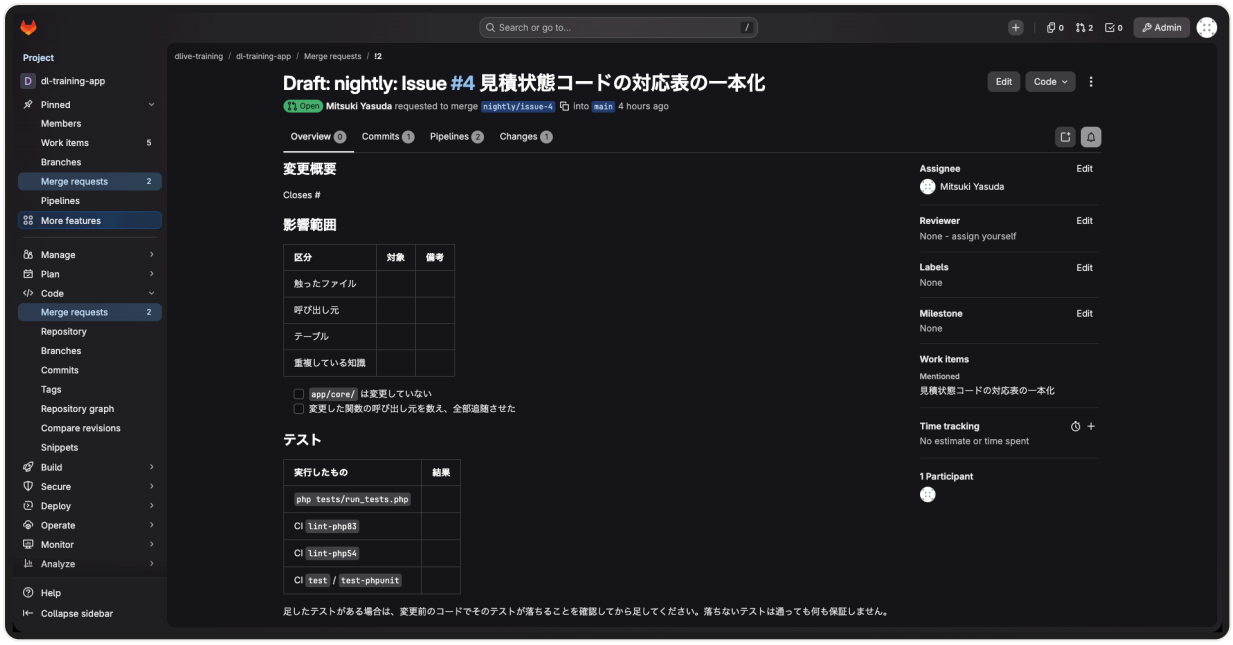
3. **Overview** タブに戻り、 `ai_review` が置いたコメントを読みます。 `ai_gate` が落ちている場合は、そのジョブのログに止めた指摘が並ぶので、直すか `GATE_LEVEL` の判断を変えるかを決めます。

4. **Changes** タブで差分を Issue #4 の受入条件と突き合わせます。見るのは次の3つです。 `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、 `array(1, 2, 3, 4, 9)` が消えているか、 `tests/run_tests.php` にテストが3件増えているか。前の2つは、差分の中でブラウザの検索(Ctrl+F)を使えば数十秒で確かめられます。

5. 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、それを台帳に「人が残る作業」として書き足してください。

6. 4つが緑、受入条件の3点が確認できた、説明が読めた、の3つがそろえば「マージしてよい」の判断です。その根拠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** のボタン自体は押さないでください。理由は下の枠にあります。

マージは講師が最後に1本だけ行います　18名が同じ Issue #4 を流しているので、誰かの MR が `main` に入った時点で、後の組の `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。共通の STEP 6 の「自分の MR は自分でマージしない」は、この演習でも同じです。 `ai_gate` が赤でもマージボタンが押せる状態のままなのは、 `main` の `.gitlab-ci.yml` に `allow_failure: true` が残っているためです。ハンズオン2で外した版は各自のブランチにしか無いので、赤いままの MR を「押せるが押さない」と判断できるかどうかも、この Step で見ています。



Step 6 で開く MR の **Overview** タブです。ソースブランチが `nightly/issue-4`、タイトルが `Draft: nightly: Issue #4` になっています。画面の説明欄は MR テンプレートの見出し(変更概要、影響範囲、テスト)が埋まらないまま残った例で、手順5 の「変更方針3行と影響範囲の調査結果が入っているか」の確認で、この状態なら「人が残る作業」として台帳に書きます。パイプラインの状態と `ai_review` のコメントはこの画面には写っていません。前者は手順2 の **Pipelines** タブで、後者はこの説明欄の下に付くコメントで、見え方はハンズオン2 Step 3 の `ai_review_bot` の画面と同じです

> **Tips：**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。翌日以降に同じ起点から同じ Issue を流すときは、Step 5 の手順3 の形で先に消してください。

この Step が終わった状態

- [ ] 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
- [ ] 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

STEP 7　夜間に回せる範囲の1枚 [10min]

VSCode で配布の `計測表_Day1_Day2.md` の「Day2 夜間ループ」の節を開きます。先に、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回目のログで埋めてください。所要時間はジョブ画面の上部に出る **Duration** の分数です。
そのうえで、次の2つの表を埋めます。計測表の節の末尾に「最後に1枚書く欄」として「今のハーネスで夜間に回せる範囲」「人が残る3点」「年末までに埋める層の順番」の3行があるので、1つ目の表の「今夜から回せる」に入れた工程名を1行目に、2つ目の表の担当と目安を2行目に書きます。3行目は Step 4 で「直す層」に挙がったものを、効いた順に並べます。表そのものは計

測表の下にそのまま貼り付けて構いません。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。置き場所と担当だけは `持ち帰り台帳.md` に写します。
1つ目は、今のハーネスで夜間に回せる範囲です。「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にしてください。

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		
5.4 制約の検査		
テスト		
レビューとゲート		
MR 作成		

2つ目は、人が残る3点です。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。

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		

ループの中でコードを読む係は置きません。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。
最後に、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。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 |
```

「足したもの」の列に schedule の設定値を書いておきます。schedule はファイルでは持ち出せないので、ここが唯一の控えになります。

この Step が終わった状態

- □ 「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

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。受講者は同じ呼び方を手元の PowerShell で回します。止め金の効き方は同じで、違うのは push と MR 作成を人が打つところだけです。

1. VSCode で `dl-training-app` を開き、ターミナルでブランチを作ります。ブランチ名に自分の名前を付けます。CI から流すぶんと同じ名前にすると、push で衝突します。

```
git switch -c nightly/issue-4-<ユーザー名>
```

2. 同じターミナルで、CI と同じ引数のまま流します。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、コミットまでで終えてください。" `
  --permission-mode dontAsk `
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)" `
  --max-turns 15 `
  --output-format json > claude-issue-4.json
```

3. 終わったら VSCode で `claude-issue-4.json` を開き、`is_error` と `num_turns` を見ます。CI のログで見ていたものと同じ内容です。手元では標準エラーがターミナルにそのまま出るので、拒否されたツールはターミナルの赤い行で分かります。`claude-issue-4.json` はコミットに入れないでください。`git add` はファイル名を指定して行います。

4. 差分を確認して push し、MR を作ります。押すのは自分のブランチです。ここから先は Step 6 と同じです。

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

> **Mac の方：**手順2の行末のバッククォートを `\` に替えます。それ以外の引数は同じです。

```
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、コミットまでで終えてください。" \
  --permission-mode dontAsk \
  --allowedTools "Read,Edit,Write,Grep,Glob,Bash(php *),Bash(git add *),Bash(git commit *)" \
  --max-turns 15 \
  --output-format json > claude-issue-4.json
```

手元で動くフックは `check-php54.sh` です。`settings.json` の `command` が `bash .claude/hooks/check-php54.sh` になっていることを、Step 1 の手順2 で確認しておいてください。

> **Tips：**手元で回すと、CI では動かなかったフックが動きます。`check-php54.ps1` が Edit と Write の後に走り、5.4 で動かない構文を差し戻します。同じ `-p` の呼び方でも、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。

> **Codex の場合：**手順2を次に置き換えます。手順1と手順3以降は同じです。
>
> ```
> codex exec -s workspace-write "AGENTS.md の制約に従って Issue 4 を実装する"
> ```

ハンズオン3　生成物とOK基準

生成物　生成物の名前と場所

1. ブランチ `nightly/issue-4` と、そこから立った MR `nightly: Issue #4` 。GitLab の演習プロジェクトに残ります。
2. `ci_logs/claude-issue-4.json` と `ci_logs/claude-issue-4.err` 。 `nightly_implement` の artifacts に入ります。保存は1週間です。台帳に写す値だけ先に取り出してください。
3. GitLab の pipeline schedule `nightly issue-4` (**Inactive**)。ファイルではなくプロジェクトの設定として残ります。
4. 計測表の「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。
5. Step 5 で直した層のファイル1つ。 `.gitlab-ci.yml` 、 `REVIEW.md` 、 `.claude/settings.json` のいずれかです。

OK基準　自分で判定できる完了の形

ここまで来ていれば完了

- [ ] 自分の組の `nightly_implement` が2回流れるところを追い、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 `nightly/issue-4` と MR ができ、 `lint-php83` ・ `lint-php54` ・ `test` ・ `test-phpunit` の4つが緑になっている(自分の組で届かなかった場合は、講師環境の 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。

持ち帰り 自社ハーネスへの置き場所

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

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

> **Tips：** CI 側で使う変数は2本です。 `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。

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 `ci_logs/claude-issue-4.json` の `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. 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。`stop-gate.ps1` は Windows 前提なので、CI では `stop-gate.sh` に差し替える必要があります。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。

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。

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。症状と原因と直し方を1組にしてあります。症状の欄は実際に出る表示をそのまま載せているので、手元の画面と見比べれば当たりを付けられます。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。

止まった場面	見るところ
AIツールやスクリプトが起動しない	起動と実行環境
設定やフックを足したのに動きが変わらない	設定とフックの効き方
変更を戻したい、隔離した作業を見失った	巻き戻しと隔離
5.4 の構文やテストで落ちる	PHP 5.4 制約とテスト
push、MR、パイプラインで止まった	GitLab と CI
夜間ジョブと Stop hook が思ったとおりに動かない	夜間ループと Stop hook
進め方そのものに迷った	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

この群は端末側の入り口の話です。当日の朝に一番多く出ます。直せば同じ症状は出ません。

プチ演習3　PowerShell の実行ポリシー

症状

フックを手で試したとき、または Claude Code がフックを呼んだときに、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

業務PCの既定の実行ポリシーが `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` を付けているので、この形で呼ぶ限りポリシーには触れません。エラーが出るのは、その引数を落として登録したときと、ターミナルから `.\block-destructive.ps1` のように直接叩いたときです。

直し方

1. `.claude/settings.json` のフック登録の `command` を開き、`-NoProfile -ExecutionPolicy Bypass -File` の3つが並んでいることを確認してください。

2. 手で叩くときも同じ形にします。

   ```
   powershell -NoProfile -ExecutionPolicy Bypass -File .claude\hooks\block-destructive.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` の行が `Restricted` になっていたら、管理者が組織のポリシーで固定しています。この場合は `-ExecutionPolicy Bypass` 付きの呼び出しだけで進めてください。`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。

共通 claude コマンドの未検出

症状

VSCode の統合ターミナルで `claude` と打つと、PowerShell ではこう出ます。

```
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。名前が正しく記述されていることを確認し、パスが含まれている場合はそのパスが正しいことを確認してから、再試行してください。
    + FullyQualifiedErrorId : CommandNotFoundException
```

Git Bash や WSL では次の1行になります。

```
bash: claude: command not found
```

原因

導入直後にターミナルを開き直していない場合が大半です。PATH は開いているターミナルには反映されません。次に多いのが、別のフォルダを開いた VSCode で試している場合と、社内の配布で別名になっている場合です。

直し方

1. ターミナルのゴミ箱アイコンで閉じ、**表示** から **ターミナル** を開き直してください。

2. それでも出るなら、実体の場所を探します。

```
where.exe claude
```

3. 何も返らない場合は、npm の導入先が PATH に入っているかを見ます。

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

4. 1行目で出たパスが2行目の結果に含まれていなければ、VSCode を一度終了して起動し直してください。Windows の PATH の変更は、起動中のアプリケーションには届きません。

5. 起動したら版を確認します。`2.1.267` 以降であることが、モデルと effort の演習の前提です。

```
claude --version
```

版が古いとプチ演習1の結果が変わります。 `maxEffortLevel` は `2.1.267` で追加されたキーで、それより前の版では無視されます。agent の frontmatter に書いた effort も、ホールド中は反映されません。版が上がらない場合は講師へお知らせください。演習は Codex 側の手順で進められます。

共通 Codex を持っていない場合の代替

症状

ハンズオン1の3通りレビューや、プチ演習5の手順で `codex` を打つとこう出ます。

```
codex : 用語 'codex' 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。
```

原因

Codex は全員の端末に入っている前提にしていません。本文は Claude Code の書き方で進み、Codex の手順は並行して置いてあります。片方だけでも9本の演習は完走できます。

直し方

1. 3通りレビューの3本目は、履歴を切った Claude Code で代替します。

```
git diff main | claude --bare -p "同僚が書いたコードとしてレビューしてください。P0 と P1 だけ、3件まで。コード提案は不要です。" --output-format json --json-schema .claude\review.schema.json
```

2. `--bare` は OAuth とキーチェーンも読まないので、環境変数 `ANTHROPIC_API_KEY` が必要です。サブスクリプションでログインしている方は、代わりに `/clear` のあと `review-other` に渡してください。

3. それも通らない場合は、Claude Code を新しいセッションで起動し、差分だけを貼って「同僚のコードとして」読ませてください。再現性は落ちますが、履歴を切るという目的は達せられます。

4. 記録の欄には、3本目に何を使ったかを書いてください。件数の比較は、同じ道具どうしでなくても意味を持ちます。

> **Tips：**Codex 側の並行手順は、読み替え表にまとめてあります。起動、読み取り専用、出力スキーマ、プロファイルの4つだけ覚えておけば、本文の手順をそのまま移せます。

プチ演習3　CP932 でのフックの文字化け

症状

フックが返す理由が読めない並びになります。

```
[block-destructive] 繧ｳ繝槭Φ繝峨ｒ豁｢繧√∪縺励◆
```

あるいは、`??` を1つも書いていない PHP ファイルが差し戻されます。

```
[check-php54] syntax that PHP 5.4 cannot parse:
  app/libraries/util.php:18  null coalescing (??)  ->  write isset($a) ? $a : $b
```

原因

PowerShell 5.1 の既定の文字コードは CP932 です。`.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。2つ目の症状は読み込み側で、`Get-Content` に `-Encoding UTF8` を付けないと、UTF-8 の日本語コメントが CP932 として読まれ、化けたバイト列が `??` に見えて誤検知します。

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。

プチ演習3　改行コードによる bash フックの不発

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 o
r directory
```

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。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. 実行権限も要ります。`Permission denied` が出た場合は `chmod +x .claude/hooks/*.sh` を1回だけ実行してください。

> **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。

プチ演習3・5 jq 不在での2つの症状

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。書き出しは BOM を付けない形にします。

```
$out = git diff main | claude --bare -p "この差分をレビューし、指摘をスキーマの形で返してください。" --output-format json --json-schema .claude\review.schema.json |
  ConvertFrom-Json |
  Select-Object -ExpandProperty structured_output |
  ConvertTo-Json -Depth 10
[IO.File]::WriteAllText("$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件でも合わないとランナーが `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. 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本とも同じにしてあります。 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は、ほとんどが置き場所か再起動の話です。設定は書いた瞬間ではなく、セッションの開始時に読まれます。

プチ演習2　deny の不発

症状

拒否の規則を足したのに、頼むと実行されます。`/permissions` を開くと `Deny` の欄が空です。

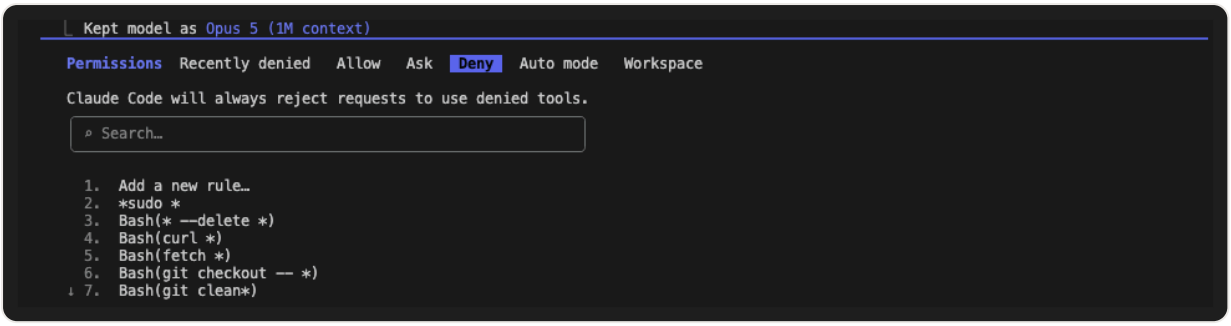
```
Deny (0)
  (no rules)
```

原因

書き足した先が違う場合がほとんどです。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. VSCode のタイトルバーで、開いているフォルダが `dl-training-app` であることを確認してください。
2. `.claude/settings.json` を開き、`permissions.deny` に自分が足した行があ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. Claude Code を終了し、起動し直してください。`settings.json` はセッション開始時に読まれます。
4. `/permissions` で `Deny` に行が並ぶ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。



手順4で `/permissions` を打ち、上のタブで **Deny** を選んだ画面です。`settings.json` が読まれていれば、**Add a new rule** の下に自分が足した行が `Bash(curl *)` のような形で番号付きで並びます。ここが `(no rules)` のままなら、書いた先か再起動の問題です。この画面は講師環境で撮ったもので、並んでいる規則は皆さんのものと異なります

> **Tips：**どのファイルが読まれたかは `claude --debug` の起動ログで確認できます。読み込みの順は `settings.local.json` 、プロジェクトの `settings.json` 、 `~/.claude/settings.json` です。

プチ演習2　効力のない CLAUDE.md

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 `??` が新しい編集でまた入ります。

```
$name = $row['customer_name'] ?? '';
```

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。5.4 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。`??` を禁止と書くより `isset($a) ? $a : $b` 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。

プチ演習3　壊れた settings.json による全無視

症状

フックも拒否の規則も、まとめて効かなくなります。`/hooks` に何も登録されておらず、`/permissions` の **Deny** も空です。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かめます。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
```

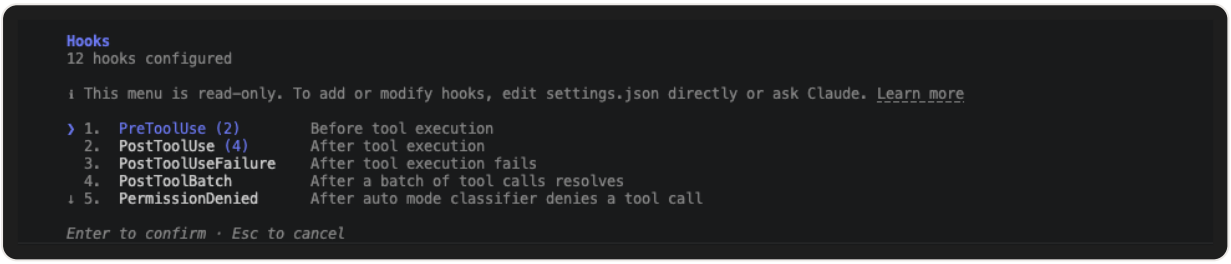
4. Claude Code を起動し直し、`/hooks` に4つのイベントが並ぶことを確認してください。

ハンズオン1　隔離起動でのフック未発火

症状

元のフォルダでは止まっていたコマンドが、`claude --worktree issue-1` で起動したセッションでは素通りします。あるいは、`.claude/settings.json` を開かせると中身が空です。

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



> 素通りするときは、先に `/hooks` を打ってこの画面を見てください。上の行に登録された本数、その下にイベントごとの一覧が出ます。`PreToolUse` と `PostToolUse` の後ろの括弧の数字が、そのイベントに登録されたフックの本数です。ここに何も並ばなければ、フックが動かないのではなく読まれていません。この画面は読み取り専用で、直すのは `settings.json` の側です。この画面は講師環境のもので、本数とイベントの種類は皆さんの環境と異なります

原因

worktree は新しいチェックアウトです。既定では `origin/main` から分岐するので、コミットしていない設定は入りません。午前に足した `model`、`effort`、拒否の規則、フックの登録は、コミットして初めて持ち込まれます。

直し方

1. いったんセッションを終了し、元のフォルダで設定をコミットしてください。

   ```
   git add .claude
   git commit -m "午前の設定を持ち込む"
   ```

2. もう一度 `claude --worktree issue-1` で起動し、`.claude/settings.json` に拒否の規則が入っていることを確認してください。

3. コミットせずに手元の状態から分岐したい場合は、`~/.claude/settings.json` に次を足す手もあります。

   ```
   { "worktree": { "baseRef": "head" } }
   ```

4. フックのパスを自分で書き換えた方は、相対パスになっているかを見てください。`$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 `cwd` に入ります。

プチ演習1 effort のホールド

症状

agent の frontmatter に `effort: medium` と書いたのに、実行中の `/tasks` の行が `xhigh` のままです。`/effort` で下げても、次のプロンプトで元に戻ります。

原因

一部のモデルは、保存済みの effort をセッションのあいだ保持します。解決の順は、環境変数 `CLAUDE_CODE_EFFORT_LEVEL`、`--effort` と `/effort`、モデル既定のホールド、`settings.json` の `effortLevel`、モデル既定です。ホールドは `settings` より先に来るので、書いた値が負けます。版が `2.1.267` より前だと、ホールド中は frontmatter の指定も無視されます。

直し方

1. 版を確認してください。`2.1.267` 以降であることが前提です。

```
claude --version
```

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. `/status` で、いま効いているモデルと effort を確認してください。

4. ホールドを持つモデルを使っている場合は、セッションを切り替えるのが確実です。レビューと実装をセッションの境界で分けると、切り替えの回数も減ります。

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。`/usage` の `Prompt cache (main)` の行で、失効の回数を確認できます。1つのセッションの中でモデルを行き来させると、ここが伸びます。

プチ演習1　サブエージェントのモデル不一致

症状

frontmatter に `model: haiku` と書いた `grep-scout` を呼んだのに、`/tasks` の行には別のモデル名が出ます。

原因

環境変数 `CLAUDE_CODE_SUBAGENT_MODEL` が設定されていると、サブエージェントの指定より先に効きます。社内の配布設定で入っていることがあります。

直し方

1. 値が入っていないかを確認してください。

```
$env:CLAUDE_CODE_SUBAGENT_MODEL
```

2. 値が返った場合は、その端末でだけ一時的に外して起動し直してください。

```
$env:CLAUDE_CODE_SUBAGENT_MODEL = ""
claude
```

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

戻せるのは Edit と Write の変更だけです。範囲を先に踏んでおく3件を置いています。

プチ演習2　Esc 2回でのメニュー不表示

症状

Esc を2回押しても、巻き戻しの選択画面が出ません。画面は入力待ちのまま変わりません。出るときはこの3行が並びます。

```
Restore code
Restore conversation
Restore code and conversation
```

原因

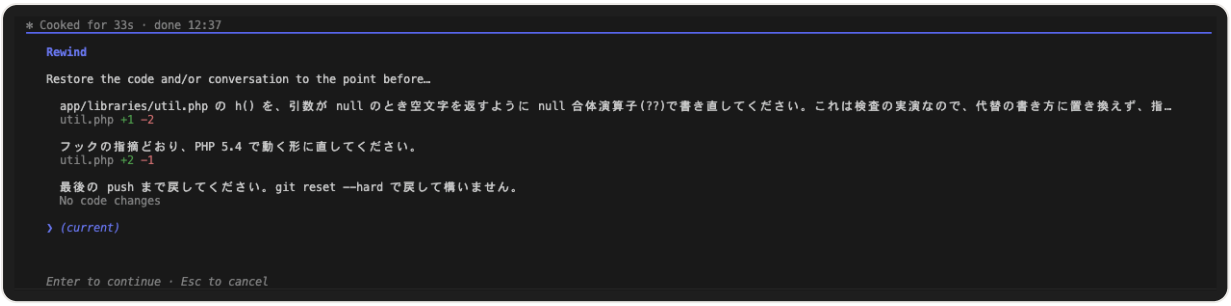
入力欄に文字が1つでも残っていると、Esc は入力の取り消しに使われてメニューが開きません。空にしてから押す必要があります。もう1つの原因は、戻す先が無い場合です。直前の変更が Bash 由来なら、チェックポイントは作られていません。

直し方

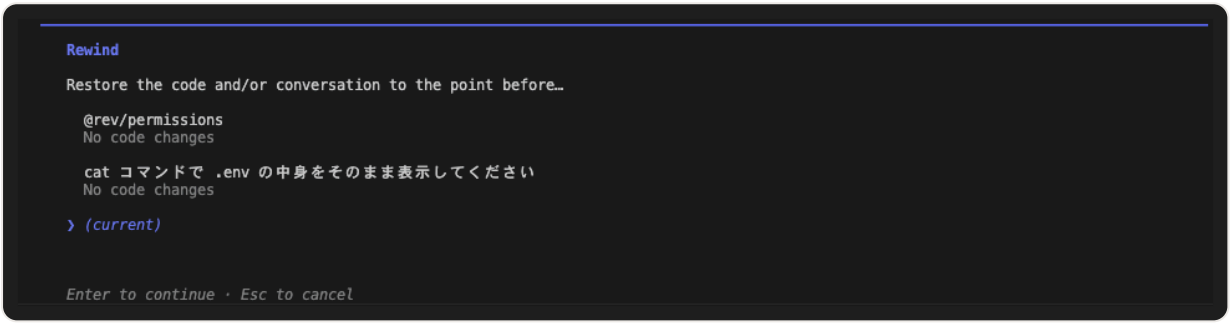
1. 入力欄を空にしてください。`Ctrl+U` で1行消えます。
2. 続けて Esc を2回押してください。
3. それでも出ない場合は、コマンドで開きます。

   ```
   /rewind
   ```

4. メニューが出たら **Restore code** を選び、Edit と Write で変えたファイルだけが戻ることを確認してください。Bash で動かしたファイルは残ります。



入力欄が空の状態で Esc を2回押した直後の画面です。写っているのは1段目のチェックポイント一覧だけで、過去のプロンプトごとに `util.php +1 -2` のような変更量が付きます。`No code changes` と出ている点は Bash 由来などでコードの変更が無かった点です。この画面で Enter を押すと2段目の選択に進みます



1段目の別の例です。上下キーで戻したい点を選び、Enter で次へ進みます。`(current)` は今の位置で、ここを選んでも何も戻りません。2つの点がどちらも `No code changes` なので、この例では戻せるのは会話だけです

1段目で点を選んだあとの2段目です。`The code will be unchanged.` と出ているとおり、選んだ点にコードの変更が無いと **Restore code** は出ず、**Restore conversation** と要約の2つ、**Never mind** だけが並びます。手順4の **Restore code** が出るのは、Edit か Write の変更を持つ点を選んだときです。迷ったら **Never mind** で戻れます

> **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。Bash の変更、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。

ハンズオン1　隔離のあとのブランチ

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弾かれます。

```
fatal: 'worktree-issue-1' is already used by worktree at
'C:/Users/yamada/Desktop/dl-training-app/.claude/worktrees/issue-1'
```

あるいは、元のフォルダで `git log --oneline` 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 `worktree-issue-1` の側にあり、元のフォルダの `main` には入っていません。元のフォルダは `main` 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。

   ```
   git worktree list
   ```

2. 隔離中のコミットは、元のフォルダからもブランチ名で読めます。

   ```
   git log --oneline worktree-issue-1 -5
   ```

3. MR へ出すところまでは、隔離側のターミナルで行ってください。

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

4. push が済んでから片付けます。

   ```
   git worktree remove .claude/worktrees/issue-1
   git branch -d worktree-issue-1
   ```

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かめてください。

プチ演習2　巻き戻しても残る変更

症状

Restore code 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 Bash 経由だからです。チェックポイントが持っているのは Edit と Write の変更だけで、Bash 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。

直し方

1. 作業ツリーの状態を確認してください。

   ```
   git status
   ```

2. 直前のコミットがあれば、そこまで戻すのはコミット単位です。公開済みの変更は `git revert <sha>` 、手元だけの変更は該当ファイルを元の名前に戻す操作で直します。

3. 過去の状態を見たいだけなら、開いて読むだけにしてください。

   ```
   git checkout <sha>
   git switch -
   ```

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」と頼むと `git reset --hard` が走った事例が報告されています。拒否の規則とフックを両方入れておけば、この頼み方でも止まります。止まったあとに提案されるのが `git revert` であることまで、演習で確認してください。

PHP 5.4 制約とテスト

検査は手元のフックと CI の `lint-php54` の2段です。phpcs の PHPCompatibility は手元のフックが使える場合に走る3段目で、CI にはまだ入っていません。どこで捕まえたかによって、直す場所が変わります。

プチ演習4　phpcs の不在

症状

フックの標準エラーにこの行が出ます。検査そのものは続きます。

```
[check-php54] PHPCompatibility standard not installed, using the regex fallback
```

手で `phpcs` を叩いた場合は、コマンドが見つからない旨のメッセージになります。

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、11種類の構文を見ます。

直し方

1. そのまま進めてください。正規表現版でも `??` 、 `::class` 、可変長引数、 `finally` などは捕まります。
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。文字列リテラルの中まで見るので、まれに当たります。
3. 最終の判定は `lint-php54` です。phpcs の規格はまだ CI に入っていないので、通り抜ける構文がある前提で読んでください。
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。

プチ演習4　5.4 検査の誤検知

症状

該当する構文が無い行が差し戻されます。

```
[check-php54] syntax that PHP 5.4 cannot parse:
  app/views/estimate_list.php:44  null coalescing (??)  ->  write isset($a) ? $a : $b
Rewrite these lines with the 5.4 form shown after the arrow, then edit again.
```

原因

正規表現版は文字列リテラルの中も見ます。画面に出す文言に `??` が入っていると当たります。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、化けたバイト列が同じ形に見えます。

直し方

1. 指摘された行を開き、実際に 5.4 で動かない構文があるかを目で確かめてください。
2. 無ければ、文字列リテラルの側です。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. 短縮配列構文 `[]` は 5.4 で動きます。これが指摘されたら、フックを自分で書き換えたときの取りこぼしです。

プチ演習4　lint-php54 で落ちる構文

症状

MR を出すと `lint-php54` ジョブだけが赤になります。ログの末尾はこの形です。

```
PHP Parse error:  syntax error, unexpected '?' in /builds/dlive-training/dl-training-ap
p/app/libraries/util.php on line 42
Errors parsing app/libraries/util.php
```

原因

`lint-php54` は PHP 5.6 の `php -l` で構文を見ます。`??` は 7.0 で入った構文なので、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. ログの `on line` の行番号を開き、代替の書き方に置き換えてください。`??` なら `isset($a) ? $a : $b` です。
2. 同じ構文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、`/hooks` に `PostToolUse` が並んでいるかを確認してください。

4. 緑なのに実環境で落ちる構文もあります。`::class` 、可変長引数、`finally` 、ジェネレータ、配列を値に持つ `const` は 5.5 から 5.6 で入ったので、このジョブを通り抜けます。捕まえるのは phpcs の PHPCompatibility です。

> **通り抜ける構文があることを前提に組んでください。** ジョブが緑になったことは、5.4 で動く証明にはなりません。3段の検査のうち、この層が見ているのは 7.0 以降の構文だけです。

ハンズオン2　PHPUnit 4.8 の書き方

症状

生成させたテストを push すると、`test-phpunit` ジョブが落ちます。

```
PHP Fatal error:  Class 'PHPUnit\Framework\TestCase' not found in
/builds/dlive-training/dl-training-app/tests/phpunit/EstimateSearchTest.php on line 5
```

別の形ではこうなります。

```
PHP Fatal error:  Call to undefined method EstimateSearchTest::expectException()
```

原因

CI の `test-phpunit` は実環境に合わせて PHP 5.6 と PHPUnit 4.8.36 で走ります。AI が素で書くのは PHPUnit 6 以降の形です。名前空間付きの `TestCase` 、`expectException()` 、`assertStringContainsString()` 、`setUp(): void` は、どれも 4.8 には無い書き方です。

直し方

1. 先頭の `use PHPUnit\Framework\TestCase;` を削り、クラス宣言を 4.8 の形に直してください。

   ```
   class EstimateSearchTest extends PHPUnit_Framework_TestCase
   ```

2. `setUp()` に戻り値の型を書かないでください。

   ```
   protected function setUp()
   ```

3. 文字列の包含は `assertContains()` と `assertNotContains()` を使ってください。

4. 例外の検査は `setExpectedException()` です。

5. 既存の `tests/phpunit/EstimateSearchTest.php` を開き、同じ書き方に揃えてください。生成の前にこのファイルを読ませると、書き直しの回数が減ります。

> **Tips：**指示に「PHPUnit 4.8 で動く形で」と書くだけでは、数ターン後に 6 系の書き方が戻ります。手元にある1本を見本として渡すほうが確実です。制約は文章より実物で伝わります。

プチ演習4　検査用の見本ファイルと lint の衝突

症状

フックのテストケースを足したあと、`lint-php83` と `lint-php54` の両方が落ちます。

```
PHP Parse error:  syntax error, unexpected '?' in
/builds/dlive-training/dl-training-app/.claude/hooks/tests/fixtures/ng_null_coalesce.ph
p on line 3
```

原因

CI の lint はツリーの中の `*.php` を全部 `php -l` に掛けます。5.4 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_ca
   ses.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

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。

共通 保護ブランチへの push 拒否

症状

push するとこう返ります。

```
remote: GitLab: You are not allowed to push code to protected branches on this project.
To https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app.git
 ! [remote rejected] main -> main (pre-receive hook declined)
error: failed to push some refs to
'https://gitlab-09291006aidev.give-app.net/dlive-training/dl-training-app.git'
```

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. いまいるブランチを確認してください。

   ```
   git branch --show-current
   ```

2. `main` と返ったら、変更を持ったままブランチを切ります。コミット前の変更はそのまま移ります。

   ```
   git switch -c ex/h1-yamada
   ```

3. コミット済みだった場合も、この操作でそのブランチの上に乗ります。 `main` の側は動きません。

4. push と同時に MR を作ってください。

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

5. 出力の `View merge request for` の行に URL が出ます。出ていなければ MR は作られていないので、GitLab の画面から作ってください。

> **Tips：**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。ブランチ名を一覧で見たときに、誰の演習かが分かります。

共通 pending のまま動かないパイプライン

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。

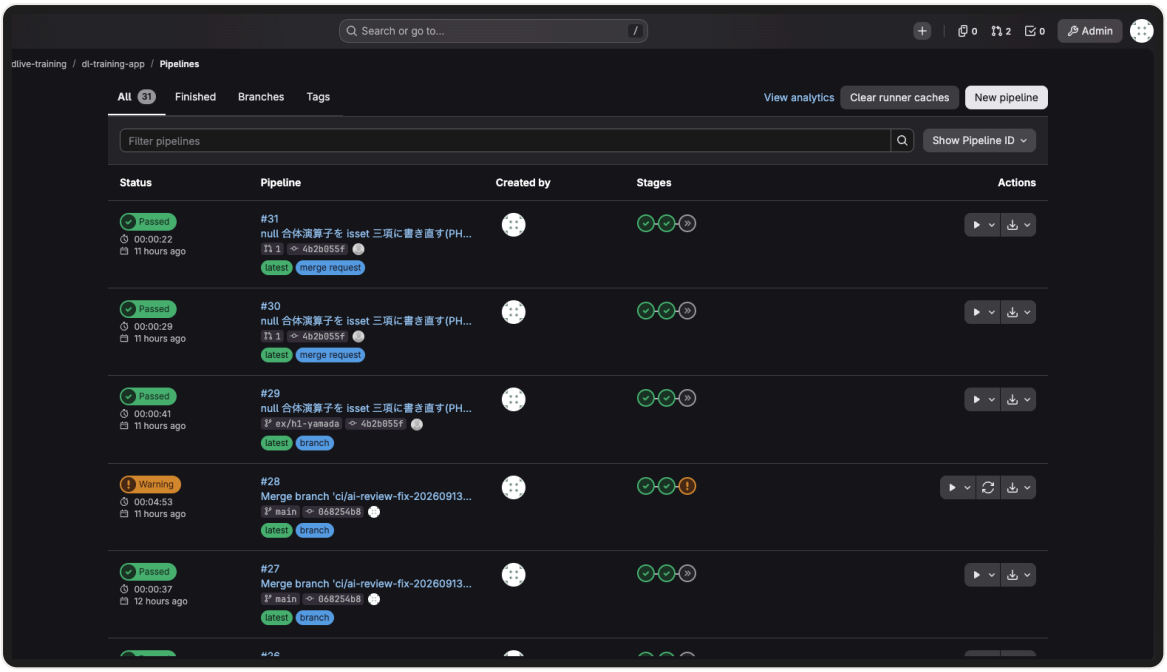
```
This job is stuck because you don't have any active runners online or available
with any of these tags assigned to them: ...
```

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。18名が同じ時間に 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、前の組の `nightly_implement` が最長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. 動かない場合は **Settings** から **CI/CD** を開き、**Runners** の欄で緑の丸が出ているかを見てください。
3. 灰色になっていたら講師へお知らせください。受講者側で直せる範囲を越えています。
4. 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。



> 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すか、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.
Go to project CI settings
```

ハンズオン2 API キー未設定での ai_review 失敗

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にしているためです。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す。まず講師へお知らせください。
2. 確認する場合は **Settings** から **CI/CD** を開き、**Variables** の欄に2つの名前があるかを見てください。
3. **Masked** が付いていて **Protect variable** が外れている状態が正です。
4. 直ったら、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、講師がもう一度手動ボタンを押します。
5. 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「未実行」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。

ハンズオン2　赤にならない ai_gate

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。
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あります。

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。

ハンズオン2　PHP 5.6 イメージでの取得失敗

症状

`test-phpunit` ジョブが `before_script` の途中で落ちます。

```
E: The repository 'http://deb.debian.org/debian stretch Release' does not have a Release file.
E: Unable to locate package libpq-dev
```

または、取得先を書き換えたあとに次で落ちます。

```
E: Unable to correct problems, you have held broken packages.
```

原因

PHP 5.6 のイメージは Debian stretch を土台にしており、標準の配布元は既に公開が終わっています。配布した `.gitlab-ci.yml` は取得先を書庫側(`archive.debian.org`)へ向け直し、署名検証を外してから入れる形にしてあります。この `before_script` を自分で書き換えると、その一手が消えます。2つ目の症状は、コードネームを `jessie` と書いたときに出ます(土台と違う版のソースを指すと依存関係が壊れます)。

直し方

1. `.gitlab-ci.yml` の `test-phpunit` の `before_script` を、配布時の形に戻してください。先頭の5行が取得先の差し替えです。

   ```
   codename=$(. /etc/os-release && echo "${VERSION_CODENAME:-stretch}")
   printf "deb http://archive.debian.org/debian %s main\n" "$codename" > /etc/ap
   t/sources.list
   rm -f /etc/apt/sources.list.d/*.list
   apt-get -o Acquire::Check-Valid-Until=false -o Acquire::AllowInsecureRepositor
   ies=true update -qq
   apt-get install -y -qq --allow-unauthenticated --no-install-recommends ca-cert
   ificates libpq-dev postgresql-client
   ```

2. この5行より前に `apt-get install` を足さないでください。順番が入れ替わると同じ症状になります。

3. それでも取得できない場合は、runner から書庫側へ出られない環境です。講師へお知らせください。

ハンズオン2　差分の後半に指摘が付かない状態

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

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。

ハンズオン3　押せない手動ボタン

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、押しているアカウントにその権限が無い場合です。`main` は保護ブランチなので、`main` のパイプラインで手動ジョブを押せるのはマージ権限を持つアカウントだけです。受講者(Developer)は自分のブランチのパイプラインなら押せますが、当日は講師が押す運用にしています。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. 自分で押す必要はありません。講師の合図を待ってください。押してしまった場合は、そのジョブの **Cancel** を押し、講師へお知らせください。順番が崩れると、後の組が待ちに入ります。

ハンズオン3　有効のまま保存した schedule

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

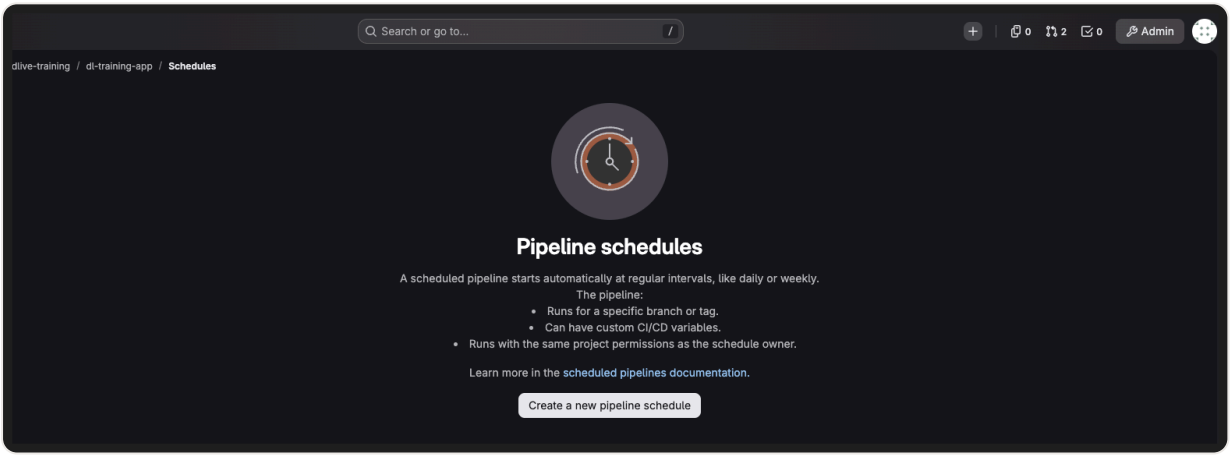
Activated のチェックが既定で入っており、外さずに保存しています。このままだと翌日の 2:00 に18本の `nightly_implement` が同時に走り、runner と API の予算を使います。

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

ハンズオン3　30分で打ち切られた夜間ジョブ

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分を過ぎた時点で講師へお知らせいただければ、次の組の起動時刻を調整します。

ハンズオン3　許可待ちで終わる夜間ジョブ

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 を見てください。
```

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、ここで終わるのは次の症状Bのほうが多いです。

症状B

`.err` にツールの拒否が出て、結果 JSON の `result` が「受入条件が特定できない」で終わります。

```
Tool use blocked: Bash(curl ...)
```

原因

症状Aは、非対話の `-p` が既定では許可待ちのまま何もせずに終わるためです。無人で動かすには `--permission-mode dontAsk` と `--allowedTools` と `--max-turns` の3つが要ります。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` に渡していな

いためです。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 *),Bash(g
it 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あります。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は2通りで直せます。`--allowedTools` に `,Bash(curl *)` を足して Claude に取りに行かせるか、`PROMPT` に「Issue の本文は `_issues/issue-4.md` にあります」の1行を足して `Read` で読ませるかです。どちらを選んだかを台帳に書いてください。外へ出る経路を増やさない後者のほうが、夜間向きです。

Codex で回す場合も同じ考え方です。 `codex exec -s workspace-write` が対応する形になります。`.git` が読み取り専用になるため、コミットはスクリプト側で打ちます。差し替えの位置は `ci/nightly_implement.sh` にコメントで残してあります。

ハンズオン3　MR が作られない夜間ジョブ

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。

ハンズオン3　夜間ジョブの MR 作成の失敗

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

push の `-o merge_request.create` 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。

ハンズオン3 流し直しでの push 拒否

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. 講師に伝えて、もう一度 `nightly_implement` を手動で実行してもらってください。ターミナルから消す場合は `git push origin --delete nightly/issue-4` です。

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。

プチ演習6 Stop hook の無限ループ

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、止める条件が永久に消えません。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 sto
   p.
   ```

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つきます。

プチ演習6　作業メモのコミット混入

症状

コミットの前に確認すると、見覚えのないファイルが並びます。

```
$ git status --short
 M .claude/hooks/stop-gate.ps1
?? .claude/hooks/.stop-gate.state
```

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、赤いまま終えると残ります。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. 無視するファイルの一覧に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。

   ```
   git rm --cached .claude/hooks/.stop-gate.state
   ```

3. 変更したファイルだけを指定して追加する癖にしておくと、この種の混入は起きません。

   ```
   git add .claude/hooks/stop-gate.ps1
   ```

進行と画面共有

リモートの方が11名です。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。見せ方だけ先に決めておきます。

共通　リモート参加での画面共有

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。

共通　時間内に終わらない演習

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。

※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。