TRAINING

Claude Code 研修

VSCode 拡張で AI駆動開発を回す

株式会社エヌアイデイ 様

2026年8月17日（月） オンライン開催

Givery

本資料の二次転載禁止について

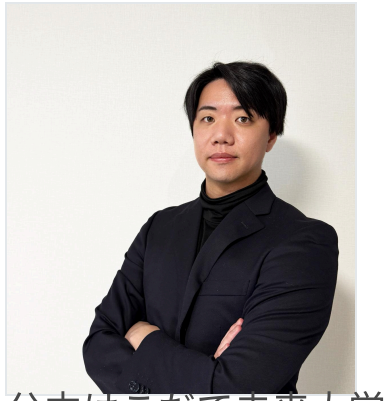
この資料は著作権により保護されています

本資料の著作権は、株式会社ギブリーに帰属します。
研修を受講される方の学習を目的として提供しています。

権利者の許可なく、本資料の全部または一部を複製・転載・
再配布・改変することを禁止します。社外への共有や、
SNS・ブログ等での無断二次転載はお控えください。

ご不明な点は、研修事務局までお問い合わせください。

講師紹介



人的資本インテリジェンス部門講師・
生成AIエバンジェリスト

安田 光喜

Mitsuki Yasuda

公立はこだて未来大学 大学院（メディアデザイン領域）を卒業し、街づくり会社での複合商業ビルの立ち上げにおける情報インフラ整備やデザイン全般業務の責任として関与。その後デザイン事務所を立ち上げ、飲食店を中心としたデザイン業務や美術館展示のアートコンテンツ開発などを行う。また、地域ブランディングにも注力し、市主催の企画運営業務を行なった。2018年10月『小中学生向けのプログラミング教育』に注目し、函館市で初めての小中学生向けプログラミングスクール『D-SCHOOL北海道』を立ち上げて運営。その後北海道のドラッグストア『サッポロドラッグストアー』からスクールの買収を受け、教育を専門とするグループ会社『株式会社シーラクンス』にて技術開発責任者として運営。スクール設計・運営、20以上の自治体連携（イベントや出張教室運営）、大人向けプログラミングスクールメンターなどさまざまな『次世代IT教育』に取り組む。
Dify公式資料の日本語翻訳や、SecondMeアンバサダーなど生成AI最新トレンドに注力。

得意領域

- 生成AIをはじめとした世界最先端トレンド
- 生成AIによる新規事業の開発、運営、補助
- アプリ開発、自動化環境の構築
- 教育（大学講師、研修、スクール運営）

実績

- 生成AI研修
- 大手企業、外資、教育機関(小中高大)
  でのITトレンド講演
- 自治体と連携した地域教育実績
- 上場企業社員対象のリスキリング支援

メッセージ

みなさんがこれから生成AIをパートナーに
新しい時代を生きていけるように、本日は
全力でサポートしていきます！
なにかご不明点などありましたら、
講師陣になんでも気軽にご質問ください。
#ClaudeCode #Codex #Cursor #Antigravity
#全国統一生成AI活用技能試験S1
#生成AIパスポート　#G検定

本日の講師体制と質問の出し方

質問は Zoom のチャットへ、いつでも書いてください。

進行　安田 光喜
講義とデモを進めます。全体の質問に答えます。

サポート　松原 孝司
個別の詰まりを見ます。画面共有はこちらへ。

場面	どうするか
講義中に分からない語が出た	チャットに1行書いてください。区切りでまとめて答えます。
演習で手が止まった	黙って待たずにチャットへ。どこまで進んで何が出ないかで十分です。
原因が切り分けられない	画面共有をお願いします。サポート講師が個別に入ります。
全員に関わりそうな質問	その場で全体に向けて答えます。遠慮なく書いてください。

分からないまま進めると後の演習で詰まります。ひとことでもチャットに書いてください。

講師紹介

株式会社ベクターデザイン 取締役
ジェネラリスト系エンジニア

松原 孝司

Koji Matsubara

東芝グループのソフトウェアエンジニアとしてキャリアをスタートし、携帯電話向け組み込みソフトウェアや車載用ポータブルナビゲーションシステムの開発に従事。約10年の勤務後、東大生・東工大生が立ち上げたジョイントベンチャーの設立に参画し、AI・機械学習につながる技術を活用したSNS投稿の自動フィルタリングシステムの開発・事業化を担当する。その後、複数のスタートアップで経験を重ね、NewsPicksのAndroidアプリ開発を行ない、エンジニア兼プレイングマネージャーとして携わる。現在は株式会社ベクターデザインの取締役として、気象観測・防災IoTシステムなどに技術面から関わる一方、複数のスタートアップでもエンジニアとして活動。幅広い技術領域を理解し、エンジニアとデザイナー、編集、営業など異なる職種の間をつなぐ「ジェネラリスト型エンジニア」。事業やプロジェクトの「0.1を2にする」段階を前進させることを得意としている。

得意領域

- 組み込み、Android、Web、
  インフラ、IoT を横断した開発
- 新技術を素早く理解し、実務へ取り入れる
  技術調査
- 新規事業やプロジェクトの
  「0.1を2にする」推進
- エンジニアと非エンジニアの橋渡し

実績

- ソフトウェア開発の実務に基づく
  生成AI活用支援

メッセージ

生成AIも、正解を教えてくれる魔法の道具
ではなく、一緒に考え、試し、成果を
磨いていくパートナーだと思っています。
20年以上のエンジニア経験を生かし、
皆さんと同じ目線で手を動かしながら
サポートします。疑問や気づきがあれば、
ぜひ気軽に共有してください！

本日のゴール

本日のゴールは、指示から差分の確認までを
自分の手で回せる状態になることです。

1. **指示・生成・差分の確認・修正を自分で回す**
   承認する前に差分を読みます。今日いちばん多く練習する動作です。
2. **ハーネス有り無しの差を自分の手で測る**
   同じ作業を演習1と演習2で行い、手数がどれだけ変わるかを見ます。
3. **5つの仕掛けを1回ずつ通す**
   CLAUDE.md 2箇所、コーディング規約、Hook、Skill、公式Skill です。
4. **明日から自社リポジトリで始められる**
   最初に何を置くか、どの順で足すかを決めて持ち帰ります。

> **ハーネスとは**
>
> 前提と制約を Claude へ渡すために、リポジトリへ置いておく仕掛けのことです。
> 本日は CLAUDE.md・コーディング規約・Hook・Skill の4種類を扱います。

本日の流れ

休憩 [10min] を4回と昼休憩 [60min] を別に挟み、予備を含めた総枠は [540min] です。

項目	区分	所要
イントロダクション（講師紹介・ゴール・進め方）	講義	[12min]
準備 環境立ち上げと公式Skillのコピー	準備	[20min]
M1 画面の見どころ	**ミニ演習**	[10min]
章1 生成AI最新トレンド	講義	[28min]
章2 Claudeファミリーの現在地	講義	[22min]
章3 Claude Code の基礎（画面・モデル・文脈）	講義	[44min]
準備とM2・M3（ユーザー設定・statusLine）	準備	[28min]
章4 ハーネス3段階の地図	講義	[12min]
演習1 Lv1 集計ツールとM4	**演習**	[59min]
演習2 Lv2 不具合修正とM5	**演習**	[63min]
M6 skill-creator・M7 automation-recommender	**ミニ演習**	[45min]
演習3 Lv3 ログ解析	**演習**	[59min]
振り返りと明日から	まとめ	[18min]

演習ごとのフォルダの開き直し

演習に入るたびに、その演習のフォルダを開き直します。

本日開き直すフォルダ	
1回目	演習1_Lv1_集計ツール
2回目	演習2_Lv2_不具合修正
3回目	予備_チケット管理
4回目	演習3_Lv3_ログ解析
毎回やること	信頼する を選ぶ
残るもの	~/.claude/ の設定

1 開くのは演習フォルダ

配布フォルダの一番上ではなく、
その演習のフォルダを開きます。

2 会話は毎回まっさら

開き直すと前の会話の文脈は残りません。
必要な前提は都度渡します。

3 ユーザー設定は残る

午前に入れた ~/.claude/ の設定は、
どのフォルダを開いても効きます。

研修の進め方

読ませる・相談する・作らせる・確かめるの
4つを、この順に回しながら進めます。

読ませる	データや既存コードを先に開かせます。読まずに書いた実装は当てになりません。
相談する	いきなり実装させず、方針と作る順番を先に出させて、そこで直します。
作らせる	差分を1件ずつ見て承認します。読まずに全部承認すると、あとで原因が追えません。
確かめる	生成物は必ず自分で実行します。数字は手で1つ数え直して突き合わせます。

> 詰まったら、次の指示を足す前に「いま何が起きているか」を1行で書き出してみてください。

補足

本日の約束

答えは配布物に入れていません。先に自分で書いてから開きます。

答えは配布物に入れていません

手順どおりに書き写す時間にしないためです。完成したコードは配っていません。

参考プロンプトは hints/ にあります

自分で書いてみて、行き詰まってから開いてください。順番が逆だと身に付きません。

_reference_完成形/ は研修中は開きません

答え合わせ用です。復習のときに、自分の実装と読み比べてください。

各演習に「自分で考える」があります

指示のない箇所を自分で決める場面です。ここが持ち帰りの中身になります。

配布ZIP の入手と展開

配布ZIP を、日本語を含まない場所へ展開します。

教材サイトを開く

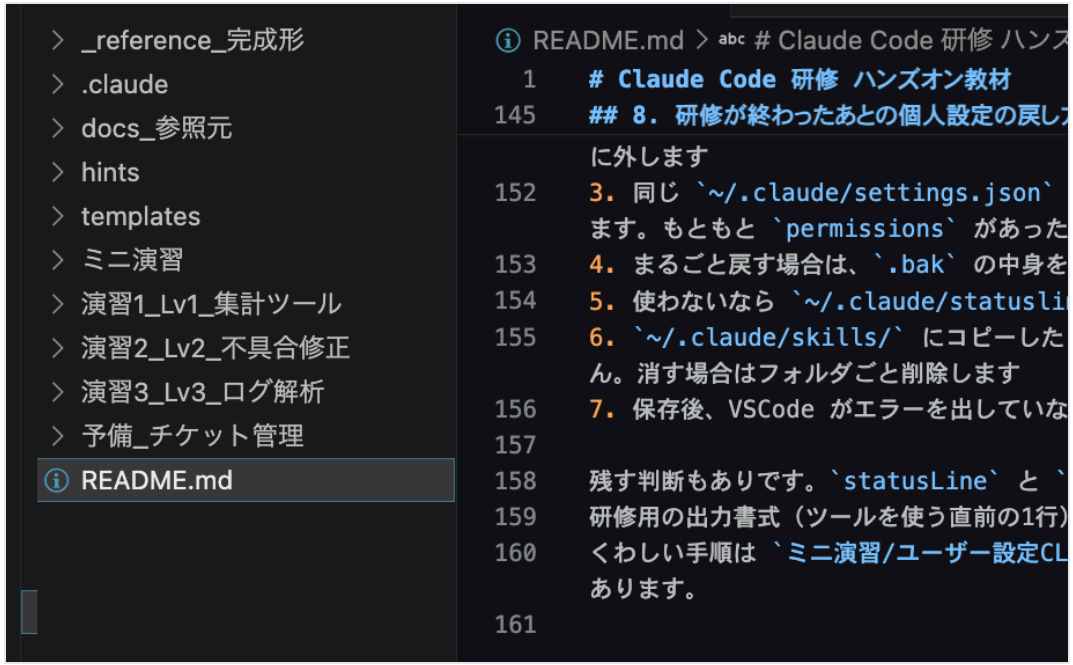
https://0817cc.give-app.net をブラウザで開きます。

ZIP をダウンロード

nid-claudecode-handson_受講者用.zip を保存します。

日本語を含まない場所へ展開

デスクトップ直下などに展開します。フォルダ名に日本語が入る場所は避けます。



画面: 教材サイトの配布物セクション

> **展開してから開きます**
>
> ZIP の中身を直接開くと、拡張から見えないファイルが出ます。

開くフォルダと信頼の確認

最初に開くのは 演習1_Lv1_集計ツール のフォルダです。

配布フォルダの中身

```
nid-claudecode-handson_受講者用/
    演習1_Lv1_集計ツール/                    最初に開く
    演習2_Lv2_不具合修正/
    演習3_Lv3_ログ解析/
    予備_チケット管理/
    docs_参照元/ hints/ templates/
    ミニ演習/ README.md
    _reference_完成形/                       研修中は開かない
```

1. **一番上のフォルダは開きません**
   演習ごとに、そのフォルダだけを開きます。

2. **確認では「信頼する」を選びます**
   制限モードのままだと拡張は動きません。

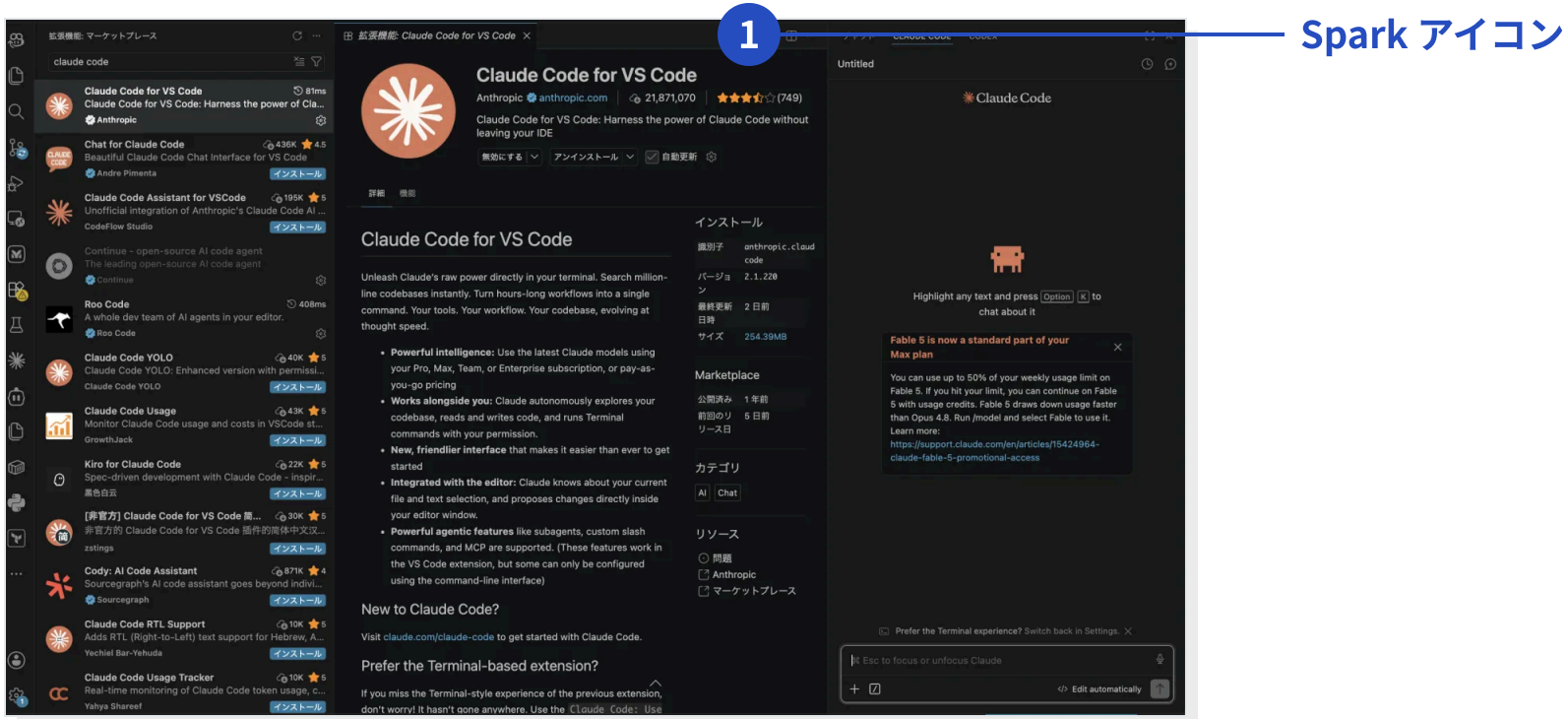
3. **本日は4回開き直します**
   そのたびにこの確認が出ます。
   毎回「信頼する」を選びます。

> 信頼したフォルダでだけ、
> そのフォルダ側の設定が効きます。

パネルを開いてサインインを確認

エディタ右上の Spark アイコンでパネルを開きます。

「このフォルダに何が入っているか教えてください」と送り、返答が返ればサインイン済みです。



画面: 拡張を入れた VSCode（右側が Claude Code のパネル）

統合ターミナルでインタプリタを確認

実行コマンドは、ここで返った結果に合わせて読み替えます。

Mac

```
$ python3 -V
```

Python 3.10 以上なら大丈夫です

返らなければ python -V も試します

Windows

```
> py -V
> python -V
```

返ってきた方を以降で使います

統合ターミナルの開き方

Windows は Ctrl+@、Mac は Cmd+@ です。VSCode の下部にターミナルが開きます。

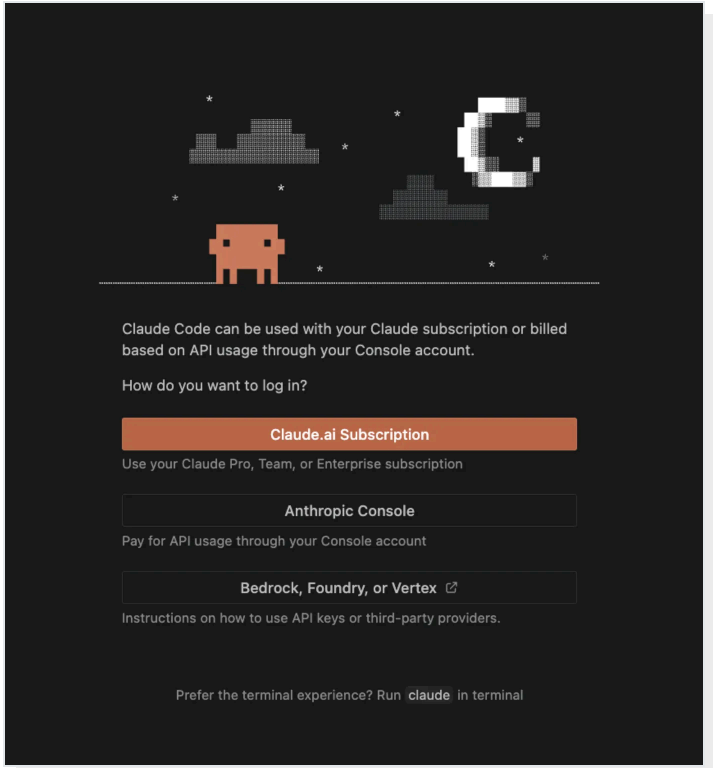
返ってきたコマンドを覚えておきます

以降のスライドでは python3 と書きます。py が返った方は読み替えてください。

補足

立ち上がらないときの3点

症状	対処
パネルが出ない	コマンドパレットで Developer: Reload Window を実行します。
サインインが切れた	右の画面の3択から Claude.ai Subscription を選びます。
通信できない	許可対象ドメインの一覧は事前セットアップガイドにあります。



画面: ログイン方法の3択

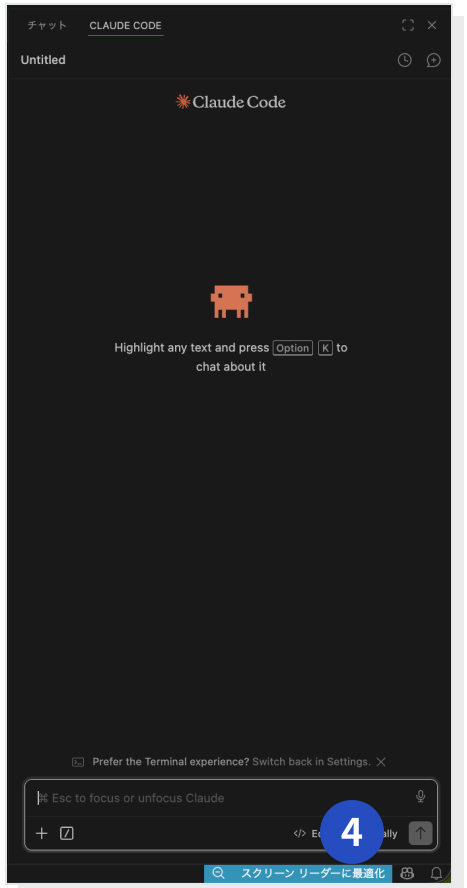
それでも直らないとき

チャットに1行書いてください。サポート講師が画面共有で見ます。

ワーク

M1 見どころ4箇所をそろえる

以降ずっと使う場所を、最初に全員で同じところに合わせます。



画面: Claude Code のパネル

1 Context usage

自動圧縮までのコンテキスト残量が割合で出ます。演習3で一気に減ります。

2 Account & Usage

使用量のバーと、コマンド別・MCP別・
プラグイン別・サブエージェント別の内訳が出ます。

3 モデル切替

「Change the AI model」で、表示名と説明つきの一覧から選びます。

4 権限モードと Effort

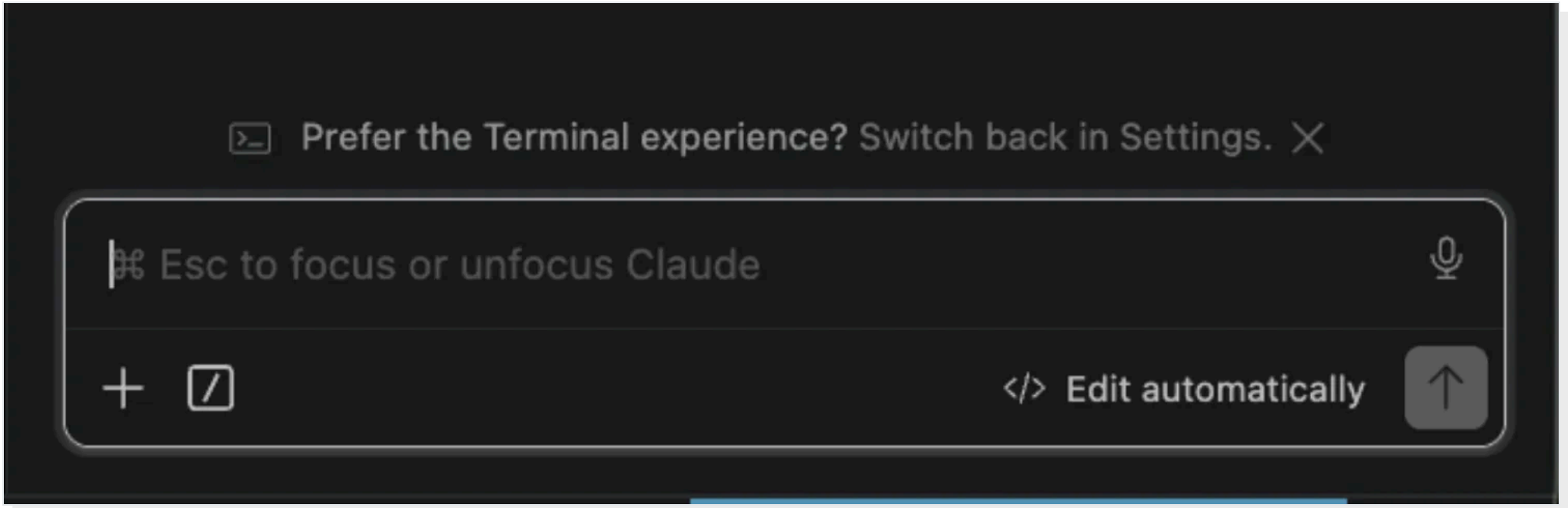
入力欄の下のモードインジケータをクリックします。左の画面の4の位置です。

Context usage と Account & Usage は会話を始めてから出ます。
当日、実機で一緒に開きます。

Context usage の読み方

Context usage は、自動圧縮までの残りを割合で示します。

表示は「% of context remaining until auto-compact.」で、残りの割合です。
圧縮が走ると「Conversation was compacted to free up context.」が出ます。



画面: パネル下部の入力欄まわり（会話前なので Context usage の行はまだ出ていません）

Account & Usage で内訳を見る

使用量は、何が食っているかまで内訳で出ます。

1. **パネルのメニューを開く**
   メニュー表記は Account & usage… です。

2. **使用量のバーを見る**
   説明は「View account info and usage limits」と書かれています。

3. **内訳を読む**
   コマンド別・MCP サーバー別・プラグイン別・サブエージェント別の4種類が並びます。

内訳に並ぶ4種

- **コマンド別**
  / で呼んだものごと
- **MCP サーバー別**
  接続先ごと
- **プラグイン別**
  入れたものごと
- **サブエージェント別**
  任せた下調べごと

開始時に1回見ておきます
締めの振り返りでもう一度見て、差をメモします。

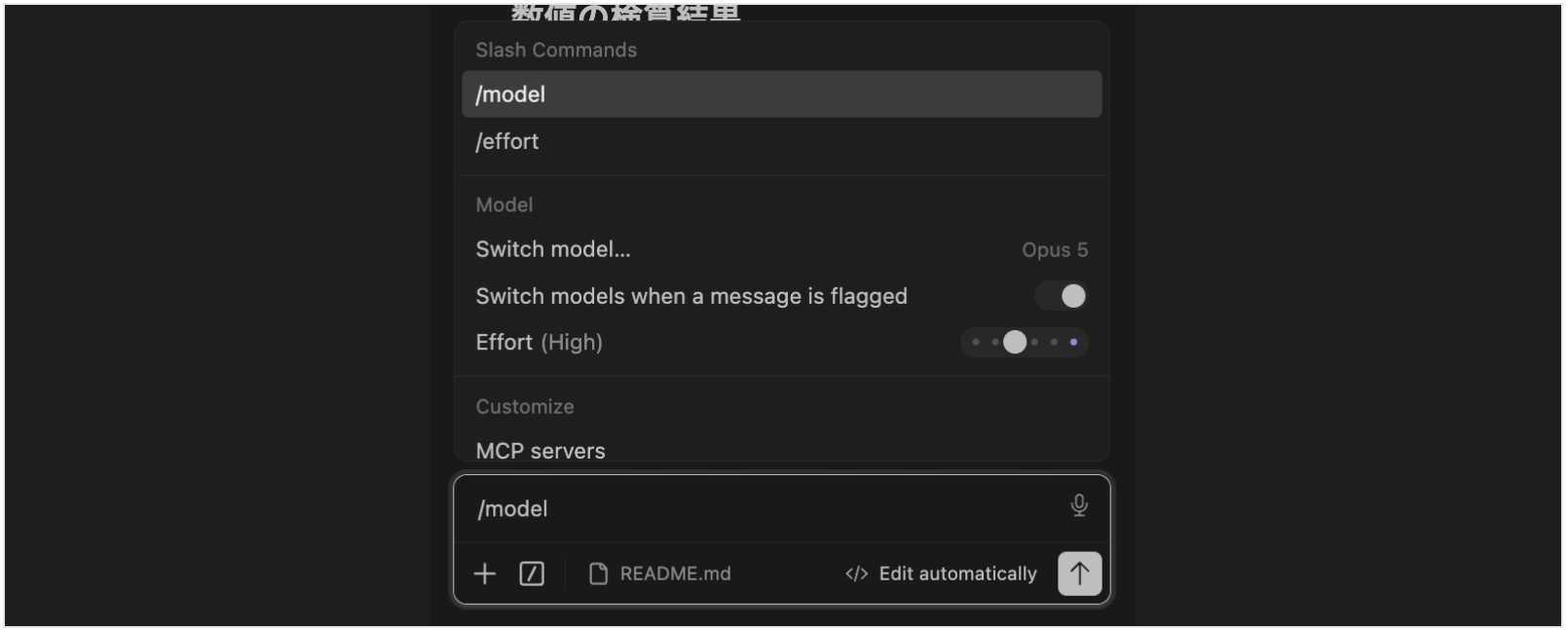
見どころ4箇所と、本日どこで使うか

見どころ	本日どこで使うか
Context usage	演習3で一気に埋まります。減り方を自分の目で見ます。
Account & Usage	開始時と、締めの振り返りの2回開きます。内訳の変化を見ます。
モデル切替	当日の版数は資料に書きません。この一覧で実機を確認します。
Effort	M1 の追加と考察で1回 low に下げ、そのあと元に戻します。

4箇所とも、当日パネルを開いて全員で同じ場所を確認してから先へ進みます。

モデルの切り替え

パネルの「Change the AI model」で、表示名と説明つきの一覧から選びます。



画面: パネルのモデル一覧

資料に版数は焼き込みません。当日、この一覧で実機を確認します。

質問

M1 の OK基準と追加と考察

OK基準

・4箇所（Context usage / Account & Usage / モデル切替 / Effort）を自分で開けます。
・権限モードが Manual になっていることを確認できています。
・生成物はありません。画面を自分で開けたかどうかだけを見ます。

追加と考察

Effort を low に落として同じ質問を1回投げ、応答の丁寧さと速さの違いを見ます。
見終わったら元の段階へ戻してから次に進みます。

振り返りシートに書くこと

開始時点の使用量と、コンテキストの残量をメモしておきます。
締めの振り返りで同じ2つをもう一度見て、差を比べます。

個人スコープとプロジェクトスコープ

演習ごとに開き直すので、
Skill は個人スコープに置きます。

プロジェクトスコープ

演習フォルダ/.claude/skills/

そのフォルダを信頼したあとだけ
効きます。別の演習フォルダを開くと
読まれません。起動したフォルダの
.claude/skills/ しか見ないので、
上位のフォルダへは遡りません。

個人スコープ

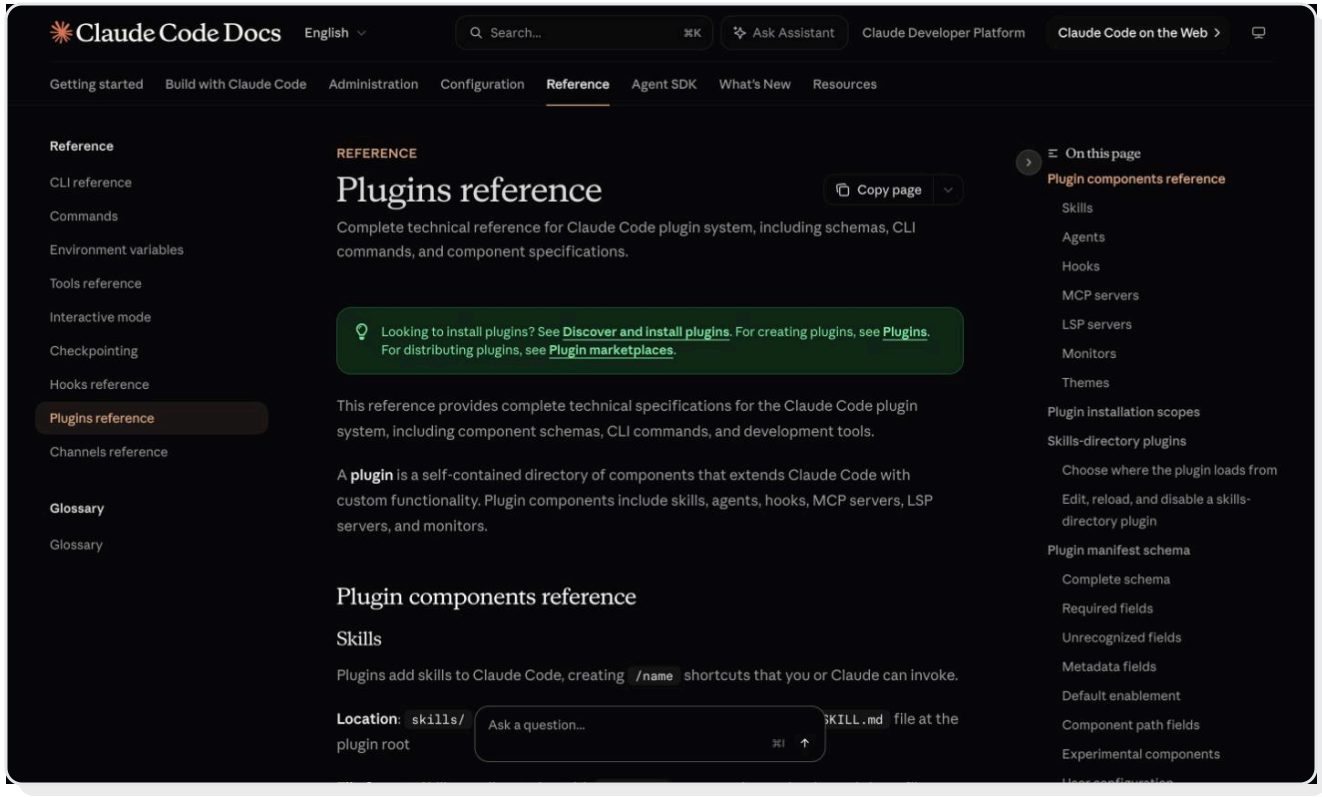
Mac ~/.claude/skills/
Windows %USERPROFILE%\.claude\skills\

すべてのプロジェクトで効きます。
本日はフォルダを4回開き直すので、
公式Skill 2本はこちらへコピーします。

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

プラグインとして読まれるフォルダ

.claude-plugin/plugin.json があるフォルダは、
名前@skills-dir のプラグインとして読まれます。



画面: 公式 Plugins reference（実画面）

配布物のどれが当てはまるか

配布ZIPの claude-code-setup が
この形です。中に入っている Skill が
claude-automation-recommender で、
コピーしたあと読み込まれるのは
次のセッションからです。

> **もう1本は素のSkill**
> skill-creator は plugin.json を持たず、
> ふつうの Skill として読まれます。

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

コピー手順は2通り

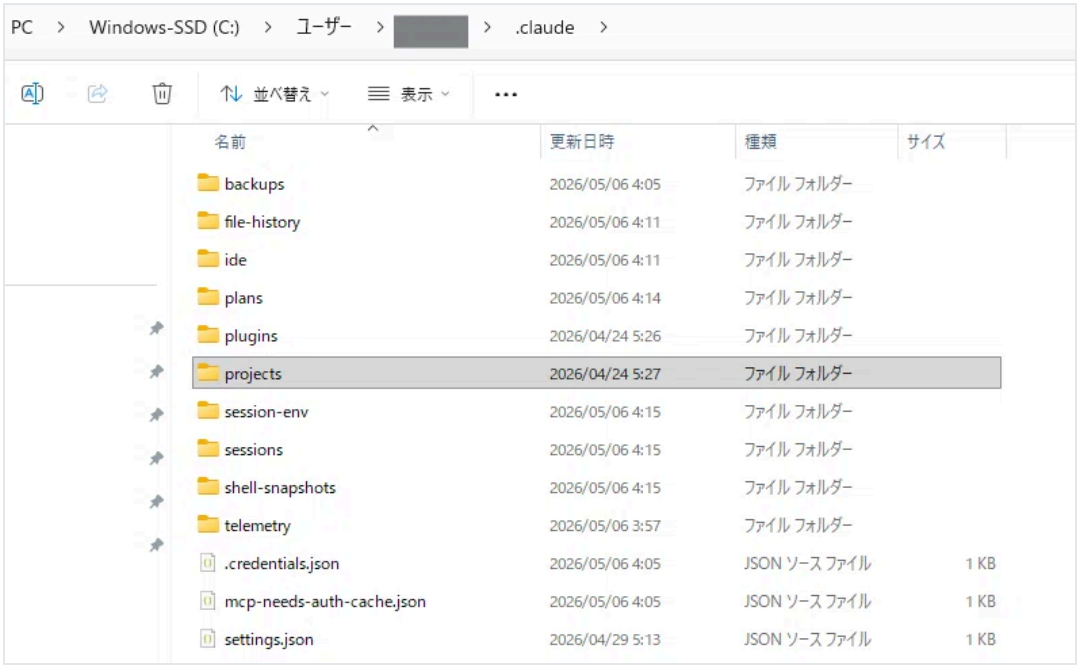
どちらの手順でも、置き場所は個人スコープです。

フォルダをドラッグする

配布フォルダの .claude/skills/ にある2本を、
エクスプローラや Finder で個人スコープへ運びます。

パネルで Claude に頼む

いま開いている演習1のパネルから頼みます。
.claude 配下への書き込みなので毎回確認が入り、
保護パスの挙動をそのまま体験できます。



.claude 配下は保護パスです
権限モードに関わらず、書き込みのたびに確認が入ります。仕様どおりの動きです。

コピー元とコピー先

コピー先のフォルダ名が、そのまま/コマンド名です。

```
# コピー元（配布フォルダの中）
nid-claudecode-handson_受講者用/.claude/skills/claude-code-setup/
nid-claudecode-handson_受講者用/.claude/skills/skill-creator/
# コピー先
Mac ~/.claude/skills/
Windows %USERPROFILE%\.claude\skills\
```

Windows はエクスプローラのアドレス欄に %USERPROFILE%\.claude\skills を貼ると開きます。
Mac は Finder で Shift+Command+G にパスを貼ると開きます。
フォルダ名は変えません。claude-code-setup と skill-creator のままにします。

出典: Skills https://code.claude.com/docs/en/skills

ここで教える3点

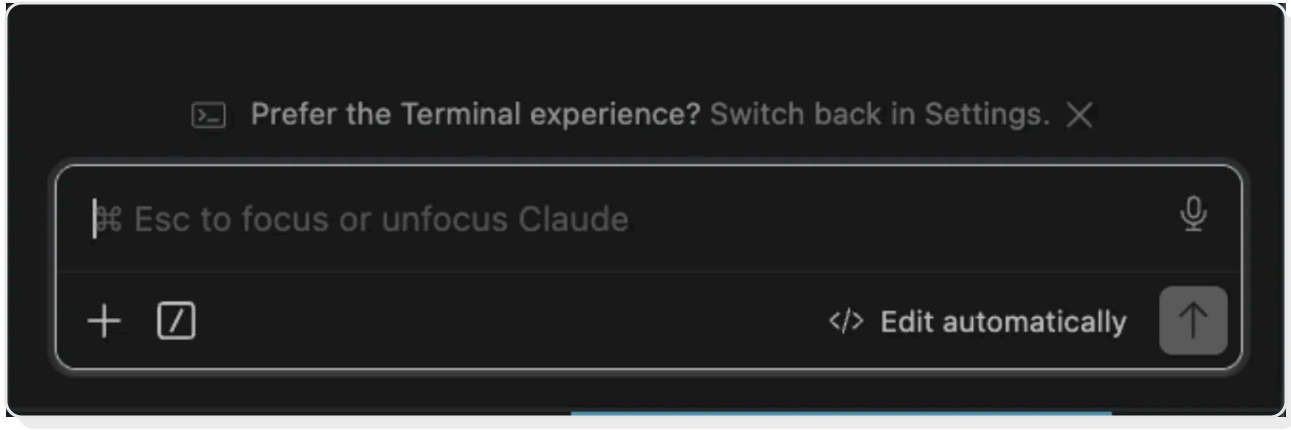
置き場所の違いが、使えるかどうかを決めます。

論点	中身
スコープの違い	個人スコープ（~/.claude/skills/）はすべてのプロジェクトで効きます。 プロジェクトスコープは、そのフォルダを信頼したあとだけ効きます。
プラグインとして読まれる形	.claude-plugin/plugin.json を含むフォルダは 名前@skills-dir として 読まれます。反映されるのは次のセッションからです。
上位へ遡らない	名前@skills-dir は起動したフォルダの .claude/skills/ しか見ません。 演習ごとに開き直すので、個人スコープへコピーします。

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

コピーできたかの確認

SKILL.md は**その場**で、プラグインは**次のセッション**から読み込まれます。



画面: パネル下部の入力欄（実画面）

1 スラッシュで一覧を開く

入力欄で / を押し、2本が並ぶか見ます。

2 出ないときは自然文で呼ぶ

「skill-creator の Skill を使って」と書けば、名前を指定して呼べます。

> **反映のタイミング**
> SKILL.md の追加は同じセッションで反映されます。plugin.json を含む claude-code-setup はプラグイン扱いのため次のセッションからです。パネルは Developer: Reload Window で開き直せます。

スラッシュ一覧に2本が並ぶかは当日リハで確認します（未確認）。

01

生成AI最新トレンド

市場の話ではなく、みなさんの仕事の話として見ていきます

この章の読み方

数字は誰が・いつ・どう測ったかまで見ます。

時点と出典を必ず添えます

この章の数値には時点と出典URLを添えてあります。値は動くので、社内で引用するときは
日付ごと引き写してください。研修の直前にも一次情報で確認しています。

ベンチマークは測り方で動きます

Terminal-Bench は 2.0 から 2.1 でベンチマーク側のバグを89問中28問直しただけで、
同じモデル構成のスコアが12.1ポイント動きました（2026年5月6日）。

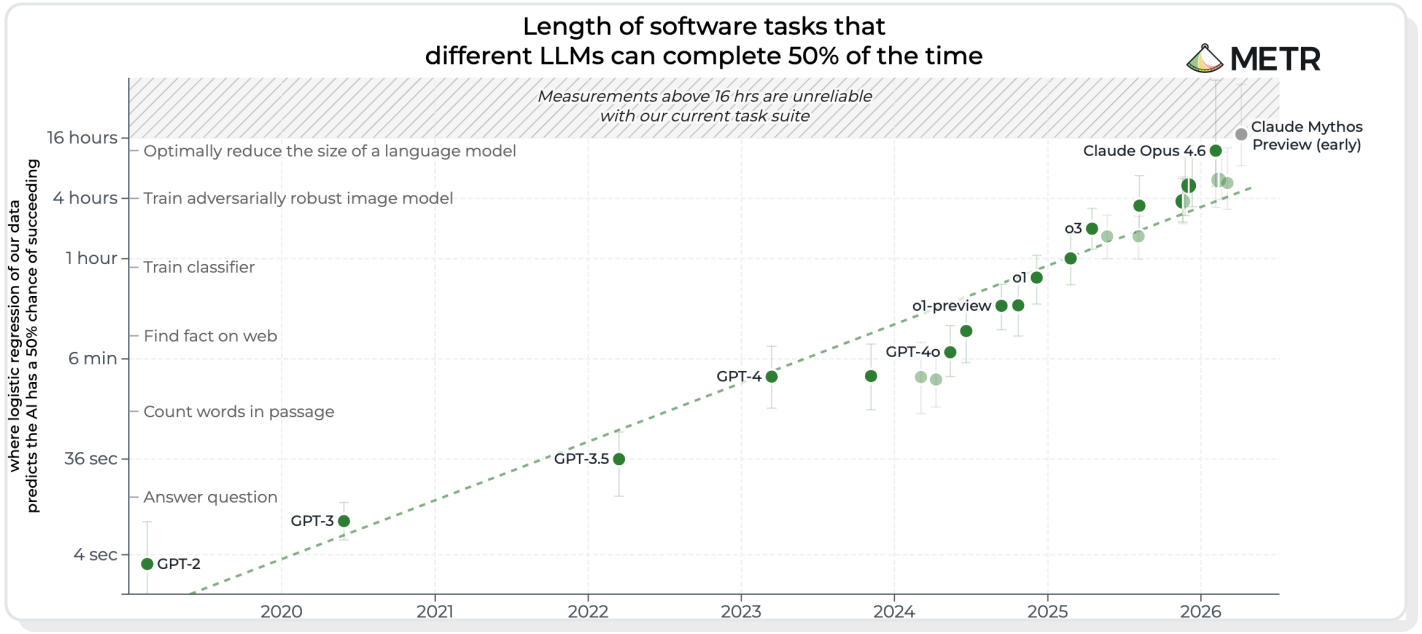
版数を出すのはこの章と章2だけです

手順の話に入ったら版数は書きません。当日、パネルのモデル一覧で実機を見ます。
この章の版数と数値は、2026年7月30日時点で一次情報を確認したものです。

出典: https://www.tbench.ai/news/terminal-bench-2-1

体感と実測のずれ

熟練開発者16名の試験で、作業時間が19%延びました。



画面: METR のランダム化比較試験の結果（実画面）

1 **実測**

作業時間は19%長くなりました
（信頼区間 +2%〜+39%）。
16名・246課題の試験です。

2 **体感**

事前予想は24%速い。実測後も
20%速いと答えています。

3 **続報（2026年2月24日）**

初回参加者 -18%、新規47名 -4%。
信頼区間が0をまたぎます。

出典: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
https://metr.org/blog/2026-02-24-uplift-update/

丸ごと任せられる範囲

使う範囲は広く、任せきれる範囲は狭いままです。

0〜20%

「完全に任せられる」と答えたタスクの割合

開発者は自分の仕事の約60%でAIを使うと答えています。
使う範囲の広さと、任せきれる範囲の狭さは別の話です。

増えたのは仕事そのもの

AI支援で行われた作業の約27%は、
AIがなければやらなかった作業でした。
ダッシュボード作成、探索的な調査、
放置されていた小さな不具合です。

出典: 2026年1月21日 Anthropic Agentic Coding Trends Report
https://resources.anthropic.com/2026-agentic-coding-trends-report

使う人と、精度への信頼

使う人は増えましたが、精度への信頼は割れています。

使っている

84% が使うか使う予定（前年76%）

51% の専業開発者が毎日使う

精度への信頼

33% が精度を信頼する

46% が精度を信頼しない

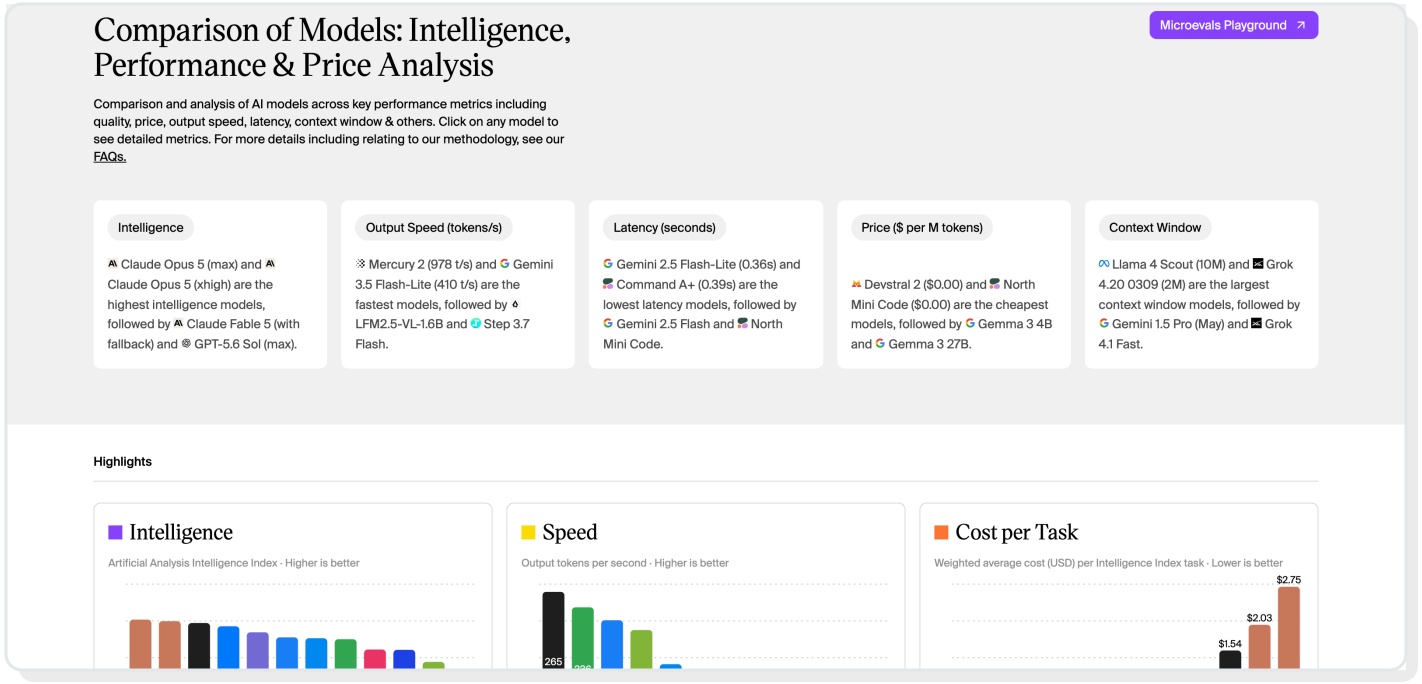
不満の中身

1位は「ほぼ正しいが少し違う出力」で66%、2位は「デバッグに時間がかかる」で45%です。

出典: 2025年調査・AI設問への回答33,662件 https://survey.stackoverflow.co/2025/ai

モデルの現在地

上位の差は2ポイント、単価は約69倍開きます。



画面: Artificial Analysis のモデル一覧（実画面）

いずれも2026年7月28日時点の値です。
出典: https://artificialanalysis.ai/models

知能指数の上位

Claude Opus 5 が61、
Claude Fable 5 が60、
GPT-5.6 Sol が59。
差は2ポイントです。

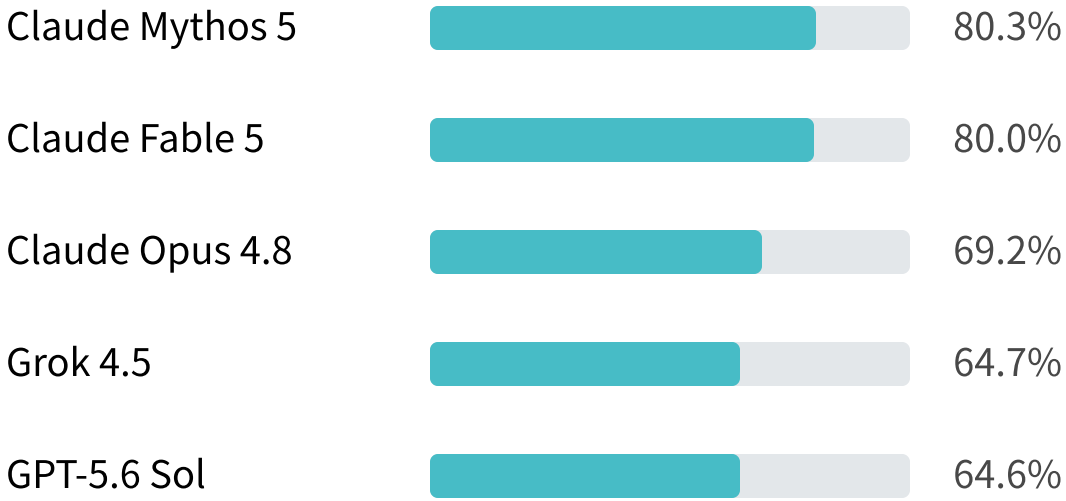
1タスクの平均コスト

0.04ドルから2.75ドルまで
開きます。スコア差17ポイント
に対して約69倍の差です。

評価軸で順位が入れ替わる

同じコーディング性能でも、軸次第で順位が動きます。

SWE-Bench Pro（実務規模の改修）



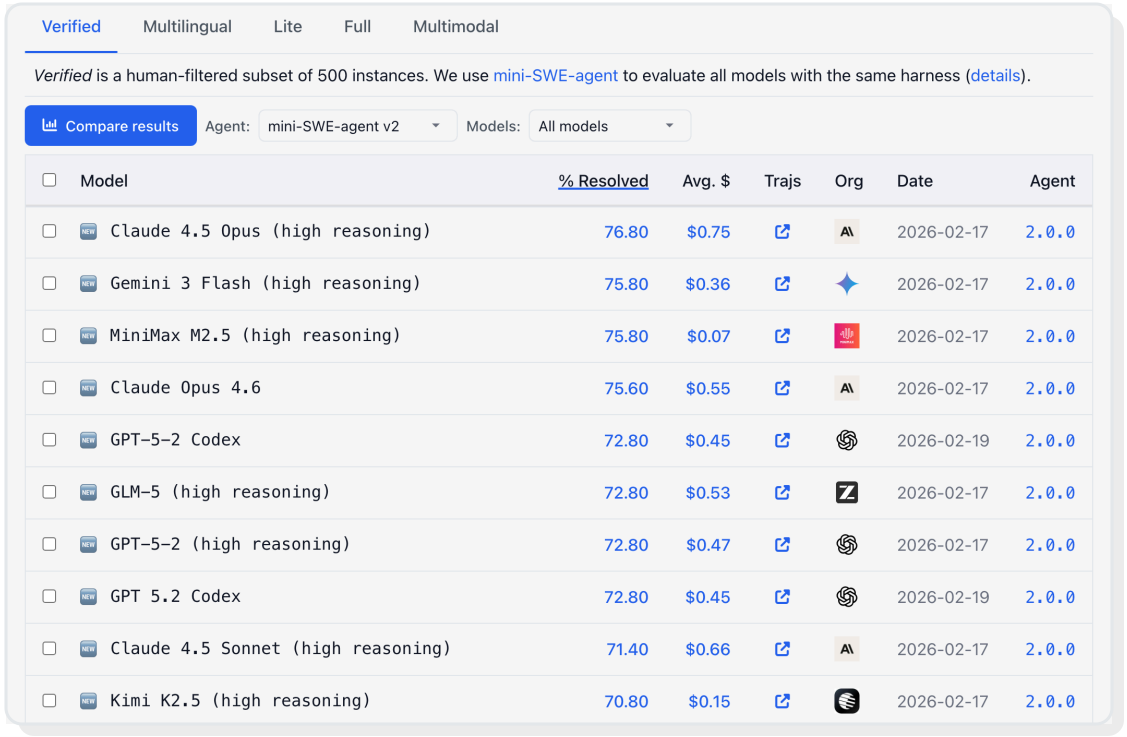
Terminal-Bench 2.1

いずれも各社が自社のハーネスで測った値です（2026年7月9日時点）。
xAI は同じ Terminal-Bench 2.1 で Claude Fable 5 (max) 84.3% と報告しています。

出典: https://openai.com/index/gpt-5-6/

同じ名前でも数字が違う

共通ハーネスの最高は76.80%、自己申告は90%台です。



Verified | Multilingual | Lite | Full | Multimodal

Verified is a human-filtered subset of 500 instances. We use mini-SWE-agent to evaluate all models with the same harness (details).

Compare results | Agent: mini-SWE-agent v2 | Models: All models

Model	% Resolved	Avg. $	Trajs	Org	Date	Agent
Claude 4.5 Opus (high reasoning)	76.80	$0.75			2026-02-17	2.0.0
Gemini 3 Flash (high reasoning)	75.80	$0.36			2026-02-17	2.0.0
MiniMax M2.5 (high reasoning)	75.80	$0.07			2026-02-17	2.0.0
Claude Opus 4.6	75.60	$0.55			2026-02-17	2.0.0
GPT-5-2 Codex	72.80	$0.45			2026-02-19	2.0.0
GLM-5 (high reasoning)	72.80	$0.53			2026-02-17	2.0.0
GPT-5-2 (high reasoning)	72.80	$0.47			2026-02-17	2.0.0
GPT 5.2 Codex	72.80	$0.45			2026-02-19	2.0.0
Claude 4.5 Sonnet (high reasoning)	71.40	$0.66			2026-02-17	2.0.0
Kimi K2.5 (high reasoning)	70.80	$0.15			2026-02-17	2.0.0

画面: SWE-bench 公式リーダーボード（実画面）

出典: https://www.swebench.com/
https://labs.scale.com/leaderboard/swe_bench_pro_public

1 共通ハーネスの値

mini-SWE-agent v2・500問での
SWE-bench Verified の最高は76.80%です
（2026年2月17日時点）。

2 ベンダーの自己申告

同じベンチマーク名で90%台が並びます。
測定元がそろっていません。

3 実務規模はさらに下がる

SWE-bench Pro の公開セット731問では
最高61.5%です（2026年7月28日時点）。

クイズ

理解度チェック

Q. ベンチマークの数字を見たとき、最初に確認するのはどれですか。

- **A** 順位表で1位になっているモデルはどれか
- **B** 上位モデルのスコア差が何ポイントか
- **C** 誰のハーネスで測った値か
- **D** 何社のモデルが参加しているか

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はC。誰のハーネスで測った値かを先に見ます。

C　正解　誰のハーネスで測った値か

共通ハーネスの最高は76.80%、同じ名前でベンダーは90%台を出しています。

A　順位表で1位のモデル

評価軸が変われば順位も入れ替わります。1位という事実だけでは決められません。

B　上位モデルのスコア差

測定元がそろっていない値の差は、比べても意味が残りません。

D　参加しているモデルの数

数が多くても、測り方がそろっている保証にはなりません。

実務でのポイント

社内で引用するときは、ハーネス名・問題数・時点の3つをセットで書き写します。

飽和したベンチマーク

従来のベンチマークは飽和し、差が見えなくなりました。

画面: Epoch AI の MMLU 評価ページ（実画面）

出典: https://epoch.ai/benchmarks/mmlu ・https://openai.com/index/gpt-5-6/

MMLU

最高88%で更新が止まりました。
評価済みは249モデルで、
2026年のモデルは未登録です。
（2026年7月28日時点）

GPQA Diamond

上位8モデルが91〜95%に密集し、
選ぶ材料になりません。
（2026年7月9日時点）

GitHub Copilot の課金の変化

補完は非課金のまま、依頼の実行分が使用量課金です。

非課金のまま

- コード補完
- Next Edit Suggestions
- 全有料プランに含まれたままです

使用量課金

- premium request units は2026年6月1日に GitHub AI Credits へ置き換わりました
- 1 AI credit = 0.01ドル
- 月あたりのクレジットはプラン料金と同額

何で数えるか
消費はモデルごとの公開API単価にもとづき、入力・出力・キャッシュのトークンで計算されます。

出典: https://github.blog/news-insights/company-news/（Copilot 課金の記事）
https://docs.github.com/en/copilot/reference/copilot-billing/models-and-pricing

手順書を版数に縛らない方が、追いつけます。

2026年7月28日

MCP 仕様

ステートレスな要求へ変更。
SDK 配布は月4億件超。

2026年7月29日

VS Code 1.131

実行中サブエージェントを
会話を開かずに確認。

毎週

Claude Code

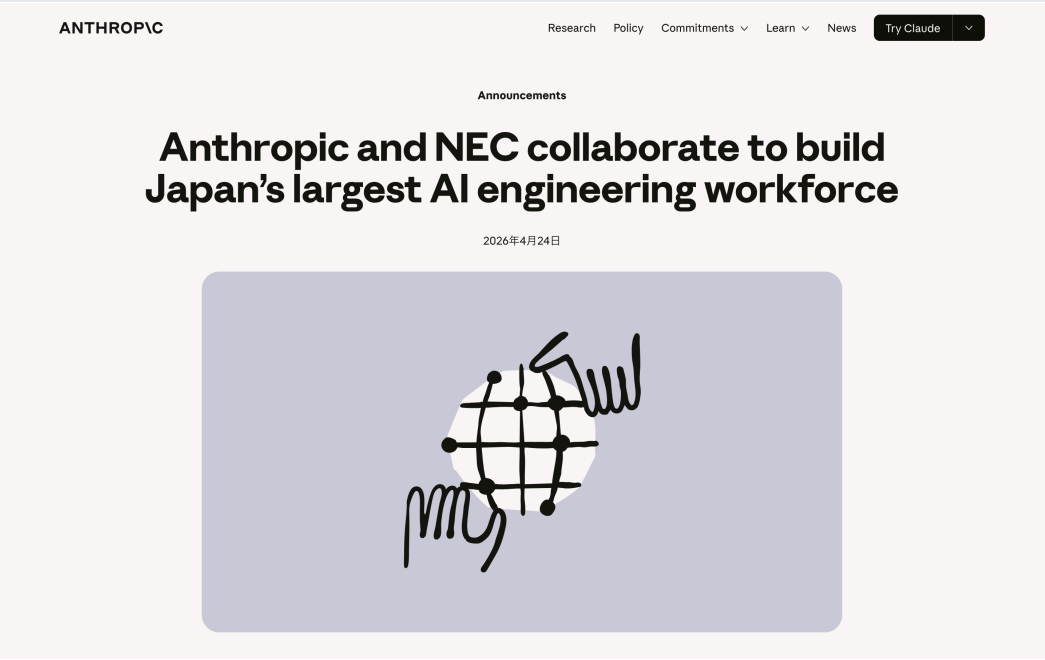
Week 29 で /fork、
Week 28 で /doctor。

資料に版数を焼き込まない理由
実際この3件も、本日時点では VS Code が 1.133 に上がっています。

出典: https://modelcontextprotocol.io/specification/2026-07-28
同 https://code.visualstudio.com/updates と https://code.claude.com/docs/en/whats-new

国内の開発現場

公表値は目標か実績かを分けて読みます。



- **NEC（2026年4月24日）**
  約3万人へ展開。Claude Code を使う組織と Center of Excellence を社内に設けています。
- **GMOデザインワン（2025年11月26日）**
  全エンジニアに導入。「作業時間約50%短縮」は目標値であって実績ではありません。
- **楽天（2026年1月21日）**
  1,250万行のOSSへの実装を7時間の自律実行で完了。参照実装と99.9%一致。計測方法は非開示です。

出典 ・https://www.anthropic.com/news/anthropic-nec
・https://prtimes.jp/main/html/rd/p/000005136.000000136.html
・https://resources.anthropic.com/2026-agentic-coding-trends-report

クイズ

理解度チェック

Q. 自社でツールの効果を判断するとき、最初に置く指標はどれですか。

- **A** 自社の実タスクを1つ決めて、同じ条件で測った時間
- **B** 公開ベンチマークでの順位
- **C** 社内アンケートで集めた体感の速さ
- **D** モデルの1タスクあたりの単価

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はA。自社の実タスクを決めて、同じ条件で測ります。

A **正解　自社の実タスクを1つ決めて、同じ条件で測った時間**
順位表は測定元も時点も違います。自分の仕事で条件をそろえた値だけが比べられます。

B **公開ベンチマークでの順位**
誰のハーネスで測ったかがそろっておらず、自社の作業とも中身が違います。

C **社内アンケートの体感**
16名の試験では19%遅くなったのに、本人たちは20%速いと答えていました。

D **1タスクあたりの単価**
費用は判断材料の1つですが、効果そのものを表す数字ではありません。

実務でのポイント
測る前に、対象タスク・条件・回数を決めて書き出しておきます。

削減時間の行き先

浮いた時間の多くは、日常業務に戻っています。

削減時間が生じた人

25.4%

タスク単位の削減幅

16.7%

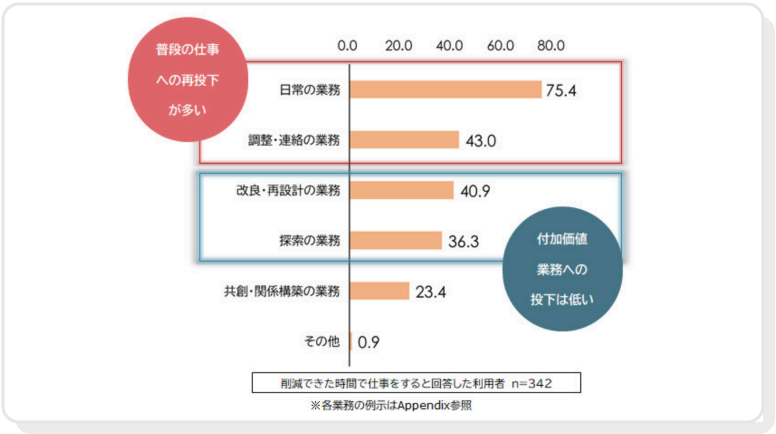
週あたり26.4分

仕事へ再投下した人

61.2%

再投下先の日常業務

75.4%



読み方

調査の対象は生成AIの利用者です。削減時間が生じた人は
4人に1人にとどまり、行き先は日常業務が中心でした。
空いた時間が新しい仕事に回るとはかぎりません。

出典: パーソル総合研究所（2026年3月13日）
https://rc.persol-group.co.jp/thinktank/
（コラム 202603130001）

章1のまとめ

見るのは順位表ではなく、
自社の1タスクです。

本日の演習3本は、その測り方をひととおり通すための練習台です。

02

Claudeファミリーの現在地

版数を覚えるのではなく、選ぶ軸と追う場所を持ち帰ります

現行ラインナップ

版数ではなく、速さ・単価・コンテキスト長で選びます。

モデル	公式の位置づけ	入力 / 出力	コンテキスト	最大出力	レイテンシ
Fable 5	長時間の自律実行向け	$10 / $50	1M	128k	Slower
Opus 5	複雑な実装と業務向け	$5 / $25	1M	128k	Moderate
Sonnet 5	速度と知能の両立	$2 / $10	1M	128k	Fast
Haiku 4.5	最速。軽い作業向け	$1 / $5	200k	64k	Fastest

単価は100万トークンあたりの入力 / 出力です。名前の先頭の Claude は省略しています。
Sonnet 5 の $2 / $10 は標準価格です。9月の値上げは行われません（2026年8月17日時点）。
出典 https://platform.claude.com/docs/en/about-claude/models/overview

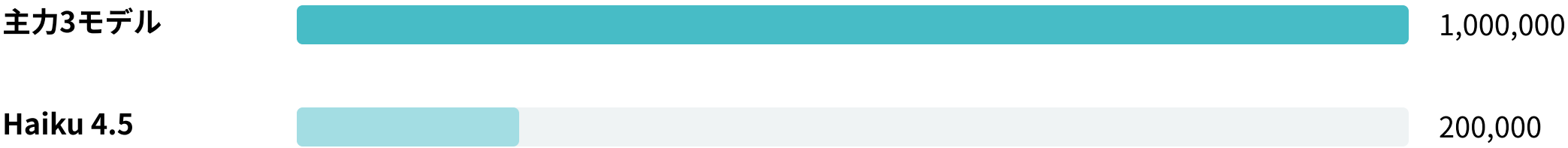
> **当日はこのページを開いて実機の一覧を見せます**
> https://platform.claude.com/docs/en/about-claude/models/overview

コンテキストの大きさ

主力3モデルは、1M トークンまで読み込めます。

1,000,000 トークン

主力3モデル（Fable 5 / Opus 5 / Sonnet 5）のコンテキスト長です。最大出力は 128k。



1M 全域が標準価格で、長文だけ割高になる料金はありません。

出典 https://platform.claude.com/docs/en/about-claude/pricing （2026年7月30日時点）

使い分けの5軸

モデル名ではなく、5つの軸で選び分けます。

公式の選択指針（2026年7月30日時点） 出典 platform.claude.com/docs/en/about-claude/models/overview

If you're unsure which model to use, start with Claude Opus 5
for complex agentic coding and enterprise work.

軸	何を見るか	判断のしかた
タスクの重さ	1回で終わるか、長く続くか	軽い整形は Haiku、通常の実装は Sonnet。 大きな改修は Opus、夜間放置は Fable。
待てる時間	対話で待つか、放置できるか	速い順に Haiku、Sonnet、Opus、Fable。 公式表の相対レイテンシの並びです。
トークン消費量	単価表だけで判断しない	新しい世代のトークナイザーは、同じ文章でも およそ30%多いトークン数になります。
Effort	モデルより先に動かす	パネルの権限モードのメニューで切り替えます。 既定の並びは low / medium / high です。
コンテキスト長	扱う資料の量で決まる	主力3モデルは 1M、Haiku 4.5 は 200k です。 追加料金なしで 1M 全域を使えます。

3つの入口の役割分担

入口は3つで、やりたいことで選びます。

Claude Code

手元のコードに
手を入れさせる

本日はここ

Claude アプリ

考えを整理して
文書にする

Claude API

他の人が使う
仕組みに組み込む

設定（CLAUDE.md・settings・MCP サーバー）は、どの入口でも共有されます。

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

※ 各ロゴは各社の商標です。

クイズ

理解度チェック

Q.「ファイルを読み飛ばして実装された」とき、先に動かすのはどれか。

A　モデルをより上位のものに切り替える

B　権限モードを Edit automatically にする

C　会話を消して最初からやり直す

D　Effort を上げてから同じ依頼をやり直す

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はD。まず Effort を上げて、同じ依頼をやり直します。

D **正解　Effort を上げてから同じ依頼をやり直す**
読み飛ばしは、確認の手数が足りない状態です。Effort を上げると読み込みが増えます。

A **モデルをより上位のものに切り替える**
知識が足りないときの手です。読み飛ばしは知識の問題ではありません。

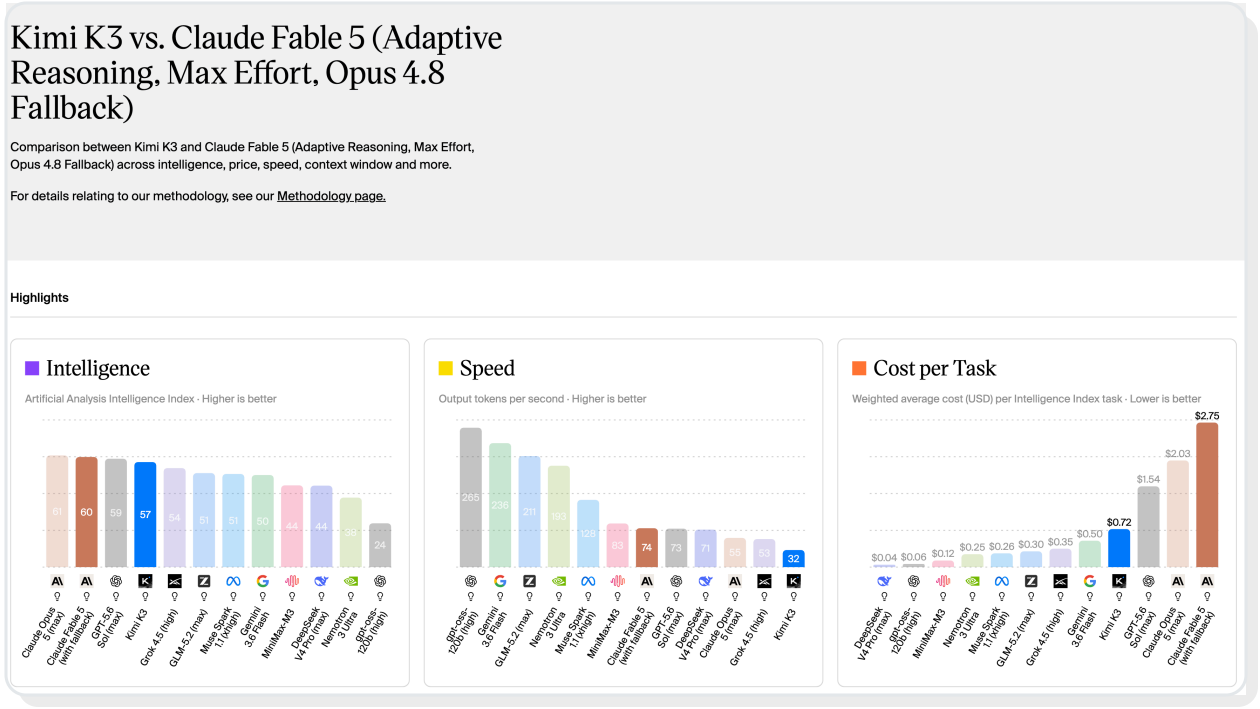
B **権限モードを Edit automatically にする**
確認の手数が減るだけで、読み込みの深さは変わりません。

C **会話を消して最初からやり直す**
同じ条件で投げ直すことになり、結果も同じになりやすいです。

実務でのポイント
モデルと Effort を同時に動かすと、どちらが効いたのか切り分けられません。

他社との比較の扱い

他社名入りの数字は、公式からは取れません。



画面: 第三者集計の比較ページ（自己申告値）

出典 https://www.anthropic.com/news/claude-opus-5
https://llm-stats.com/benchmarks/swe-bench-verified （2026年7月30日時点）

1 公式発表に他社名は出ません
比較対象は自社モデルと
the next-best model という表記です。

2 第三者集計は自己申告です
掲載104モデルすべてがベンダーの
自己申告で、独立検証は0件です。

3 使うなら性質を必ず併記します
測定元と時点が違う数字を並べても
比較にはなりません。

解説

入口をまたいで共有される設定

設定はどの入口でも同じです。

どこでも同じ	CLAUDE.md、settings、MCP サーバーの設定は ターミナル・拡張・デスクトップで共通です。
拡張だけの機能	インライン差分、@ によるファイル参照、 計画のレビュー、会話履歴の4つです。
第三者プロバイダ	Bedrock ほかを経由して使えるのは ターミナルの CLI と VS Code の2つです。

研修での意味

午前に入れるユーザー設定は、後でターミナルやデスクトップに移っても効きます。

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

直近の主要発表

直近3か月でも、認証と課金と接続の仕様が動いています。

出典 https://platform.claude.com/docs/en/release-notes/api （2026年7月30日時点）

- **2026-05-04** Workload Identity Federation が一般提供。短命トークンで認証
- **2026-05-11** Claude Platform on AWS を公開。AWS 課金と IAM 認証で利用
- **2026-05-13** cache diagnostics が公開ベータ。キャッシュが外れた箇所が分かる
- **2026-05-28** mid-conversation system messages。会話の途中で指示を差し込める
- **2026-06-26** レート制限のティアが Start / Build / Scale の3つに統合
- **2026-07-08** Console で API キーに有効期限を設定できるように

当日の注意 旧 Workbench は2026-08-17（研修当日）でアクセス終了です。案内には出しません。

最新動向の追い方

追う場所を8つ決めておくと、版数を覚えずに済みます。

情報システム部門へ共有するなら、status.claude.com の購読を先に入れます。

code.claude.com/docs/en/whats-new	週次ダイジェスト。毎週
code.claude.com/docs/en/changelog	全バージョンの変更点。ほぼ毎日
github.com/anthropics/claude-code/blob/main/CHANGELOG.md	上と同じ内容の原本。ほぼ毎日
platform.claude.com/docs/en/release-notes/api	API・SDK・Console。数日ごと
platform.claude.com/docs/en/about-claude/models/overview	モデル一覧と価格。公開時
platform.claude.com/docs/en/about-claude/model-deprecations	非推奨と引退の日付。告知時
status.claude.com	稼働状況と障害履歴。発生時
www.anthropic.com/news	製品発表・研究・事例。随時

> **週次ダイジェストはここにまとまっています**
> https://code.claude.com/docs/en/whats-new

クイズ

理解度チェック

Q. Claude Code の機能追加を最も早くまとめて確認できる場所はどれか。

- **A** 変更点が日ごとに並ぶ changelog
- **B** 週ごとの更新をまとめたダイジェスト
- **C** 現行モデルの一覧と価格のページ
- **D** 非推奨と引退の日付を載せたページ

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はB。週次ダイジェストにまとまっています。

B **正解　週ごとの更新をまとめたダイジェスト**
各週にバージョン範囲とコード例、本編ドキュメントへのリンクが付きます。

A **変更点が日ごとに並ぶ changelog**
ほぼ毎日更新で粒度が細かく、追いかけ続けるには向きません。

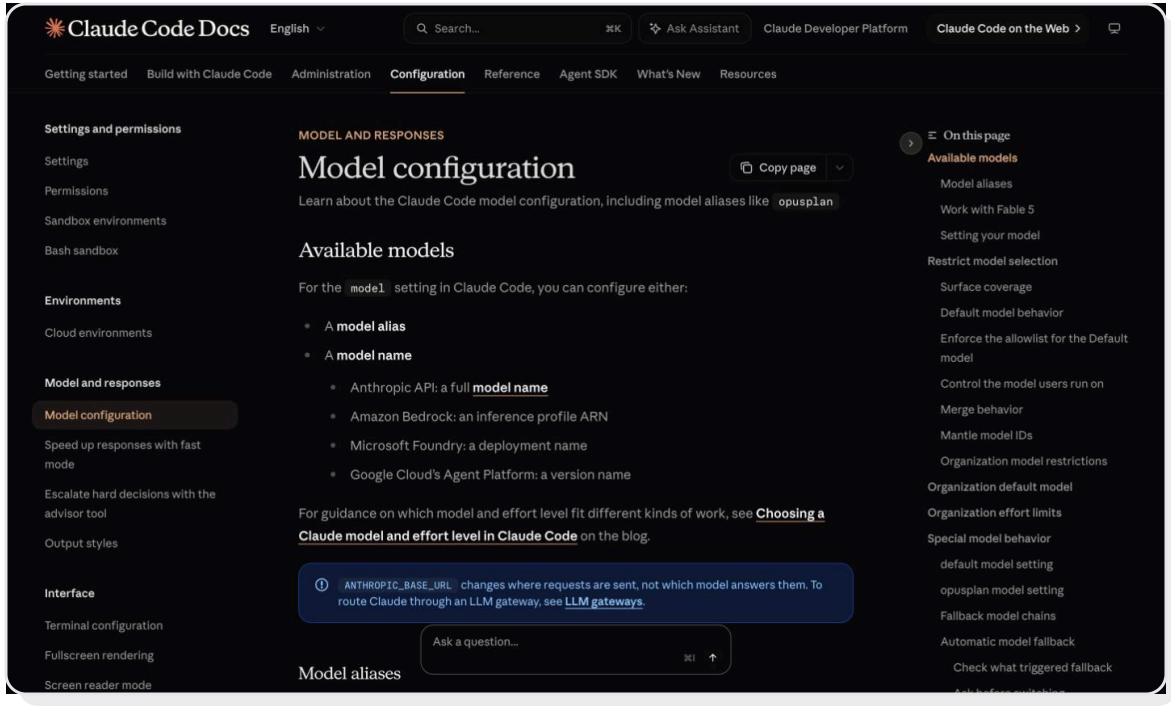
C **現行モデルの一覧と価格のページ**
更新はモデル公開時だけで、機能追加は載りません。

D **非推奨と引退の日付を載せたページ**
使っているモデルの寿命を見る場所で、機能追加とは別です。

実務でのポイント
障害の切り分けが要る人は、status.claude.com の購読を先に入れておきます。

料金の考え方

コストは、モデルを落とす前に Effortを見直します。



画面: 公式 Model configuration（effort level の記述）

1. **Effort を下げる**
   同じモデルのまま、確認の手数を減らせます。

2. **プロンプトキャッシュを効かせる**
   読み出しは基本入力の0.1倍です。
   5分のキャッシュなら1回読まれた
   時点で元が取れます。

3. **それでも足りなければモデルを見直す**
   モデルと Effort を同時に変えると、
   どちらが効いたか分からなくなります。

Claude Code は Pro 以上に含まれます。Free プランには記載がありません（2026年7月30日）。

出典 https://claude.com/pricing ・ https://platform.claude.com/docs/en/about-claude/pricing

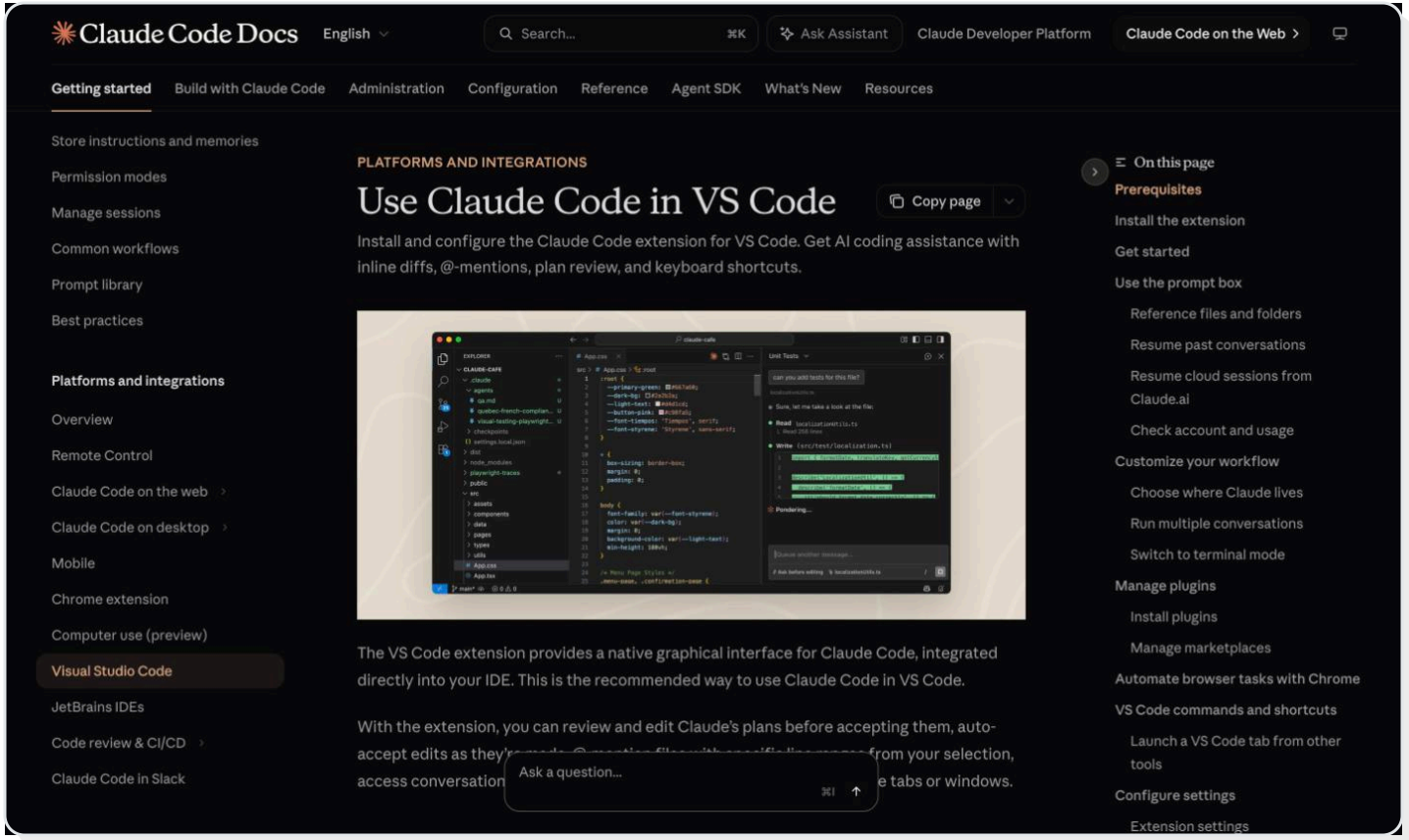
03

Claude Code の基礎（画面と位置づけ）

拡張パネルのどこに何があるかを、先に全員でそろえます

拡張の位置づけ

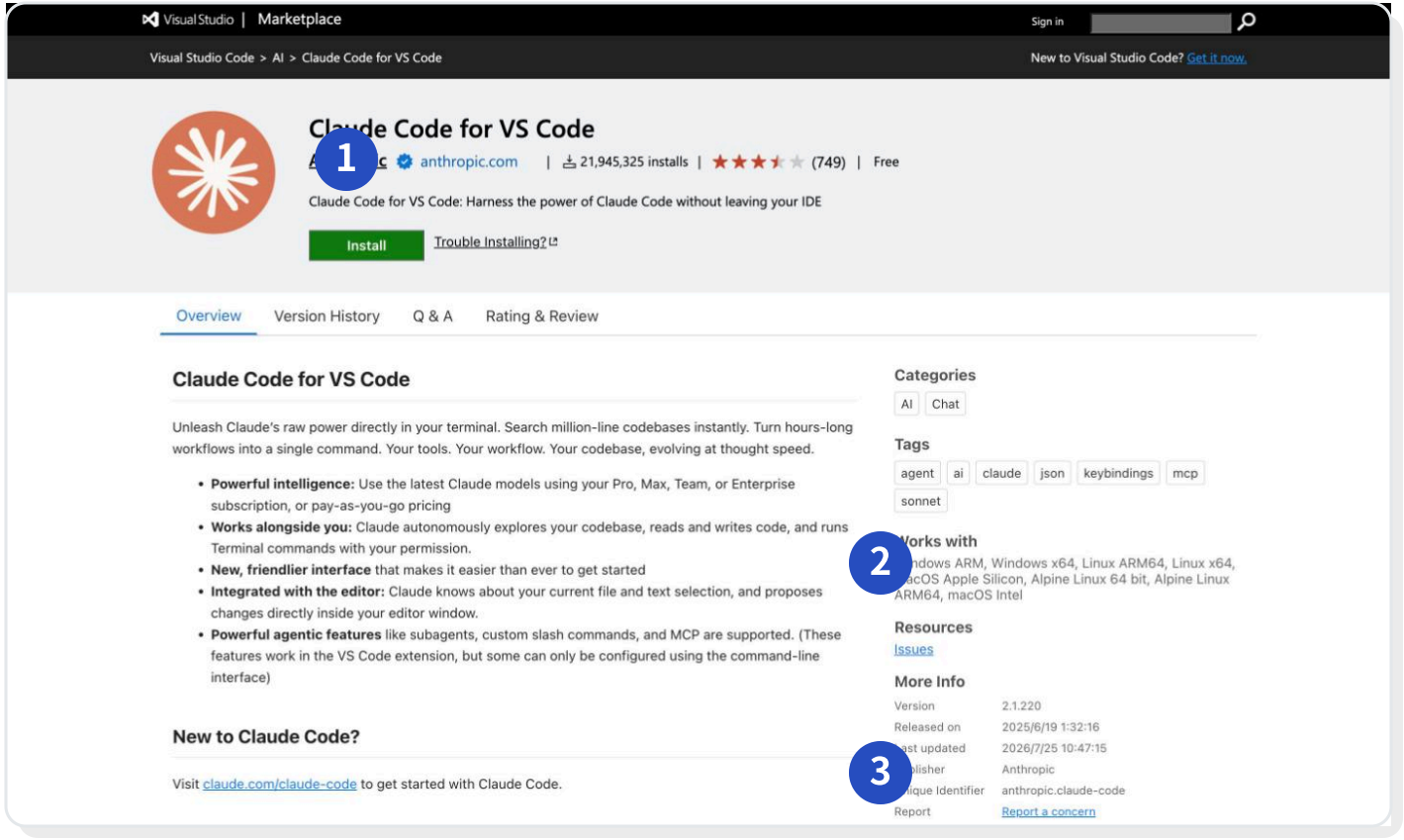
VS Code で使うなら、拡張パネルが公式の推奨手段です。



画面: 公式ドキュメント Use Claude Code in VS Code
出典: https://code.claude.com/docs/en/vs-code

拡張の身元を確認する

入れる前に、発行元と識別子を確認します。



画面: Marketplace の拡張ページ（実画面）

1 発行元
Anthropic（青いバッジ付き）

2 対応OS
Windows / macOS / Linux

3 識別子
`anthropic.claude-code`

似た名前の拡張が並びます。
発行元と識別子で選びます。

起動口4つ

クリックで開くのが基本、ショートカットは補足です。

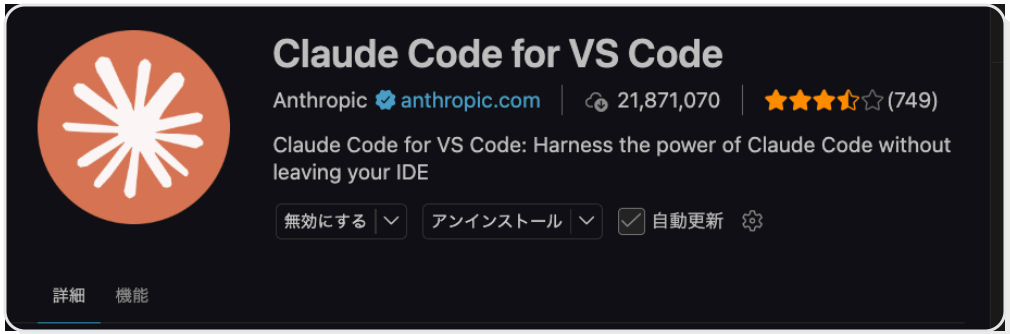
1. **エディタ右上の Spark アイコン**
   ファイルを開いているときに出ます。最短の経路です。

2. **アクティビティバーの Spark アイコン**
   左端に常に出ます。セッション一覧が開きます。

3. **コマンドパレット**
   Windows Ctrl+Shift+P / Mac Cmd+Shift+P
   一覧から Claude Code を選びます。

4. **ステータスバー右下の Claude Code**
   ファイルを開いていなくても使えます。

フォーカス切替は Windows Ctrl+Esc / Mac Cmd+Esc です。
macOS Tahoe 以降は効かないことがあります。



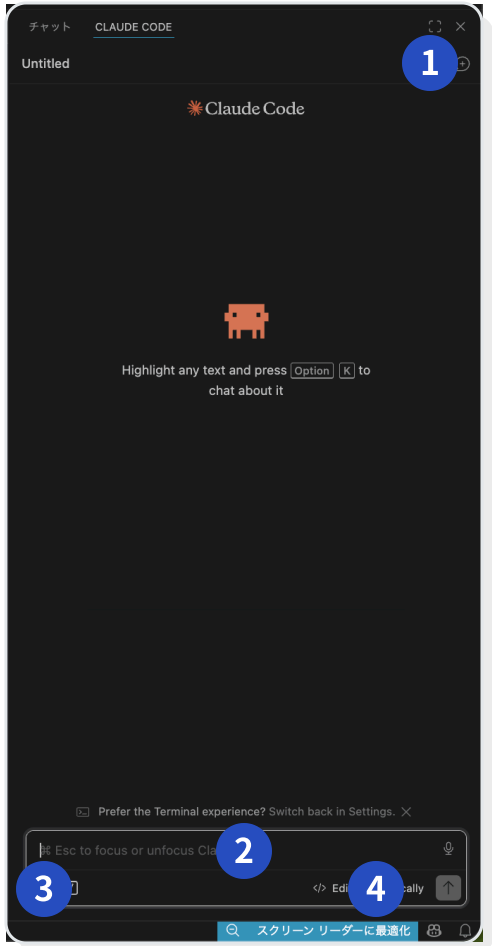
画面: 拡張の詳細ページ（実機）。
Spark アイコンが Claude Code の目印です。

キーボードで開く

パネルへフォーカス	Ctrl+Esc / Cmd+Esc
コマンドパレット	Ctrl+Shift+P / Cmd+Shift+P

パネルの部位

入力欄のまわりに、状態表示が並びます。



1 セッション履歴
Local と Web の2タブで過去の会話を開きます。

2 入力欄
指示を書きます。Shift+Enter で改行します。

3 添付（Add context）
入力欄の左下の + からファイルを追加します。

4 モードインジケータ
現在の権限モードが出ます。クリックで切り替えます。

Context usage は自動圧縮までの残量の割合を出します。
選択範囲インジケータの目に斜線は、選択が隠れている状態です。
差分ビューはエディタ領域に、元の内容と並べて出ます。
Account & Usage はコマンドメニューから開きます。

画面: 実機のパネル。この例のモードは Edit automatically。本研修は Manual に固定します

拡張と CLI の違い

本研修は拡張だけで進めます。
ターミナルで claude と打つ場面はありません。

CLI（本研修では使いません）

コマンドとスキル
すべて使えます

MCP サーバーの設定
設定できます

! のシェルショートカット
あります

Tab 補完
あります

VS Code 拡張（本研修）

コマンドとスキル
一部です。/ を押して確認します

MCP サーバーの設定
一部です。/mcp で管理します

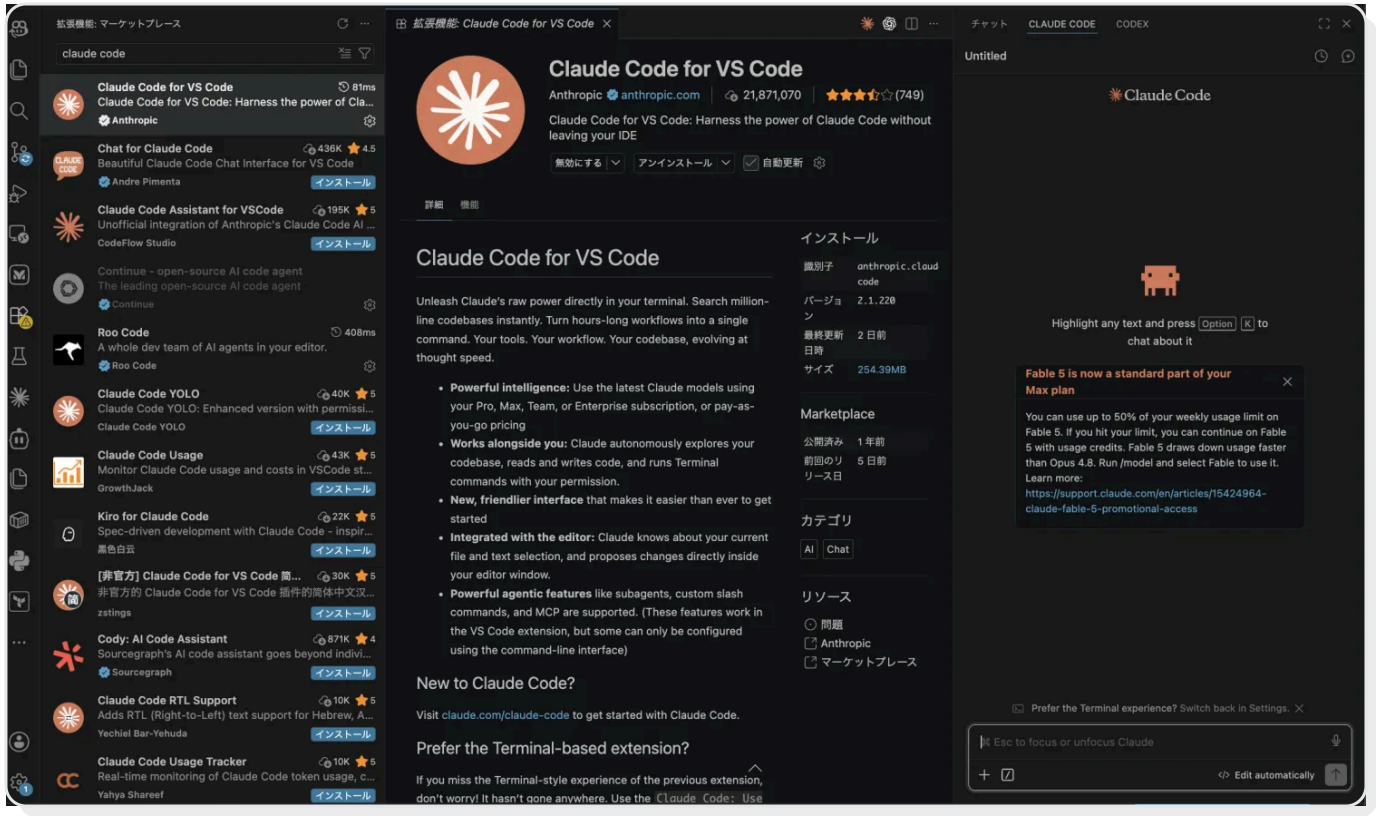
! のシェルショートカット
ありません

Tab 補完
ありません

> 設定と会話履歴は拡張と CLI で共有されます。拡張が内包する CLI は PATH に入りません。

パネルの置き場所

置き場所は3通りです。研修では右サイドバーを使います。



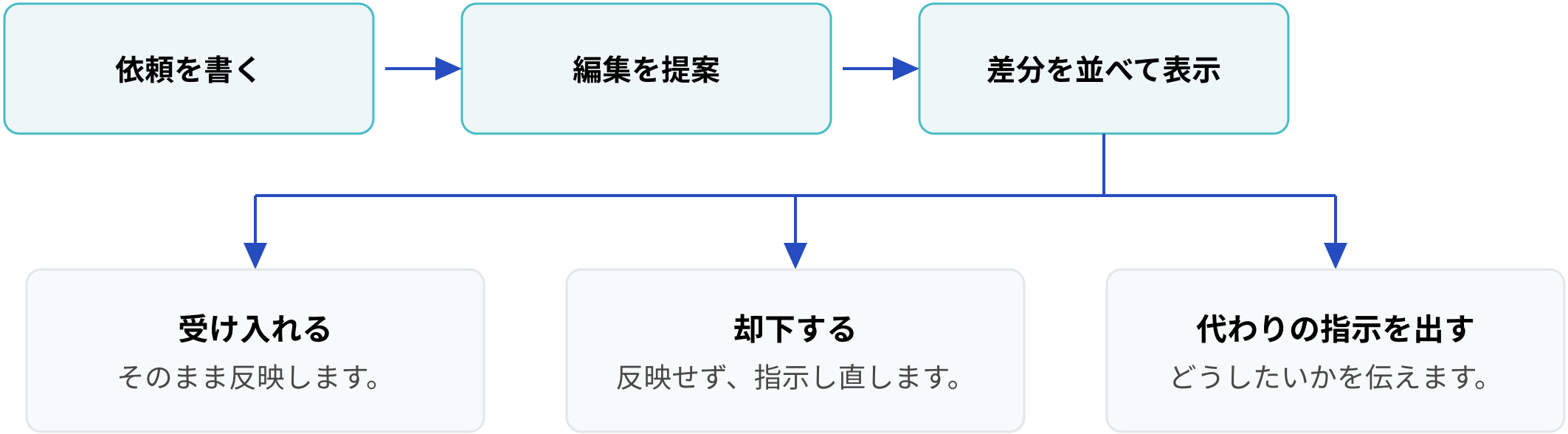
画面: パネルを右サイドバーに置いた状態（実機）

- **右サイドバー**
  Secondary sidebar
  コードを書きながら常に見えます。
- **左サイドバー**
  Primary sidebar
  Explorer と同じ側に置きます。
- **エディタ領域のタブ**
  Editor area
  ファイルと並べて開きます。

移動はパネルのタブをドラッグします。

差分の流れ

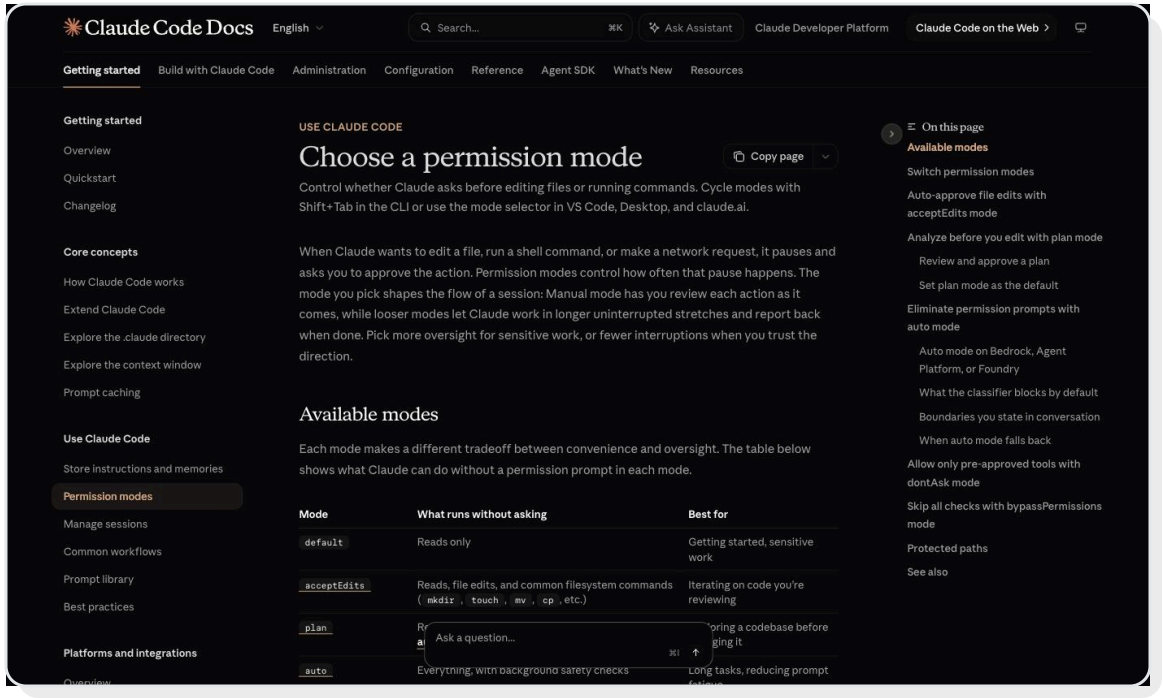
編集の提案は差分で出ます。
受け入れる・却下する・言い直すの3択です。



差分ビューの中で直してから受け入れると、手を入れたことが Claude に伝わります。
設定 autosave は既定で有効で、読み書きの前に自動保存されます。

権限モード

モードは Manual / Plan / Edit automatically の3つです。



画面: 公式ドキュメント。VS Code はモードセレクタを使うと記載

- **Manual（本研修で固定）**
  読み取りだけが確認なしで動きます。
- **Plan**
  読み取り中心。編集の前に計画を出します。
- **Edit automatically**
  読み取りとファイル編集が確認なしで動きます。

> 2026年8月14日から Pro / Max / Team の既定は auto mode です。本研修は Manual に固定します。
> チャットで頼んで変えるものではありません。CLI の Shift+Tab は拡張では使いません。

クイズ

理解度チェック Q5

Q. VS Code 拡張で権限モードを切り替える操作はどれか。

A 入力欄に「Plan モードにして」と入力する

B 入力欄の下のモードインジケータをクリックする

C Shift+Tab を押してモードを循環させる

D コマンドパレットで Developer: Reload Window を実行する

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック A5

正解はB。モードインジケータをクリックします。

B **正解 入力欄の下のモードインジケータをクリックする**
公式も「VS Code ではモードセレクタを使う」と書き分けています。

A モードはチャットの依頼では変わりません。画面の操作で切り替えます。

C Shift+Tab の循環は CLI の操作で、拡張のページには書かれていません。

D Reload Window はパネルが出ないときの対処で、モードは変わりません。

モードは画面の操作で切り替えます。会話の中で頼む方法は用意されていません。
本研修は Manual に固定します。読み取り以外は実行の前に確認が入ります。

ワークスペースの信頼

制限モードのままだと拡張は動きません。

1. **フォルダを開く**
   ファイル > フォルダーを開く で演習フォルダを開きます。

2. **信頼する を選ぶ**
   確認ダイアログで信頼する側を選びます。

3. **パネルが動くか確かめる**
   制限モードのままだと拡張は反応しません。

この1枚だけ、実機で撮影します

演習1_Lv1_集計ツール を開いた直後
ワークスペースの信頼ダイアログ

> プロジェクトの Hooks と allow ルールは、信頼したあとに効きます。
> deny と ask は信頼を待たずに効きます。本日は4回開き直すので、そのたびに出ます。

解説

保護パス

.git / .vscode / .claude への書き込みは、権限モードに関わらず毎回確認が入ります。

`.git` `.vscode` `.idea` `.claude`

これらのフォルダへの書き込みは、権限モードを Edit automatically にしていても毎回確認が入ります。エディタや Claude 自身の設定を、気づかないうちに書き換えられないようにするための仕組みです。

> **.claude/ の確認に出る選択肢**
>
> Yes, and allow Claude to edit its own settings for this session を選ぶと、そのセッションのあいだは、再確認なしで自分の設定を編集できます。

.claude/worktrees は保護の対象から外れています。

Plan モードの拡張固有の動き

計画は Markdown で開き、コメントで注文できます。

Plan モードで変わること	
Manual（本研修で固定）	1件ずつ承認しながら編集します
Plan	編集の前に、まず計画を出します
計画の開き方	Markdown のファイルとして開きます
直したいとき	その行にインラインコメントを付けます
承認	Yes, manually approve edits を選びます

> Plan モードでは、ファイルを読み、探索のコマンドを動かし、計画を書きます。
> ソースは編集しません。直してほしい点はコメントで伝えてから承認します。

計画承認の選択肢

研修では Yes, manually approve edits を選びます。

選択肢	内容
Yes, manually approve edits	承認して、編集を1件ずつ確認します。 本研修はこれを選びます。
Yes, and use auto mode	承認して auto モードで始めます。 使えないときは Yes, auto-accept edits になります。
No, refine with Ultraplan on Claude Code on the web	計画をブラウザ側のレビューへ送ります。
No, keep planning	Plan モードのまま、変えてほしい点を伝えます。

03

Claude Code の基礎

モデル・文脈・コマンド

モデルと Effort の選び方、ファイルの渡し方、会話の区切り方を扱います

モデルの切り替え

モデルはパネルの一覧から切り替えます。

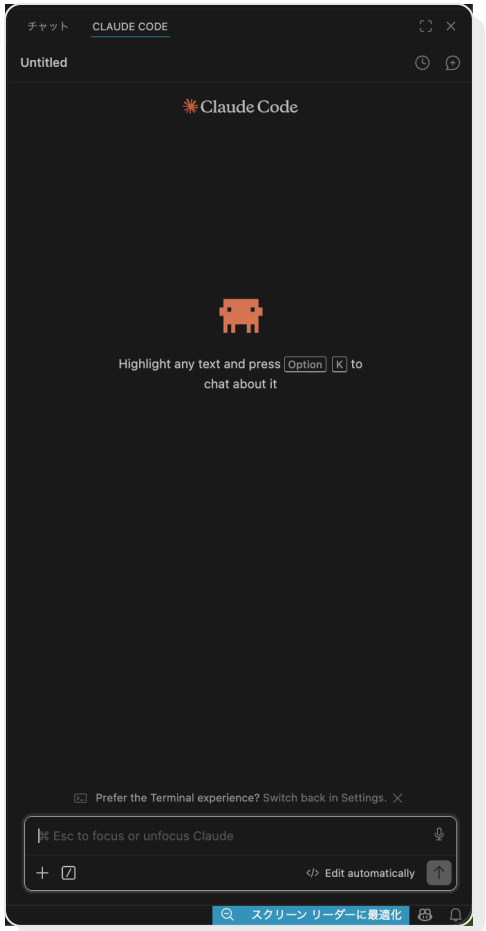
1. **パネルのメニューで「Change the AI model」を開く**
   表示名と説明つきの一覧から選びます。

2. **エイリアスで指定する**
   エイリアスは7つです。default / best / fable /
   opus / sonnet / haiku / opusplan から選びます。
   1M コンテキストは sonnet[1m] / opus[1m] と書きます。

3. **版数は資料に書きません**
   エイリアスが指す版数は、プロバイダとアカウント種別と
   時期で変わります。当日この一覧で実機を確認します。

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

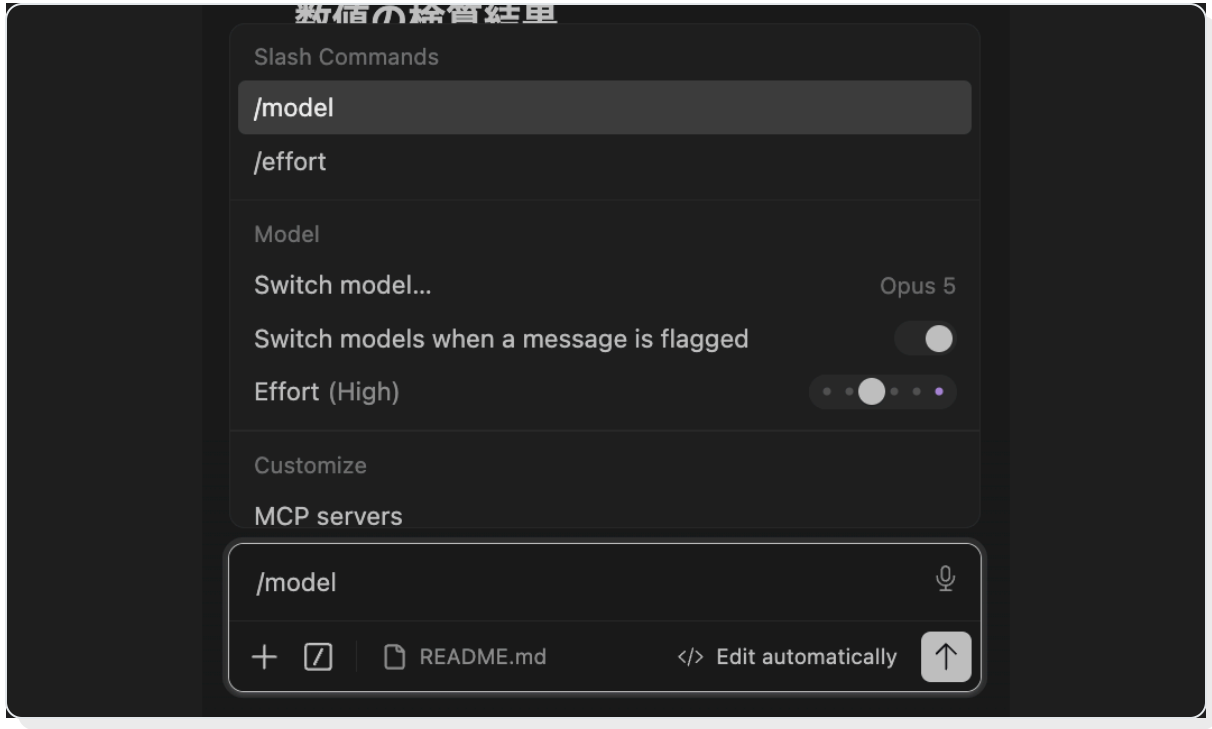
> **切り替えるときの確認**
> 会話に出力がある状態で変えると、履歴を読み直すため確認が入ります。



画面: 実機のパネル

Effort の切り替え

Effort は権限モードのメニューで切り替えます。



画面: パネルの Effort（実機）

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

1. **既定の並び**
   low / medium / high / xhigh です。
   その先に ultracode が並ぶ場合もあります。
2. **操作**
   「Click or drag to set effort level」
   「Click to cycle effort level」
3. **選べる段階**
   モデル側の申告に従います。対応
   していないモデルもあります。
4. **設定ファイル**
   settings.json の effortLevel は
   low / medium / high / xhigh の4値です。

症状から動かすノブを決める

足りないのが知識ならモデル、丁寧さなら Effort です。

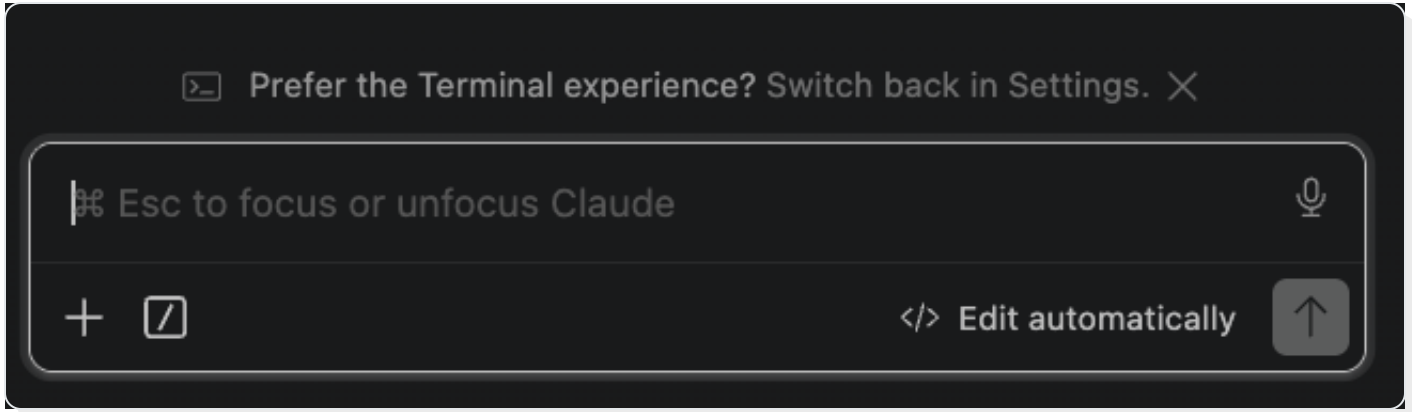
症状	動かすノブ
見つけにくいバグ、不慣れな領域 アーキテクチャの判断で迷う	**モデルを上げる** まず1段だけ上げて、変化を見ます
読み飛ばしや未実行が起きる ファイルを読まない、テストを走らせない	**Effort を上げる** モデルは変えずに1段だけ動かします
正確に書ける機械的な修正 すでに文脈にあるコードへの質問	**小さいモデルで足ります** 上げる理由がないときは戻します

切り分けの約束

モデルと Effort を同時に動かすと、どちらが効いたのか分かりません。1つずつ動かします。
プロンプトの中で効くキーワードは ultrathink だけで、think hard などは文章として扱われます。

ファイルの渡し方は4経路

ドラッグはShift を押しながら行います。



画面: 入力欄まわり（実機）

1. **@ 参照（第一手順）**
   ファイル名やフォルダ名を打つと候補が出ます

2. **Add context**
   入力欄の左下の + がこれにあたります

3. **ドラッグ＆ドロップ**
   Shift を押しながら入力欄へ運びます。押さないとエディタでファイルが開くだけです。

4. **エディタの範囲選択**
   選んだ行はそのまま共有されます

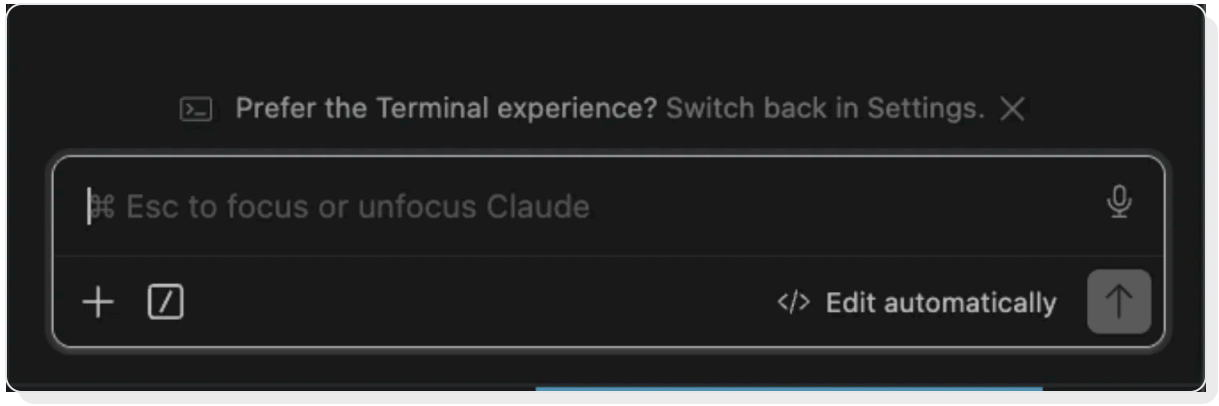
渡し方の実演

@ のあいまい一致で、候補から選びます。

1. **@auth と打つ**
   部分入力でも候補が出ます。フォルダは末尾にスラッシュを付けます。

2. **行範囲付きの参照を挿入する**
   エディタで範囲を選び、Windows は Alt+K、Mac は Option+K を押します。

3. **添付を外す**
   添付の X をクリックすると文脈から外れます。



画面: パネル下部の入力欄（実機）

自然文でのファイル指定
独立した機構としての記述はありません。
@ のあいまい一致で拾われる補助として
扱います。第一手順は @ です。

クイズ

理解度チェック

Q. 拡張でファイルを渡す操作のうち、公式に記述があるのはどれか。

- **A** ファイル名を文章で書いて伝える
- **B** エディタのタブを右クリックして送る
- **C** モードインジケータをクリックする
- **D** Shift を押しながら入力欄へドラッグする

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はD。Shift を押しながらドラッグします。

D **正解　Shift を押しながら入力欄へドラッグする**
受け側に「Drop to attach as context」が出て、添付として文脈に入ります。

A **ファイル名を文章で書いて伝える**
独立した機構としての記述がありません。@ のあいまい一致で拾われる補助です。

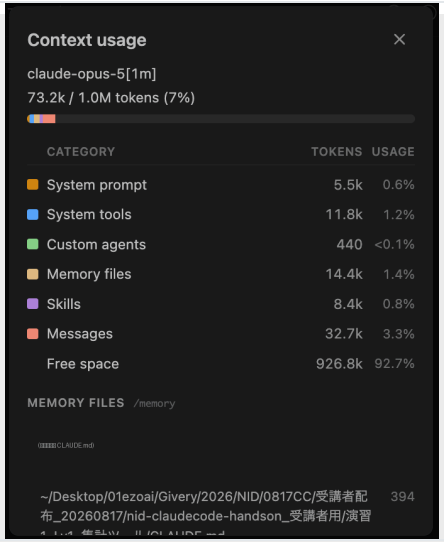
B **エディタのタブを右クリックして送る**
公式ドキュメントに、この操作の記述はありません。

C **モードインジケータをクリックする**
権限モードの切り替えです。ファイルはパネルに渡りません。

実務でのポイント
Shift を押さないと、エディタでファイルが開くだけでパネルには渡りません。

コンテキストと使用量の見方

残量はContext usage、内訳は Account & Usage です。



画面: /context の Context usage（実機）

Context usage

自動圧縮までの残りの割合が出ます。圧縮が走ると、その旨のメッセージが会話に出ます。

Account & Usage

使用量のバーに加えて、コマンド別・MCP 別・プラグイン別・サブエージェント別の内訳が出ます。

/compact と /clear

切り替えが目的なら/clear のほうが安く済みます。

/compact

- 同じ会話を続けたまま、要約に置き換えます
- 話の流れは引き継がれます
- 要約のために会話全体を読むので、
  そのぶん費用がかかります

/clear

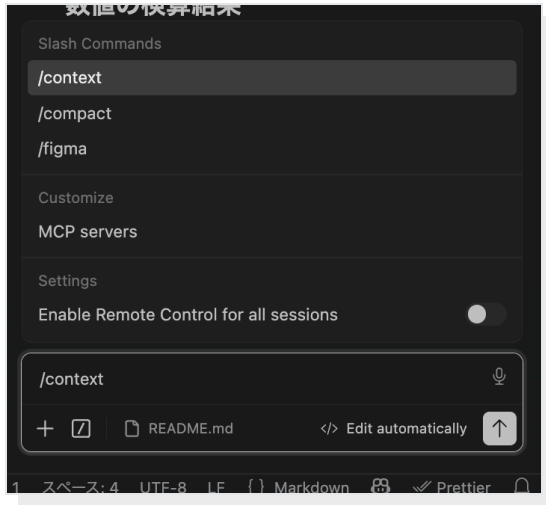
- 会話を捨てて、空から始めます
- 前の話は引き継ぎません
- トークンを消費しません
- 無関係な作業へ移るときに使います

自動の圧縮があります

上限に近づくと自動で圧縮されるので、コンテキストが埋まってもセッションは止まりません。

スラッシュコマンド

/init でプロジェクトの CLAUDE.md の雛形を作ります。



画面: / で開くコマンドメニュー（実機）

/init　　**CLAUDE.md の雛形を作る**

プロジェクトの構成を読み取って作ります。本日は演習ごとに毎回実行します。

そのほかに使うもの

/context	使用量の内訳を色付きのグリッドで見ます
/memory	CLAUDE.md の編集と auto memory の切り替え
/permissions	承認ルールを設定します
/status	セッションの設定を表示します
/help	使えるコマンドの一覧を表示します

> **未確認**
> ／ を押すと入力に合わせて候補が絞り込まれます。
> 全一覧は公式に列挙が無いため、当日実機で確認します。

クイズ

理解度チェック

Q. 無関係な作業に切り替えるときに使うのはどれですか。

A /clear を実行する

B /compact を実行する

C /init を実行する

D /status を実行する

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はA。/clear で会話を捨てて空から始めます。

A　正解　/clear を実行する
前の話は引き継ぎません。トークンも消費しないので、切り替えには一番安く済みます。

B　/compact を実行する
同じ会話を続けたまま要約に置き換えます。会話全体を読むぶん割高です。

C　/init を実行する
プロジェクトの CLAUDE.md の雛形を作るコマンドです。

D　/status を実行する
現在のセッションの設定を表示します。文脈は変わりません。

実務でのポイント
残したい観点があるときだけ、自分で /compact に指示を添えて打ちます。

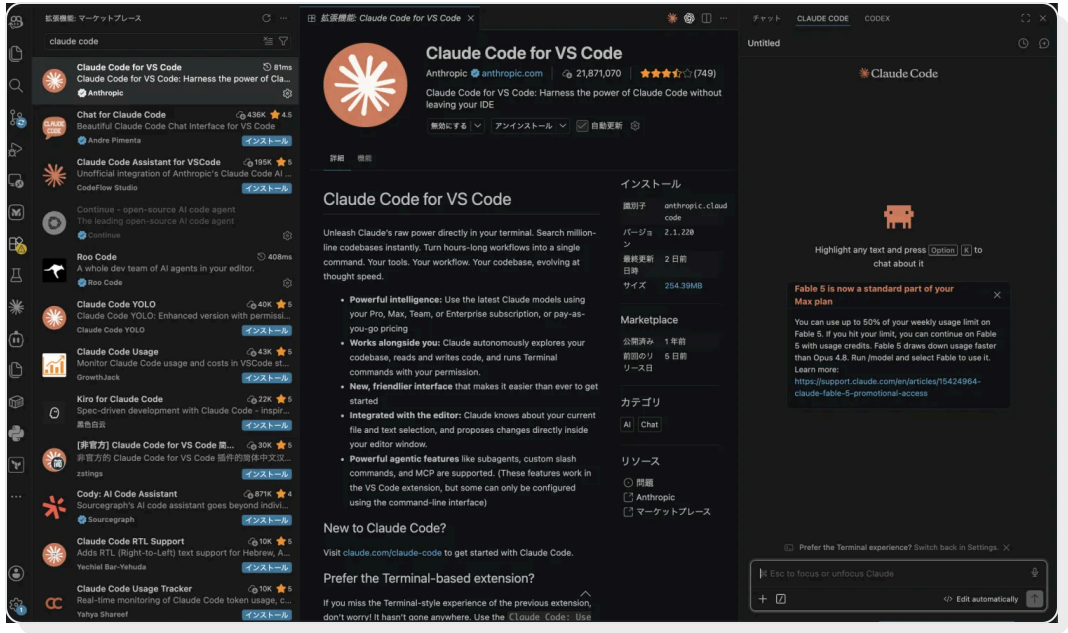
個人設定のバックアップ

.bak を取ってから、ユーザー設定を触ります。

1 **「ファイル > 開く」で開く**
ユーザー設定の settings.json を開きます。

2 **「名前を付けて保存」で退避する**
同じフォルダに settings.json.bak として保存します。

3 **CLAUDE.md も同じ手順で退避する**
CLAUDE.md.bak を作ります。ファイルが
無い人は、この時点では作りません。



画面: 拡張を入れた VSCode（実機）

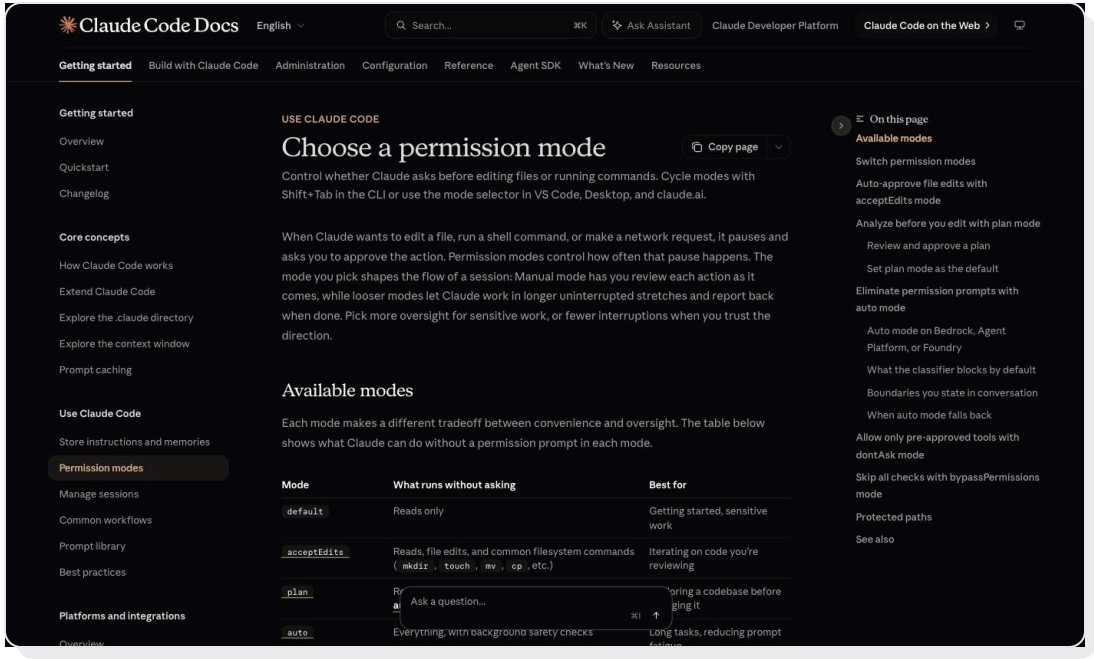
なぜ先に取るのか
社内プロキシの設定が入っている JSON を壊すと、その日つながらなくなります。

破壊的コマンドを止める

deny に、戻せない操作を並べておきます。

```
{
"permissions": {
"deny": [
"Bash(rm -rf:*)",
"Bash(rm -fr:*)",
"Bash(sudo:*)",
"Bash(mkfs:*)",
"Bash(dd:*)",
"Bash(shutdown:*)",
"Bash(reboot:*)"
]
}
}
```

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



画面: 公式の権限モードのページ

貼る場所

すでに permissions がある人は、deny の配列へ
追記します。コピー元は配布フォルダにあります。

パスと戻し方

壊れたら.bak から上書きして戻します。

項目	Mac	Windows
設定フォルダ	`~/.claude/`	`%USERPROFILE%\.claude\`
設定ファイル	`~/.claude/settings.json`	`%USERPROFILE%\.claude\settings.json`
個人の指示	`~/.claude/CLAUDE.md`	`%USERPROFILE%\.claude\CLAUDE.md`

壊れたときの戻し方

.bak の名前を settings.json に戻して上書きします。

研修終了時に戻す手順は、配布物の README.md 末尾にも同じ文言で載せてあります。

補足

保存後の確認

保存したら、エラーが出ていないことを確かめます。

壊れやすいのはこの2箇所

```
# 正しい形
{
"statusLine": { "type": "command", "command": "..." }
}

# 1 最後の項目にカンマを残す "command": "...",
# 2 閉じ括弧の数が合わない } が1つ多い・少ない
```

どこを見るか

・エディタに波線が出ていないこと
・問題パネルにエラーが並んでいないこと
・壊れやすいのはカンマの位置と閉じ括弧

直らないとき

.bak から上書きで戻し、もう一度
貼り直します。

このあと触るもの

ユーザー設定の CLAUDE.md と settings.json を、ここから続けて書き足していきます。

ワーク

ミニ演習M2 ユーザー設定の CLAUDE.md

~/.claude/CLAUDE.md に置いた指示は、どのフォルダを開いても効きます。

このミニ演習ですること

~/.claude/CLAUDE.md の末尾に、指示を1つだけ追記します。
追記したら新しい会話を始めて、その1行が出るかを見ます。
開くフォルダに関係なく効き続ける層だと分かれば達成です。

触る場所	~/.claude/CLAUDE.md（Windows は %USERPROFILE%\.claude\CLAUDE.md）
生成物	追記した CLAUDE.md 1ファイル。見出し1つと本文3行だけです。
前提	settings.json と CLAUDE.md の .bak を取ってあること

逐語の正本は配布物の ミニ演習/ユーザー設定CLAUDE_md/追記する内容.md です。

追記する内容

見出しを1つ付けて、次の内容をそのまま貼ります。項目は足しません。

~/.claude/CLAUDE.md の末尾に追記する内容

```
## ツールを使う直前の1行

ファイル編集・コマンド実行・検索のツールを呼ぶ直前に、次の1行だけを
出力します。前置き・敬語・補足説明は書きません。

要約：使用するツール

「使用するツール」の位置には、これから呼ぶツールの名前を書きます。
```

顧客に指定された文言です。項目を足したり順番を変えたりしません。
コロンは全角です。プロジェクト側（M4）で使う半角の 要約: とは
別の行として、両方が同時に出ます。

逐語の正本: 配布物の ミニ演習/ユーザー設定CLAUDE_md/追記する内容.md

動作確認

追記のあと、新しい会話で読み取りだけの依頼を1つ出します。

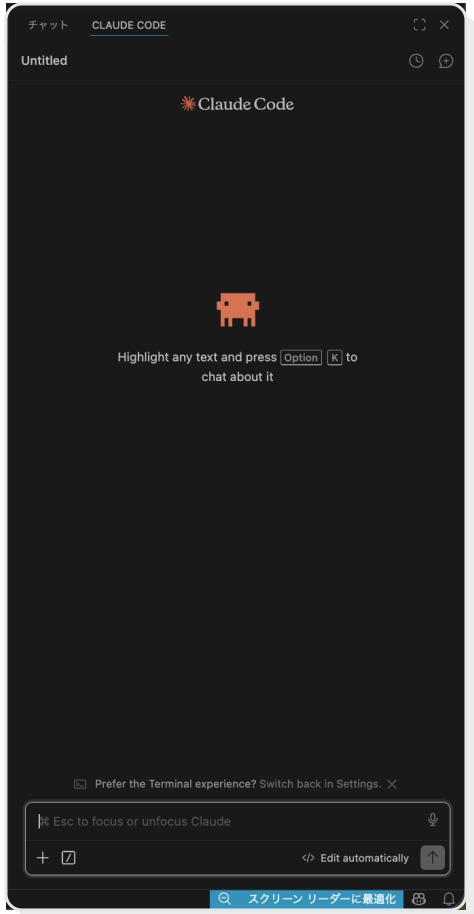
1. **追記して保存する**
   ~/.claude/CLAUDE.md の末尾に貼り、保存します。

2. **新しい会話を始める**
   設定ファイルの変更を反映させるためです。

3. **読み取りだけの依頼を出す**
   data/work_log.csv の先頭3行を見せてください。

4. **1行が出るかを見る**
   ツールを呼ぶ直前に 要約： で始まる1行が出れば成功です。

読み込みの確認は /context の Memory files でもできます。
開けない場合は、1行が出たことをもって確認とします。

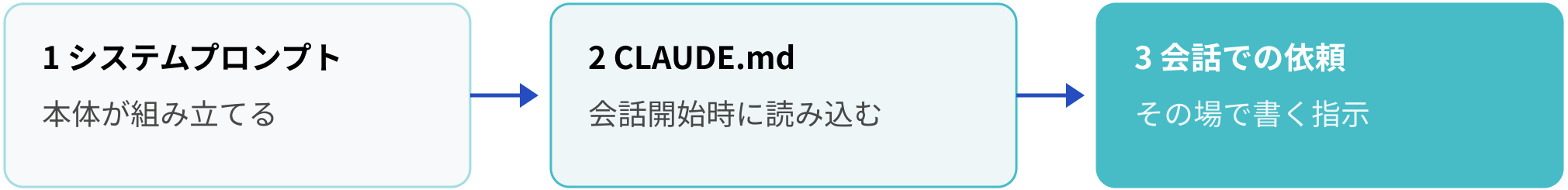


画面: 実機のパネル

解説

CLAUDE.md の届き方

CLAUDE.md はシステムプロンプトの一部ではなく、その後に**ユーザーメッセージ**として届きます。



覚えておくこと

読んで従おうとしますが、厳密な遵守は保証されません。1行が出ない回があっても壊れているわけではありません。必ず走らせたいものは、フック（M5）で書きます。

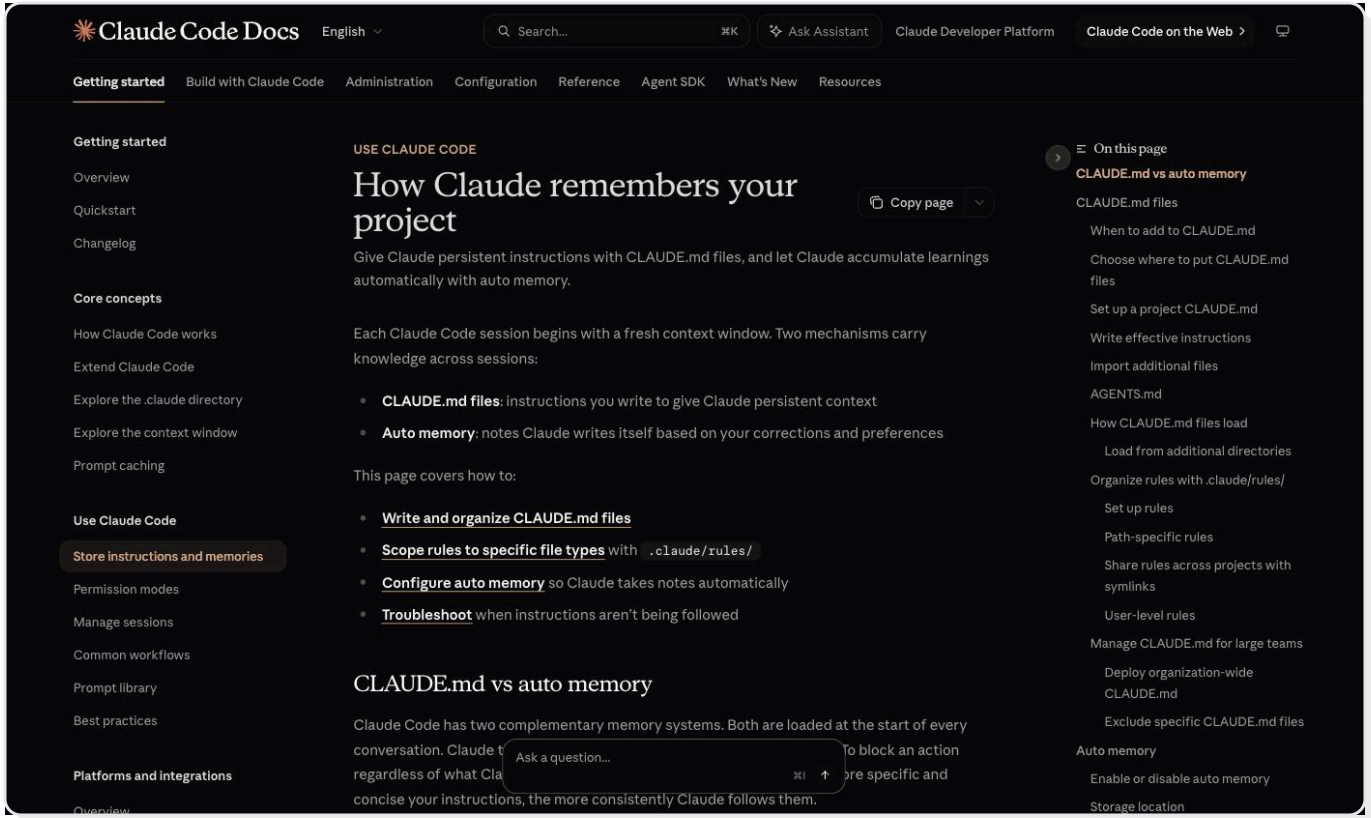
出典: https://code.claude.com/docs/en/memory

追加と考察: 「前置き・敬語・補足説明は書きません」の1行を消して、出力の長さが変わるかを見ます。

公式ドキュメント How Claude remembers your project

記憶の仕組みは、人が書く CLAUDE.md と Claude が書き溜める auto memory の2つです。

ファイル種別ごとに分けたいときは .claude/rules/ を使います。どちらも会話の開始時に読み込まれます。



出典: How Claude remembers your project https://code.claude.com/docs/en/memory

ワーク

ミニ演習M3 statusLine の設定

statusLine は正しい設定ですが、
拡張のチャットパネルには出ません。

先に伝える事実

拡張本体の画面実装に statusLine の参照は1件もありません。
設定項目 disableAllHooks の説明が「すべてのフックと statusLine の実行を無効化する」となっており、フックと同じ実行系の機能です。
だから M1（パネルで見る）と M3（設定を書く）を分けています。

このミニ演習のOK基準

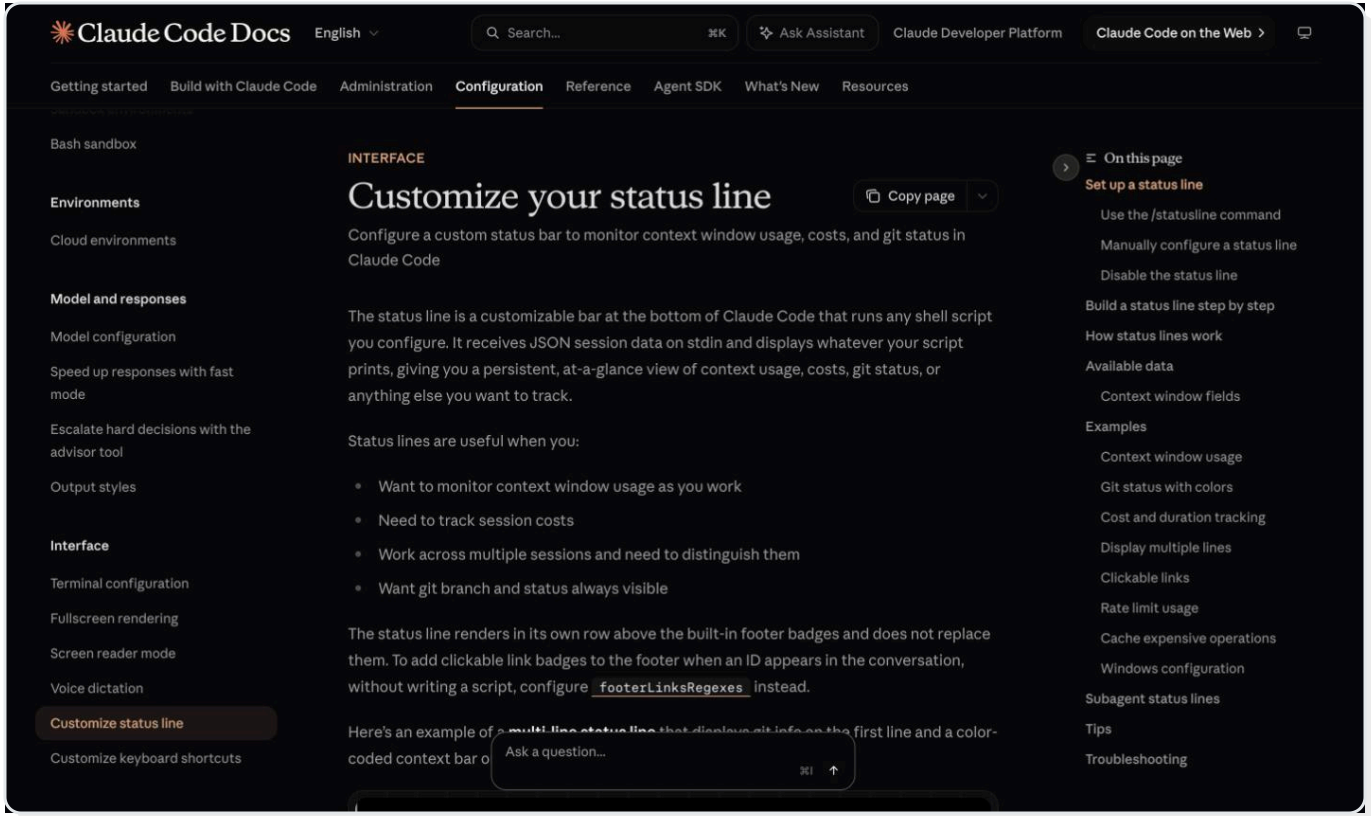
設定ファイルとスクリプトが正しく書けたことです。パネルでの表示確認は求めません。

出典: https://code.claude.com/docs/en/statusline
画面実装の確認は、制作時に実機の拡張本体を調べた結果です。

公式ドキュメント Customize your status line

公式ページはコンテキスト量・コスト・レート上限を出せると書いていますが、記述はターミナル前提です。

拡張のパネルに描画されるという記載は、公式ページのどこにもありません。



出典: Customize your status line https://code.claude.com/docs/en/statusline

statusLine の3層

本日手を動かすのは必修Bで、表示の確認は求めません。

層	見るもの	操作	扱い
必修A	モデル名・コンテキスト 残量・使用量	パネルで見る	M1 で確認済み
必修B	**設定ファイルと スクリプト**	**~/.claude/ に 2つ書く**	**本演習。表示確認は 求めません**
任意C	statusLine の表示	ターミナルモードに 切り替える	任意。切り替えた人だけ

必修A は M1 で確認済みです。任意C は必修ではなく、切り替えた人だけ表示を見ます。

statusline.py の骨

標準入力の JSON から3つの値を取り出して、1行だけ印字します。

~/.claude/statusline.py（配布物 ミニ演習/statusline/statusline.py）

```
# 標準入力の JSON を読む。壊れていても {} を返す
data = read_payload()
model = pick(data, ("model", "display_name"), "unknown")
used = to_float(pick(data, ("context_window", "used_percentage"), 0))
cost = to_float(pick(data, ("cost", "total_cost_usd"), 0))
print("{0} | context {1:.0f}% | ${2:.3f}".format(model, used, cost))
```

決めておくこと

- used_percentage はセッション序盤に null になります。既定値は 0 です。
- キーが欠けても値が null でも、必ず1行を印字して終了コード 0 で終わります。
- 出力が空か終了コードが非ゼロだと、表示そのものが消えます。

模擬 JSON での動作確認

1行が出て、終了コードが 0 になることを見ます。

期待する出力

```
# 模擬 JSON を流す（Mac）
echo '{"model":{"display_name":"Opus"}, ...}' \
| .claude/statusline.sh ; echo "exit=$?"

Opus | context 42% | $0.123
exit=0
```

確認するもの

1 出力が1行だけ出ること
2 終了コードが 0 であること
3 モデル名・コンテキスト%・
セッションコストの3つが入っていること
4 Windows の確認コマンドは
hints/M3_statusline_hint.md にあります

Mac の確認コマンドと期待する出力

```
echo '{"model":{"display_name":"Opus"},"context_window":{"used_percentage":42},
"cost":{"total_cost_usd":0.1234}}' | python3 ~/.claude/statusline.py

Opus | context 42% | $0.123
```

settings.json の書き方

必須のキーは type と command の2つだけです。

OS	settings.json に貼るブロック	注意
Mac	`"type": "command",` `"command": "python3 ~/.claude/statusline.py"`	~ と python3 をそのまま使えます。

Windows	`"type": "command",`
	`"command": "python C:/Users/<ユーザー名>/`
	`.claude/statusline.py"`
		絶対パスで書きます。
		区切りはスラッシュです。
		コマンド名は python です。

1つの設定例で両方を賄いません。自分の OS の側だけを、settings.json の一番外側の { } の中に貼ります。

貼り方の詳細は配布物の ミニ演習/statusline/README.md にあります。

質問

OK基準と追加と考察

設定ファイルとスクリプトが書けていれば、このミニ演習は達成です。

OK基準（3つ）

1 statusLine が入っていて、JSON が壊れていないこと

2 スクリプトが模擬 JSON を受け取り、1行を返して終了コード 0 で終わること

3 出力にモデル名・コンテキスト%・セッションコストの3つが含まれること

追加と考察

/clear を1回実行して、金額が 0 に戻ることを見ます。
セッション単位の数字であることの確認です。

パネルでの表示確認は求めません。表示を見るのはターミナルモードに切り替えた人だけです。

補足

任意C ターミナルモードでの表示

切り替えた人だけ表示を確認します。必修ではありません。

切り替えと戻し方

切り替える
拡張設定 claudeCode.useTerminal を有効にする
確認する
ターミナルモードで開き直し、1行が出るか見る
戻す
設定を元に戻す。戻さないと以降の画面と合わなくなる

切り替えと戻し方

1 拡張設定 claudeCode.useTerminal
を有効にします
2 ターミナルモードで開き直します
3 確認したら、この設定を元に戻します

元に戻さないと、以降の説明と
画面が合わなくなります

ターミナルモードで実際に描画されるかは未確認です。当日の実機で確かめます。
パネルで数字を見る場所は M1 のとおりです（Context usage と Account & Usage）。

04

ハーネス3段階の地図

作り手が実装する層と、利用者が設定する層を分けて整理します

2つのハーネス

作り手が実装する層と利用者が設定する層は別ものです。
本日さわるのはユーザーハーネスのほうです。

エージェントハーネス

作り手がモデルの周りに実装する足回り

- 制御ループ（モデル呼び出しの司令塔）
- ツールとメモリ
- コンテキスト管理
- ガードレール
- 検証ループ

Claude Code の内部がこれにあたります

ユーザーハーネス

利用者が自分の環境に合わせて作る制約

- CLAUDE.md（ルールファイル）
- Skills（手順の標準化）
- Hooks（決定論的な行動強制）
- MCP（外部ツール連携）
- lint と型チェック
- パイプライン化

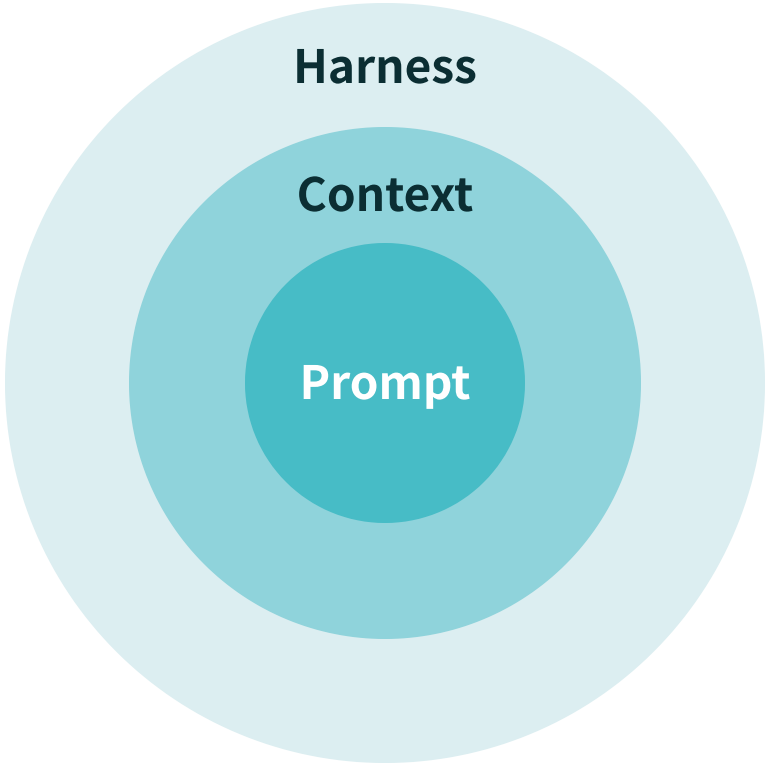
本日さわるのはこちらです

出典 r.kagaya「ハーネスエンジニアリングの定義」Zenn 2026年4月23日

※ ロゴは各社の商標です。

ハーネス・コンテキスト・プロンプト

関心の射程は、プロンプトから外側のハーネスへ広がります。



Harness ⊇ Context ⊇ Prompt

外側ほど設計対象が広くなります

プロンプトを磨く段階から、渡す情報を
組み立てるコンテキスト設計へ、さらに
実行の足回りを決めるハーネス設計へと進みます。

- **Prompt** 1回の指示の書き方
- **Context** 渡す情報の組み立て
- **Harness** 実行の足回りと制約の設計

出典 Martin Fowler が紹介した同心円モデル

ユーザーハーネスの4象限

CLAUDE.md と Skill が埋めるのは右上の一角だけです。

	計算的（決定論・安価・高速）	推論的（LLM の判断）
ガイド 事前・予防	**LSP、CLI、コードモッド** 書き始めを機械で揃える	**CLAUDE.md と Skills、設計資料** 読ませて従わせる層。本日の中心
センサー 事後・自己修正	**linter、型チェック、テスト** 決定論で必ず止める	**AI レビュー、LLM-as-judge** 意味のずれを見つける

本日埋める象限　　手つかずになりがちな3つ

Lv1 / Lv2 / Lv3 の定義

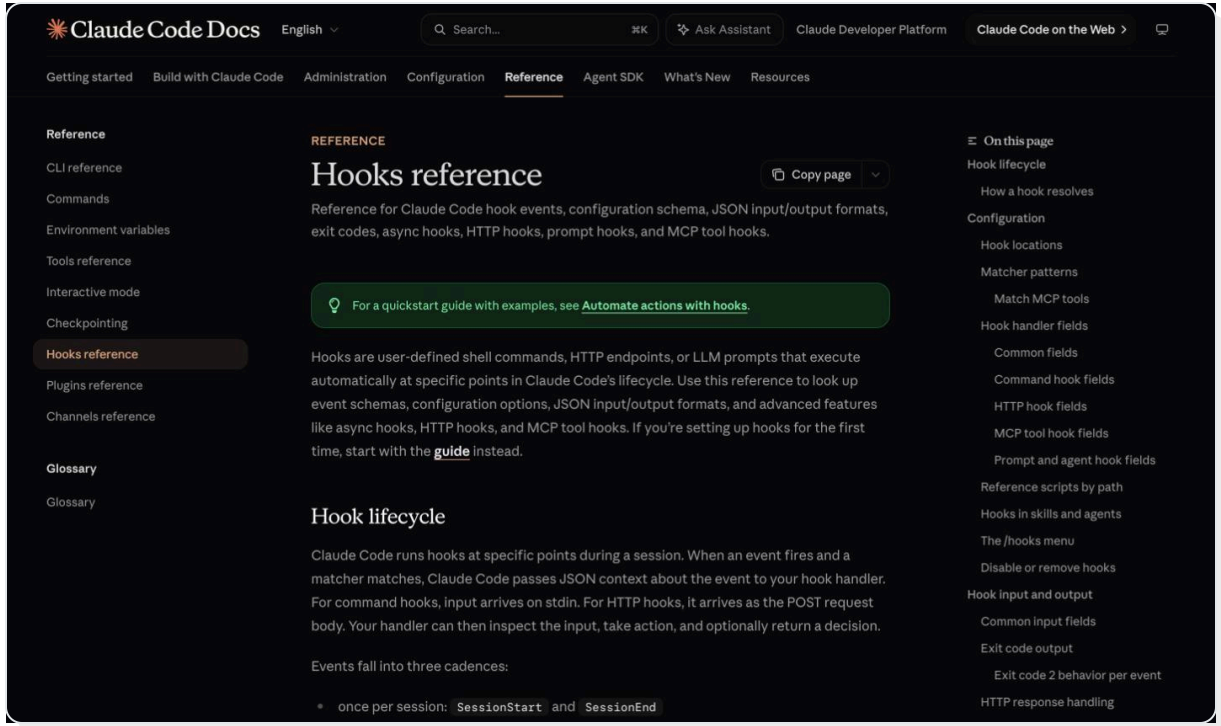
3つの段階は、環境に何が置いてあるかで分かれます。

段階	環境の状態	受講者がやること	何を体で分かってもらうか
Lv1 用意された ハーネスを使う	CLAUDE.md 1枚 / Skill 2本 docs 4種と settings.json	/init で読み込みを確認 用意された Skill を呼ぶ	前提が書いてあると 指示が短くて済むこと
Lv2 ハーネスが無い 状態から作る	プロジェクト側は空 ユーザー設定は効いている	/init で1枚作る 規約のひな型を持ち込む Hook を自然文から作る	前提が無いと毎回 同じことを書かされる
Lv3 組み合わせて 回す	rules 2本 / Skill 3本 Subagent 2本 / Hook 1本 deny と docs 5種	仕様から実装する 下調べとレビューを任せる	指示の量ではなく 確認の質が変わること

Lv3 が正解ではありません。まず Lv1 相当を1枚置き、効いた規約を Hook に上げる順で育てます。

抜けやすい3つと効かせ方の原則

必ず走らせたいものはHooksに書きます。



画面: 公式ドキュメント Hooks reference

抜けやすい3つ

1. **計算的センサー**
   lint と型チェックを、実行ループそのものに組み込む
2. **検証ループ**
   出力を別の仕組みで確かめ、誤りを自己修正させる
3. **評価と計測**
   効果をデータで確かめ、次に直す場所を決める

本日の対応は 演習1=Lv1、演習2=Lv2、演習3=Lv3。ミニ演習7本がその間を埋めます。
出典 https://code.claude.com/docs/en/hooks

クイズ

理解度チェック

Q. 機械で判定できる規約を毎回止めたいとき、どの象限に置きますか。

- **A** 事前・予防 × 推論的（CLAUDE.md に強く書く）
- **B** 事前・予防 × 計算的（ひな型を先に配る）
- **C** 事後・自己修正 × 計算的（lint と型チェックを実行ループに入れる）
- **D** 事後・自己修正 × 推論的（AI にレビューを頼む）

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はC。決定論で判定できるものは Hook と lint に置きます。

C　正解　事後・自己修正 × 計算的（lint と型チェック）
毎回同じ基準で止まるので、Hook で実行ループに組み込みます。

A　事前・予防 × 推論的（CLAUDE.md に強く書く）
読んで従おうとする層で、毎回守られる保証はありません。

B　事前・予防 × 計算的（ひな型を先に配る）
書き始めは揃いますが、書いた後の違反は止まりません。

D　事後・自己修正 × 推論的（AI にレビューを頼む）
意味のずれには強い一方、判定が毎回同じとは限りません。

実務でのポイント
必ず守らせたい規約は Hook に、方針や背景は CLAUDE.md に分けて書きます。

開き直しの3手

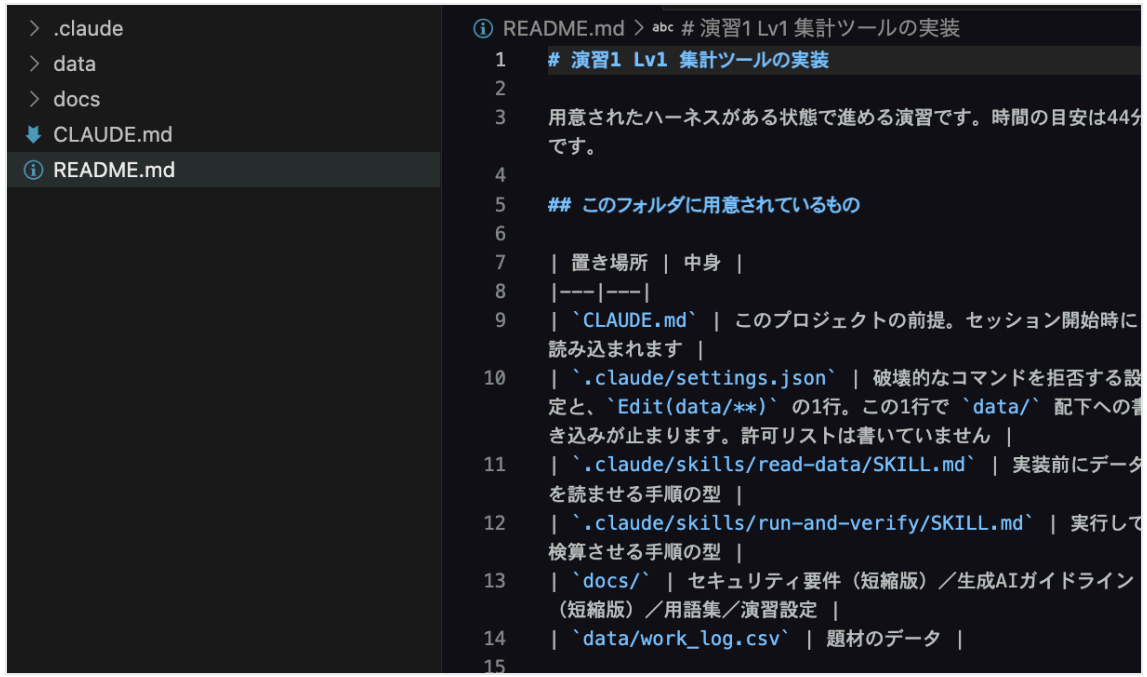
開き直すたびに会話は新しく始まります。
前の演習の文脈は持ち越されません。

- **「ファイル > フォルダーを開く」**
  ①で開くのは 演習1_Lv1_集計ツール

- **信頼するを選ぶ**
  制限モードのままだと拡張は動きません

- **Spark アイコンでパネルを開く**
  エディタ右上のアイコンです

午前に入れたユーザー設定（~/.claude/）は、
新しいセッションでも効いています。



画面: 演習1 のフォルダを開いた直後（実機）

開き直し4回の一覧

演習ごとに開き直して、置いてあるものの違いを見ます。

回	開くフォルダ	そこで確かめること
①	演習1_Lv1_集計ツール 昼休憩の直後	午前に入れたユーザー設定が、新しい セッションでも効いていること
②	演習2_Lv2_不具合修正 演習2 の前	プロジェクト側に .claude も CLAUDE.md も無いこと
③	予備_チケット管理 M7 の前	.claude を持たないフォルダ M7 の「入れる前」の状態
④	演習3_Lv3_ログ解析 演習3 の前	rules・Skill・Subagent・Hook が 揃っていること

開き直しの手順は毎回同じで3手です。前のページに戻って確認できます。

05

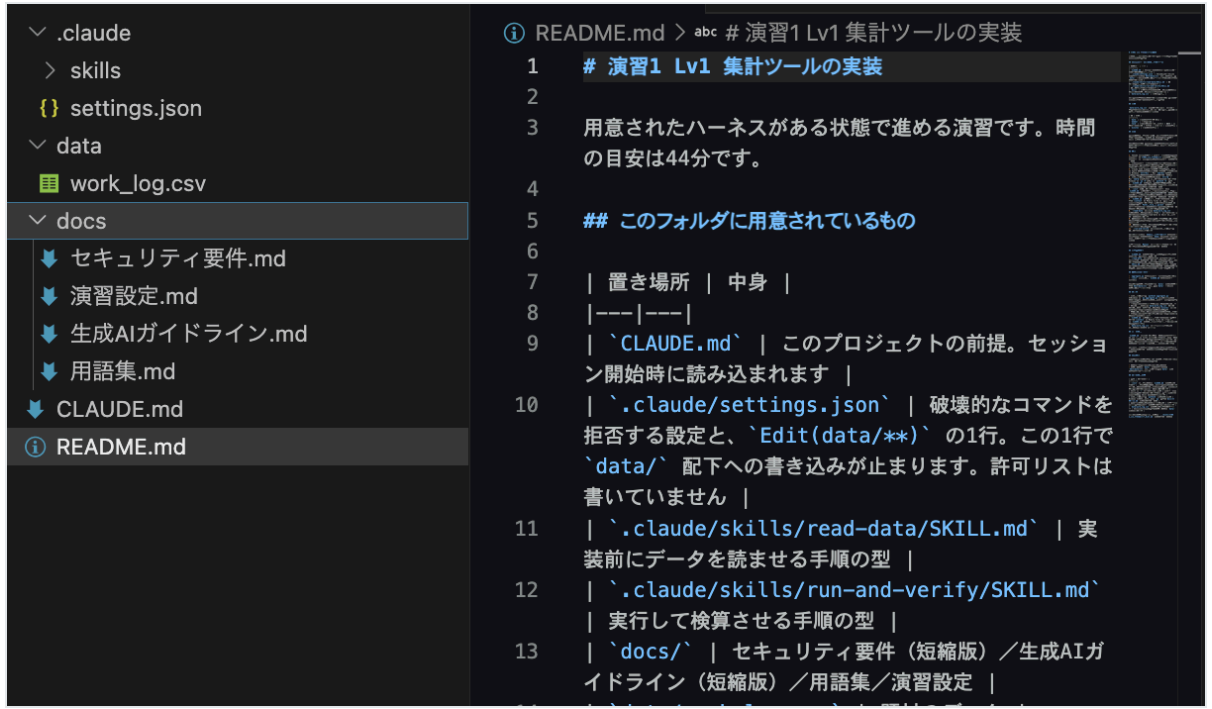
演習1 Lv1 集計ツールの実装

ハーネスが効いている状態で、指示から検算までの1周を通します

ワーク

演習1 のねらいと用意されたハーネス

ハーネスが効いている状態を通してから、演習2 へ進みます。



CLAUDE.md・.claude・docs・data が並んでいることを見ます。

画面: 演習1 のフォルダを展開した状態（実機）

- **1周を通す**
  指示、生成、差分の確認、
  実行、検算の順で進めます。
- **効いている状態を体験**
  CLAUDE.md と Skill が
  先に入っています。
- **演習2 と比べる基準**
  無い状態との差を、
  自分の手数で測ります。

使うハーネス

このフォルダには、6つの部品が先に置いてあります。

置き場所	中身
CLAUDE.md	プロジェクトの前提。開始時に読み込まれます
.claude/settings.json	破壊的なコマンドの拒否と Edit(data/**) の deny
.claude/skills/read-data/	実装前にデータを読ませる手順の型
.claude/skills/run-and-verify/	実行して検算させる手順の型
docs/	セキュリティ要件・生成AIガイドライン・用語集・演習設定
data/work_log.csv	題材データ。58行、minutes の総計は4875

コーディング規約は置いていません。演習2 で「無いものを持ってくる」体験のためです。
個人スコープの公式Skill 2本は、この演習では呼びません。

CLAUDE.md に書いてある前提

開いて読み、どの行が効くかを先に予想します。

```
- Python 3.10 以上。標準ライブラリだけで書く
- 関数には日本語の docstring を1行付ける
- CSV から読んだ数値は、計算する前に int または float へ変換する
- データファイルのパスは Path(__file__).parent を基準に解決する
- 既存の出力フォーマットを変えない
- 実装のあとは統合ターミナルで実行して確認する
```

> この6行が、今日の生成物の形を決めます。演習2 では、同じ前提が
> どこにも書かれていない状態から始めます。

/init と既存 CLAUDE.md の扱い

既存の行を削る提案は却下し、追記だけを承認します。

1. **/init を実行する**
   既存の CLAUDE.md があるので、
   追記の提案が差分で出ます。

2. **差分を1件ずつ読む**
   既存行を削る提案が混ざって
   いないかを見ます。

3. **迷ったら却下して言い直す**
   「既存の行は変えずに、不足して
   いる項目だけ追記してください」

参考の依頼文は hints/ にあります。
答えは配布物に書いていません。

差分の読み分け

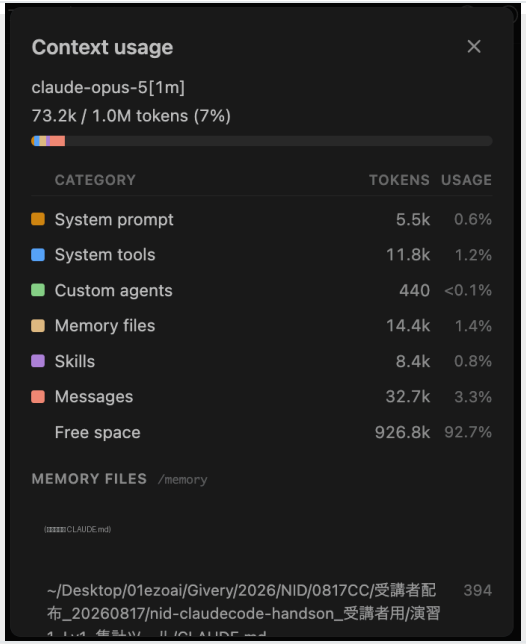
受け入れる提案
既存に無い項目を足すもの
実行や検算の手順を書き足すもの

却下する提案
既存の行を書き換えるもの
既存の行を削るもの

却下したときの言い直し
「既存の行は変えずに、不足分だけ追記してください」

CLAUDE.md の読み込み確認

確認の経路は2つあり、どちらか片方で足ります。



画面: /context の Memory files（実機）
出典 https://code.claude.com/docs/en/memory

経路1　/context の Memory files

一覧に2つの CLAUDE.md が
並んでいるかを見ます。

経路2　語を指定して聞き返す

下の文で聞き、書いた語が
返るかで判定します。

```
CLAUDE.md に書いてある
Path(__file__).parent という語を含めて、
このプロジェクトの前提を3つ挙げてください。
```

※ 拡張の /context で Memory files が開けることは実機で確認済みです。

使うコマンドと生成物

/init は必須です。ほかは必要な場面で呼びます。

コマンド	何をするか
/init	既存の CLAUDE.md への追記提案を差分で出す（必須）
/read-data	実装前に data/ を読ませ、列と行数と分布を報告させる
/run-and-verify	実行して、出力の数値を元データから数え直す
/context	読み込まれているファイルを確認する
/model	当日のモデル一覧を見る
/compact	会話を圧縮して、コンテキストの残量を空ける

生成物の名前と場所

aggregate.py（このフォルダの直下に新規作成）と、/init が追記した CLAUDE.md

進め方5段

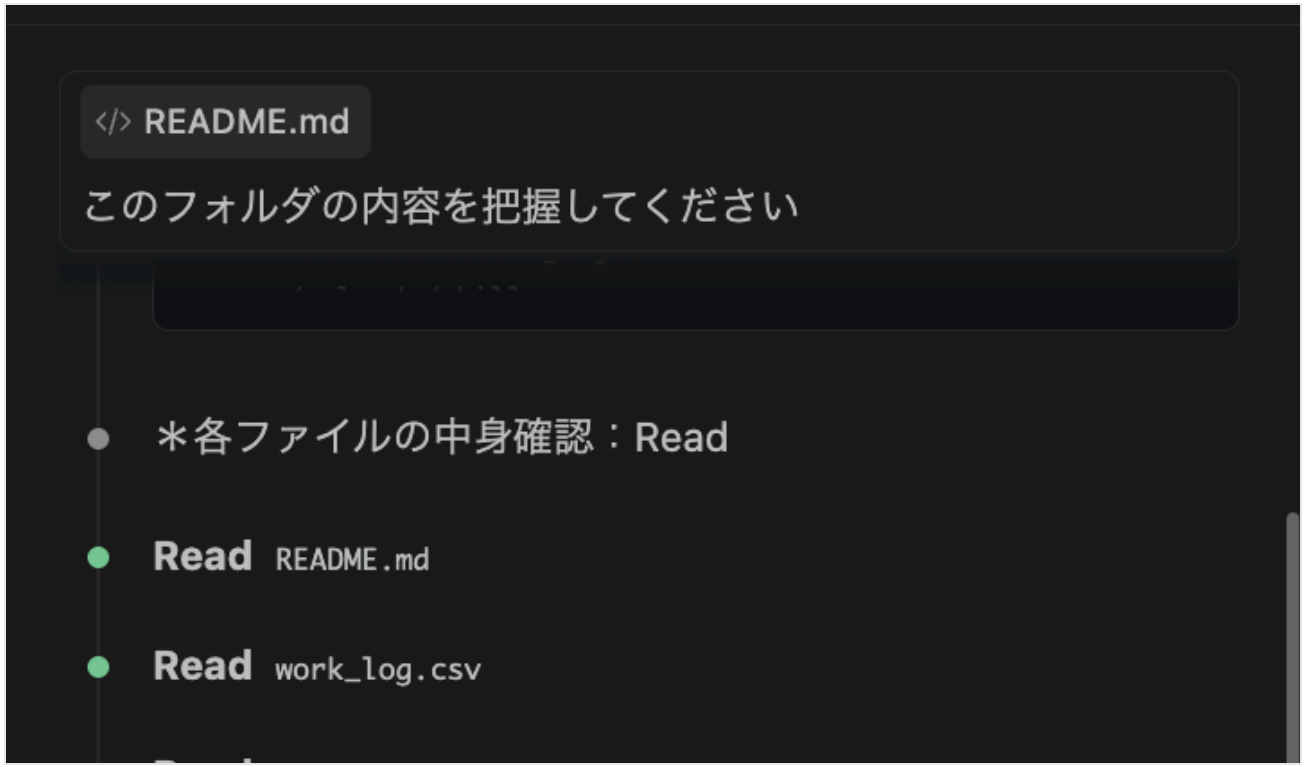
読ませて相談してから作らせ、最後に検算します。

1	**データ確認**	/read-data で列構成と行数を報告させます
2	**設計の相談**	読み込み・集計・表示の分け方と、集計結果を持つクラスを決めます
3	**実装と承認**	差分は1件ずつ読んで承認します。まとめて承認しません
4	**実行**	統合ターミナルで python3 aggregate.py（Windows は py aggregate.py）
5	**検算**	/run-and-verify で合計を数え直し、手で確かめた値と合わせます

実演

講師実演 /read-data

実装の前に、列構成と行数を報告させます。



画面: パネルがファイルを読んでいるところ（実機）

「/」のメニューに出ない場合は「read-data の Skill を使ってください」と自然文で呼びます。

題材データ work_log.csv

列は4つ、ヘッダーを除いて58行です。

```
date,member,task,minutes
2026-07-01,sato,設計レビュー,150
2026-07-01,takahashi,実装,45
2026-07-01,suzuki,ドキュメント作成,30
```

列の意味

date	作業日（YYYY-MM-DD）
member	メンバー名
task	作業の種別（6種）
minutes	作業時間（分）

検算に使う値

行数（ヘッダーを除く） 58
メンバー 5名、作業種別 6種
minutes の総計 4875
期間 2026-07-01 から 2026-07-20

統合ターミナルの作業ディレクトリは開いたフォルダなので、実行時のパス指定は要りません。

質問

自分で考える

手を止めて、この3問を自分の言葉で答えてみてください。

1. CLAUDE.md に書いてあったおかげで、自分が書かずに済んだ指示はどれでしたか。

2. /read-data を呼ばずに「CSV を集計するツールを作ってください」とだけ頼んだ場合、何が違ったと思いますか。時間があれば1回試して比べてください。

3. SKILL.md を2本とも開いて読み、自分の言葉で書ける範囲の内容かどうかを判断してください。書ける範囲であれば、自分の職場の手順も同じ形にできます。

実行と検算

エラーが出なかったことは、検算ではありません。

1. **/run-and-verify を呼ぶ**
   実行と検算の手順が
   型として入っています。

2. **統合ターミナルで実行する**
   Mac は python3 aggregate.py
   Windows は py aggregate.py

3. **数え直した値と合わせる**
   行数58 と minutes の総計4875 を
   別の経路で数え直します。

> 一致しないときは、直す前に
> どちらが誤りかを先に言葉にします。

実行と、突き合わせる2つの数

```
# 実行
python3 aggregate.py # Mac
py aggregate.py # Windows

# 別の経路で数え直す
行数 58
minutes の総計 4875

# エラーが出なかったことは検算ではありません
```

OK基準7項目

7つすべてを満たせたら、この演習は完了です。

	確認する項目
1	python3 aggregate.py（Windows は py aggregate.py）がエラーなく動く
2	メンバー別の合計稼働分数が、多い順に表示される
3	表示された合計が、数え直した値と一致する（行数58・総計4875）
4	読み込み・集計・表示が、別々の関数に分かれている
5	集計結果を保持するクラスがあり、メンバー別と作業種別の両方を取り出せる
6	CLAUDE.md の読み込みを確認できている（2経路のどちらか片方で足ります）
7	data/work_log.csv の中身が変わっていない（行数58・総計4875 のまま）

質問

追加と考察

1行を消して同じ依頼を出し、この回の差を記録します。

一時的に消す1行

CSV から読んだ数値は、計算する
前に int または float へ変換します。

この行が効いている前提で、
いまのコードが出ています。
消したあとに同じ依頼を出します。

消したあとに見るところ

数値を文字列のまま足していないか
合計がずれていないか
docstring や出力の書式は残るか

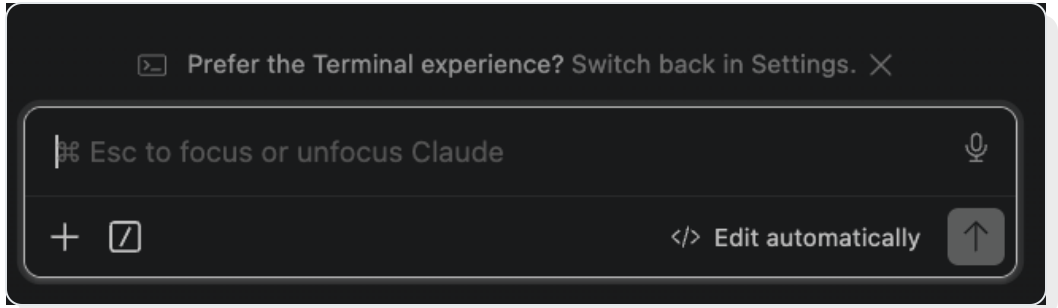
確認が済んだら、消した行を
元に戻します。

結果は「毎回こうなる」ではなく「この回はこうなった」と記録します。
1回の比較で分かるのは、そこまでです。

補足

つまずきどころ

詰まったら、この4つを先に見ます。



画面: Claude Code パネルの入力欄まわり（実機）

パネル下部の入力欄です。
Skill が一覧に出ないときも、
この欄に自然文で書いて呼びます。

症状	見るところ
/init の差分が既存行を書き換えている	却下して、追記だけを頼み直します
Skill が / のメニューに出ない	信頼を確認し、自然文で Skill 名を書きます
python3 が見つからない	Windows は py です。演習設定.md の表を見ます
パネルが開かない、拡張が反応しない	制限モードです。開き直して信頼を選びます

クイズ

理解度チェック

Q. 実装を頼む前に、最初にやることはどれですか。

A　思いついた列名で、いきなり実装を頼む

B　データを読ませて、列構成と件数を確認する

C　発展課題の CSV 書き出しから先に着手する

D　クラス名と関数名を全部先に決めきる

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はB。先にデータを読ませると、後戻りが減ります。

B **正解　データを読ませて、列構成と件数を確認する**

列名や集計軸のずれを、実装前に直せます。/read-data がこの型です。

A 思いついた列名で、いきなり実装を頼む

列名を勘で書くと実データと食い違い、やり直しが増えます。

C 発展課題の CSV 書き出しから先に着手する

発展は基本の集計ができてからです。順番が逆になります。

D クラス名と関数名を全部先に決めきる

先に固めすぎると、データを見たあとの見直しがしにくくなります。

実務でのポイント

読ませる、相談する、書かせるの順にすると、後戻りが小さくなります。

発展

発展課題

作業種別のランキングを CSV に書き出すサブコマンドを足します。

条件は2つ

既存のメンバー別の表示を変えない

書き出し先は data/ の外にする

止まったら想定どおりです

settings.json の deny にある
Edit(data/**) の1行で、data/ へ
書こうとすると止まります。

deny の1行で止まります

```
# .claude/settings.json
"deny": ["Edit(data/**)"]

# data/ へ書こうとすると承認を求められて止まる

# 書き出し先を data/ の外にすれば通る
./ranking.csv など
```

ワーク

M4 プロジェクトの CLAUDE.md に実行サマリー

同じ CLAUDE.md でも、層を分ければ矛盾しません。

ユーザー層

~/.claude/CLAUDE.md

午前の M2 で入れた1行が、いまも効いています。ツールを呼ぶ直前に1行を出す指示で、どのフォルダを開いても同じように効きます。

プロジェクト層

演習1_Lv1_集計ツール/CLAUDE.md

これから3行を足します。応答の末尾に、要約・読んだファイル・使ったハーネスを出す指示で、この演習フォルダでだけ効きます。

役割を分けておけば、両方が同時に効いても指示はぶつかりません

出力の書式を逐語で書くほど、出せているかを人が一目で判定しやすくなります。

追記する3行

「無い場合は なし」まで、逐語で書き切ります。

```
## 応答の最後に必ず3行を付ける

要約: この応答でしたことを1文で書く

読んだファイル: 読んだパスをカンマ区切り。
無い場合は なし

使ったハーネス: CLAUDE.md・Skill名・
Subagent名・Hook名をカンマ区切り。
無い場合は なし

# ラベルは同じ文字列のまま出す
```

ラベルを固定する理由

- **一目で判定できる**
  付いているかを人が数えられます
- **言い換えを防げる**
  同じ意味の別表現に散りません
- **後から機械にも渡せる**
  固定文字列なら検索で拾えます

ラベルの文言は変えません。

この3行は演習2 でも同じものを使います。写しで足ります。

動作確認の4手

追記したら、実際に依頼を1つ出して確かめます。

1. **追記する**
   演習1フォルダの CLAUDE.md の末尾に足します。
2. **依頼を1つ出す**
   演習1の発展課題を、いつもどおり頼みます。
3. **並び順を見る**
   3行が指定の順で付いているか見ます。
4. **中身を見る**
   実際に呼んだ Skill 名が入っているか。

3行の書式

```
要約: 1行で何をしたか
読んだファイル: パスをカンマ区切り
使ったハーネス: CLAUDE.md / Skill名 /
Subagent名 / Hook名

# どちらも無いときは
読んだファイル: なし
```

3行が出ないときは、CLAUDE.md を保存したか、開いているフォルダが演習1 かを確かめます。

CLAUDE.md が効く書き方

守れる粒度で書くほど、**指示は守られます**。

観点	書き方	効き方
分量	200行未満に収める	長いほど後半の行が効きにくくなります
構造	見出しと箇条書きで書く	どこに何が書いてあるか拾いやすくなります
粒度	検証できる言い方にする	「丁寧に」ではなく「実行して確認する」
一貫性	矛盾する指示を置かない	層ごとに役割を分けます

ラベルは固定文字列にする

「要約:」のように決めておくと、出せているかを人が一目で判定できます。

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

質問

追加と考察

ラベルを曖昧な言い方に変えると、出力は崩れます。

「使ったハーネス:」を「使ったもの:」に変えて出し直します

出力が崩れるかを見て、確認したら元のラベルに戻します。

崩れたなら、何が曖昧だったかを1行で書きます

語の抽象度か、対象の範囲か。どちらが効かなかった原因かを言葉にします。

発展課題 「毎回必ず」を機械で担保する方法を考えます

Stop フックで応答を検査する構成が候補です。実装と挙動は未検証なので本日は触れるだけにします。

変えた設定は必ず元に戻してから次へ進みます。

圧縮のあとも3行は続くか

プロジェクトの CLAUDE.md は圧縮を越えて読み直されます。

応答の末尾に付く3行（例）

```
# 依頼を1つ出したときの応答の末尾
要約: 作業種別のランキングを CSV に書き出す処理を追加
読んだファイル: aggregate.py, data/work_log.csv
使ったハーネス: CLAUDE.md, run-and-verify

# /compact を挟んで、もう1つ依頼を出したあと
Conversation was compacted to free up context.
要約: 書き出し先を data/ の外へ変更
読んだファイル: aggregate.py
使ったハーネス: CLAUDE.md
```

1. **/compact を1回実行する**
   会話が要約にたたまれます
2. **もう1つ依頼を出す**
   同じ会話のまま続けます
3. **末尾の3行を見る**
   同じ順で付いていれば OK

> 圧縮が走ると Conversation was compacted to free up context. が出ます。

OK基準3点

3つそろって、M4 は完了です。

確認する項目
1 応答の末尾に、要約 → 読んだファイル → 使ったハーネス の順で3行が付く
2 「使ったハーネス」に、その応答で実際に使ったものの名前が入っている
3 /compact のあとも、同じ3行が続いて付く

> **演習2 では同じ3行を自分で書きます**
>
> ゼロから作った CLAUDE.md に、いま決めた3行をそのまま写します。文言は変えません。

フォルダの開き直し②

次に開くのは 演習2_Lv2_不具合修正 です。

1. **ファイル > フォルダーを開く**
   配布フォルダの中の 演習2_Lv2_不具合修正 を選びます。

2. **信頼する**
   制限モードのままだと拡張は動きません。

3. **パネルを開く**
   Spark アイコンから開き、会話は新しく始まります。

演習2_Lv2_不具合修正 の中身

```
演習2_Lv2_不具合修正/
  monthly_report.py      直す対象
  data/                  入力データ
  README.md              演習の指示

ここに無いもの
  .claude/               設定なし
  CLAUDE.md              規約なし
  docs/                  参照先なし
```

エクスプローラで確認すること
.claude も CLAUDE.md も docs も無いこと。
無いのはプロジェクト側だけで、ユーザー設定は効いています。

06

演習2 Lv2 不具合修正

プロジェクト側のハーネスが無い状態から、詰まった箇所を材料にして組み立てます

ワーク

演習2 のねらい

無い状態から始めて、詰まった箇所を材料に
ハーネスを自分で組み立てます。

演習1 では、CLAUDE.md も Skill も設定も先に置いてありました。演習2 のフォルダにはそれがありません。何を毎回書かされるかを自分の手で確かめてから、足りないものを1つずつ足していきます。

無い状態を体験する
効いていた前提が消えます

詰まりを材料にする
書かされた前提をメモします

手数の差を測る
1から書くか持ってくるか

前半の到達点

症状は3件ありますが、前半は1件目までです。前半と後半のあいだに休憩と M5 が入ります。

効いているのはユーザー設定側

Lv2 で無いのは、プロジェクト側だけです。

効いている層と、無い層	
~/.claude/CLAUDE.md	ツールを呼ぶ直前の1行
~/.claude/settings.json	statusLine と deny
~/.claude/skills/	公式Skill 2本
演習2/.claude/	ありません
演習2/CLAUDE.md	ありません
演習2/docs/	ありません

1. **ユーザーの CLAUDE.md**
   `~/.claude/CLAUDE.md`
   ツールを呼ぶ直前の1行
2. **ユーザーの設定**
   `~/.claude/settings.json`
   statusLine と deny
3. **個人スコープの Skill**
   `~/.claude/skills/`
   午前にコピーした公式Skill 2本

> ここを混ぜて説明すると「設定が消えた」と誤解されます。消えていません。

症状

まず実行して、止まるところを自分の目で見ます。

```
# Mac の場合
$ python3 monthly_report.py 2026-07

# Windows の場合
> py monthly_report.py 2026-07

# 統合ターミナル（Windows Ctrl+@ / Mac Cmd+@）で実行します
```

原因は、この資料にも配布物にも書いていません

エラーの本文は自分の画面で読みます。どこで止まったか、何を言っているかを先に
読んでから、Claude に調べさせます。参考プロンプトは hints/ にあります。

質問

症状は3件

1件直すと、次の症状が出ます。

出る順番は人によって違います

順序は問いません。出た順に1件ずつ直します。隣と違っていても問題ありません。

3件のうち1件は、エラーを出しません

合計が静かにずれるだけです。実行が最後まで通っても、合計そのものを確かめます。

前半の到達点は1件目の修正までです

残りは休憩と M5 のあと、後半で直します。ここで全部やり切らなくて構いません。

自分で考える: 静かにずれる1件には、どうやって気づけますか。

前半の操作6手

実行 → 調査 → 1件だけ修正 → 再実行 の順で進めます。

1. **実行して症状を見る**
   エラーの本文をそのまま読みます。
2. **/init を実行する**
   何も無い状態から CLAUDE.md が作られます。
3. **原因を調査させる**
   この時点では修正させません。
4. **1件目を修正させる**
   差分を確認してから承認します。
5. **再実行する**
   次の症状が出ることを確認します。
6. **前提を1行メモにする**
   毎回書かされた前提を書き留めます。

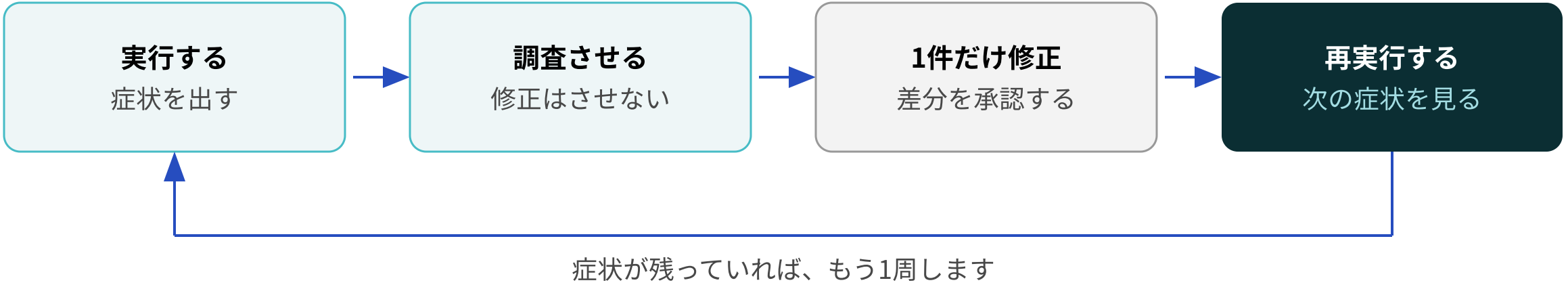
実行するコマンド

```
# Mac
python3 monthly_report.py 2026-07

# Windows
py monthly_report.py 2026-07

# 症状は、見えたまま1行で伝えます
```

答えと参考プロンプトは
hints/ にあります。

調査と修正を分ける

まとめて直させると、どの修正で消えたかが追えません。



まとめて頼まない

3件をまとめて直させると、どの修正でどの症状が消えたのかが追えなくなります。
1件ずつ直して再実行すると、原因と結果が1対1で残ります。

補足

前半のつまずきどころ

順番の差も長い CLAUDE.md も、想定どおりです。

起きること	どうするか
症状の出る順番が人によって違う	順序は問いません。出た順に1件ずつ直します
最後の1件に気づかない	実行が通っても、合計の妥当性まで確認します
/init が作った CLAUDE.md が長い	200行未満を目標に、その場で削ります
.claude 配下の編集で毎回確認が出る	保護パスの仕様です。中身を読んで承認します
調査に深入りして時間を使い切る	前半の到達点は1件目の修正までです

詰まったら黙って待たずに、Zoom のチャットへ1行入れてください。講師が画面を見に行きます。

作られた CLAUDE.md を削る

公式の指針は 200行未満。削る判断そのものが練習です。

生成された行を、どう削るか

/init が書きがちな見出し

プロジェクト概要 / ディレクトリ構成 / 主要ファイル
ビルドとテストのコマンド / コーディング規約

残す基準

毎回のやりとりで、実際に行動が変わるもの

削る基準

コードを読めば分かること。書いても行動が変わらないこと

削った行は手元に残します。何を前提として書くかを決める練習になります。

質問

自分で考える

前半で同じ前提を何回書いたかを数えます。

同じ前提を何回書きましたか

依頼のたびに書いた言葉を数えます。3回以上出たものが CLAUDE.md の候補です。

/init が作った CLAUDE.md のうち、実際に効いた行はどれですか

応答の中身に反映された行を挙げます。挙がらない行は次の判断材料になります。

効かなかった行は、何が曖昧でしたか

語の抽象度か、対象の範囲か。どちらが原因だったかを言葉にします。

ここで取った1行メモが、次の M5 で規約を書くときの材料になります。

ワーク

M5 のねらい

規約を直すと、機械の判定も**その場で変わります**。

人が読む説明と、機械が読む設定を1枚のファイルにまとめます。説明を直せば判定が変わり、判定を直せば説明とのずれに気づきます。M5 では、その1枚を置いてから、判定を受け持つ Hook を日本語の依頼文で作ります。

docs/コーディング規約.md
人が読む説明と
機械が読む設定

→

PostToolUse の Hook
編集した直後に
規約を読んで判定

→

会話に指摘が返る
違反の箇所と
直し方が最大3点

20分の枠です。規約は白紙から書きません。配布フォルダのひな型をコピーして、2項目だけ自分の言葉に直します。

規約ファイルのコピー

ひな型をコピーして、2項目だけ自分の言葉に直します。

1. **ひな型を開く**
   配布フォルダの docs_参照元/ にあります。
   演習フォルダの外なので、@ の候補には出ません。

2. **docs/コーディング規約.md として保存**
   エクスプローラや Finder でコピーして貼り付けます。
   Shift を押しながらパネルへドラッグして、保存を
   頼む方法もあります。

3. **2項目だけ書き換える**
   max_line_length の値と、人が読む説明のうち
   1行を、自分たちの言葉に直します。

4. **設定キーの名前は変えない**
   キー名は Hook が読みます。変えるのは値だけです。

docs_参照元/ のひな型

```
docs_参照元/
    コーディング規約_ひな型.md          これを使う
    CLAUDE_md_ひな型.md
    セキュリティ要件_ひな型.md
    生成AIガイドライン_ひな型.md
    用語集_ひな型.md
    設定断片/
    README.md
```

規約ファイルの形

同じ1枚に、人が読む説明と機械が読む設定を並べます。

```
# コーディング規約（Python）

## 人が読む説明
- 命名は意味を推測できる語にする
- コメントには「なぜ」を書く
- 既存の出力フォーマットを変えない

## 機械が読む設定
{
  "forbid_tabs": true,
  "max_line_length": 100,
  ...
  "forbid_print": false
}
```

読み方

- **上半分は人が見る項目**
  機械では測れないので
  レビューで見ます。
- **下半分は Hook が読む**
  この設定ブロックが
  判定の正本になります。
- **キー名は変えない**
  変えるのは値だけです。
  名前を変えると
  判定が動きません。

機械が判定する8項目

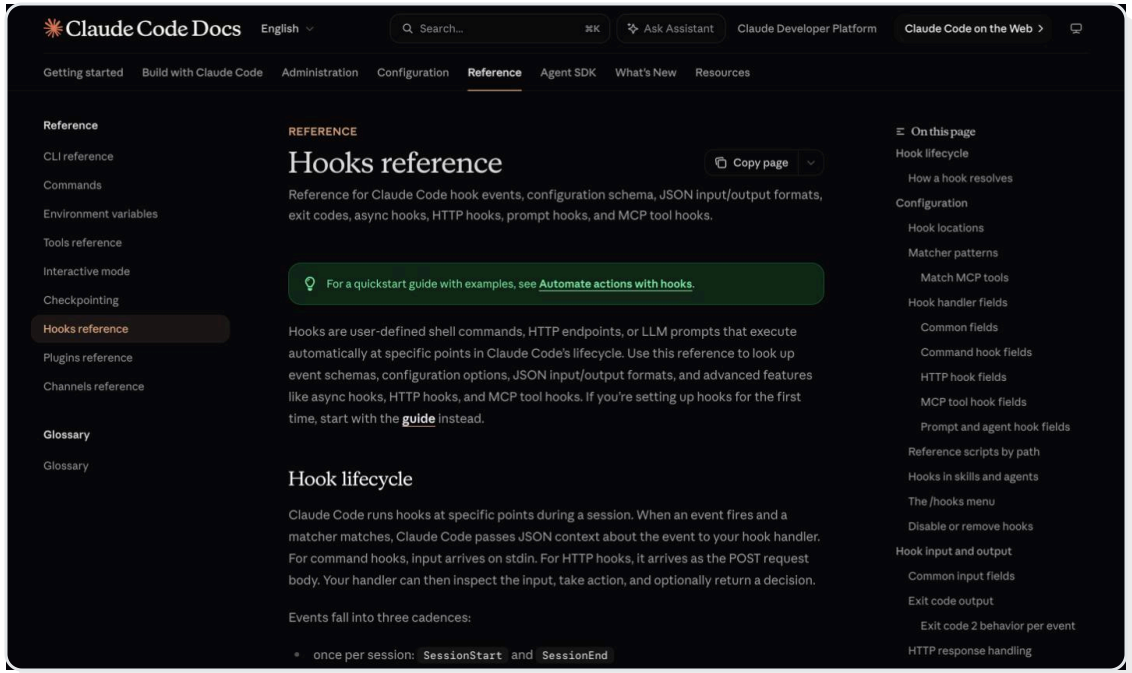
値は変えてかまいません。キー名は変えません。

設定キー	既定値	見るところ
forbid_tabs	true	タブと空白が混ざるとインデントが崩れます
max_line_length	100	ここを自分たちの値に直します
require_docstring	false	true にすると関数を書くたびに指摘が返ります
require_type_hints	false	同じ理由で、配布時は off にしてあります
forbid_bare_except	true	例外を飲み込むと不具合が正常終了になります
forbid_mutable_default	true	既定値の list は呼び出しをまたいで残ります
max_function_lines	50	1画面に収まらない関数は影響範囲が読めません
forbid_print	false	題材が標準出力にレポートを出すため許可します

docstring と型注釈の2項目は、発展課題で true に切り替えて指摘の増え方を見ます。

Hook を日本語で作らせる

依頼文に4つの条件を書き、差分を見てから承認します。



公式ドキュメント Hooks reference（発火の場所と入力の一覧）
出典: https://code.claude.com/docs/en/hooks

1. **発火のタイミング**
   PostToolUse、matcher は Write|Edit|MultiEdit
2. **判定の正本**
   docs/コーディング規約.md の機械が読む設定のブロック
3. **返し方**
   違反の箇所と直し方が会話に返る
4. **指摘の上限**
   1回の応答で最大3点まで

> 「同じファイルは1回だけ」は prompt 型では効きません。編集のたびに独立して起動するためです。

生成された settings.json

登録は1件だけです。差分を1行ずつ読んで承認します。

```
// .claude/settings.json
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit|MultiEdit",
        "hooks": [
          { "type": "prompt",
            "prompt": "規約に反していないか
            判定。最大3点まで",
            "timeout": 60 }
        ]
      }
  ] } }
```

読むところ

PostToolUse
編集した直後に走ります

matcher
この3つの編集で走ります

type: "prompt"
シェルを使わない判定です

壊れたら、このブロックだけ
消します。元の内容は
hints/ のヒントにあります。

Hook の型の選び方

本線はprompt 型です。command 型は演習3 で触ります。

型	判定のしかた	本日の扱い
prompt 推論的センサー	LLM がその場で読んで判定します。 シェルを使わないため、環境の違い （Git Bash の有無、py と python3） に左右されません。	M5 でこの型を作ります。 差分を見て承認します。
command 計算的センサー	スクリプトが機械的に判定します。 同じ入力なら同じ結果が返ります。 そのぶん、実装と実行環境の準備が 必要になります。	演習3 に配布済みのもの で体験します。

推論的センサーは日本語で書けます。計算的センサーは同じ入力に同じ結果を返します。
どちらを置くかは、判定したい中身で決めます。

指摘が会話に出るところ

わざと規約に反するコードを書かせ、承認した直後に**指摘が返る**ことを確かめます。

会話に返る指摘（形）

```
[コーディング規約] monthly_report.py
L12 タブでインデントされています
L28 可変オブジェクトが既定引数になっています
L45 1行が 100 文字を超えています

# 1回の応答で3点まで。まず1点目を直して、また走らせます
```

> 指摘は1回の応答で最大3点までに絞ってあります。まず1点目を直し、
> 承認してもう一度走らせます。出ないときの切り分けはこのあと扱います。

解説

編集の後か、前か

PostToolUse は編集の後に走るので、編集は止まりません。

今日作った Hook は、Claude がファイルを
編集した直後に走ります。書き換えはもう
済んでいて、指摘が会話に返るだけです。

編集そのものを止めたい場合は、編集の前に
走る PreToolUse になります。止める仕組みは、
指摘を返す仕組みとは別に考えます。

覚えておくこと

編集の直後に走る＝直った状態を確かめる係。
編集の前に走る＝入る前に止める係。

	計算的	推論的
事前 予防	先に縛る	文章で伝える
事後 センサー	機械で測る	読んで判定する

今日の Hook はどのマスに入るでしょうか。
自分の言葉で1行にしてみてください。

動かないときの切り分け

登録 → 信頼 → インタプリタの順に、上から見ます。

登録が見えるか

.claude/settings.json をエディタで開き、PostToolUse の登録が1件あるかを読みます。

信頼しているか

プロジェクトの Hook は、ワークスペースの信頼を受け入れたあとに有効になります。開き直して信頼を選び直します。

インタプリタ名

command 型にした場合は、Windows と Mac でコマンド名が違います。py と python3 を読み替えます。

> /hooks の一覧が拡張のパネルで開けるかは未確認です（未確認#2）。開けない場合は .claude/settings.json を開いて読み合わせます。

質問

M5 の OK基準

4つそろえば完了です。指摘が3点以内かどうかも見ます。

規約の設定キー（既定値）

```
forbid_tabs true
max_line_length 100
forbid_bare_except true
forbid_mutable_default true
max_function_lines 50
require_docstring / require_type_hints /
forbid_print false（演習で切り替える）
```

- [ ] PostToolUse の登録が1件ある
- [ ] わざと規約に反するコードで指摘が出る
- [ ] 修正を承認するともう一度走り、指摘が出なくなる
- [ ] 指摘が1回の応答で3点を超えない

追加と考察

forbid_print を true に変えて、print を含むコードで指摘が出るかを確かめ、false に戻します。

発展課題

ひな型を見ずに、規約を1から5項目書きます。説明と設定の両方をそろえます。

ワーク

演習2 後半のねらい

規約と Hook が効いた状態で、残りの症状を直します。前半との手数の差を、自分の手で測ります。

やること

残りの症状を直す

1件ずつ直して、そのつど実行して
症状が消えたかを見ます。

Hook の指摘を読む

指摘が出たら直してから次へ進みます。
指摘は最大3点までに絞ってあります。

測ること

前提を書いた回数

前半で毎回書いていた前提が、
後半で減ったかを見ます。

1回ぶんの記録として残す

1回の比較なので、一般化は
しません。1行だけ記録します。

後半の操作

4手で進めます。最後に合計を手で確かめます。

1. **残りの症状を修正させる**
   1件ずつ直し、そのつど実行して症状が消えたかを確かめます。
2. **Hook の指摘が出ることを見る**
   規約に反していれば、承認の直後に会話へ指摘が返ります。
3. **最後まで実行が通ることを見る**
   エラーで止まらず、レポートが出るところまで進めます。
4. **合計を突き合わせる**
   表示された合計と、データから手で確かめた値を比べます。

直ったかどうかの見分け

実行が通る、では足りません

見るもの

画面に表示された合計
データから手で数えた値

この2つが一致して、はじめて直ったと言えます

```
# Mac
python3 monthly_report.py 2026-07
# Windows
py monthly_report.py 2026-07
```

前半の手数と後半の手数

同じ前提を何回書いたかを、1行で記録します。

前半：何も無いところから

依頼のたびに前提を書き足しました。

```
この関数を直してください。
日付は文字列のまま比較しない、
直したら実行して確認する、
合計は手で数えた値と比べる…
```

書き忘れると前提が抜けます。
同じ説明を何度も書きました。

後半：規約と Hook がある

依頼は用件だけで済みました。

```
この関数を直してください。

前提は CLAUDE.md、規約は
docs/コーディング規約.md、
判定は Hook が受け持ちます。
```

言い忘れても前提が効きます。
規約違反は Hook が拾います。

1回の比較です。「毎回こうなる」とは書きません。差は振り返りシートに1行で残します。

演習2 の OK基準

次の7点がそろえば完了です。3つ目は、実行が通ることとは別に確かめます。

- [ ] 症状3件が出なくなり、最後までエラーなく実行できる
- [ ] メンバー別の時間と合計が、手で確かめた値と一致する
- [ ] 実行が通ったことだけで判定せず、合計の妥当性を確かめている
- [ ] CLAUDE.md の読み込みを確認できている
- [ ] docs/コーディング規約.md が機械が読む設定ブロックを含む
- [ ] .claude/settings.json に PostToolUse の登録が1件ある
- [ ] わざと規約に反するコードで指摘が出る

発展課題　4件目の不具合を自分で仕込み、症状だけを伝える文にして調査させます。
原因は書かずに渡すところが要点です。

解説

静かなずれ

最後まで通ったかではなく、合計そのものを見ます。

数え直しの手順

手で数える
対象月の行だけを、データから拾って合計する

突き合わせる
表示された合計と並べて、差を見る

ずれたら
原因ではなく症状を1行で伝えて、調べさせる

1. **実行が通る＝正しい、ではない**
   症状の1件はエラーを出さず、
   合計が静かにずれるだけです。
2. **別の手段で数え直す**
   データから手で数えた値と
   突き合わせます。
3. **ずれたら症状として書く**
   原因ではなく症状を1行で
   伝えて、調べさせます。

追加と考察
forbid_print の値を1つだけ書き換えて、指摘の出かたが変わることを見ます。
規約と Hook が同じ正本を見ている関係が目で確かめられます。確認したら戻します。

クイズ

理解度チェック

Q. 修正後、最後まで実行できました。完了と判断できるのはどれですか。

- **A** 実行がエラーなく最後まで終わったことを確かめたとき
- **B** Hook の指摘が会話に出なくなったことを確かめたとき
- **C** 修正の差分をすべて承認したことを確かめたとき
- **D** 表示された合計が手で確かめた値と一致したとき

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はD。合計を確かめるまでは完了ではありません。

D **正解　表示された合計が手で確かめた値と一致したとき**

症状の1件はエラーを出さず、合計が静かにずれるだけです。値を突き合わせて初めて分かります。

A 実行がエラーなく最後まで終わったことを確かめたとき

通ることは条件の1つですが、静かなずれはこれでは見つかりません。

B Hook の指摘が会話に出なくなったことを確かめたとき

指摘が消えたのは規約に沿った印で、集計の値の正しさとは別です。

C 修正の差分をすべて承認したことを確かめたとき

承認は変更を適用しただけで、結果の確認はこれからです。

実務のポイント

出てきた数字は、別の手段で1回数え直します。突き合わせるところまでが修正の作業です。

フォルダの開き直し③

M6 と M7 は 予備_チケット管理 で続けて進めます。

1. **フォルダを開く**
   ファイル > フォルダーを開く から
   予備_チケット管理 を選びます。

2. **信頼するを選ぶ**
   制限モードのままだと拡張が動きません。

3. **中身を確認する**
   .claude も CLAUDE.md も無いことを
   エクスプローラで確かめます。

予備_チケット管理 の中身

予備_チケット管理/
- ticket_app.py　　M6 と M7 の対象
- data/
- README.md

無いもの
- .claude/
- CLAUDE.md

ここが M7 の「入れる前」の状態です
何も入っていない状態を見てから始めると、あとで入れたものの効き目が分かります。

ワーク

ミニ演習 M6 のねらい

うまくいった手順を、次回も再現できる形に変えます。作って終わりにしません。

題材にする流れ

演習2 で回した「実行して落ちる → 原因を調べさせる → 1件ずつ直す → 再実行 → 合計を検算する」を、そのまま Skill にします。

この演習で確かめること

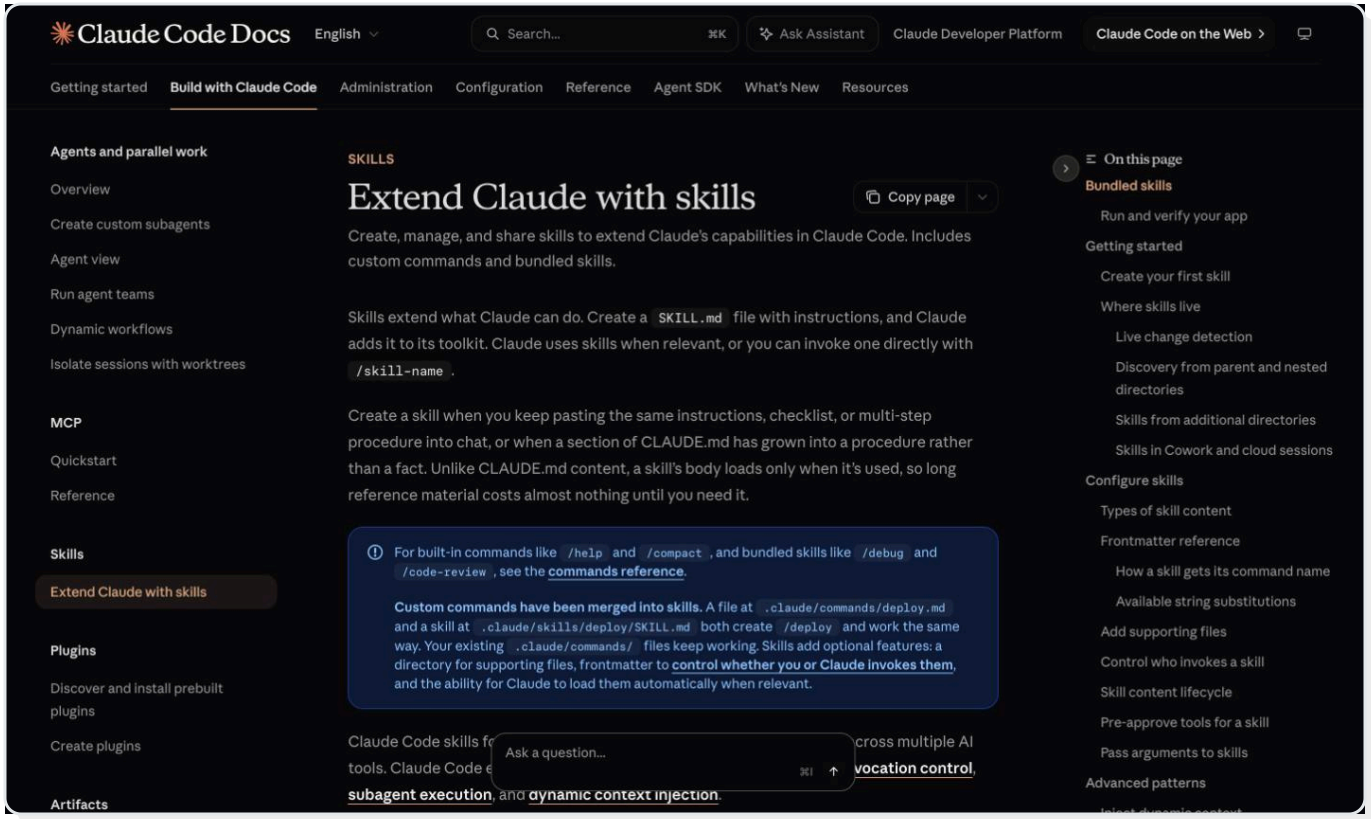
skill-creator に何を聞かれ、どう答えると SKILL.md が出来上がるか。
作った Skill を、いま開いているフォルダのファイルに1回使うところまで。
description に「いつ使うか」を書くと、呼ばれ方がどう変わるか。

公式ドキュメント Extend Claude with skills

カスタムコマンドは Skills に統合され、
どちらの置き方でも同じ /名前 を作ります。

.claude/commands/ に置いた既存のファイルは、そのまま動きます。

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



M6 の依頼文

先に /clear を実行してから skill-creator を呼びます。

skill-creator の SKILL.md は英語で約34KB あり、読み込むだけでコンテキストを大きく使います。
演習2 の会話を残したまま呼ぶと、途中で圧縮が走ります。

```
いまの会話でうまくいった手順を Skill にしたいです。
日本語で質問してください。
評価（eval）やテストケースの実行はしません。
  SKILL.md を書くところまでで止めてください。
対象は、Python スクリプトを実行して落ちたところから、
  原因を調べ、1件ずつ直し、再実行して合計を検算する流れです。
```

範囲を先に切る理由

SKILL.md には評価ループへ進むよう強く書かれています。1行目で範囲を切ると、
テストケースの作成とサブエージェントの並列起動に入らずに止まります。

聞かれる4点への答え方

聞かれるのは4点です。答え方を先に決めます。

聞かれること	答え方
何をできるようにするか	Python スクリプトが落ちたところから原因を調べて直し、 再実行して合計を検算するまでを1本の手順にする
いつ発火してほしいか	Python スクリプトを実行してエラーで止まったとき、と伝える
期待する出力形式	原因・直した箇所・再実行の結果・検算の結果の順に短くまとめる
テストケースを作るか	「作りません」と答える。ここで評価ループに入らずに止まる

出典: skill-creator の SKILL.md（意図の確認で聞かれる4点）

skill-creator の対話を進める

できた SKILL.md は、自分で読んで直します。

1 **4点に答える**

前のページで決めた答え方を、
そのまま返します。

2 **SKILL.md を読む**

生成されたファイルを開き、手順が
抜けていないか突き合わせます。

3 **description を直す**

いつ使うかを description に
書き切るのが原本の方針です。

skill-creator に聞かれる4点

1 何をする Skill か
2 いつ使うか
3 どんな手順で進むか
4 生成物は何か

いつ使うかを description に書き切ります

直す場所はここだけです

本文の手順は生成されたままで構いません。

作る場所と名前の決まり

Skill は個人スコープに作ります。演習ごとに フォルダを開き直すためです。

```
Windows      %USERPROFILE%\.claude\skills\<名前>\SKILL.md
Mac          ~/.claude/skills/<名前>/SKILL.md
```

名前の決まり

コマンド名はフォルダ名で決まります。frontmatter の name は表示名だけに使われます。
フォルダ名は英数字とハイフンだけにします。日本語や空白が入ると呼び出せません。
SKILL.md はフォルダの直下に置きます。1段深いところに置くと読まれません。

出典: Skills https://code.claude.com/docs/en/skills

補足

研修で使わない部分

使わない部分は依頼文の1行目で切ります。

同梱されているもの	使わない理由
run_eval.py / run_loop.py / improve_description.py	claude の CLI を subprocess で起動します。本研修は CLI を入れないため動きません。
quick_validate.py / package_skill.py	PyYAML が必要です。追加インストールができない環境では読み込みで止まります。
評価ループ（eval）	サブエージェントを並列で起動します。30名が同時に回すと時間もトークンも読めません。

使う範囲は、意図の確認から SKILL.md を書くところまでです。ここで止めます。

作った Skill を1回使う

対象は、いま開いている ticket_app.py です。

呼び方は2通り

```
# 1 スラッシュで呼ぶ
/フォルダ名

# 2 自然文で呼ぶ
ticket_app.py に、さっき作った手順を使ってください

# 再現するか見る順番
実行 → 症状の説明 → 修正 → 再実行 → 検算
```

1. **同じ手順が出るか**
   実行 → 症状の説明 → 修正 → 再実行 → 検算の順に進むか
2. **呼び方は2通り**
   /フォルダ名 でも、自然文で Skill 名を伝えても呼べます
3. **直したくなったら**
   SKILL.md の書き換えは同じセッションで反映されます

演習2 で踏んだ手順が再現されるかを見ます。順番が変わっていたら SKILL.md の本文を直します。

M6 の OK基準

3つそろったら、できたと判断します。

見るところ	OK と言える状態
置き場所	個人スコープの skills フォルダに SKILL.md があり、 frontmatter に name と description がある
再現するか	ticket_app.py に対して呼ぶと、演習2 で踏んだ手順が 同じ順番で出てくる
呼び出し	/フォルダ名 でも、自然文で Skill 名を伝えても呼べる

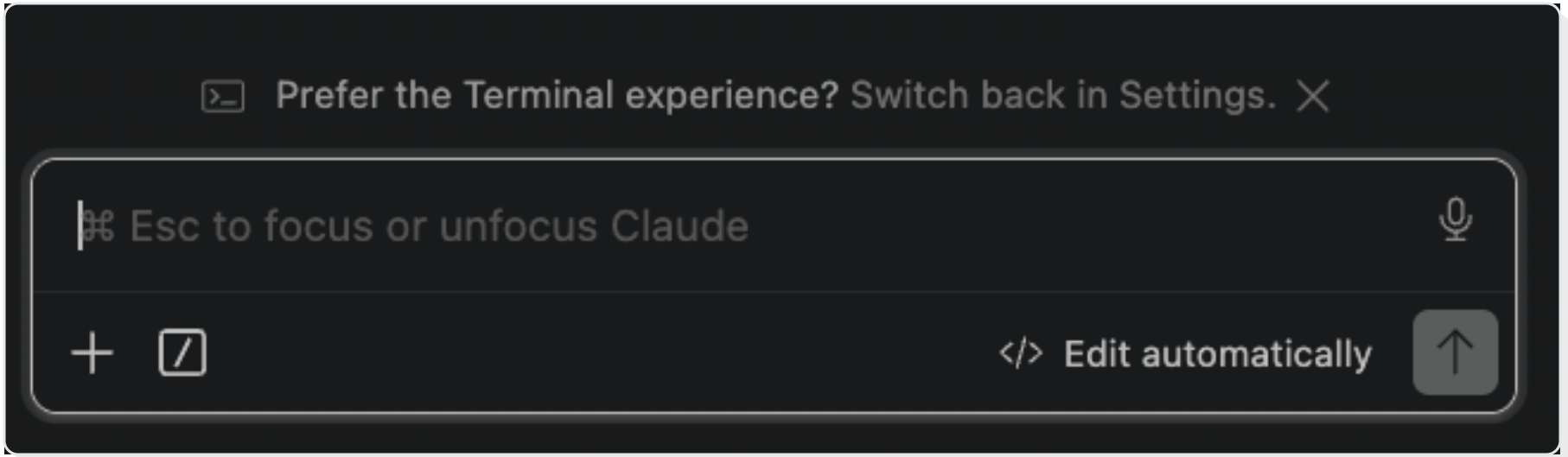
追加と考察

description から「いつ使うか」を消して呼び直し、自動で選ばれにくくなるかを見ます。
確認できたら元に戻します。

コマンドメニューに出るか

自作した Skill が / のメニューに並ぶかは環境によって違います（未確認）。

出ない場合は、自然文で Skill 名を指定して呼びます。Skill は description から選ばれる経路があるため、メニューに出なくても動きます。当日リハで実機を確認します。



画面: 実機のパネル入力欄。この欄で / を押すとコマンドメニューが開きます。

ワーク

ミニ演習 M7 のねらい

何を自動化するかを人が決める前に、公式Skill に棚卸しさせます。提案と導入は別です。

先に棚卸しさせる

このフォルダに何を入れられるかを Skill に並べさせます。決めるのはそのあとです。

提案が出ることと、入ることは別

提案を出す Skill はファイルを作りません。導入は別の依頼として投げます。

効いているかを測る

入れたものが効いているかを、自分で決めた3項目で導入前と導入後に分けて記録します。

入れる前の状態

予備_チケット管理 には .claude も CLAUDE.md も ありません。ここから始めます。

M6 のあとに /clear を実行してから呼びます。会話が残っていると、提案が M6 の話に引きずられます。

入れる前と、このあと	
いま入っているもの	ticket_app.py と data/ だけ
.claude/	ありません
CLAUDE.md	ありません
このあと増えるもの	settings.json と CLAUDE.md
比べ方	同じ依頼を、前と後で1回ずつ

この画面を、導入したあとの画面と見比べます。

提案の依頼文

全員が同じ依頼文を使い、条件を3つ入れます。

```
このプロジェクトに入れるべき Claude Code の自動化を
  提案してください。条件は3つです。

1. Web検索は使わず、同梱の references/ と
    このフォルダの中身だけで判断してください。

2. Hooks・サブエージェント・Skill を各1〜2件に絞り、
    なぜ必要かと、設置するファイルのパスを書いてください。

3. この時点ではファイルを作らないでください。
```

条件を入れる理由

SKILL.md は Web検索を薦めます。社内網で止まるので1行目で縛ります。
件数を絞らないと出力が長くなり、全員で見比べる材料になりません。

提案を出させる

許可の確認が続きます。Manual のまま1つずつ承認します。

1. **Skill を呼ぶ**

   claude-automation-recommender を呼び、このフォルダを対象にします。

2. **承認の中身を読む**

   ls や cat を Bash で叩くため、確認が何度も出ます。

3. **事前承認は効かない**

   frontmatter の tools は公式の項目ではないためです。

提案に並ぶもの

CLAUDE.md に書く規約
settings.json の permissions
Hooks の登録
Skill の候補

それぞれ1〜2件と、なぜ要るかの理由が付きます

読み取りしかしません

承認を読む練習として扱います。

ファイルは1つも増えません

出力を見たら、エクスプローラで増えていない ことを自分の目で確かめます。

SKILL.md の冒頭にある記述

```
This skill is read-only. It analyzes the codebase and outputs
recommendations. It does NOT create or modify any files.
```

2段構えで進みます

提案を出す依頼と、導入する依頼を分けます。頼み方を変えてもファイルは作られません。
導入は次のページで、別の依頼として投げます。
提案の中身は人によって違うので、入れるものは講師が3本に決めます。

導入する3本と検証項目

入れるのは3本です。確かめ方も先に決めます。

入れるもの	確かめること
permissions.deny に Edit(data/) を1行足す**	data/ のファイルを書き換えてくださいと頼み、拒否されるかを見る
PostToolUse に type: "prompt" のフックを1本入れる	規約に反する編集をしたときに、指摘が会話に返るかを見る
CLAUDE.md を1枚置く 型変換・パス解決・実行確認	CSV から読んだ数値の型変換と、実装後の実行確認を、言わなくても守るかを見る

> 提案は人によって違います。入れるものは講師が指定し、全員を同じ3本にそろえます。

導入は別の依頼として投げる

差分を1行ずつ読んでから承認します。

settings.json に増える行（例）

```
{
"permissions": {
"deny": ["Edit(data/**)"]
},
"hooks": { "PostToolUse": [ ... ] }
}

# .claude 配下は保護パス。モードに関わらず確認が出ます
```

1 差分で読む

settings.json のどの行が
増えたのかを見ます

2 毎回確認が入る

.claude 配下は保護パスで、
モードに関わらず確認されます

3 壊れたら戻す

JSON が壊れると設定が全部
効きません。該当行を消します

依頼は「提案のうち3つを実際に入れてください。差分を見せてから適用してください」の形にします。

Edit(data/**) を使う理由

パス付きのルールは Edit と Read しか参照されません。

```
{
  "permissions": {
    "deny": ["Bash(rm -rf:*)", "Edit(data/**)"]
  }
}
```

止められない範囲

deny が効くのは Claude のファイル
ツールと、ファイルを扱うと認識された
Bash コマンドです。Python スクリプトが
自分で開いて書き込む場合は止まりません。

効いているかの確かめ方

data/tickets.csv の1行目を書き換えて
くださいと頼み、拒否されることを
確認します。ファイルの中身も
変わっていないかを見ます。

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

比べる3項目

比べる項目は、測る前に3つに固定します。

見るもの	観測のしかた	導入前	導入後
data/ への書き込み	data/tickets.csv が変わったか	記録欄	記録欄
規約違反の編集	指摘が会話に返り、その指摘どおりに直したか	記録欄	記録欄
型変換と実行確認	int や float への変換が入り、実行して確かめたか	記録欄	記録欄

項目を先に固定しておくと、あとから都合のよい結果だけを拾う読み方になりません。

効果検証のしかた

同じ依頼・同じ初期ファイルで、1回ずつ比べます。

1. **戻せるようにする**
   対象ファイルをコピーしておくか、
   Rewind code to here で巻き戻します。

2. **同じ依頼を投げる**
   文面を変えると比較になりません。
   導入前に投げた文をそのまま使います。

3. **その場で書く**
   前ページの3項目に、思い出しでなく
   見たまま記入します。

比べるときに揃えるもの

同じにするもの

依頼の文面
初期のファイル

その場で控えるもの

やり直しの回数
承認を押した回数
実行結果の数字

> **見るのは会話とファイルの両方**
> 実行結果の数字も一緒に控えます。

質問

記録のしかたと考えること

1回の比較なので、この回はこうなりましたと書きます。毎回こうなるとは書きません。

自分で考える

提案のうち、この題材では効かないものはどれで、なぜ効かないと判断しましたか。
自分のプロジェクトなら、最初に入れる1本はどれですか。

発展課題

採用しなかった提案（MCP サーバー、プラグイン）を持ち帰り候補に整理します。
読むのは references/mcp-servers.md と plugins-reference.md の2本です。
この場では導入しません。持ち帰って社内の申請手順と突き合わせます。

補足

OK基準と、進まないときの切り替え

3点そろえば OK です。2回失敗したら切り替えます。

OK基準	
提案の時点	ファイルが1つも増えていない
導入したあと	3本が入った状態で、検証3項目の結果を記録できている
データ	data/tickets.csv が1バイトも変わっていない

進まないときの切り替え	
1回目のエラー	そのまま承認して様子を見る
同じ人が2回失敗	「フォルダの中身は自分で伝えます」と宣言し、@ でフォルダを参照する
それでも進まない	講師画面の提案結果から、導入の手順へ合流する

演習3フォルダの開き直し

4回目です。演習3_Lv3_ログ解析 を開いて信頼します。

1. **フォルダを開く**
   ファイル > フォルダーを開く を選び、
   演習3_Lv3_ログ解析 を開きます。

2. **信頼するを選ぶ**
   制限モードのままだと拡張は動きません。

3. **.claude/ を展開する**
   agents・rules・scripts・skills・
   settings.json の5つが揃っています。

> hooks/ というディレクトリはありません。フックは
> settings.json に登録し、本体は
> .claude/scripts/check_conventions.py にあります。

演習3_Lv3_ログ解析 の中身

```
演習3_Lv3_ログ解析/
  .claude/
    agents 2 / rules 2 / scripts 1 / skills 3
    settings.json
  CLAUDE.md                          参照表
  docs/                              5種
  spec.md                            7要件
  data/                              解析対象のログ
  README.md
```

ここまでの3フォルダと、見た目から違います。

07

演習3 Lv3 ログ解析パイプライン

予防と強制と分業が同時に効く環境で、仕様書から実装を作り切ります

ワーク

演習3のねらい

指示の量ではなく、確認の質が変わることを見ます。

自分が書いた指示の文字数を数える

演習1・演習2 と比べて、どれだけ変わったかを見ます。依頼文をそのままコピーして
文字数を数えるだけで足ります。

減った分を誰が書いていたかを言う

CLAUDE.md・rules・docs・Skill・Hook・Subagent のどれが肩代わりしたかを、
最後の振り返りで1つ挙げます。

確かめる対象が移る

動いたかどうかではなく、spec.md の要件と合っているかを見ます。この演習では
要件との突き合わせまでが1回の作業です。

揃っているハーネスの現物

演習3のフォルダには、予防・強制・分業の材料が入っています。

3段階で、揃うものが増えます

Lv1 演習1_Lv1_集計ツール
CLAUDE.md / Skill 2本 / docs 4種

Lv2 演習2_Lv2_不具合修正
プロジェクト側は何もありません

Lv3 演習3_Lv3_ログ解析
rules 2 / skills 3 / agents 2 / hook 1 / docs 5

使う道具が増えるほど、決めておく手間も増えます

1. CLAUDE.md　いつ何を読むかの参照表
2. .claude/rules/　2本のルール
3. .claude/skills/　3本の手順
4. .claude/agents/　2本の担当
5. .claude/scripts/check_conventions.py
6. .claude/settings.json　フックと deny
7. docs/　5種の参照ドキュメント
8. spec.md　7要件の仕様書
9. data/app_access.log　解析対象のログ

CLAUDE.md は docs を @ で取り込まず、
いつ読むかを書いた参照表を置いています。

同時に効く3つ

予防・強制・分業が同時に効く状態を体験します。

予防

先に読ませて外れを減らす

- CLAUDE.md
- .claude/rules/ 2本
- docs/ 5種
- Skill 3本

使う前から文脈を食わせません。

強制

外れたら止める・指摘する

- PostToolUse のフック1本
- permissions.deny

Claude が編集した直後に
走ります。保存では走りません。

分業

別の文脈に任せる

- code-reviewer
- verify-data

本会話に返るのは要約だけ。
auto memory は渡りません。

ハーネスの一覧と役割

この演習で効いているものを、置き場所ごとに並べます。

置き場所	中身	役割
`.claude/rules/`	python.md（**/*.py に限定）と data.md（data/** ほかに限定）	ファイル種別ごとの規約
`.claude/skills/`	spec-to-checklist / log-sample / verify-report の3本	手順を / で呼べる形に
`.claude/agents/`	code-reviewer と verify-data の2本 どちらも tools は Read, Grep, Glob	別の文脈で下調べ
`settings.json`	PostToolUse に command 型のフック1本 check_conventions.py を呼ぶ	編集の直後に規約を判定
`settings.json`	permissions.deny に Edit(data/**) と 破壊的なコマンド	触らせない場所を止める
`CLAUDE.md`	docs は @ で取り込まず、いつ読むかを 書いた参照表を置いている	使う前から食わせない

paths 付きのルールは /compact を越えると、該当ファイルが再度読まれるまで失われます。

/init を実行する

CLAUDE.md と docs が揃った状態で、何が起きるかを見ます。

1. **/init を実行する**
   パネルで / を押し、init を選びます。

2. **既存行を削る提案は却下する**
   演習1・演習2 と同じ判断です。足す提案だけ受け入れます。

3. **読み込みを確認する**
   /context の Memory files を見るか、rules に書いた語を含む応答が返るかで判定します。

/context の Memory files

```
Memory files
~/.claude/CLAUDE.md user
演習3_Lv3_ログ解析/CLAUDE.md project
.claude/rules/python.md rules
.claude/rules/data.md rules

# rules に書いた語が応答に出るかでも判定できます
```

すでに CLAUDE.md があるので、提案は追記の差分として出ます。

spec.md の7要件

解析ツール analyze_log.py に求める出力は、仕様書に7つ並んでいます。

要件	出力する内容	区分
1	総リクエスト件数	必修
2	ステータスコード別の件数（コード昇順）	必修
3	エラー総数とその割合（ステータス400以上）	必修
4	リクエストパス別の件数（多い順）	必修
5	応答時間の平均と最大（ミリ秒）	必修
6	応答時間が遅い上位5件（日時・パス・応答時間）	余力
7	**形式が崩れた行はスキップし、件数を報告する**	必修

要件7を必修にしているのは、壊れた行で処理が止まると要件1から5も出力されないためです。
時間内に終わらない場合は要件6を落とします。段階線は docs/演習設定.md と README にあります。

補足

つまずきどころ

詰まったら、原因の当たりをここから探します。

command 型のフックが動かない

Windows のインタプリタ解決です。
settings.json の command を1語だけ替えます。

指摘が続いて進まない

指摘は1回の実行で最大3点までです。時間内に
直せない項目は規約側を false に落とします。

サブエージェントが呼べない

.claude/agents/ はセッション開始時に置きます。
出ないときはコマンドパレットの
Developer: Reload Window を実行します。

コンテキストが埋まる

/compact に観点を渡します。何を残したいかを
書くと、残るものが変わります。

計画の承認で自動承認を選んだ

もう一度 Plan に戻し、計画から出し直します。
承認は Yes, manually approve edits です。

自作Skill が / メニューに出ない

M6 で作ったものが出ないことがあります。
自然文で名前を書いて呼びます。

Skill で仕様を分解する

/spec-to-checklist で7要件をチェックリストに落とします。

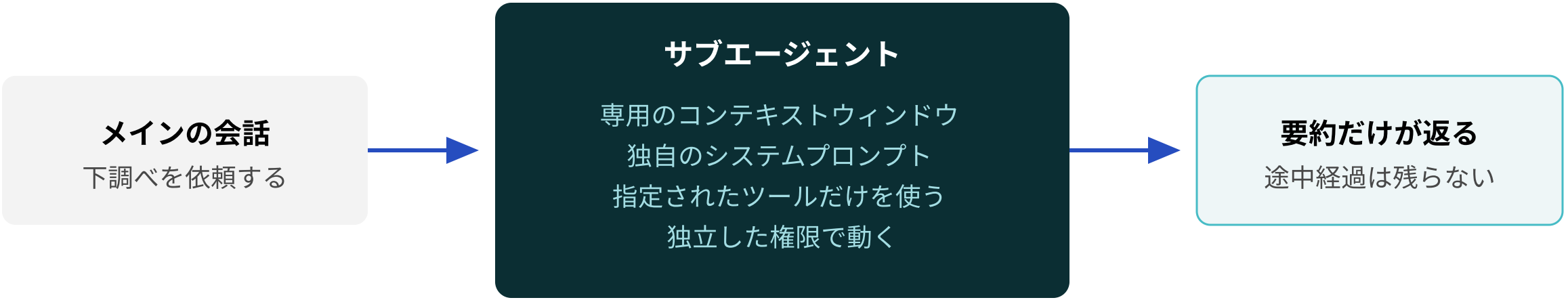
/spec-to-checklist が出すもの

```
# checklist.md
- [ ] R1 ...
- [ ] R2 ...
spec.md の7要件を、1件ずつ項目に落とします

# あわせて /log-sample を呼び、
# 正常な行と壊れた行の実物を1件ずつ見ます
```

生成物は checklist.md です。要件ごとにチェック項目が並びます。
あわせて /log-sample を呼び、正常な行と壊れた行の実物を見ます。

サブエージェントの文脈分離

サブエージェントは別の文脈で動き、返るのは要約だけです。



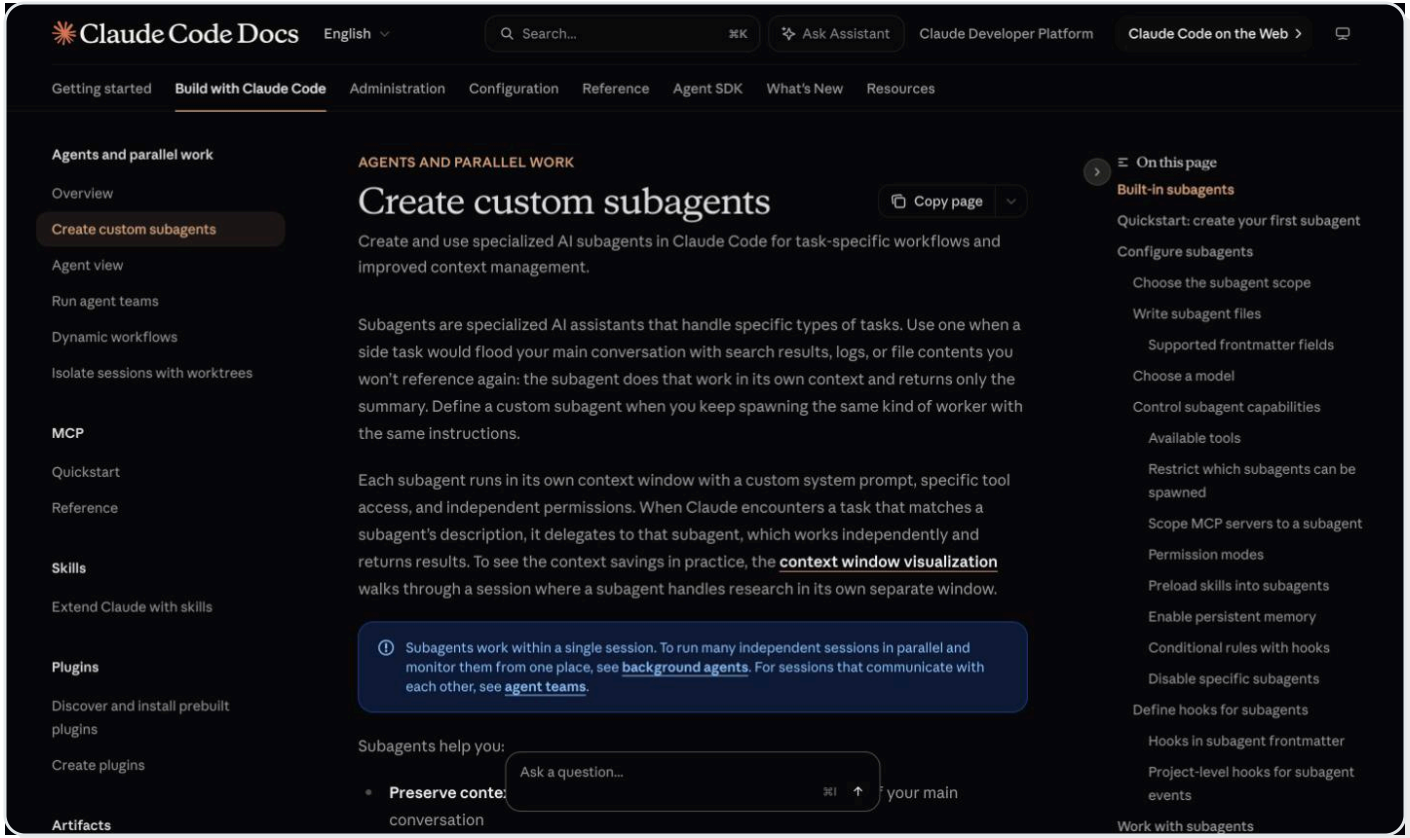
メインの会話の auto memory は渡りません

ログの走査や件数の数え直しを任せると、本会話のコンテキストは埋まりません。

出典: サブエージェント https://code.claude.com/docs/en/sub-agents

サブエージェントの公式ドキュメント

tools は使えるツールを絞る指定で、外しても起動できます。



画面: 公式ドキュメント Create custom subagents
出典: https://code.claude.com/docs/en/sub-agents

verify-data に下調べを任せる

下調べと数え直しを任せ、本会話には**要約だけ**を返させます。

1. **verify-data を呼ぶ**
   data/app_access.log の件数と形式の
   下調べを任せます。

2. **返ってきた要約を読む**
   ログの中身そのものは本会話に載りません。

3. **Context usage を見る**
   下調べの前後でどれだけ動いたかを控えます。

本会話へ返るのは要約だけ

```
# verify-data が返すもの
総行数  数えた結果
壊れた行の数  内訳つき
期間  最初と最後

# ログ本体は本会話に載りません
```

自分で考える
同じ下調べを本会話でやっていたら、
Context usage はどう動いたと思いますか。

ログそのものは本会話に載りません。
返ってくるのは要約だけです。

Plan モードで計画に手を入れる

計画は Markdown で開き、インラインコメントで直せます。

計画にコメントを付ける

```
# 計画が Markdown で開いたら
## 手順3 壊れた行の扱い
壊れた行はスキップする
← この行にコメントを付ける
「捨てずに、件数として数えてください」

# 直してから承認します
```

1. **モードインジケータをクリック**
   入力欄の下にあります。
2. **Plan を選ぶ**
   編集の前に計画を出す動きに変わります。
3. **計画が Markdown で開く**
   拡張だけの挙動です。
4. **1箇所だけ直して出し直す**
   直したい行にコメントを付けます。

モードの切り替えは、パネルのモードインジケータで行います。
ラベルは Manual / Plan / Edit automatically です。

クイズ

理解度チェック

Q. 演習3で「できあがった」と判断する正解条件はどれですか。

- **A** 「完成しました」という応答が返ったこと
- **B** 規約チェックのフックが何も指摘しなくなったこと
- **C** spec.md に書かれた要件を満たしていること
- **D** code-reviewer が指摘を出さなくなったこと

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はC。spec.md に書かれた要件が正解条件です。

C　正解　spec.md に書かれた要件を満たしていること

/verify-report で要件ごとに突き合わせ、結果を verify_result.md に残します。

A　「完成しました」という応答が返ったこと

作業の報告であって、要件を満たした証拠ではありません。

B　規約チェックのフックが何も指摘しなくなったこと

フックが見るのは書き方の規約で、要件そのものは見ていません。

D　code-reviewer が指摘を出さなくなったこと

レビューは書き方の指摘です。仕様と合っているかは別に確かめます。

実務でのポイント

正解条件を先に文書にしておくと、完了の判断が読み比べの作業になります。

計画の承認

承認はYes, manually approve edits を選びます。

承認で出てくる選択肢

Yes, manually approve edits
これを選びます。差分を1件ずつ見ます

Yes, auto-accept edits
No, keep planning

自動承認だと、差分を読む練習になりません

自動承認を選ぶと、差分を1件ずつ見る練習になりません。
権限モードのラベルは Manual / Plan / Edit automatically の3つです。

演習3のOK基準

9項目すべてを確認できたら、演習3は完了です。

区分	確認すること
出力	要件1から5と要件7が出力に現れる（余力は要件6だけ）
実行	Mac は python3 analyze_log.py、Windows は py analyze_log.py が動く
実行	別のフォルダから実行しても同じ結果になる
数値	総件数と、ステータス別件数の合計が一致する
数値	エラー（400以上）の割合が手計算と一致する
出力	壊れた行を数件混ぜても落ちず、スキップ件数が報告される
データ	data/app_access.log が1バイトも変わっていない
ハーネス	規約チェックのフックが、最終状態で何も指摘しない
ハーネス	code-reviewer の指摘1件以上に対応した記録がある

生成物　analyze_log.py　checklist.md　verify_result.md

質問

追加と考察・発展課題

手が空いたら、次の順で試します。

追加　tools から Read を外して呼ぶ

.claude/agents/code-reviewer.md の tools から Read を外して呼び、
ファイルが読めなくなることを確認して元に戻します。起動そのものは失敗しません。

考察　Hook と CLAUDE.md の線引き

機械で止めるべきものと、文章で伝えるべきものの線をどこに引きますか。
自分のプロジェクトに当てはめて、1つずつ挙げてみてください。

発展課題　ハーネスを別の題材へ移植する

予備_チケット管理/ticket_app.py にこの演習のハーネスを移植し、機能を1つ足します。
M7 で入れたものと重ならない部分（rules と Skill）を足すと、差が見えます。

突き合わせと最後のレビュー

実行して /verify-report で要件ごとに突き合わせます。

verify_result.md の形

```
| 要件 | 実装 | 確認方法 | 判定 |
|------|------|----------|------|
| R1 | ... | ... | OK |
| R2 | ... | ... | 未 |

# 未 が残っている間は、次へ進みません
```

1. **規約チェックの指摘に対応する**
   編集した直後にフックが出します。
2. **統合ターミナルで実行する**
   Mac は python3 analyze_log.py、
   Windows は py analyze_log.py です。
3. **/verify-report で突き合わせる**
   生成物は verify_result.md です。
4. **code-reviewer にレビューさせる**
   指摘を1つだけ直します。

規約チェックのフックが何も指摘しない状態にしてから、最後のレビューへ進みます。

今日測ったもの

ハーネスが有る状態と無い状態の差は、体感ではなく
記録した数で振り返ります。

演習1 Lv1	**書かずに済んだ指示** CLAUDE.md と Skill が置いてあったので、毎回書かずに済んだ前提を数えます。
演習2 Lv2	**毎回書いた前提の回数** プロジェクト側のハーネスが無い状態で、同じ前提を何回打ち直したかを数えます。
演習3 Lv3	**減った指示の文字数** 仕様とルールが読まれる状態で、依頼1回の文字数がどれだけ減ったかを見ます。

差の大きさより、どこに効いたかを言葉にできるほうが役に立ちます。
振り返りシートに、この3つの数と、いちばん効いた指示を1つ書いておいてください。

4象限の埋まり具合

本日で3つの象限に手が入りました。残る1つは現場で足します。

	計算的（決定論・安価・高速）	推論的（LLM の判断）
ガイド 事前・予防	**LSP、CLI、コードモッド** 本日は触れていません	**CLAUDE.md と Skill** M2・演習1・M6 で作りました
センサー 事後・自己修正	**規約チェックの Hook** 演習3 で機械判定を受けました	**code-reviewer のレビュー** M5 と演習3 で使いました

■ 本日の中心　■ 本日さわった層　□ 手つかず

自分のプロジェクトでは、どの象限から足すかを1つ決めておいてください。

個人設定の後始末

午前に入れたユーザー設定を、.bak から戻します。

戻すものと、残してよいもの

戻すもの

~/.claude/settings.json .bak を上書きで戻す
~/.claude/CLAUDE.md 研修用に足した1行を外す

残してよいもの

statusLine の設定
~/.claude/skills/ の公式Skill 2本

同じ手順は README.md の末尾にもあります

1. **settings.json を戻す**
   .bak の名前を settings.json に戻して上書きします。
2. **CLAUDE.md を戻す**
   研修用に足した出力の書式は、業務では外しておきます。
3. **残してよいもの**
   statusLine の設定は、そのまま残しても差し支えありません。
4. **公式Skill 2本の扱い**
   使い続けるなら残します。消す場合はフォルダごと削除します。

同じ手順は、配布物の README.md 末尾とハンズオンガイドの末尾にも載せてあります。

クイズ

理解度チェック

Q. 明日いちばん最初にやることとして、いま勧められるのはどれですか。

- **A** 自分のリポジトリを VSCode で開き、/init で CLAUDE.md を1枚作る
- **B** 演習3 と同じ Hook・Skill・サブエージェントを一式そろえる
- **C** 全社で共通の CLAUDE.md を先に決めてから配る
- **D** 今日の演習フォルダで、同じ課題をもう一度やり直す

正解は次のページで確認します。まず自分で考えてみてください。

解説

理解度チェック 解説

正解はA。自分のリポジトリで CLAUDE.md を1枚作ります。

A　正解　自分のリポジトリを開いて /init を実行する
1枚あれば、毎回書いていた前提が消えます。効いたかどうかは翌日すぐ分かります。

B　演習3 と同じ一式をそろえる
Lv3 は今日の到達点です。動かす前に道具が増えると、育てる順序が見えません。

C　全社共通の CLAUDE.md を先に決める
1つのリポジトリで効いた規約が無いまま決めると、守られない文言だけが残ります。

D　演習フォルダで同じ課題をやり直す
題材が固定なので、自分のコードで測らないと差が出ません。

順序は CLAUDE.md、規約、Hook です。
毎回言っている1件が決まってから、機械で止める側へ上げます。

明日からの一歩

1週間の目標は、CLAUDE.md 1枚と、規約1つの Hook です。

1. **自分のリポジトリを VSCode で開く**
   フォルダを開いて、信頼するを選びます。

2. **/init で CLAUDE.md を1枚作る**
   生成された行から、要らないものを削ります。

3. **効いた規約を1つだけ Hook に上げる**
   毎回言っている1件を選んで機械判定にします。

今週やること

1 自分のリポジトリを開く
2 /init で CLAUDE.md を1枚作る
3 要らない行を削る
4 効いた規約を1つ Hook に上げる

やらないこと
最初から Lv3 を揃えること

Lv3 が正解ではありません。
Lv1 相当を1枚置いて、効いたものだけ足していくのが実務の順序です。

情報源と更新の粒度

明日から自分で追う先は、この8か所です。

週次ダイジェスト https://code.claude.com/docs/en/whats-new	毎週
変更点の一覧 https://code.claude.com/docs/en/changelog	ほぼ毎日
変更点の原本 https://github.com/anthropics/claude-code/blob/main/CHANGELOG.md	ほぼ毎日
API のリリースノート https://platform.claude.com/docs/en/release-notes/api	数日ごと
モデル一覧 https://platform.claude.com/docs/en/about-claude/models/overview	モデル公開時
非推奨と引退の予定 https://platform.claude.com/docs/en/about-claude/model-deprecations	廃止告知時
稼働状況 https://status.claude.com/	障害発生時
製品発表と事例 https://www.anthropic.com/news	随時

情報システム部門へ共有するなら、status.claude.com の購読を最初に入れます。

質問

振り返りと Q&A

1. 今日いちばん効いた指示はどれでしたか
2. どこで手が止まりましたか
3. 現場で最初に使う場面はどこですか

Claude Code

Visual Studio Code

GitHub

Python

今日さわった4つです。更新情報は前のページの8か所で追えます。

※ 各ロゴは各社の商標です。

出典一覧 ①

本日の説明は、公式ドキュメントを出典にしています。

Claude Code 公式ドキュメント

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

overview, vs-code, permission-modes, commands, model-config,
hooks, skills, sub-agents, plugins, statusline, context-window,
whats-new, changelog

Claude Platform（モデルと API）

https://platform.claude.com/docs/en/

about-claude/models/overview, about-claude/pricing,
release-notes/api, about-claude/model-deprecations

稼働状況と障害履歴

https://status.claude.com/

いずれも2026年7月30日に開いて内容を確認しています。

出典一覧 ②

章1と章2で出した数値は、次の資料を2026年7月30日に開いて確認したものです。

開発トレンド調査	https://resources.anthropic.com/2026-agentic-coding-trends-report
METR 実験 2025	https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
METR 追試 2026	https://metr.org/blog/2026-02-24-uplift-update/
開発者調査 2025	https://survey.stackoverflow.co/2025/ai
モデル横断比較	https://artificialanalysis.ai/models
SWE-bench 本家	https://www.swebench.com/
SWE-bench Pro	https://labs.scale.com/leaderboard/swe_bench_pro_public
検証済みスコア	https://llm-stats.com/benchmarks/swe-bench-verified
ベンチマーク飽和	https://epoch.ai/benchmarks/mmlu
Copilot 課金の変更	https://github.blog/news-insights/company-news/github-copilot-is-moving-to-usage-based-billing
Copilot 単価表	https://docs.github.com/en/copilot/reference/copilot-billing/models-and-pricing
VS Code の更新	https://code.visualstudio.com/updates
MCP 仕様	https://modelcontextprotocol.io/specification/2026-07-28
NEC 協業発表	https://www.anthropic.com/news/anthropic-nec
国内の導入事例	https://prtimes.jp/main/html/rd/p/000005136.000000136.html
就業者の意識調査	https://rc.persol-group.co.jp/thinktank/thinktank-column/202603130001/
ハーネスの原典	https://zenn.dev/r_kaga/articles/329afdc151899f

株式会社ギブリー

〒150-0036 東京都渋谷区南平台町15-13 帝都渋谷ビル8F