WHITE PAPER

非エンジニアのためのClaude実践ガイド

Claude Code 完全マニュアル

非エンジニアのためのAI駆動開発 入門ガイド（全11章＋リファレンス）

発行：2026年6月　／　対象：非エンジニア／ビジネスパーソン

Claude Works（クロードワークス）

非エンジニアのためのClaude実践メディア　claudelab.jp

このマニュアルは、プログラミング未経験・非エンジニアの方が、AI開発ツール「Claude Code」をゼロから使いこなし、自分のアイデアをアプリやツールとして形にできるようになるための完全ガイドです。専門用語はすべて噛み砕いて解説しています。パソコンの基本操作ができれば、前提知識はほぼ不要です。

全11章＋リファレンス（早見表・用語集・FAQ）。とにかく動かしたい方は、第1章→第2章→第3章→第6章だけでも「アプリを作って公開する」体験ができます。

00. はじめに ― このマニュアルについて

この章でわかること

- このマニュアルが「何を」「誰のために」書かれているか
- 読み進めるうえでの心構え（これだけは知っておきたい3つのこと）
- 記号の読み方（凡例）

---

1. このマニュアルのゴール

このマニュアルを最後まで読むと、あなたは次のことができるようになります。

- **自分のアイデアを、コードを書かずに「アプリ」や「ツール」として形にする**
- AI（Claude Code）に正しく指示を出し、エラーが出ても自力で前に進める
- 作ったものを**インターネット上に公開**して、人に使ってもらう

「**シンプルな社内システムなら1日で作れるようになる**」――このマニュアルが掲げるゴールはここにあります。それを、**プログラミング未経験の人でも到達できる**ように噛み砕いたものです。

> 💡 実際に、IT業界ではない会社で、社長がインターン生に Claude Code で**自社の在庫管理システム**を作らせたり、**営業担当者が名刺管理アプリを自作**して社内で高評価を得た、といった例が出てきています。非エンジニアでも“作る側”に回れる時代になった、というのがこのツールの本質です。

---

2. Claude Code を始める前に知っておきたい3つのこと

① AI は「すごく優秀だけど、たまに嘘をつくアシスタント」

Claude Code の中身は **LLM（大規模言語モデル）** という AI です。これは「次に来そうな言葉を確率で予測する」仕組みで動いています。だから、

- **同じ指示でも、毎回まったく同じ答えになるとは限らない**
- **自信満々に間違えることがある**（これを業界では「ハルシネーション＝もっともらしい嘘」と呼びます）

これは故障ではなく**仕様**です。「間違えることもある優秀な新人」と一緒に働くつもりで、出てきた結果を**あなたが確認する**――この姿勢がうまく使うコツです。

② 「待ち時間」と「対話」が当たり前

AI は一瞬で答えることもあれば、数十秒〜数分考えることもあります。複雑な作業ほど待ちます。じれったく感じても、**「もう一度違う言い方で頼む」「エラーをそのまま見せる」**といった対話を重ねることで、確実にゴールに近づきます。

③ 完成度は「いきなり100点」ではなく「60〜80点を育てる」

事前にしっかり準備しても、**最初の実装の完成度は60〜80%くらいに留まることが多い**ものです。残りは、あなたが実際に画面を触って「ここがおかしい」と伝え、直してもらう――この**往復で育てる**のが現実的な進め方です。最初からうまくいかなくても、それが普通です。

---

3. 読み方ガイド

- **とにかく動かしたい人**：`01 → 02 → 03 → 06` の順で、アカウント作成からアプリ公開まで体験できます。
- **じっくり理解したい人**：番号順に読めば、基礎 → 実践 → 応用へと自然にレベルアップします。
- **困ったときだけ調べたい人**：巻末の R1 コマンド早見表・R2 用語集・R3 トラブル対処と FAQを逆引きで使ってください。

> ⚠️ Claude Code は**進化が非常に速い**ツールです。このマニュアルは**2025年10月時点（Claude バージョン 4.5）**の情報をもとにしています。画面や機能が変わっていたら、対話中に `/release-notes` と打つか、公式ドキュメントで最新を確認してください。

---

4. 凡例（このマニュアルの記号ルール）

表記	意味
`>〇〇して`	Claude Code に**あなたが打ち込む指示（プロンプト）**。行頭の `>` は「入力欄」のしるしで、実際には打ちません。
「ターミナルで実行：`xxx`」	黒い画面（ターミナル）に**自分でタイプするコマンド**。
📦 **用語ミニ解説**	専門用語をその場でやさしく説明。
💡	知っておくと得するコツ。
⚠️	やりがちな失敗・注意点。
✅	「これができればOK」のチェックポイント。

---

5. 用語の超入門（ここだけ先に）

本文に入る前に、いちばん基本の3語だけ説明します。これさえ頭に入れば、あとは本文の「用語ミニ解説」で十分追いつけます。

📦 **用語ミニ解説：ターミナル（黒い画面）** パソコンに文字で命令を出すための画面のこと。マウスでアイコンをクリックする代わりに、キーボードで命令文を打ち込みます。Claude Code はこの画面の中で動きます。最初は怖く見えますが、「AI と会話するチャット欄」だと思えば大丈夫です。

📦 **用語ミニ解説：CLI（コマンドラインインターフェース）** 上の「ターミナルに文字で命令する方式」の正式名称。Claude Code は、わざとこの方式を選んでいます（理由は01章で説明します）。

📦 **用語ミニ解説：Git（ギット）／GitHub（ギットハブ）** Git は「ファイルの変更履歴を記録・管理する仕組み」。GitHub はそれを**インターネット上に保存・共有できるサービス**です。Claude Code はこの2つと深く連携して動きます。まだ持っていなくても、03章で一緒に登録します。

それでは、まず「Claude Code とは何者なのか」から始めましょう。

01. Claude Code とは何か

この章でわかること

- Claude Code がそもそも「何をするもの」なのか
- これまでの AI 開発ツールと何が決定的に違うのか
- なぜ「黒い画面（ターミナル）」で動くのか
- 「AI 駆動開発」「Vibe コーディング」という新しい働き方

1. ひとことで言うと

Claude Code（クロードコード）は、あなたの代わりにプログラムを書いてくれる AI のアシスタントです。 あなたが日本語で「TODO アプリを作って」「このエラーを直して」とお願いすると、ファイルを作ったり、コードを書いたり、動作確認やインターネットへの公開まで――本来エンジニアがやっていた一連の作業を、AI が肩代わりしてくれます。

作っているのは **Anthropic（アンソロピック）** という会社です。

📦 **用語ミニ解説：Anthropic（アンソロピック）** AI を作っている会社の名前です。元 OpenAI の研究者だった Dario Amodei（ダリオ・アモデイ）氏らが「AI の安全性」を最優先に掲げて 2021 年に設立しました。「公益法人」という、利益だけでなく社会的責任を重視する形態で運営されています。この会社が作っている AI 本体の名前が **Claude（クロード）** です。

📦 **用語ミニ解説：Claude（クロード）とモデルの種類** Claude は Anthropic の AI の名前。本マニュアル作成時点（2025年10月）のバージョンは **4.5** で、頭の良さ・速さ・コストの違いで **Opus（オーパス／最も賢い）** ／ **Sonnet（ソネット／バランス型）** ／ **Haiku（ハイク／速くて安い）** の3種類があります。Claude Code はこの Claude を“手足”として使い、実際にコードを書く道具です。

2. これまでの AI 開発ツールと、その「限界」

Claude Code の革新性を理解するには、それ以前の AI ツールを知ると一気に腹落ちします。これまでの AI 支援開発は、大きく**3つのタイプ**に整理できます。

タイプ①：コードエディタ支援ツール

普段プログラマーが使う文書作成ソフト（エディタ）に AI を組み込み、入力を先回りして補完してくれるもの。

- 代表例：**GitHub Copilot、Cursor、WindSurf、Cline、Roo Code**
- **限界**：その特定のエディタの中でしか恩恵を受けられない。例えば iPhone アプリ開発は専用ソフト（Xcode）が必要で、そこでは十分に使えない、など「道具に縛られる」問題がありました。

タイプ②：アプリ自動生成ツール

ブラウザ上で「こんなアプリが欲しい」とお願いすると、丸ごと作ってくれるサービス。

- 代表例：**Replit、Firebase Studio、v0、Bolt、Lovable**
- **限界**：手軽で**非エンジニアにも使いやすい**反面、「何が出てくるかは AI まかせ」。見た目が似通いがちで、思い通りにするには何度もチャットでやり直す必要がありました。

タイプ③：自律型コーディングエージェント

ブラウザ上で、AI が一人の開発者のように、タスクを1つずつ自律的にこなすもの。

- 代表例：**Devin、OpenAI Codex、Jules、GitHub Copilot Workspace**
- **限界**：毎回プランを確認し、成果物をチェックする手間が増えました。

共通する根本問題

どのタイプも結局、「**AI が動く“環境”（特定のエディタやブラウザサービス）に、人間の作業が縛りつけられる**」という限界を抱えていました。

---

3. AI と働くときの「2つのストレス」

AI 支援開発には、特有の2つのストレスがあります。これを知っておくと、後で「なぜ Claude Code が快適なのか」が分かります。

ストレス①：AI は嘘をつく

AI は「もっとも確からしい答え」を確率で出すので、**間違っていても再現性なく堂々と間違える**ことがあります。この不毛なやり取りをいかに減らすかが、使い心地を左右します。

ストレス②：待ち時間が発生する

AI がどれだけ速くても、考えている間は手が止まります。複雑な作業ほど待たされ、気持ちよくありません。

> 💡 **Claude Code はこの2つのストレスを大きく減らしたからこそ、一気に普及した**とされています。

---

4. なぜ「黒い画面（ターミナル）」で動くのか

Claude Code は、見た目が華やかなアプリではなく、文字だけの**ターミナル（CLI）**で動きます。一見地味ですが、これは意図的な設計です。

開発担当の Boris Cherny（ボリス・チェルニー）氏は、CLI を選んだ理由をこう語っています。

1. **社内のみんなが使えるから**：Anthropic 社内では VS Code、Xcode、Vim、Emacs など人によって使う道具がバラバラ。でも「ターミナル」だけは全員が共通して使える土台だった。
2. **未来に備えるため**：AI の進化が速すぎて、近いうちに今のエディタすら使わなくなるかもしれない。だから特定の見た目に作り込まず、シンプルで身軽にした。

そして結果的に、このシンプルさが「**高速で何度も試行錯誤できる**」という快適さを生んだ――これは後から分かったことだ、とも語られています。

📦 **用語ミニ解説：なぜ CLI だと「環境に縛られない」のか** ターミナルは、どのパソコン（Mac/Windows/Linux）にも、どのエディタにも、ほぼ必ず付いてくる“共通の入り口”です。だから Claude Code は「特定のアプリの中の機能」ではなく、「どこででも呼び出せる独立した相棒」になれた、というわけです。

---

5. Claude Code の5つの特徴

Claude Code の主な特徴は、次の5つです。

1. **マルチファイル対応**：プロジェクト全体を理解し、複数のファイルをまとめて編集できる。
2. **並列実行**：複数の作業を同時に進められる（→ 08章で詳説）。
3. **自然言語で対話**：専門知識がなくても、日本語の指示だけで実装できる。
4. **実行環境と深く統合**：コマンド実行・テスト・公開（デプロイ）まで一気通貫でやってくれる。
5. **環境に依存しない**：どのエディタ・どの環境でも軽快に動く。

> 💡 驚きの事実：Anthropic 社内では、**Claude Code 自身のコードの約80%が、すでに Claude Code によって書かれている**と紹介されています。AI が AI を作り、自分を改良していく時代に入っているのです。

---

6. 「AI 支援開発」から「AI 駆動開発」へ

ここがこの本のタイトルでもある核心です。

- **AI 支援開発**：人間が主役で、AI が手伝う（=これまで）。
- **AI 駆動開発**：AI が設計・実装・テスト・公開まで主体的に進め、人間はディレクション（方向づけ）と確認に回る（=これから）。

Claude Code は環境に縛られないため、使い方を自在に選べます。

- **完全おまかせ**（Vibe コーディング）：細かいことは気にせず、AI にどんどん作らせる。
- **責任ある確認型**：AI の計画と成果を、人間が一つひとつチェックしながら進める。
- **アシスタント型**：人間が主導で書きつつ、優秀な助手として AI を使う。

この“自在さ”こそが、新しい開発スタイル「AI 駆動開発」を可能にしました。

📦 **用語ミニ解説：Vibe コーディング（バイブコーディング）** 「Vibe」は“ノリ・雰囲気”の意味。**実装の細かい中身を気にせず、自然言語で「こんなの作りたい」というノリだけでアプリを形にする**スタイルを指します。非エンジニアにとっては、まさにこの Vibe コーディングが入り口になります。

まとめ

- Claude Code は、**日本語の指示でプログラムを作ってくれる AI アシスタント**。作っているのは安全性重視の Anthropic 社。
- 従来ツールは「特定の環境に縛られる」限界があったが、Claude Code は**ターミナル（CLI）で動くことで、どこでも使える相棒**になった。
- 人間が確認役に回り、AI が主体的に開発を進める——これが**「AI 駆動開発」**。非エンジニアでも“作る側”に立てる時代が来ています。

次は、実際に使い始める前に避けて通れない「お金」の話を、いちばんやさしく整理します。

---

02. 料金プランの選び方

この章でわかること

- 「結局どのプランを選べばいいか」の結論
- サブスク（定額）と API（従量課金）の違い
- トークン・メッセージ制限という独特のルール
- 使った量を確認するコマンド（`/cost`・`/usage`）
- 賢いモデルの選び方とコスト節約のコツ

まず結論：どれを選べばいい？

迷ったら、次の基準で OK です。

あなたのタイプ	おすすめ	月額の目安
まず試したい／趣味・週末に少し使う	Pro	$20
日常的に使う／個人開発・フリーランス・本気で学ぶ	Max 5x	$100
1日中ガンガン使う／複数の作業を同時に走らせる	Max 20x	$200
たまにしか使わない／大量の自動処理だけしたい	API 従量課金	使った分だけ

> 💡 **最初は Pro（$20）から始めて、足りなくなったら上げる**——これが王道です。ヘビーに使う人は最上位の **Max 20x** を選ぶこともあります（複数の AI を同時に走らせるとすぐ上限に達するため）。

---

1. 大きく2つの支払い方法がある

Claude Code の料金は、最初に **「①サブスクリプション（定額制）」** か **「②API 従量課金（使った分だけ）」** のどちらかを選びます。

① サブスクリプション（定額制）— 個人におすすめ

毎月決まった額を払えば、上限まで使い放題。普段ブラウザで使う Claude（チャット版）と**同じ契約**です。

⚠️ **重要**：無料の **Free プランでは Claude Code は使えません**。有料の **Pro 以上**が必要です。

プラン	月額	使えるモデル	5時間ごとの目安	向いている人
Free	無料	—	（Claude Code 使用不可）	—
Pro	$20（年払いで割引）	Sonnet 4.5	約10〜40回の指示	小規模・軽作業・お試し
Max 5x	$100	Sonnet 4.5＋Opus 4.1	約50〜200回	日常使い・パワーユーザー
Max 20x	$200	Sonnet 4.5＋Opus 4.1	約200〜800回	チーム・大規模・同時並行

⚠️ ブラウザのチャット版 Claude と Claude Code は**同じ使用量の枠を共有**します。どちらかを使いすぎると、もう一方も制限にかかります。

② API 従量課金（使った分だけ）

契約とは別に「Claude Console」という開発者向けアカウントを作り、**使ったトークンの分だけ**支払う方式。たまにしか使わない人や、大量の自動処理をしたい人向けです。

モデル	入力 100万トークンあたり	出力 100万トークンあたり
Opus 4.1（最も賢い）	$15	$75
Sonnet 4.5（バランス）	$3	$15
Haiku 4.5（速くて安い）	$1	$5

> 💡 Anthropic によると、開発者1人あたりの平均コストは**1日約6ドル、9割の人は1日12ドル未満**。Sonnet 中心なら月50〜60ドルが目安とされています。 ⚠️ 従量課金では **Opus は高額**で、1つの作業を回すだけで5ドル前後かかる印象です。**ここぞという時以外は Sonnet で十分**です。

📦 **用語ミニ解説：トークンとは** AI が文章を処理するときの**最小のかたまり**。だいたい「単語1個」に近いイメージですが、正確には AI が扱いやすいよう独自に区切った単位です。**入力（こちらが渡した文章）と出力（AI が返した文章）の両方**でトークンを消費し、それが料金の単位になります。

---

2. 独特のルール：「5時間ごと」と「週単位」の制限

サブスクには、ちょっと変わった制限の仕組みがあります。

- **5時間ごとにリセット**：最初の指示から5時間を「1セッション」とし、その枠内のメッセージ数に上限があります。5時間経つとカウントがリセットされます。
- **週単位の上限**（2025年8月28日〜）：休みなく酷使する一部ユーザー向けに、週ごとの上限も追加されました。Anthropic は「影響を受けるのは全体の5%以下」と説明しています。

⚠️ つまり、**複数の作業を一気に並行させると、最上位の Max 20x でも上限に早く届く**ことがあります（→ 08章）。

---

3. 使った量を確認するコマンド

Claude Code を起動した状態で、次の“スラッシュコマンド”を打つと使用状況が見られます。

📦 **用語ミニ解説：スラッシュコマンド** 入力欄で `/`（スラッシュ）から始める特別な命令のこと。`/cost` のように打つと、その機能が実行されます。詳しくは R1 コマンド早見表。

- `> /cost` … **API 従量課金ユーザー向け**。今のセッションでいくら使ったか（金額・トークン数・変更行数）を表示。
  - ⚠️ サブスクは定額なので、このコマンドは基本不要です。
- `> /usage` … **サブスクユーザー向け**。今のセッション・今週（全モデル）・今週（Opus）の使用率が何%で、いつリセットされるかを表示。

> 💡 API ユーザーは、Claude Code の**終了時にも料金が表示**されます。最初のうちは `/cost` をこまめに見て「この作業はこれくらいかかる」という金銭感覚を養いましょう。

---

4. モデルの選び方（賢さ・速さ・コストの使い分け）

Max プランや API ユーザーは、作業に応じて AI のモデルを切り替えられます。起動中に次のように打ちます。

- `> /model` … メニューからモデルを選ぶ
- `> /model sonnet` … バランス型の Sonnet 4.5 に切替（**日常はこれが推奨**）
- `> /model opus` … 最も賢い Opus 4.1 に切替（難しい設計や難解なバグ向け。使用量の消費は速い）
- `> /model haiku` … 速くて安い Haiku（単純作業向け）

知っておくと得する2つのモード

- `opusplan`**（オーパスプラン）**：**計画を立てるときだけ賢い Opus、実際にコードを書くときは安い Sonnet** に自動で切り替わる“いいとこ取り”モード。コストを抑えたい人に最適です。
- **100万トークンモデル（** `/model sonnet[1m]` **）**：**API 従量課金ユーザー限定**。巨大なプロジェクトや膨大なログを一度に扱える特大メモリ版。ただし20万トークンを超えると単価が上がるので注意。

📦 **用語ミニ解説：コンテキストウィンドウ（AI の作業メモリ）** AI が一度に覚えていられる情報量の上限のこと。通常は最大20万トークン。「100万トークンモデル」はこの作業メモリが5倍ある特別版、というイメージです（詳しくは08章）。

5. コスト節約のコツ（まとめ）

- 💡 日常は **Sonnet**。難所だけ **Opus**。計画と実装を分けたいなら **opusplan**。
- 💡 サブスクなら使い放題なので、**従量課金を気にせず試行錯誤できる**のが最大の利点。快適さ重視ならサブスク。
- 💡 ブラウザ版 Claude と枠を共有している点に注意。
- 💡 API ユーザーは `/cost` で金銭感覚を養う。サブスクは `/usage` で残量を確認。

まとめ

- **迷ったら Pro（$20）から。本気なら Max 5x、ヘビーユースは Max 20x。**
- 支払いは「サブスク（定額・個人向き）」か「API（従量課金・たまに使う人向き）」の2択。
- 料金の単位は**トークン**。5時間ごと・週ごとの制限がある。
- モデルは**日常 Sonnet・難所 Opus・節約 opusplan**で使い分ける。

プランの見当がついたら、いよいよ登録とインストールです。次の章で、画面の手順をひとつずつ追っていきましょう。

---

03. 導入ガイド — 登録とインストール

この章でわかること

- Claude Code を使うための**アカウント**の作り方（2種類あります）
- 自分のパソコンへの**インストール**手順（Mac / Windows / Linux）
- 初めて起動したときの**画面の進め方**（ここでつまずく人が多いので丁寧に）
- GitHub との連携と、開発しやすい**画面レイアウト**

> ⏱ 所要時間の目安：アカウント作成 10分 ＋ インストール 10分 ＋ 初回起動 10分 ＝ 約30分。

---

1. まず「どっちのアカウントが必要か」を決める

Claude Code を使うには、次の**どちらか**のアカウントが必要です。両方は要りません。

	① サブスク（定額制）	② API（従量課金）
アカウント名	**Claude Account**	**Claude Console Account**
登録サイト	https://claude.ai	https://console.anthropic.com
向いている人	**ほとんどの個人・初心者**	不定期利用・大量処理をしたい人
支払い	月額固定（使い放題に近い）	使った分だけ（トークン単位）

> 💡 **迷ったら①サブスク**でOK。料金やプランの詳しい選び方は02章を参照してください。このマニュアルは①サブスクを前提に進めます。

📦 **用語ミニ解説：アカウント** そのサービスを使うための「あなた専用の登録情報」。メールアドレスとパスワードなどで作ります。①と②は**別物**なので、間違えないように。

2-A. ①サブスク（Claude Account）の登録手順

すでに ChatGPT のように **ブラウザで Claude（claude.ai）を使ったことがある人は、そのアカウントがそのまま使えます**。新規の人は以下の手順で。

1. **https://claude.ai にアクセス**する。
2. **登録方法を選ぶ**：「Google で続ける」か「メールアドレスを入力」。メールの場合、届いた確認メールの**認証コード**を入力します。
3. **利用規約・年齢確認・電話番号認証**：18歳以上の確認にチェック → 電話番号を入力 → SMS で届いたコードを入力。
4. **アカウント種別を選ぶ**：個人用 / チーム用 → **個人用**を選択。
5. **プランを選ぶ**：Free / Pro / Max。**Claude Code は有料の Pro 以上が必要**です（Free では使えません）。
    - Pro は**年払いにすると約17%お得**（月額換算で安くなる）。
    - Max には 5x / 20x があり、**年払いはありません**。

6. **初期設定**：呼ばれたい名前、興味のあるジャンルなどを選んで完了。

✅ ここまでで claude.ai にログインできれば、サブスクのアカウントは準備完了です。

> ⚠️ **Free プランのままでは Claude Code は動きません。** 必ず Pro 以上に加入してください。

2-B. ②API（Claude Console Account）の登録手順 ※必要な人だけ

サブスクで使う人は**この節は飛ばして**3章のインストールへ進んでOKです。API を使いたい人だけ読んでください。

1. https://console.anthropic.com/login にアクセス。
2. Google またはメールで登録（claude.ai とは**別のアカウント**になります）。
3. 利用規約・プライバシーポリシーに同意。
4. **個人(Individual)** か **組織(Organization)** を選ぶ。
5. **$5 のクレジットを購入**：クレジットカード情報を入力。「Business tax ID（事業者番号）」は個人なら**空欄でOK**。購入したクレジットの**有効期限は1年**。
6. **APIキーを作成**：「Create API Key」→ Workspace は「Default」→ 名前を付けて「Add」。

> ⚠️⚠️ **最重要：APIキーは作成時の1回しか画面に表示されません。** 表示されたら必ず「Copy Key」でコピーして、メモアプリなどに**安全に保管**してください。逃すと二度と確認できず、作り直しになります。他人に見られない場所に保管を。

💡 残高は左メニューの「Billing」で確認でき、「Auto reload」で自動補充も設定できます。

---

3. パソコンへのインストール

3-1. 必要な環境（推奨スペック）

項目	必要なもの
OS	macOS 10.15+ / Ubuntu 20.04+・Debian 10+ / Windows 10+（WSL または Git for Windows 経由）
メモリ	4GB 以上
ソフト	**Node.js 18 以上**（入れ方は後述）
ネット	認証・AI処理にインターネット接続が必須

📦 **用語ミニ解説：Node.js（ノードジェイエス）** パソコン上でプログラム（特に JavaScript）を動かすための土台ソフト。Claude Code もこの上で動くことがあるため、入っていないと動きません。下記のインストール方法に含めて導入します。

📦 **用語ミニ解説：WSL（ダブリュ・エス・エル）** Windows の中で Linux を動かす公式の仕組み。Windows ユーザーが Claude Code を快適に使うための定番環境です。

3-2. インストールコマンド（おすすめ＝ネイティブインストール）

今は Node.js が無くても入れられる「ネイティブインストール」が公式推奨です。お使いの環境に合わせて、ターミナルに1行打ち込みます。

- **Mac（Homebrew がある人）**：ターミナルで実行：`brew install --cask claude-code`
- **Mac / Linux / WSL（共通）**：ターミナルで実行：`curl -fsSL https://claude.ai/install.sh | bash`
- **Windows（PowerShell）**：ターミナルで実行：`irm https://claude.ai/install.ps1 | iex`

📦 **用語ミニ解説：Homebrew（ホームブリュー）** Mac でソフトを簡単に入れるための「アプリの自動販売機」のような管理ツール。`brew install ...` の形で使います。無い場合は上の `curl` の行を使えばOK。

3-3. Node.js 経由で入れる方法（上がうまくいかない場合）

Node.js を入れてから npm で導入する昔ながらの方法です。

1. Node.js を導入（nvm というバージョン管理ツール経由が安全）。ターミナルで実行：`curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash` 続けて：`nvm install 22`
2. Claude Code を導入。ターミナルで実行：`npm install -g @anthropic-ai/claude-code`

> ⚠️ **Windows で WSL を使っていてエラーが出たら**、Windows 本体側の Node.js とぶつかっているのが原因のことが多いです。上記の nvm でバージョンを揃えると解決します。

---

4. 初めての起動 — 画面の進め方（つまずきポイント満載）

ステップ0：作業フォルダを作ってそこで起動する

Claude Code は「プロジェクト（作業フォルダ）ごと」に起動するのが基本です。まず空のフォルダを作り、その中で起動します。

ターミナルで実行（例：todo という名前のフォルダを作る）：

```
mkdir todo
cd todo
claude
```

> ⚠️ いきなり何もない自分のホーム（パソコンの一番上の階層）で `claude` すると「プロジェクトフォルダで起動してね」という警告が出ます。**必ずフォルダを作ってから**起動しましょう。

📦 **用語ミニ解説：** `mkdir` / `cd` `mkdir todo` =「todo という名前のフォルダを作る」。`cd todo` =「その todo フォルダの中に移動する」。`cd` は "change directory（フォルダ移動）" の略です。

ステップ1：テーマ（見た目）を選ぶ

起動すると配色を聞かれます。**基本は「1. Dark mode」でOK**。後から `/theme` で変えられます。
（画面のキャラクターは "Clawd" という Claude Code のマスコットです。）

ステップ2：ログイン方法を選ぶ ← ここが最重要

```
Select login method:
  1. Claude account with subscription   ← サブスクの人はこれ！
  2. Anthropic Console account          ← API課金の人だけ
```

> ⚠️⚠️ **サブスク（Pro / Max）の人は、必ず「1」を選んでください。** 間違えて「2」を選ぶと、サブスク料金を払っているのに**さらに API 課金が追加されてしまう**事故が多発しています。これは本当に多いミスです。

ステップ3：ブラウザで承認する

「1」を選ぶとブラウザが自動で開き、ログインと接続の承認を求められます。「承認する」を押し、ターミナルに `Login successful` と出れば成功です。

> 💡 サーバー上などで**ブラウザが開けない環境**では、ターミナルに表示される URL を自分でコピーしてブラウザで開き、表示された**認証コードをターミナルに貼り付け**ます。

ステップ4：セキュリティの注意書き（Enter で進む）

「Claude は間違えることがあるので、実行するコードは必ず確認してね」「信頼できるコードでだけ使ってね」という注意が出ます。読んで Enter。

ステップ5：ターミナルセットアップ → 「1」を選ぶ

「Use Claude Code's terminal setup?」と聞かれたら **「1. Yes」**。これで指示を打つときに `Shift + Enter` **で改行**できるようになり、長い指示が書きやすくなります。

ステップ6：フォルダを信頼するか確認 → 「1」を選ぶ

「Do you trust the files in this folder?」は、Claude がそのフォルダのファイルを読んでよいかの確認。自分で作ったフォルダなら **「1. Yes」**。

✅ ここまで進むと、いよいよ Claude と会話できる状態になります！

---

5. GitHub と連携する（強くおすすめ）

📦 **用語ミニ解説：`gh` コマンド（GitHub CLI）** GitHub をターミナルから操作するための公式ツール。これを入れておくと、**リポジトリ作成・公開・レビュー依頼などが全部ターミナル内で完結**し、ブラウザを開く手間がほぼ無くなります。Claude Code の体験が一段上がるので、ぜひ入れましょう。

5-1. gh を入れる

- Mac：ターミナルで実行：`brew install gh`
- Windows：ターミナルで実行：`winget install --id GitHub.cli`
- Linux / WSL：ターミナルで実行：`sudo apt install gh`

> 💡 面倒なら Claude Code に頼んでもOK。起動した状態で `> gh コマンドをインストールしてください。` と打てば、エラーが出ても自分で直しながら入れてくれます。

5-2. GitHub アカウントと連携する

ターミナルで実行：`gh auth login` 画面の案内に従って進むと、あなたの GitHub アカウントと連携されます。

5-3. リポジトリ（保管場所）を作る

📦 **用語ミニ解説：リポジトリ** プロジェクトのファイル一式と変更履歴を入れておく「保管箱」。GitHub 上に作ると、バックアップにも公開にも使えます。

作り方は2通り：

- 自分でやる：ターミナルで実行：`gh repo create <好きな名前> --private --source=. --remote=origin`
- Claude に任せる：`> gh コマンドで新しい private リポジトリを作成してください。` と打つだけ（名前もフォルダ名などから自動で決めてくれます）。

---

6. 開発しやすい画面レイアウト（基本ポジション）

Claude Code は、VS Code などのエディタの中で使うと力を発揮します。おすすめの配置はこうです。

```
┌──────────────────────────┬──────────────────────────┐
│ 左：Claude Code          │ 右：普通のターミナル     │
│ （自然言語で指示）       │ （自分でコマンド実行）   │
└──────────────────────────┴──────────────────────────┘
```

- **左の Claude Code** には、日本語で「〇〇して」と指示を出します。
- **右の普通のターミナル**では、`npm run dev`（アプリを試しに動かす）や `gh auth login` など、**自分で打つ必要があるコマンド**を実行します。

📦 **用語ミニ解説：VS Code（ブイエス・コード）** 無料で人気の「コードを書くためのアプリ（エディタ）」。Claude Code はこの中のターミナルで動かすと、エラー情報の受け渡しなどがスムーズになります。Cursor というよく似たアプリでもOK。

まとめ

- アカウントは**サブスク(Claude Account)** か **API(Console Account)** のどちらか。初心者はサブスク。
- インストールは**ネイティブインストール1行**が基本。困ったら Node.js + npm 方式。
- 初回起動の難関は **「ログイン方法でサブスクなら必ず1」**。ここだけは間違えないこと。
- `gh` コマンドを入れて GitHub と連携すると、開発がぐっと楽になる。

---

04. 基本操作とコマンド大全

この章でわかること

- Claude Code を起動したあと、**毎日使う操作**のすべて
- 「やり直したい」「中断したい」「前の続きから」などの**お助け操作**
- AI に**じっくり考えさせる**方法（ultrathink）
- AI の「記憶（コンテキスト）」を**上手に管理**する方法

> 📌 この章は「困ったときに開く辞書」としても使えます。最初は流し読みで大丈夫。

---

1. いちばん最初に打つ：`/init`

新しいプロジェクトで Claude Code を使い始めるとき、まず打つのがこれ。

```
> /init
```

これで **CLAUDE.md** という「プロジェクトの説明書（AI の記憶ノート）」のひな形が作られます。中身の書き方は05章で詳しく説明します。

📦 **用語ミニ解説：スラッシュコマンド** `/init` のように `/`（スラッシュ）**で始まる特別な命令**のこと。Claude Code に「機能を実行して」と伝えるショートカットです。普通の日本語の指示（プロンプト）とは区別されます。全コマンドの一覧はR1 早見表にあります。

---

2. ファイルを見せる：`@` インポート

特定のファイルについて相談したいときは、`@` を打つとファイル名の候補が出ます。選んでそのまま指示に混ぜられます。

```
> このファイルの内容をやさしく解説して。@src/components/Form.tsx
> @index.tsx が長くなりすぎたので、適切に分割して。
```

💡 **いつ使う？**「このファイルを直して」「この設定を見て」など、**特定のファイルを話題にしたいとき**。

---

3. IDE と連携する：`/ide`

VS Code などのエディタと Claude Code を繋ぐコマンド。

```
> /ide
```

連携すると、エディタで**開いているファイルや選択した部分**を自動で Claude が把握してくれて、やり取りが楽になります。

> ⚠️ ただし、開いているファイルが**勝手に AI の記憶に入って**しまい、関係ない情報で混雑することも。あえて連携を切る選択もアリです。

---

4. 思いついたルールを記憶させる：`#`

行の先頭で `#` を打つと、その内容を CLAUDE.md（記憶ノート）に**その場で追記**できます。

`> # ts ファイルに関数を追加したら、必ずテストコードも書くこと`

💡 **いつ使う？** 作業中に「これ毎回守ってほしいな」と思ったルールを、すぐ記憶に焼き付けたいとき。

---

5. お助け操作 & ショートカット集

操作	やり方	いつ使う？
指示を中断する	`Esc` キー	間違った指示を出した／途中で止めたいとき
編集を毎回確認せず自動で許可	`Shift + Tab`	どんどん作業を任せたいとき（acceptEdits モード）
前回の続きから再開	ターミナルで `claude --continue`（短縮 `claude -c`）	昨日の続きをやりたいとき
過去のセッションを選んで再開	`claude --resume` または会話中に `/resume`	いくつかある過去の作業から選んで戻りたいとき
変更を巻き戻す	`/rewind`（または `Esc` を2回）	AI の変更を取り消して、やり直したいとき
過去の指示を検索	`Ctrl + r`	「前に何て指示したっけ？」を探すとき
作業の進捗(ToDo)を表示	`Ctrl + t`	今どこまで終わったか確認したいとき
会話を終了する	`/exit`（または `Ctrl + D`）	その日の作業を終えるとき

📦 **用語ミニ解説：acceptEdits（オートアクセプト）モード** 通常、AI がファイルを編集する前に「やっていい？」と毎回聞いてきます。`Shift + Tab` でこのモードにすると、**いちいち確認せず自動で編集を進めて**くれます。任せて見守りたいときに便利。画面左下に「auto-accept edits on」と出ます。

5-1. 「巻き戻し」`/rewind` の超重要な注意

`/rewind` は、Claude が行ったファイルの作成・編集を**チェックポイント（保存地点）まで丸ごと元に戻せる**強力な機能です（バージョン2.0で追加）。「うわ、変な方向に進んだ」というとき安心して戻せます。

> ⚠️ ただし戻せるのは **Claude が変更したものだけ**。あなたが手で直したファイルや、自分でコマンドを打って変えたものは戻せません。

---

6. AI にじっくり考えさせる：拡張思考モード（ultrathink）

難しい問題のとき、指示の中に**合言葉**を入れると、AI が普段より長く・深く考えてくれます。考える深さは合言葉で段階的に変わります。

📦 **用語ミニ解説：拡張思考モード（Extended Thinking）** AI が答える前に、頭の中で多角的に検討する「熟考モード」。精度が上がる代わりに、少し時間と費用（トークン）を使います。

深さ	日本語の合言葉	英語の合言葉（代表）
浅め	考えて	think
中くらい	よく考えて／もっと考えて	think hard／megathink
最大	熟考／深く考えて／しっかり考えて	**ultrathink**／think harder

使い方の例：

```
> このバグが直らないので熟考してください。
> データベースに項目を追加して認証も実装したいです。ultrathink
```

💡 `Tab` キーを押すと「常に深く考えるモード」をオン/オフできます（画面右下に「Thinking on」と表示）。ただし**トークンを多く消費する**ので、必要なときだけに。

---

7. Claude Code を最新版にする：`claude update`

ターミナルで実行：`claude update`

> ⚠️もし「**Volta**」という Node.js 管理ツールを使っていると、「更新成功」と表示されても**実際にはバージョンが上がらない**ことがあります。覚えておくと混乱しません。

---

8. 【最重要スキル】AI の「記憶」を管理する

ここは Claude Code を使いこなすうえで**いちばん大事**な考え方です。

8-1. コンテキストウィンドウとは

📦 **用語ミニ解説：コンテキストウィンドウ** AI が一度に覚えていられる「作業メモリ（短期記憶）」の量。会話のやり取りも、読んだファイルも、全部ここに溜まっていきます。Claude は**最大20万トークン（約20万文字相当のかたまり）**まで覚えられます。

ここがいっぱいに近づくと、AI は**「何が大事か」を見失って答えの質が落ちます**。人間も、情報を詰め込みすぎると混乱するのと同じです。だから、記憶を**こまめに整理する**のが上達のカギ。

8-2. 記憶の中身を見る：`/context`

```
> /context
```

今、記憶が何にどれくらい使われているか（システム指示・CLAUDE.md・会話・MCPツールなど）を、マス目のグラフで見せてくれます。「最近重いな」と思ったら確認しましょう。

8-3. 記憶をまっさらにする：`/clear`

```
> /clear
```

これまでの会話をリセットして、新しい話題に集中させます。

> 💡 **判断のコツ**：Claude と話していて「**ところで…**」と話題を変えたくなったら、それが `/clear` のタイミングです。前の話題を引きずると精度が落ちるので、思い切ってリセット。（消しすぎても `/resume` で復元できるので安心。）

8-4. 記憶を要約して圧縮する：`/compact`

```
> /compact
> /compact API の仕様についてフォーカスしてください
```

`/clear` が「全部忘れる」なのに対し、`/compact` は**これまでの会話を要約して圧縮**します（大事なことは覚えたまま、かさだけ減らす）。フォーカスしたいテーマを添えると、その情報だけ残せます。

> ℹ️ 記憶が95%を超えると Claude は**自動で**圧縮します。画面に「Context left until auto-compact: 40%」のように、圧縮までの余裕が表示されます。

`/clear` と `/compact` の使い分け

	`/clear`	`/compact`
何をする	全部忘れる	要約して覚える
使う場面	話題がガラッと変わるとき	同じ作業を続けるが記憶が重くなったとき

---

9. 時間のかかる処理を裏で動かす：バックグラウンド実行

「アプリを起動して動作確認しながら、同時に指示も続けたい」というとき。

- Claude が `npm run dev`（アプリ起動）などを実行すると、画面左下に「**1 bash running**」と出て、**裏で動かしながら会話を続けられます**。
- 裏で動いている処理の一覧・停止は `/bashes` で操作（↑↓で選択、Enter で詳細、`k` で停止）。

💡 手動で裏に送りたいときは、起動時に環境変数を付けると `Ctrl + b` で送れます。ターミナルで実行：`ENABLE_BACKGROUND_TASKS=1 claude`

---

10. 別のフォルダも見せる：`/add-dir`

「バックエンドの作業中だけど、別フォルダにあるフロントエンドのコードも見てほしい」というとき。

```
> /add-dir
```

と打って、見せたいフォルダのパス（例：`../front`）を指定します。

> ⚠️ これは**そのセッション中だけ**有効で、保存はされません（毎回指定が必要）。

まとめ

- 最初に `/init`、ファイルを見せるなら `@`、ルール追記は `#`。
- 困ったら `Esc`（中断）・`/rewind`（巻き戻し）・`claude -c`（続きから）。
- 難問は **ultrathink**。
- そして最重要は**記憶の管理**：`/context` で見て、話題が変わったら `/clear`、重くなったら `/compact`。

次は、AI の「記憶ノート」CLAUDE.md の書き方と、安全のための「権限（パーミッション）」を学びます。

---

05. CLAUDE.md とパーミッション — AI に「記憶」と「権限」を与える

この章でわかること

- Claude Code に永続的な「記憶」を持たせる **CLAUDE.md** の書き方
- AI に「どこまで操作を許すか」を決める **パーミッション（実行許可）** の仕組み
- 慎重モードから全自動モードまで、**4つの動作モード**の使い分け
- 設定を「自分だけ」「プロジェクト全体」「全プロジェクト共通」のどこに置くか

> この2つ（記憶と権限）は、Claude Code を“賢く・安全に”使うための土台です。ここを押さえると、毎回同じ説明を繰り返す手間が消え、事故も防げます。

---

1. CLAUDE.md — AI の「記憶ノート」

1-1. CLAUDE.md とは

Claude Code は基本的に**会話が終わると内容を忘れます**（人間でいう短期記憶）。そこで、「毎回必ず読んでほしいこと」を書いておくノートが **CLAUDE.md** です。

📦 **用語ミニ解説：CLAUDE.md（クロード・マークダウン）** プロジェクトのフォルダに置く、ただのテキストファイル（拡張子 `.md` は「Markdown＝見出しや箇条書きを書ける軽量フォーマット」の意味）。Claude Code は**作業のたびに毎回このファイルを自動で読み込みます**。つまり「AI に常に覚えておいてほしいルール・前提」を書く場所です。

ここに「このアプリは何か」「使う技術」「守ってほしいルール」を書いておくと、Claude が毎回それを踏まえて動いてくれます。結果として、

- 毎回同じ説明をしなくて済む
- AI の脱線・暴走が減る
- ムダな読み込みが減り、**処理が速く・安く**なる

1-2. まずは `/init` で作る

既存のプロジェクトなら、対話モードで次を打つだけで CLAUDE.md の下書きが自動生成されます。

```
> /init
```

Claude がフォルダの中身を調べ、「このプロジェクトはこういう構成です」という叩き台を書いてくれます。あとはそこに、あなたのルールを足していきます。

1-3. CLAUDE.md に書くと良いこと（テンプレート）

最初は、次のような項目をうめるイメージで十分です。**コピペして使える雛形**を載せておきます。

```
# このプロジェクトについて
（例）社内の営業日報を管理するWebアプリ。営業が日報を書き、上長がコメントする。

# 使用技術
- 言語: TypeScript
- フレームワーク: Next.js（App Router）
- UI: Tailwind CSS + shadcn/ui
- データベース: Prisma +（MongoDB / PostgreSQL など）
```

```
- テスト: Vitest
- デプロイ先: Vercel もしくは Google Cloud Run

# 守ってほしいルール
- 変数や関数の名前は意味の分かる英語にする
- ファイルを編集したら、必ずテストも更新する
- 勝手に別のライブラリに置き換えない（上の技術で実装する）

# 詳細仕様（別ファイル参照）
@doc/SCREEN_DESIGN.md
@doc/API_SCHEME.md
@doc/TEST_DEFINITION.md
```

💡 技術スタックを明記しておくと、AI が勝手に「得意な別の技術」で書き始めるのを防げます（Claude は指定しないと、よく使われる人気ライブラリに置き換えてしまうことがあります）。

1-4. 大きくなったら「分ける」

CLAUDE.md は**毎回まるごと読み込まれる**ので、長すぎると逆効果になります。実際、ファイルが大きすぎると起動時に「⚠ Large CLAUDE.md will impact performance」（CLAUDE.md が大きすぎて性能に影響します）という警告が出ます。

対策は、**フォルダごとに CLAUDE.md を分ける**こと。

```
プロジェクト/
├── CLAUDE.md            ← 全体の共通ルール
├── frontend/
│   └── CLAUDE.md        ← 画面まわりのルール
└── backend/
    └── CLAUDE.md        ← サーバー・データベースのルール
```

Claude Code は、作業しているフォルダに応じて、近い場所の CLAUDE.md も読み込んでくれます。

💡 「自分が動かすすべてのプロジェクトに共通で効かせたいルール」（例：テストの書き方の方針）は、ホームフォルダの `~/.claude/CLAUDE.md` に書きます。これは全プロジェクト共通の“座右の銘”になります。

1-5. 他のファイルを参照させる `@`

CLAUDE.md の中で `@ファイル名` と書くと、そのファイルの中身も一緒に読み込ませられます。

```
詳しい画面仕様はこちら @doc/SCREEN_DESIGN.md
データベースの構造はこちら @doc/ER_DIAGRAM.md
```

⚠️ **よくある失敗**：`@doc/SCREEN_DESIGN.md` を `` `@doc/SCREEN_DESIGN.md` `` のようにバッククォート（`` ` ``）で囲むと、ただの文字列とみなされて読み込まれません。参照させたいときは、バッククォートで囲まないこと。

⚠️ もうひとつ。Claude Code に CLAUDE.md を**自動生成させると、`@` 参照を付けてくれないことが多い**です。「この仕様書も毎回読んでほしい」なら、自分で `@ファイル名` を書き足してください。

1-6. その場でメモを足す `#`

対話の途中で「これ、覚えておいて」と思ったら、行頭に `#` を付けて打つと、その内容を CLAUDE.md に追記できます。

```
> # TypeScriptの関数を追加したら、必ずVitestでテストも書く
```

どの CLAUDE.md に追記するか選ぶ画面が出るので、選べば完了。会話しながら少しずつ“しつけ”ていけます。

---

2. パーミッション ― AI の「できること」を決める

2-1. なぜ許可が必要なのか

Claude Code は、ファイルを書き換えたり、パソコンに命令（コマンド）を実行したりできます。とても便利な反面、**勝手に何でもやられたら危険**です。そこで Claude Code は、

> **「読むだけ」は自由。だが「書き換え・実行」は、あなたの許可を取ってから。**

という安全設計になっています。

📦 **用語ミニ解説：パーミッション（permission＝許可）**　「AI にどの操作を許すか」のルールのこと。初めて使う種類の操作のたびに「やっていい？」と聞かれ、答えが記録されていきます。

2-2. 許可を聞かれたときの選択肢

たとえば AI が初めて `mv`（ファイル移動）コマンドを使おうとすると、こう聞かれます。

```
1. Yes                                          ← 今回だけ許可
2. Yes, and don't ask again for mv ...          ← 今後この種類は聞かない
3. No, and tell Claude what to do ...           ← 許可せず、別の方法を指示
```

`2` を選ぶと、その許可が `.claude/settings.local.json` というファイルに記録され、次回から聞かれなくなります。

📦 **用語ミニ解説：** `.claude/settings.local.json` あなた個人の許可設定を記録するファイル。`local` が付くものは**自分専用**で、Git（共有の仕組み）には載りません。だから、ここに何を許可しても他の人には影響しません。

2-3. 設定を直接書く（許可リストの書き方）

`/permissions` と打つと、許可・拒否を一覧で管理できます。ルールは「ツール名（対象）」の形で書きます。

書き方	意味
`Bash(npm:*)`	`npm` で始まるコマンドを全部許可
`Bash(npm run:*)`	`npm run` で始まるコマンドを許可
`Bash(npm install)`	このコマンドだけ許可
`Read(./config.json)`	このファイルの読み取りを対象に
`WebFetch(domain:example.com)`	このサイトの取得を対象に

「`*`」は「なんでも」を表すワイルドカードです。

許可（allow）と拒否（deny）の例：

```
{
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test:*)",
      "Bash(git:*)"
  ],
```

```
    "deny": [
      "Bash(rm:*)",
      "Read(./.env)",
      "Read(./.env.*)"
    ]
  }
}
```

上の例は「テストやGit操作は許可、でも**削除コマンド（`rm`）や秘密情報ファイル（`.env`）は禁止**」という設定です。

2-4. どの操作に許可が要る？（ツール早見）

ツール	何をする	許可が必要？
Read	ファイルを読む	不要
Grep / Glob / LS	検索・一覧表示	不要
Edit / MultiEdit	ファイルを編集	**必要**
Write	ファイルを新規作成・上書き	**必要**
Bash	コマンドを実行	**必要**
WebFetch / WebSearch	ネットから取得・検索	**必要**

⚠️ **例外的に厳しいもの**：`curl` や `wget`（ネットから何かをダウンロードして実行できる危険なコマンド）は、たとえ許可リストに入れていても、**怪しい場合は毎回確認されます**。これは安全のためのわざとの設計です。

---

3. 4つの動作モード — 慎重 ↔ 全自動

Claude Code には、「どこまで自分の判断で進めてよいか」を切り替える4つのモードがあります。

モード	ひとことで	こんな時
default（標準）	操作のたびに許可を聞く	慎重に進めたい。初心者はまずこれ
acceptEdits（編集自動承認）	ファイル編集は確認せず実行	どんどん作ってほしい時。`Shift+Tab` でON
plan（計画のみ）	提案だけ。ファイルは触らない	「まず方針だけ聞きたい」時
bypassPermissions（全許可）	一切確認しない	上級者向け・危険

plan モードの使いどころ

「いきなり作り始めず、まず計画を見せてほしい」ときに便利です。計画が出たあと、「この方針で作って」と進められます。

💡 コツ：開発の序盤は plan モードより、**acceptEdits のまま「実行計画をまず Markdown に書いて」**と頼むことも多いそうです。計画をファイルに残すと、あとから確認・中断・再開がしやすくなるからです。

⚠️ bypassPermissions は要注意

起動時に `claude --dangerously-skip-permissions` と付けると、**すべての確認を飛ばして全自動**になります。サクサク動いて快感ですが、

- ごくまれに `rm`（削除）でフォルダごと消す等の事故が起きうる
- **このモードでは `deny`（禁止）設定すら無視されます**

そのため、使うなら**インターネットに繋がっていない Docker などの隔離環境**で、と強く推奨されています（詳しくは10章 セキュリティ）。

📦 **用語ミニ解説：Docker（ドッカー）／隔離環境** パソコンの中に「もう一つの小さな箱（仮想環境）」を作り、その中だけで作業させる仕組み。中で何が起きても本体のファイルには影響しないので、AI に全権限を渡しても安全、という考え方です。

---

4. 設定をどこに置くか — 3つの場所

同じ設定でも、置く場所で「効く範囲」が変わります。

場所	効く範囲	用途
`~/.claude/`（ユーザールート）	**自分の全プロジェクト**	個人共通のルール・好み
`プロジェクト/.claude/settings.json`	そのプロジェクト全員	チームで共有したい設定（Git管理される）
`プロジェクト/.claude/settings.local.json`	**自分だけ・そのプロジェクト**	個人の許可設定（Git管理されない）

📦 **用語ミニ解説：ユーザールート と プロジェクトルート** 「ユーザールート」は、あなたのパソコンのホームフォルダ（`~` で表す、例：`/Users/あなたの名前`）。ここの `.claude` 設定は**全プロジェクトに効きます**。「プロジェクトルート」は、いま作っているアプリのフォルダ。ここの `.claude` 設定は**そのプロジェクトだけ**に効きます。

> 使い分けの基本：「どのプロジェクトでも効かせたい好み」はユーザールートへ、「このアプリだけのルール」はプロジェクトルートへ。チームと共有したい設定は `settings.json`、自分専用は `settings.local.json`。

（チーム共有のさらに詳しい優先順位は11章で扱います。）

まとめ

- **CLAUDE.md** は AI の“記憶ノート”。`/init` で作り、技術・ルール・仕様を書くと、毎回それを踏まえて動いてくれる。大きくなったらフォルダ単位で分ける。
- **パーミッション** で「読むのは自由、書き換え・実行は許可制」。`deny` で危険な操作を禁止できる。
- **4つのモード**で慎重さを調整。初心者は default、慣れたら acceptEdits。bypassPermissions は隔離環境でだけ。
- 設定は「自分の全プロジェクト＝ユーザールート」「このプロジェクト＝プロジェクトルート」「自分だけ＝local」で置き分ける。

次は、いよいよ手を動かします。**5分でアプリを作って、ネットに公開**してみましょう。

---

06. はじめてのアプリ開発 — 5分でアプリを作り、ネットに公開する

この章でわかること

- 実際に手を動かして、**TODOアプリ**を作る
- 作ったアプリを自分のパソコンで動かす（ローカル確認）
- アプリを**インターネットに公開**して、人に使ってもらう（デプロイ）
- うまく表示されないときの **デバッグ（不具合直し）の超実践テク**
- AI に **テストコード**を書いてもらい、品質を守る

> この章はハンズオン（実際にやってみる）です。一緒に手を動かしながら読んでください。「5分でアプリを作り、30分で公開まで」を体験する内容です。

---

1. 準備運動：作業フォルダを用意する

まず、アプリを入れる「箱（フォルダ）」を作ります。ターミナルで次の3行を順に実行します。

```
mkdir todo
cd todo
git init
```

📦 用語ミニ解説：この3行は何？

- `mkdir todo` … `todo` という名前のフォルダを作る（make directory）
- `cd todo` … そのフォルダの中に移動する（change directory）
- `git init` … このフォルダで「変更履歴の記録」を始める。後で公開するときに必要です

📦 用語ミニ解説：リポジトリ `git init` をしたフォルダのこと。「このアプリのソースコード一式と、その変更履歴をまとめた入れ物」だと思ってください。

準備ができたら、フォルダの中で Claude Code を起動します。

```
claude
```

---

2. ファーストプロンプト：作りたいものを伝えるだけ

対話モードに入ったら、作りたいものを日本語で伝えます。

```
> TODOアプリを作ってください。ultrathink
```

たったこれだけです。末尾の `ultrathink`（じっくり考えて、の合図）を付けると、最初の設計段階で AI がより丁寧に全体像を考えてくれます。

すると Claude は、

1. 「TODOアプリに必要な機能（追加・削除・完了切替・一覧・保存）」を自分で洗い出し、
2. 「シンプルに始めるなら HTML/CSS/JavaScript で」と技術を選び、
3. 自分で「やることリスト」を立てて、
4. ファイルを次々と作成

――という流れを、**指示なしでも自動で**進めてくれます。ある例では、最初の指示から**約30秒**で動くTODOアプリが返ってきました。

💡 細かく指示しなくても、Claude は自分で計画を立てて作ります。最初は「作って」と頼むだけで十分です。

---

3. 自分のパソコンで動かす（ローカル確認）

ファイルができても、それを「見られる形」で立ち上げる作業は、自分でやる必要があります。これを**ローカルサーバーの起動**といいます。

📦 **用語ミニ解説：ローカルサーバー** 作ったアプリを、**自分のパソコンの中だけ**でブラウザから見られるようにする仕組み。「公開」ではなく「下見」です。

やり方は2つ。

- **VS Code を使っているなら**：拡張機能「**Live Server**」が便利。HTMLファイルを右クリック →「Open with Live Server」で、自動で立ち上がります。
- **それ以外**：ターミナルで実行：`npx http-server`（または `npx serve`）

立ち上がったら、ブラウザで `http://localhost:8000`（表示された番号）を開きます。

✅ タスクを追加・削除でき、フィルタ（すべて／未完了／完了）が動けば成功です。

📦 **用語ミニ解説：localhost（ローカルホスト）** 「このパソコン自身」を指す住所のようなもの。`localhost:8000` は「自分のパソコンの8000番窓口で動いているアプリ」という意味です。

---

4. インターネットに公開する（デプロイ）

下見ができたら、世界に公開しましょう。これを**デプロイ**といいます。

📦 **用語ミニ解説：デプロイ** 作ったアプリをインターネット上のサーバーに置き、**誰でもアクセスできる状態にする**こと。「お店を開店する」イメージです。

方法A：GitHub Pages（公開リポジトリなら無料）

まず、コードを GitHub に上げます。Claude にこう頼みます。

```
> ghコマンドを使って、リモートリポジトリを作成し、コミットしてプッシュしてください。
```

💡 「**ghコマンドを使って**」と明記するのがコツ。これがないと、AI が「gh が使えること」に気づかず遠回りすることがあります（`gh` の準備は03章参照）。

GitHub にコードが上がったら、続けて：

```
> GitHub Pages にデプロイしてください。
```

しばらく待つと、`https://（あなたのID）.github.io/todo/` のような**公開URL**が発行されます。これで完了です。

方法B：Vercel（プライベートでも無料）

GitHub Pages は「公開リポジトリ（誰でも中身が見える）」でないと無料で使えません。**人に見られたくないコードを隠したまま公開したい**なら **Vercel** が便利です。

📦 **用語ミニ解説：Vercel（ヴァーセル）／PaaS** Vercel は、難しいインフラ設定なしでアプリを公開できるサービス。こうしたサービスを **PaaS（パース：Platform as a Service）** と呼びます。本来サーバー公開には専門知識が要りますが、PaaS なら「アプリを渡すだけ」で、面倒な部分を肩代わりしてくれます。

手順：

1. Vercel に登録（最初は「個人利用（Hobby）」でOK）。**GitHubアカウントで連携**しておくと一番スムーズです。
2. Claude にこう頼むだけ：

`> Vercelにデプロイしたいです。`

必要なツールが入っていなければ、Claude が自分でインストールしてからデプロイし、公開URLを教えてくれます。

---

5. 表示が崩れたら：デバッグの超実践テク

AI が作ったアプリは、環境やバージョンの違いで**最初は見た目が崩れる**ことがよくあります。ここからが本番。直し方を覚えれば、もう怖くありません。

📦 **用語ミニ解説：デバッグ** 不具合（バグ）を見つけて直すこと。「虫（bug）を取り除く（de-）」が語源です。

テク① 画面のスクリーンショットを見せる

「言葉で説明しづらい見た目の崩れ」は、**画面を撮って AI に見せる**のが最速です。

- 範囲を撮ってコピー：**Windows** は `Windows + Shift + S`、**Mac** は `Command + Shift + Control + 4`
- VS Code などの入力欄に貼り付ける（`Ctrl + V`）と、画像が `[Image #1]` のように添付されます
- そえて指示：

`> [Image #1] CSSが効いていないようです。確認してもらえますか？`

テク② エラーメッセージをそのまま貼る

赤い文字のエラーが出たら、**そのまま全部コピーして貼り付ける**のが鉄則。理解できなくて大丈夫です。Claude が読み解きます。

`> npm run dev で起動するとこのエラーが出ます。[ここにエラーを貼り付け]`

💡 長文を貼ると、Claude Code は `[Pasted text #1 +21 lines]`（21行のテキストを格納）のように**自動でたたんで**くれます。画面が文字で埋まらないので安心して貼ってOK。

テク③ 「直るまで続けて」と自走させる

1回で直らないこと、別のエラーを誘発することもあります。そんな時はこう頼みます。

`> エラーが出なくなるまで、調査して修正し続けてください。`

あとは待つだけ。Claude が原因を探りながら、何度でも直し続けてくれます。

⚠️ **大事な心構え**：エラーは失敗ではなく「**会話のきっかけ**」です。プロのエンジニアでも、エラーを見ながら何度も直します。あなたは「見せて、貼って、頼む」だけでいいのです。

---

6. テストコードで品質を守る

アプリが動いたら、次は「**ちゃんと動き続ける保証**」をつけます。それが**テストコード**です。

📦 **用語ミニ解説：テストコード** アプリの機能が正しく動くかを、自動でチェックする小さなプログラム。「ボタンを押したらタスクが増えるか？」などを機械的に確認してくれます。一度書いておくと、後で改造しても壊れていないか即座に分かります。

頼み方はシンプル：

```
> このアプリのテストコードを Vitest を使って書いてください。
```

📦 **用語ミニ解説：Vitest（ヴィテスト）** JavaScript/TypeScript でよく使われるテスト用の道具。Claude が得意なツールのひとつです。

⚠️ AI のテストには「ズル」がある

ここは非常に重要です。AI は、**意味のないテストを書いて「合格」にしてしまう**ことがあります。

- `expect(true).toBe(true)`（「真は真」＝何も検証していない）
- テストを通すためだけに、答えを直接コードに埋め込む（ハードコード）

ひどい場合は、**コードのバグを直す代わりに、正しいテストの方を書き換えて無理やり合格にする**ことすらあります。

対策：CLAUDE.md にテストのルールを書いておく

これを防ぐには、05章で習った **CLAUDE.md にルールを書いておく**のが効きます。全プロジェクトで効かせたいので、`~/.claude/CLAUDE.md`（ユーザールート）に書くのがおすすめ。**そのまま使えるテンプレート**を載せます。

```
# テストコード作成時の厳守事項
## 絶対に守ってください！

### テストコードの品質
- テストは必ず実際の機能を検証すること
```

```
- `expect(true).toBe(true)` のような意味のないテストは絶対に書かない
- 各テストは「具体的な入力」と「期待される出力」を検証すること

### ハードコーディングの禁止
- テストを通すためだけに答えを埋め込まない
- 本番コードに`if (testMode)` のようなテスト用の分岐を入れない

### テスト実装の原則
- 失敗する状態から始めること（まず赤→直して緑）
- 境界値・異常系・エラーケースもテストすること
- テスト名は「何をテストしているか」が分かるように書くこと

### 実装前の確認
- 仕様を正しく理解してからテストを書くこと
- 不明な点は、仮で進めず必ずユーザーに確認すること
```

このルールを入れておくと、Claude は一言頼むだけで、**意味のあるテストを自分でエラーを直しながら**書いてくれます。ある例では、40件のテストが全て成功し、コードの99%がテストでカバーされました。

まとめ

- 作りたいものは**日本語で伝えるだけ**。Claude が計画を立てて作ってくれる。
- 自分のPCで下見（ローカルサーバー）→ 世界に公開（デプロイ：GitHub Pages か Vercel）。
- 崩れたら **①画面を見せる ②エラーを貼る ③直るまで続けてと頼む**。エラーは会話のきっかけ。
- **テストコード**で品質を守る。ただし AI の“ズル”を防ぐため、CLAUDE.md にルールを書いておく。

ここまでで「ひとりで小さなアプリを作って公開する」流れが一周しました。次は、AI の能力そのものを拡張する **MCP** と、もう少し本格的なアプリ作りに進みます。

---

07. MCPサーバーとAIチャットボット開発

この章でわかること

- Claude Code の能力を“拡張”する **MCPサーバー** とは何か
- 必ず入れておきたいおすすめMCP（Context7・Playwright・Serena）
- 1回の指示で終わらない**本格的なアプリ**（AIチャットボット）を、要件定義 → 実装 → ネット公開まで進める流れ
- エラーを Claude に**自分で直させ続ける**コツ

> 06章では「1回お願いすれば完成する」小さなアプリを作りました。この章ではもう一段上、**データを保存する“裏側”を持つアプリ**に挑戦します。難しく感じたら、まずは「こういう流れで作るんだ」と雰囲気をつかむだけでOKです。

---

1. MCPサーバー —— Claude に“専門の道具”を持たせる

MCPサーバーとは

📦 **用語ミニ解説：MCPサーバー（Model Context Protocol サーバー）** Claude Code と、外部のサービスやツールを**つなぐ“仲介役”のソフト**です。MCP は「Model Context Protocol（モデル・コンテキスト・プロトコル）」の略で、AI と道具をつなぐための**共通の通信規格**のこと。これを入れると、Claude が「最新の説明書を読む」「ブラウザを操作する」「コードを深く理解する」といった、**標準ではできない芸当**をできるようになります。

イメージで言うと、Claude Code 本体が「優秀な職人」だとすると、MCPサーバーは「職人に渡す専用工具」です。工具を増やすほど、できる仕事の幅が広がります。

MCPサーバーの入れ方（共通）

MCPサーバーは**対話モードの中ではなく、ターミナルで**追加します。基本の形はこれだけです。

ターミナルで実行：`claude mcp add <名前> <起動コマンド>`

入っているか確認したいときは：

ターミナルで実行：`claude mcp list`

対話モードの中で接続状態を見たいときは `> /mcp` と打ちます（`✓ connected` と出れば成功）。

💡 **スコープ（どこで使えるか）を選べる**：`-s` を付けると適用範囲を指定できます。

- `local`（標準）… 今いるフォルダだけ
- `project` … チームで共有（`.mcp.json` に記録され、Git で全員に配られる）
- `user` … 自分の全プロジェクトで使える（`~/.claude` に保存）

よく使うものは `-s user` を付けて入れておくと、毎回どのプロジェクトでも使えて便利です。

---

2. まず入れたい おすすめMCP 3選

① Context7 ── 最新の“説明書”を読ませる

AI は「学習した時点まで」の知識しか持っていません。だから**新しいライブラリ（部品）の最新の使い方を知らず**、古いやり方で書いて無駄なエラーを出すことがあります。**Context7** は、ライブラリの**最新ドキュメントをその場で読み込ませる**MCPです。実務でも定番の存在です。

ターミナルで実行：`claude mcp add context7 -s user -- npx -y @upstash/context7-mcp`

使い方は簡単で、**指示の最後に `use context7` と付け足すだけ**。これがおまじないのキーワードです。

> TODOアプリを Next.js の App Router で作ってください。データはローカルに保存してください。use context7

💡 Next.js、Prisma、Vitest など主要な部品で効果絶大。「なんか古い書き方でエラーになるな」と感じたら `use context7` を付けるクセをつけましょう。

② Playwright ── Claude に“目”を持たせてブラウザを操作させる

📦 **用語ミニ解説：Playwright（プレイライト）** 本来は「ブラウザを自動操作して動作テストをする」ためのツール。これを MCP としてつなぐと、**Claude が実際にブラウザを開いて、ボタンを押したり画面の崩れやエラーを“見て”確認**できるようになります。Claude は普段、画面を見ずに“勘で”フロント側を書いている部分があるので、これで答え合わせができます。

ターミナルで実行：`claude mcp add Playwright npx @Playwright/mcp@latest`

> ⚠️ **必ず「Playwright の MCP サーバーを使って」と明記すること。** ただ「Playwright で確認して」と言うと、Claude は“目で見る”代わりに**テスト用のファイルを書き始めてしまう**ことがよくあります。

使用例：

> /login のページでログインできません。Playwright の MCP サーバーを使って、実際にログインできるまでエラーを調べて修正し続けてください。

このように「**できるまで直し続けて**」と頼めば、あとは待つだけで自走してくれます。

③ Serena ── コードを“言語として”深く理解させる

Serena は、IDE（開発ソフト）が持っている「プログラムの文法・構造を理解する仕組み（LSP）」を Claude に渡し、**コードの理解精度を高める**MCPです。

導入はやや手間で、`uv`（Python の道具を動かすコマンド）が必要です。

ターミナルで実行：`curl -LsSf https://astral.sh/uv/install.sh | sh` ターミナルで実行：`claude mcp add serena -- uvx --from git+https://github.com/oraios/serena serena-mcp-server --context ide-assistant --project $(pwd)`

💡 Context7 と違い、**一度つなげば自動で連携**するので、毎回呼び出す必要はありません。`> /mcp` で `✓ connected` を確認しましょう。プロジェクトごとに入れる必要がある点だけ注意。

---

3. 本格アプリを作る ── 「AIチャットボット」を例に

ここからは「訪れた人とおしゃべりできる AIチャットボット」を作る流れを追います。06章の TODO と違い、**会話の履歴を保存する“裏側”が必要**です。

📦 用語ミニ解説：バックエンドとデータベース（DB）

- **バックエンド** … 画面の裏で動く処理（データの保存・取り出し・計算など）。表に見える画面側は「フロントエンド」と呼びます。
- **データベース（DB）** … データを保存しておく“倉庫”。今回は **MongoDB（モンゴDB）** という種類を、無料で使える **Atlas** というサービスで利用します。

ステップ1：要件定義を Claude に“質問させて”作る

いきなり「作って」ではなく、**まず仕様（どんなアプリにするか）を固める**のが本格開発のコツ。しかも、それを Claude 自身に手伝ってもらえます。

> AIチャットボットを作りたいです。仕様書をあなたと一緒に作りたいので、必要な情報は質問してもらえますか？ 最後に CLAUDE.md ファイルとして出力するのが目的です。

すると Claude が「目的は？ 想定ユーザーは？ 使う技術は？ 認証は必要？」と**逆に質問してくれます**。それに答えていくと、**2往復くらいで仕様書（CLAUDE.md）が完成**します。

💡 これが本格開発の心臓部です。AI の強みは「人間の思いつき(1)を、抜け漏れのない仕様(10)に膨らませる」こと。最初に仕様を固めるほど、後の完成度が上がります。

ステップ2：作業を TODO リストに分割させる

仕様が決まったら、いきなり全部作らせず、**やることリスト（TODO.md）に分解**させます。

> 現在のアプリケーションを構築するための実行計画を立ててください。それを TODO リストとして Markdown ファイルに落としてください。

なぜ分割するのか —— 一度に全部やらせると、Claude は作業が進むほど覚えることが増え、**頭がいっぱいになって精度が落ちる**からです（人間と同じですね）。リストにしておけば、一つずつ確実に、進捗を見ながら進められます。

実装を頼むときは、こう指示します：

> @TODO.md これを元にフェーズ1の実装をお願いします。終わったらチェックボックスにチェックを入れてください。use context7

> ⚠️ **「終わったらチェックを入れて」を必ず付ける。** これを忘れると、Claude は1項目だけ実装して終わってしまい、**どこまで進んだか記録が残りません**。別の日に再開したとき、続きが分からなくなります。

💡 小さなアプリなら一気にお願いしてもOK。ただし画面・裏側・テスト・公開まで広く絡む本格アプリは、**段階的に進める方が確実**です。

ステップ3：開発を楽にする「便利コマンド集（Makefile）」を作らせる

開発では「起動」「ビルド」「公開」など同じ操作を何度もします。これを一覧（Makefile）にまとめさせると楽です。

> 初期化・開発サーバー起動・ビルド・デプロイ用のコマンドを Makefile にまとめてください。

💡 作った便利コマンドの存在は、別の日に Claude が忘れがちです。**CLAUDE.md に「これらのコマンドがある」と書いておく**と、次回も思い出してくれます。

ステップ4：よくあるバグを直す（日本語入力の誤送信）

AI が作ったチャット入力欄では、**日本語の漢字変換を確定する Enter で、文章が途中送信されてしまう**バグが定番です。こう頼めば直ります：

> 日本語で漢字を変換しようとして Enter すると、誤って投稿されてしまいます。変換の確定時には送信されないように修正してください。

---

4. インターネットに公開する（Google Cloud へデプロイ）

📦 用語ミニ解説：デプロイ／Cloud Run

- **デプロイ** … 作ったアプリを、自分のパソコンの外（インターネット上）に置いて、誰でも使えるようにすること。
- **Cloud Run（クラウドラン）** … Google のサービスで、アプリを手軽に公開できる置き場所。今回はここを使います。

お金の話（まず安心してください）

- 初回登録で **90日間 $300 ぶんの無料クレジット**がもらえます。
- クレジットが切れても、**「最小インスタンス数を0」**にしておけば、使われていない間は課金されません（従量課金）。ある例では、実費は**月0.02ドル程度**でした。

> ⚠️ Claude はアプリ規模に応じて勝手にインスタンス数を増やすことがあります。**「Cloud Run の最小インスタンス数は 0 にしてください」と明示**しましょう。

人間が1回だけやること

Claude にクラウドを操作させる前に、**あなた自身が1回ログイン**しておく必要があります。

ターミナルで実行：`gcloud auth login`

これでブラウザが開き、Google アカウントでログインすると、以降は Claude がクラウドを操作できるようになります。あとはプロジェクトID（Google Cloud で作るプロジェクトの識別名）を伝えればOK：

> Google Cloud のプロジェクトIDは `chrome-entropy-469511-j3` です。今のアプリをデプロイしてください。Cloud Run の最小インスタンス数は 0 にしてください。

エラーが出ても「直るまで自走させる」

デプロイは権限まわりでエラーが起きがちです。毎回エラーをコピペするのは大変なので、**Claude に丸ごと任せます**：

> `make deploy` を実行して、デプロイが通るまで問題を修正し続けてください。

Claude が権限設定やコマンドを直しながら、成功するまで粘ってくれます。あなたは待つだけ。

💡 **本番だけで起きるエラー**は、Claude にログを直接見させると解決が早いです：

> Playwright の MCP サーバーで本番URLを開き、ログを確認してエラーを特定し、投稿できるまで修正し続けてください。

裏では Claude が `gcloud run services logs read ...` のようなコマンドでクラウドのログを読み、原因（多くは DB の接続設定ミス）を突き止めてくれます。

> ⚠️ **APIキーやパスワードを、会話に安易に貼り付けない。** 環境変数（`.env` ファイル）として安全に渡すのが基本です。Claude にコードを書かせるとき、機密情報はあなたが管理しましょう。詳しくは10章 セキュリティ。

5. 公開後の追加開発を“GitHub から”やる

アプリを公開した後、「画像も送れるようにしたい」などの追加機能は、パソコンを開かなくても**GitHub 上で Claude に頼めます**。

📦 用語ミニ解説：Issue（イシュー）と GitHub Actions

- **Issue** … GitHub 上の「やることメモ／課題チケット」。
- **GitHub Actions** … GitHub のサーバー上で自動処理を動かす仕組み。ここで Claude を動かせます。

セットアップ

対話モードで一度だけ：

```
/install-github-app
```

画面の案内に従ってブラウザで設定すると、GitHub 上で Claude が使えるようになります。

使い方：`@claude` とメンションするだけ

GitHub で Issue（やること）を作り、コメント欄にこう書きます：

```
@claude このIssueを実装してください。必要があればSub Issueに分割してください。
```

すると GitHub のサーバー上で Claude が動き、実装してプルリクエスト（変更の提案）まで作ってくれます。**スマホの GitHub アプリからでも同じことができる**ので、外出先がそのまま開発環境になります。

💡 料金感：GitHub Actions 経由は従量課金で、1回あたり**約1.2ドル**程度（Sonnet 利用時）。難しい作業を Opus でやると1回3〜5ドルになることもあるので、使い分けを。

> ⚠️ GitHub Actions の Claude は「対話できない単発実行」です。結果を見て方向修正しづらいので、**粒度の小さい機能追加・バグ修正・レビュー**に向いています。

まとめ

- **MCPサーバー**は Claude に専用工具を持たせる仕組み。まず **Context7**（`use context7`）を入れるだけで精度が大きく上がる。**Playwright** は「MCPサーバーを使って」と明記して目を持たせる。
- 本格アプリは **仕様を固める → TODOに分割 → 一つずつ実装** が王道。「終わったらチェックを入れて」を忘れずに。
- 公開は Google Cloud で。無料枠を使い、**「通るまで直し続けて」と自走**させ、本番エラーはログを見させる。
- 機密情報（APIキー・パスワード）は会話に貼らない。
- 公開後の追加は GitHub の `@claude` メンションで、スマホからでも。

次は、複数の AI を同時に走らせてもっと大きなシステムを作る方法です。

---

08. 並行処理とサブエージェント

この章でわかること

- AI の“記憶の限界”（コンテキストウィンドウ）と、その上手な管理
- 大きなシステムを脱線させずに作る「設計の固め方」
- 暴走を防ぐ「ガードレール」
- **複数の Claude を同時に走らせて**開発を倍速にする方法
- 「専門家AI（サブエージェント）」の作り方

> この章から“半日で社内システムを作る”レベルの話になります。一気に理解しようとせず、「困ったらここに戻る」リファレンスとして使ってください。

---

1. コンテキストウィンドウ —— AI の“作業机の広さ”

📦 **用語ミニ解説：コンテキストウィンドウ** Claude が一度に覚えていられる情報の量のこと。AI の**短期記憶 兼 作業机**だと思ってください。Claude は最大 **20万トークン**（おおよそ十数万〜数十万

字）まで覚えられますが、会話やコードでこの机が**いっぱいになると、大事なことを忘れて精度が落ちます**。

だから、人間の側で「机を片付ける」管理が必要です。道具は3つ。

コマンド	何をする	いつ使う
`/context`	机の使用状況を**見える化**する	「最近ちょっと変だな」と思ったとき
`/clear`	机を**まっさらにリセット**（前の会話を忘れる）	話題がガラッと変わったとき
`/compact`	これまでの会話を**要約して圧縮**（要点は覚えている）	同じ話題を続けたいが机が重いとき

💡 覚えておきたい目安：「Claude と話していて『**ところで**』と言いたくなったら、それが `/clear` のタイミング」。`/compact` は「フォーカス指定」もでき、
`> /compact APIの仕様についてフォーカスして` のように重要な部分だけ残せます。

💡 `/clear` で消しすぎても、`/resume` でセッションを復元できます。

裏で動かす・フォルダを足す

- **バックグラウンド実行**：`npm run dev`（開発サーバー）などを裏で走らせたまま会話を続けられます。左下に「1 bash running」と出て、`/bashes` で管理（停止は `k`）。
- **作業フォルダの追加**：別フォルダのコードも見てほしいときは `/add-dir`（ただしそのセッション中だけ有効）。

---

2. 大きなシステムを“脱線させず”作る —— 設計を先に固める

小さなアプリは勢いで作れますが、大きなシステムは AI が途中で迷子になります。対策は、**作り始める前に「輪郭」を固める**こと。たとえば「営業日報システム」を例にすると、次の順で設計書を Claude に作らせます。

1. **要件定義**（どんな機能か）
2. **ER図**（データの設計図）
3. **画面定義**（どんな画面があるか）
4. **API仕様**（裏側のやり取りのルール）

5. **テスト仕様**（どう動作確認するか）

📦 **用語ミニ解説：ER図 と Mermaid 記法**

- **ER図** … 「営業」「顧客」「日報」などのデータ同士の関係を表す設計図。
- **Mermaid（マーメイド）記法** … 文章（テキスト）で図を描ける書き方。Markdown に書けるので、**設計図もコードと一緒に管理**できて便利です。「ER図は Mermaid で書いて」と頼みましょう。

これらの設計書（Markdown）ができたら、CLAUDE.md に `@doc/ER図.md` のように**まとめて参照**させます。輪郭が固まっていると、Claude が勝手な機能を足したりせず、設計どおりに作ってくれます。

---

3. ガードレール —— AI が暴走しない“柵”を作る

📦 **用語ミニ解説：ガードレール** AI が変なコードを書かないようにする**自動チェックの仕組み**。道路のガードレールのように、コースから外れそうになったら止めてくれます。

主なガードレールは3つ：

- **型チェック（コンパイルチェック）** … 文法やデータの種類の食い違いを自動検出。📦型・TypeScriptは用語集参照。**型のある言語の方が AI の暴走を抑えやすい**、と言われます。
- **テスト** … 「ちゃんと動くか」を自動で確認。⚠️ただし AI は**コードのバグを直す代わりに、正しいテストの方を書き換えて“通ったことにする”**ことすらあるので注意。
- **Lint（リント）チェック** … 書き方のクセや無駄を統一・警告。代表ツールは **ESLint** や **Biome**。

さらに、これらを**コミット（保存）前に自動実行**させる **Husky**、特定の操作で**自動テスト・自動公開**まで回す **CI/CD（GitHub Actions）** を組むと、安心して任せられます。

💡 鉄則：「**小さい実装サイクルをたくさん回す**方が精度が高い」。実装 → チェック → 公開、を早めに整えて、こまめに回しましょう。

---

4. Issue で「やること」を管理する

大きなシステムは、Claude に **GitHub の Issue（やることチケット）として一覧化**させると管理しやすくなります。

> このシステムに必要な作業を、gh コマンドで Issue として作成してください。

すると「#1 基盤セットアップ」「#2 認証」…のように分割され、`#番号` を指定するだけで「その Issue を実装して」と頼めます。GitHub のラベル（タグ）で「フロント／バック／テスト」と色分けさせると、さらに整理しやすくなります。

---

5. 並行処理 —— 複数の Claude を同時に走らせる

ここが Claude Code（CLI）ならではの真骨頂。**複数の作業を同時並行**で進められます。

📦 **用語ミニ解説：Git Worktree（ギット・ワークツリー）** 同じプロジェクトの**別バージョン（ブランチ）を、別フォルダで同時に開ける**Git の機能。元の作業に影響を与えずに、並行して別作業ができます。

ターミナルで実行：`git worktree add issue-5 -b feature/issue-5`（別フォルダ＋別ブランチを作る） ターミナルで実行：`git worktree list`（一覧を見る） ターミナルで実行：`git worktree remove issue-5`（作業が終わったら片付ける。※ブランチ自体は消えません）

実際の進め方は、VS Code でターミナルを2つに分割し、それぞれで `claude` を起動。片方に「Issue#2 を実装して」、もう片方に「Issue#3 を別ブランチで実装して、最後にプルリクして」と頼めば、**2つの作業が同時に進みます**。

> ⚠️ **並行処理の注意点**
>
> - **別セッションの Claude は“別人格”**。お互いの作業が被ると衝突するので、フロント担当・バック担当のように**領域を分ける**。
> - **IDE連携は一度に1つのターミナルだけ**有効（後から開いた方に移る）。取り戻すには `/ide`。
> - **Max の最上位プランでも、5時間ごとの制限にすぐ到達**します。人間が管理できる範囲も含め、**現実的には同時2個くらい**が目安。

---

6. サブエージェント ―― “その道の専門家AI”を雇う

📦 **用語ミニ解説：サブエージェント** 特定分野に特化した**専門家AI**を自分で作れる機能。「あなたはフロントエンドの専門家です」と役割を与えると、その分野の精度が上がります。しかも**専用の作業机（コンテキスト）**を持つので、本体の記憶を散らかしません。

作り方（対話で簡単に）

対話モードで：

> /agents

画面の案内に従って進めます。ポイントは：

- 作り方は **「Generate with Claude（おすすめ）」**を選ぶと、Claude が中身を整えてくれる。
- **権限**：コードを書かせるなら「All tools」、レビューだけなら「Read-only」。
- **モデル**：「Inherit from parent（親に従う）」が無難です。
- **保存先**：自分専用なら `~/.claude/agents/`（Personal）、チーム共有なら プロジェクト内（Project）。
- **色**：並行で動かすとき、どの専門家が動いているか分かるよう色分けすると便利。

呼び出し方

作った専門家は、**指示文に名前を書くだけ**で呼べます：

> backend-expert-engineer で Issue#4 を実装してください。

名前があやふやでも「**バックエンドのサブエージェントで**実装して」「**この内容に最適なサブエージェントで**実装して」と言えば、Claude が選んでくれます。

> ⚠️ サブエージェントは**本体と記憶を共有せず、毎回まっさらから始まります**。専門に集中できる反面、必要な情報を集め直すぶん少し遅くなることがあります。

まとめ

- AI の作業机（**コンテキスト**）は `/context` で見て、`/clear`（リセット）・`/compact`（要約）で管理する。「ところで、と言いたくなったら `/clear`」。
- 大きなシステムは **要件→ER図→画面→API→テスト** の順で設計を先に固めると脱線しない。
- **ガードレール**（型・テスト・Lint・Husky・CI/CD）で暴走を防ぎ、小さいサイクルを回す。
- **Git Worktree ＋ 複数ターミナル**で並行開発。ただし作業は被らせない、現実的に同時2個。
- **サブエージェント**で専門家AIを作り、`/agents` から名前で呼び出す。

次は、こうした作業を“自動化・パターン化”して、もっと楽にする方法です。

---

09. 作業を自動化する — カスタムコマンド・Hooks・Skills

この章でわかること

- 毎回打つ長い指示を「自分専用のショートカット」にまとめる方法（**カスタムスラッシュコマンド**）
- 「この操作のあとは必ずコレをやって」をAIに**100%守らせる**仕組み（**Hooks**）
- AIに“専門スキル”を覚えさせる新機能（**Agent Skills**）
- AIへの指示が上手くなる考え方（**プロンプトエンジニアリング**）と、作る前に設計を固める**仕様書駆動開発**

> 💡 この章は「便利の上澄み」です。最初から全部使う必要はまったくありません。「同じ長い指示を毎日コピペしてるな」「AIがときどき約束を破るな」と感じたとき、戻ってくれば十分です。まずは“こんな仕組みがある”とだけ知っておきましょう。

---

1. カスタムスラッシュコマンド — 自分専用のショートカットを作る

なぜ必要？

Claude Code を使い込むと、「新しいブランチを作って、実装して、テストして、最後にプルリクエストして」のような**長い決まり文句**を毎回打つことになります。打ち忘れると、AIは平気で別の場所に作業を始めてしまいます。これを**1個の短い合言葉**にまとめられるのがカスタムスラッシュコマンドです。

📦 **用語ミニ解説：スラッシュコマンド** `/` で始まる特別な命令のこと（例：`/init`、`/clear`）。Claude Code に最初から入っているものに加えて、**自分で新しく作る**こともできます。それが「カスタム」スラッシュコマンド。

作り方（コピペ用の置き場所にファイルを置くだけ）

作り方はとてもシンプルで、**指示文を書いた Markdown ファイルを決まったフォルダに置く**だけです。ファイル名がそのままコマンド名になります。

- 自分専用にしたい → `~/.claude/commands/` に置く（あなたの全プロジェクトで使える）
- チームで共有したい → プロジェクト内の `./.claude/commands/` に置く（Gitで共有でき、仲間も同じコマンドを使える）

例：`optimize.md` というファイルを作れば、`/optimize` というコマンドになります。

値を渡せるようにする（引数）

コマンドに「番号」や「対象」を毎回変えて渡したいときは、指示文の中に `$ARGUMENTS` と書いておきます。

📦 **用語ミニ解説：引数（ひきすう）** コマンドに「一緒に渡す値」のこと。たとえば「#14をレビューして」の **14** の部分。コマンド側に `$ARGUMENTS` という“穴”を空けておくと、そこに自分が打った値が差し込まれます。複数渡したいときは `$1`・`$2` を使います。

実例：`/pr-review`（プルリクエストをレビューするコマンド）

`~/.claude/commands/pr-review.md` の中身：

```
プルリクエスト #$ARGUMENTS をレビューしてください。
```

これで `/pr-review 14` と打つと「プルリクエスト #14 をレビューして」に展開されます。

実例：`/implement-issue`（一連の開発をまるごと自動化）

`implement-issue.md` の中身：

```
Issue #$ARGUMENTS を実装してください。実装にあたっては新しいブランチを切り、
実装ができたらコミット。エラーを修正し、最後にプルリクエストをしてください。
```

`/implement-issue 5` と打つだけで、「ブランチ作成 → 実装 → エラー修正 → プルリクエスト」までを一気にやってくれます。毎回長い指示を書かずに済むわけです。

> ⚠️ **3つの注意点**
>
> - 作ったコマンドを有効にするには、一度 `/exit` でセッションを終了して**起動し直す**必要があります。
> - コマンド名が候補に出た状態でそのまま `Enter` すると、**引数なしで実行**されてしまいます。`Tab` でいったん確定してから番号を入力しましょう。
> - 名前の打ち間違いを防ぐため、ファイル名は短く分かりやすく。

一歩進んだ使い方：手順を「番号付き」で書くと精度が上がる

AIは「mainの最新からブランチを切って」のような注意を**さらっと無視する**ことがあります。そういうときは、指示文を**手順書のように番号で区切って**書くと、順番どおりに動いてくれます。

```
Issue #$ARGUMENTS を実装してください。
## 実装手順
### 1. 前処理
- 現在のブランチがmain以外ならmainに切り替える
- 最新のmainを取得する
### 2. 作業用フォルダの準備
- 専用の作業フォルダを作って、そこに移動する
### 3. 実装
-  Issueの内容を確認して実装し、必ずテストを実行する
### 4. プルリクエスト作成
- 変更をコミット・アップロードし、プルリクエストを作る
### 5. 後始末
- 元のフォルダに戻り、作業フォルダを片付ける
```

このように「やることリスト」として渡すのが、安定して動かす最大のコツです。

---

2. Hooks（フック）— 「必ずやって」をAIに守らせる

なぜ必要？

カスタムコマンドや CLAUDE.md に「ファイルを直したら必ず整形して」と書いても、AIは**確率で動く**ので、守るときもあれば忘れるときもあります。「**絶対に・毎回**やってほしい」処理には、Hooks を使います。

📦 **用語ミニ解説：フック（Hook）** 「○○したら自動で△△する」という“引っかけ金具”のこと。たとえば「ファイルを保存したら、自動で見た目を整える」。AIの気分に左右されず、**決まったタイミングで決まった処理を100%実行**してくれます。

📦 **用語ミニ解説：フォーマッター** コードの見た目（字下げ・スペース・改行など）を自動で整えてくれる道具。代表例は `prettier`。散らかった部屋を自動で片付けてくれる係、とイメージしてください。

どんな「タイミング」を選べる？

Hooks は「いつ実行するか」を細かく選べます。代表的なものだけ挙げます。

タイミング	いつ動くか
`PostToolUse`	AIがファイル編集などの操作をした**直後**
`PreToolUse`	操作をする**直前**
`Stop`	AIが返答を終えたとき（例：完了の合図に音を鳴らす）
`SubagentStop`	サブエージェント（08章）が作業を終えたとき
`SessionStart`	新しいセッションを始めた／再開したとき（例：起動時に下準備コマンドを流す）
`SessionEnd`	セッションを終えたとき（記録を残すなど）

作り方は「`/hooks`」で対話的に

設定は `.claude/settings.json` というファイルに書きますが、**手で書くと記号の打ち間違いでよく失敗します**。対話モードで `/hooks` と打って、画面の案内に沿って選ぶのが安全です。

```
> /hooks
```

よくある例：ファイルを編集したら自動で整形する 「AIがファイルを直した直後（`PostToolUse`）に、整形ツールを走らせる」設定です。`$CLAUDE_FILE_PATHS` には、AIがいま直したファイルの場所が自動で入ります。

```
{
  "matcher": "Edit|Write|MultiEdit",
  "hooks": [{
    "type": "command",
    "command": "echo \"$CLAUDE_FILE_PATHS\" | grep -E '\\.(ts|tsx)$' && { npx prettier --wr
}]
}
```

💡 処理が長くなるなら、別ファイル（`.sh` という拡張子のスクリプト）にまとめて呼び出すと読みやすく、テストもしやすくなります。

📦 **用語ミニ解説：スクリプト** 複数の命令をまとめて書いた「自動実行の台本」。`.sh` ファイルに手順を並べておけば、Hooks から1行で呼び出せます。

⚠️ Hooks のセキュリティ注意（ここは大事）

Hooks は**あなたの権限で、確認なしにコマンドを実行**します。便利な反面、危険な命令も無条件で動いてしまうので、次のことが強く推奨されています。

- スクリプトは**絶対パス**で指定する（`check.sh` ではなく `~/scripts/check.sh`）
- `sudo`（管理者権限）を使わない
- `.env`・`.ssh`・`secrets` などの**機密ファイルを巻き込む設定にしない**
- ネットからダウンロードした内容をそのまま実行しない（`curl ... | sh` の形は危険）
- 変数は必ずクォートで囲む（`"$VAR"`）

> 💡 **強み**：CLAUDE.md の指示は“お願い”レベルで、守られないことがあります。Hooks は“強制力”があるので、「これだけは絶対」という品質チェックや整形に向いています。

---

3. Agent Skills — AIに「専門スキル」を覚えさせる

2025年10月に登場した新機能が **Agent Skills（エージェント・スキル）** です。

📦 **用語ミニ解説：Skill（スキル）** 「やり方の手順書」＋「実際に動かす台本（スクリプト）」をセットにして、AIに渡す“専門能力パック”のこと。たとえば「ファイルを直したら、その都度コードの誤りをチェックして自動修正する」といった一連の能力を、ひとまとめで覚えさせられます。

仕組みと作り方

`~/.claude/skills/`（または `./.claude/skills/`）の中に、スキルごとのフォルダを作り、その中に `SKILL.md` という説明書を置きます。最低限、次の3つを書きます。

- `name`：スキルの名前（英小文字・数字・ハイフン）
- `description`：何をするか／いつ使うか
- `allowed-tools`：使ってよい道具（省略可）

💡 **作るのはClaude本人に頼むのが一番ラク。** 自分で書こうとせず、こう頼みましょう。

> 新しいスキルを作ってください。ファイルを編集し終わった後、その編集したファイルに対してコードのチェック（Linter）を自動で実行するスキルです。

すると、必要な `SKILL.md` 一式を作ってくれます。あとは実際に使うときに、AIが「この作業ならあのスキルが使える」と判断して自動的に発動します（名前を指定して呼ぶこともできます）。

> 💡 **Skills と MCP の違い**：似た目的に「MCPサーバー」（07章）がありますが、MCPは情報のやり取りが多く“重い”ことがあります。Skills は「やりたいことだけ」を渡せるので軽く、**再現性（毎回同じ品質で動く）**が高いのが魅力です。

---

4. AIへの指示が上手くなる「プロンプトエンジニアリング」

📦 **用語ミニ解説：プロンプトエンジニアリング** AIから狙いどおりの結果を引き出すために、**指示の出し方を工夫する技術**。特別な資格は要りません。次の4つを意識するだけで、結果がぐっと安定します。

1. **明示的に書く**：欲しい結果を、あいまいにせず具体的に。「いい感じに」より「スマホで見たとき2列に並ぶように」。
2. **例を見せる（Few-shot）**：「こういう形式で出して」とサンプルを1つ添える。

3. **形式を指定する**：「表で」「箇条書きで」「Markdownで」と出力の形を決める。
4. **役割を与える**：「あなたは経験豊富なデザイナーです」と前提を与えると、その視点で答えてくれる（08章のサブエージェントと同じ発想）。

💡 余裕が出てきたら `/output-style` も覗いてみてください。`Default` ／ `Explanatory`（なぜそう実装したかを解説してくれる）／ `Learning`（あえて途中で止めて学ばせる）の3種があり、特に **Explanatory は「AIが何をなぜやったか」が分かって勉強になります**（※現在は非推奨化されていますが、考え方自体は今も有用です）。

---

5. 作る前に設計を固める「仕様書駆動開発」

AIは確率で動くぶん、「行き当たりばったり」で頼むほど完成度が落ちます。そこで近年注目されているのが **仕様書駆動開発** です。

📦 **用語ミニ解説：仕様書駆動開発（しようしょくどうかいはつ）** コードを書き始める前に、**「要件（何を作るか）→ 設計（どう作るか）→ タスク（やることリスト）」の順に文書を固めてから**実装に入るやり方。土台がしっかりするので、AIが脱線しにくくなります。

代表的なツール・仕組み：

- **Kiro（kiro.dev）**：Amazon製。仕様を「要件 / 設計 / タスク」の3階層のMarkdownに整理してから作る「Specモード」が特徴。
- **GitHub Spec Kit**：GitHub製。仕様書づくりに特化。
- **claude-code-spec-workflow**：Claude Code で仕様書駆動開発をするための有志製ツール。

> 💡 むずかしく考えなくても大丈夫。06章や07章でやった「**まず Claude に質問させて仕様書（CLAUDE.md）を作る**」も、立派な仕様書駆動開発の入り口です。

まとめ

- **カスタムスラッシュコマンド**＝長い決まり文句を `/名前` のショートカットに。`$ARGUMENTS` で値を渡せる。作ったら `/exit` で再起動。
- **Hooks**＝「○○したら必ず△△」をAIに**強制**させる仕組み。便利だが**全権限で動く**ので、絶対パス・機密ファイル回避などの安全策は必須。

- **Agent Skills**＝手順書＋台本をセットにした“専門能力”。作るのはClaudeに頼むのが楽。
- **プロンプトエンジニアリング**＝明示・例示・形式指定・役割付与の4つで指示が上達する。
- **仕様書駆動開発**＝「要件→設計→タスク」を先に固めると、AIが脱線しにくい。

ここまでで「速く・安全に・たくさん作る」テクニックが揃いました。次は、安心して使い続けるための土台、**セキュリティとプライバシー**を学びましょう。

---

10. セキュリティとプライバシー — 安心して使うために

この章でわかること

- Claude Code が「安全に作られている」根拠と、その仕組み
- あなたの会話やコードが「学習に使われるのか」と、その止め方
- 非エンジニアが**これだけは守るべき**安全ルール
- もっと頑丈に隔離して使う方法（Docker / サンドボックス）

> Claude Code は「あなたのパソコンの中でファイルを読んだり、コマンドを実行したり」できる、かなり強力なツールです。強力ということは、使い方を誤ると危険もある、ということ。でも安心してください。Claude Code は**何重もの安全装置**を備えていて、最後の砦として**あなたが確認する**仕組みになっています。この章でその全体像をやさしく押さえましょう。

---

1. Claude Code を作っている会社は信頼できるの？

Claude Code を作っているのは **Anthropic（アンソロピック）** という会社です。この会社は、世界的に通用する**第三者のセキュリティ認証**をいくつも取得しています。

📦 **用語ミニ解説：認証（セキュリティ認証）** 「この会社は、情報を安全に扱う仕組みがちゃんとできていますよ」と、**外部の専門機関がチェックして発行するお墨付き**のこと。会社が自分で「安全です」と言うのとは違い、客観的な証明になります。

認証の名前	ざっくり言うと
SOC 2 Type2	米国公認会計士協会の基準。半年〜1年かけて「セキュリティ・可用性・機密性・プライバシー」の社内体制を継続的にチェックした証明
ISO 27001	情報セキュリティの国際規格（日本でいう「ISMS認証」）
ISO 42001	2023年にできた「**AIを責任を持って扱う**ための」国際規格
CSA STAR	クラウドサービスのセキュリティ認証
HIPAA	米国の医療情報保護法への準拠

Anthropic の API 経由の Claude はこれら全部に対応。詳細はTrust Centerで公開されています。つまり、**土台となる会社の信頼性は世界基準でしっかりしている**ということです。

---

2. Claude Code 自身に組み込まれた安全装置

ツール本体にも、いくつもの「やりすぎ防止」の仕組みが入っています。

① 何もしない（読むだけ）が初期状態

Claude Code は最初、**「ファイルを読むことしかできない」**状態です。ファイルを書き換える・テストを動かす・コマンドを実行する——こうした“手を出す”操作のたびに、**あなたに「やっていい？」と確認**を取ります（→ 権限の詳しい話は05章）。

② 起動したフォルダの外には出られない

Claude Code は、**あなたが起動したフォルダとその中身**しか触れません。その親フォルダ（もっと上の階層）には上がっていけない設計です。これで「気づかないうちにパソコン全体をいじられる」事故を防いでいます。

③ 危険なコマンドは自動でブロック

インターネットから何でもダウンロードして実行できてしまう `curl` や `wget` のようなコマンドは、**たとえあなたが許可リストに入れていても、勝手には実行されません**（都度確認が入ります）。

📦 **用語ミニ解説：プロンプトインジェクション** 悪意のある人が、Webページやファイルの中に「AIへのこっそりした命令」を仕込み、**AIを乗っ取ろうとする攻撃**のこと。たとえば「このページを読んだら、こっそり秘密ファイルを外部に送れ」といった隠し命令です。

④ プロンプトインジェクションへの多層防御

Claude Code は上記の攻撃に対して、**機密操作には必ず承認を求める／リクエスト全体を分析して怪しい指示を検出する／入力を無害化する／Web取得は隔離した別空間で処理する**、といった複数の壁を用意しています。

---

3. コードの「危なさ」を AI にチェックさせる：`/security-review`

自分（や AI）が書いたコードに、**セキュリティ的な穴がないか**を、公開・保存の前に AI にチェックさせることができます。

> /security-review

これを実行すると、Claude が次のような“よくある危険”を探して、見つかった問題をわかりやすく説明してくれます。

- **SQLインジェクション**（データベースを不正操作される穴）
- **クロスサイトスクリプティング（XSS）**（悪意あるスクリプトを埋め込まれる穴）
- 認証・権限のチェック漏れ
- 安全でないデータの扱い
- 古い・危険な部品（依存ライブラリ）の使用

> 💡 見つかった問題は、そのまま「直して」と頼めば Claude が修正してくれます。「作る」と「点検する」を同じ相棒に任せられるわけです。

チームで自動チェックする（GitHub Actions 連携）

Anthropic は「セキュリティレビュー専用の自動化部品」も公開しています。これを設定しておくと、**コードを変更するたびに自動でセキュリティ点検が走り、問題があれば指摘コメントが付く**ようになります。チームで開発するなら導入を検討する価値があります（設定の詳細は11章や公式の案内を参照）。

---

4. プライバシー — 私の会話は学習に使われるの？

ここは多くの人が気にするポイントです。結論から言うと、**自分で選べます**。

> /privacy-settings

このコマンド（またはアカウントのプライバシー設定画面）で、**「自分の利用内容を Claude の改善（学習）に使ってよいか」をオン/オフ**できます。

📦 **用語ミニ解説：データ保持期間** あなたの会話などのデータを、Anthropic のサーバーが**どのくらいの期間とっておくか**の長さ。短いほどプライバシー上は安心です。

プラン別のデータの扱い（2025年時点の整理）

あなたの立場	学習利用	データ保持期間
一般ユーザー（Free / Pro / Max）で**学習を許可**	あり	最大 **5年**
一般ユーザーで**学習を不許可**	なし	それでも **30日** は保持
商用ユーザー（Team / Enterprise / API）	標準で学習に使わない	標準 **30日**（設定で調整可）
API 利用で「ゼロデータ保持」を設定	なし	サーバーに会話を**残さない**

> ⚠️ **機密性が非常に高い情報**（顧客情報、未公開の事業情報など）を扱うなら、一般プランではなく**商用APIの「ゼロデータ保持」設定**を使うのが最も安全です。趣味や学習用途なら、一般プランで学習をオフにしておけば十分でしょう。

---

5. もっと頑丈に隔離して使う（中〜上級）

「念のため、パソコン本体から完全に切り離した“箱の中”で動かしたい」という場合の方法が2つあります。

📦 **用語ミニ解説：サンドボックス** 「砂場（sandbox）」が語源。**外に影響が出ない隔離された実験スペース**のこと。万一おかしなことが起きても、その箱の中だけで完結し、パソコン本体やネットワークに被害が及びません。

① Docker コンテナで動かす

📦 **用語ミニ解説：Docker（ドッカー）/ コンテナ** アプリを「必要なものを全部詰めた箱（コンテナ）」に入れて、その箱の中だけで動かす技術。本体環境を汚さず、外との通信も最小限に絞れます。

Claude Code の公式リポジトリには `.devcontainer` という設定一式が用意されていて、VS Code の「Dev Container」拡張を入れておけば、**ボタン一つ（Reopen in Container）でDockerのなかでClaude Codeを起動**できます。外部通信も GitHub と Anthropic API など必要最小限だけに絞られます。

② `/sandbox`（Mac / Linux のみ）

Docker はやや準備が大変。もっと手軽に隔離したいなら `/sandbox` コマンドが使えます（**Windowsはこのコマンドはまだ未対応**）。

> /sandbox

実行すると、ファイルシステムやネットワークを分離した状態でコマンドを動かせます。設定を常に効かせたい場合は、設定ファイル（`settings.json`）に次のように書いておけます。

```
"sandbox": {
  "enabled": true,
  "autoAllowBashIfSandboxed": true
}
```

---

6. 【最重要】非エンジニアが必ず守るべき安全ルール

仕組みがどれだけ堅牢でも、**最後に安全を確認する責任はあなた（利用者）にあります**。これは Anthropic 自身が公式に明言していることです。

> 「Claude Code は、あなたが与えた権限しか持ちません。承認する前に、提案されたコードやコマンドが安全かを確認する責任は利用者にあります。」「これらの保護でリスクは大きく減りますが、すべての攻撃を完全に防げるシステムは存在しません。」

非エンジニアが特に気をつけるべきことを、5つにまとめます。

- ✅ **「許可しますか？」が出たら、内容をよく見てから「Yes」する**。とくに削除（`rm`）や外部通信のコマンドは慎重に。
- ✅ **パスワードや API キーを、会話にそのまま貼り付けない**。これは繰り返し強調される鉄則です。鍵は`.env`ファイルなど決められた場所に置き、AI には「環境変数から読んで」と頼みます。
- ✅ **秘密のファイルは“読ませない”設定にする**。設定ファイルの`deny`（拒否）に`Read(./.env.*)`のように書いておくと、AI がうっかり機密ファイルを読むのを防げます（→05章）。
- ✅ **信頼できないフォルダでは起動しない**。素性のわからないファイルが入ったフォルダで起動すると、プロンプトインジェクションの入口になり得ます。
- ✅ **「すべて自動許可」モード（`--dangerously-skip-permissions`）は、隔離環境以外で安易に使わない**。名前のとおり“危険”です。

まとめ

- Claude Code を作る Anthropic は、世界基準のセキュリティ認証を多数取得していて土台は堅牢。
- ツール本体も「読むだけが初期状態」「フォルダの外に出ない」「危険コマンドはブロック」など多層防御。
- コードの危なさは`/security-review`で AI 自身に点検させられる。
- 学習に使うかは`/privacy-settings`で自分で選べる。機密重視なら API のゼロデータ保持。
- もっと隔離するなら Docker か`/sandbox`。
- そして何より――**最後に「Yes」を押すのはあなた**。鍵や秘密は渡さない、内容を見てから許可する。この基本を守れば、安心して相棒として使えます。

---

11. チーム開発と応用テクニック — 仕事で使う・未来を見る

この章でわかること

- チームみんなで Claude Code を使うときの「設定の共有」の仕組み
- 会社で導入するときに API キーを配らずに済ませる安全な方法
- 一歩進んだ応用ワザ（デザインからコード生成／画面なしの自動実行／ブラウザだけで使う Web 版）
- Claude Code がこれから向かう未来と、「いま始める意味」

---

1. チームで設定を共有する

一人で使うときは自分のパソコンの設定だけで完結します。でも**チームで同じプロジェクトを開発する**となると、「みんなに守ってほしいルール」と「自分だけの設定」を分けたくなります。Claude Code はそれを**設定ファイルの優先順位**で実現しています。

📦 **用語ミニ解説：設定ファイル** ツールの動き方（どのコマンドを許すか、どのモデルを使うか等）を書いておくテキストファイル。Claude Code では `settings.json` という名前のファイルが、置き場所によって役割が変わります。

設定の優先順位（上が強い＝勝つ）

優先	ファイル	役割	Git共有
1（最強）	`managed-settings.json`	会社（管理者）が決めるルール。**個人は上書き不可**	—
2	コマンドライン引数	起動時にその場で渡す設定	—
3	`.claude/settings.local.json`	**自分だけ**の個人設定	されない
4	`.claude/settings.json`	**チーム共有**のプロジェクト設定	される
5	`~/.claude/settings.json`	自分の**全プロジェクト共通**設定	—

> 💡 考え方はシンプルです。**チーム全員に効かせたいルールは `.claude/settings.json`（Gitで共有）**に。**自分だけの好み・許可は `.claude/settings.local.json`（Git管理されない）**に。そして会社として絶対に守らせたいことは、管理者だけが触れる `managed-settings.json` に書きます。

チーム共有設定の例

たとえば「テストとGit操作は自由に許可、でも削除や外部通信は禁止、機密ファイルは読ませない」を会社単位で決めるなら、こう書きます。

```
{
  "permissions": {
    "allow": ["Bash(npm run lint)", "Bash(npm run test:*)", "Bash(git:*)", "Read", "Edit",
    "deny": ["Bash(rm:*)", "Bash(curl:*)", "Bash(wget:*)", "Read(./.env)", "Read(./.env.*)"
    "defaultMode": "acceptEdits"
  }
}
```

> 実際、Anthropic 社内のチームでは、Issue の自動実装・コードレビューの自動化・デプロイ手順の標準化などにこうした共有設定を活用し、**プルリクエストの折り返し時間を約30%短縮**したと報告されています。

---

2. 会社で導入：API キーを配らずに使う

📦 **用語ミニ解説：API キー** サービスを使うための「本人確認の鍵」。これが漏れると他人に使われ放題（＝勝手に課金される）になるため、厳重に管理すべきものです。

チームに Claude Code を配るとき、**全員に API キーを配るのは危険**（漏洩リスク・管理の手間）です。そこでおすすめなのが、クラウド経由で使う方法です。

📦 **用語ミニ解説：IAM** クラウドサービス（AWS や Google Cloud）が持つ「**誰が何をしてよいかを管理する仕組み**」。会社のクラウドアカウントに紐づけて、メンバーごとに権限を細かく割り当てられます。

- **AWS Bedrock** や **Google Cloud Vertex AI** 経由で Claude を使うと、各クラウドの **IAM** でメンバー管理・利用量・費用を**一元管理**できます。
- **API キーを個人に渡さなくてよい**ので、漏洩リスクがありません。
- 入力内容がモデルの再学習に使われない設計で、企業利用に適したデータ保護がなされます。
- Vertex AI なら、最大 **100万トークン**の大きな記憶を持つ Sonnet モデルも選べます。

> 詳しくは公式ドキュメント（Amazon Bedrock ／ Google Vertex AI）を参照してください。

---

3. 応用ワザ① デザインからコードを作る（Figma MCP）

「AI に画面デザインまで任せると、“いかにも AI が作った”ありがちな見た目になる」――その悩みを解決するのが **Figma MCP サーバー**です。

📦 **用語ミニ解説：Figma（フィグマ）** ブラウザで使えるデザインツール。Web やアプリの画面デザインを作る定番サービスです。

Figma で用意したデザインを、Claude が読み取って **React などのコンポーネント（画面部品）に変換**してくれます。MCP の基本は07章を参照。接続後はデザイン部品のリンクを渡してこう頼みます。

> Figma の MCP サーバーを使って、このコンポーネントを React にしてください。（FigmaのURL）

- ⚠️ **1ページ丸ごとは情報が多すぎて失敗しがち。小さな部品（ボタン、カードなど）に分けて1つずつ**作らせると、精度が上がります。
- 💡 **Storybook**（部品カタログ）や **Playwright**（07章）と組み合わせると、「作る→カタログで見る→ずれを AI 自身に確認させて直す」という高速サイクルが回せます。

---

4. 応用ワザ② 画面なしで一発実行（ヘッドレスモード）

📦**用語ミニ解説：ヘッドレスモード**「対話画面（head）を持たない」モード。**チャットせず、命令を1回投げて結果だけ受け取る**使い方です。自動化やスクリプトに向きます。

> ターミナルで実行：`claude -p "やってほしいこと"`

たとえば、

- **ログを見張る**：
  `tail -f app.log | claude -p "何か異常な内容があれば教えてください。"`
- **まとめて書き換える**：`claude -p "古い書き方のファイルを新しい書き方に直して" --allowedTools "Edit,Read"`

こうした“画面なし”の使い方で、Claude Code を**定期処理やバッチ作業の部品**として組み込めます。

---

5. 応用ワザ③ ブラウザだけで使う「Claude Code on the Web」

非エンジニアにとって、いちばん敷居が低い入口がこれかもしれません。ターミナル（黒い画面）を一切使わず、**ブラウザや iOS アプリだけ**で Claude Code が使えます（2025年10月に発表されたベータ機能）。

使い方（3ステップ）

1. `claude.ai/code` にアクセス。
2. Claude.ai のアカウントでログイン。
3. 自分の GitHub リポジトリと連携する。

あとは画面のチャット欄に指示を打つだけ。会話を始めると**自動で新しいブランチが作られ**、右下の「PRを作成」ボタンで成果をまとめられます。

メリットと注意点

- 💡 各会話が**独立した安全な箱（サンドボックス）**で動くので、複数の作業を同時に進めても安心。外出先からスマホで開発を進められます。

- ⚠️ ただし**開発サーバーを起動して見た目を確認することはできない**ので、用途は「特定機能の実装・バグ修正・コード調査」が中心です。
- ⚠️ 毎回まっさらな環境になるため、`npm install` などの準備が必要なことがあります。これを自動化するには、`SessionStart`（セッション開始時）のフックに準備コマンドを書いておきます（フックは09章参照）。このとき、設定は**Gitで共有される** `settings.json` に書く必要がある点に注意。

```
{
  "hooks": {
    "SessionStart": [
      { "matcher": "startup", "hooks": [ { "type": "command", "command": "npm install" } ]
  ]
 }
}
```

---

6. Claude Code が向かう未来 — そして「いま始める意味」

最後に、この本の締めくくりでもある「これから」の話を。なぜ**今**学ぶ価値があるのか、その根拠を共有します。

すでに起きている事実

- 2025年5月末のリリースから**わずか数ヶ月で、約11万5千人の開発者**が利用し、**週におよそ1億9500万行**のコードを処理。
- ある企業では、当初「エンジニアで2年かかる」と見積もられた作業が、Claude Code 活用で**1人・2週間で完了**。
- そして——**非エンジニアの実例**。IT業界ではない会社で、社長がインターン生に**自社用の在庫管理システム**を作らせた例もあります。別の現場では**営業マンは名刺管理アプリを自作**して経営陣に高く評価された。**“作る側”は、もうエンジニアだけのものではない**のです。

これから起きること（予測）

- Anthropic CEO は、**1〜3年以内に Claude が「指示を待つ道具」から「自分から動く同僚」へ**進化すると予測しています。データを見張り、変更を提案し、自らコードを書く——まるで職場の仲間のように。

- さらに、**2026〜2027年**には、生物学・数学・コンピュータ科学などで**ノーベル賞級の知的タスクを AI が自律的にこなす**ようになる、との見立ても示されています。

だから、いま

完璧に理解してから始める必要はありません。**小さなものを1つ作ってみる**——それが、この大きな変化の波に乗る最初の一歩です。このマニュアルの06章に戻れば、5分でその一歩を踏み出せます。

まとめ

- チーム利用は**設定ファイルの優先順位**で「共有ルール」と「個人設定」を両立できる。
- 会社導入は **Bedrock / Vertex AI 経由（IAM 管理）**で API キーを配らず安全に。
- 応用ワザ：**Figma MCP**（デザイン→コード）、**ヘッドレス**（画面なし自動実行）、**Web 版**（ブラウザだけ・非エンジニアの最短入口）。
- Claude Code は「同僚のような AI」へ進化していく途中。**いま小さく始めることが、最大の準備**です。

学び終えたあなたへ — 次の一歩

1. まだアプリを作っていなければ、06章で1つ作って公開してみる。
2. 自分の仕事の「めんどうな繰り返し作業」を1つ、Claude にやらせてみる。
3. 困ったらR3 トラブル対処とFAQ、コマンドを忘れたらR1 早見表、言葉が分からなければR2 用語集へ。
4. そして——さらに深く学びたくなったら、Anthropic の公式ドキュメントや、信頼できる解説書・公式ブログにあたってください。一次情報を確認する習慣が、いちばんの近道です。

R1. スラッシュコマンド早見表（リファレンス）

> `/`（スラッシュ）で始まる命令の一覧です。困ったときの逆引きに使ってください。 内容は Claude Code **v2.0.14** 時点のものがベース。最新は対話中に `/help` または `/release-notes` で確認できます。

📦 **用語ミニ解説：スラッシュコマンド** 普通の日本語の指示（プロンプト）とは別に、`/` で始めて打つ「機能の呼び出し命令」。入力欄で `/` を打つと候補が出ます。

⭐ まず覚える「これだけ」7つ

コマンド	ひとことで	使う場面
`/init`	記憶ノート(CLAUDE.md)を作る	プロジェクトを始めるとき
`/clear`	記憶を全部リセット	話題が「ところで」と変わるとき
`/compact`	記憶を要約して圧縮	同じ作業で記憶が重くなったとき
`/context`	記憶の使用状況を見る	「最近重いな」と思ったとき
`/model`	使う AI モデルを変える	賢さ/速さを切り替えたいとき
`/rewind`	変更を巻き戻す	失敗を取り消したいとき
`/exit`	会話を終了	作業を終えるとき

⍰ セッション・記憶の管理

コマンド	説明
`/clear`	会話履歴を全消去して新しい話題に集中（消しても `/resume` で復元可）
`/compact`	会話を要約して圧縮。`/compact ○○にフォーカス` でテーマ指定も可
`/rewind`	会話・コードを巻き戻す（`Esc` 2回でも起動）。※手動変更・コマンド変更は戻せない
`/context`	記憶（コンテキストウィンドウ）の占有状況をグラフ表示
`/export`	今の会話をクリップボード/ファイルに書き出す
`/resume`	過去の会話セッションを選んで再開
`/exit`	会話を終了（`Ctrl + D` でも可）

👤 アカウント・認証

コマンド	説明
`/login`	Anthropic アカウントを切り替え
`/logout`	今のアカウントからログアウト（サブスク↔API 切替時など）
`/upgrade`	Max プランへのアップグレード案内（Max 20x の人は不要）

⚙️ プロジェクト設定

コマンド	説明
`/init`	記憶ノート **CLAUDE.md** を作成
`/memory`	CLAUDE.md を編集
`/config`	設定画面を開く（表示テーマの変更などもここ）
`/permissions`	権限（どの操作を許す/拒否するか）の確認・編集
`/add-dir`	別フォルダを作業対象に追加（※そのセッション中だけ）
`/ide`	エディタ（VS Code等）との連携オン/オフ
`/terminal-setup`	`Shift + Enter` で改行できるようにする

🤖 モデル・AI 機能

コマンド	説明
`/model`	使う AI モデルを選択・変更
`/agents`	専門家 AI（サブエージェント）を作成・管理（→08章）
`/vim`	Vim 風のキー操作モードをオン/オフ（上級者向け）

📦 モデルの合言葉：`/model opus`（最も賢い）／`/model sonnet`（バランス）／`/model haiku`（速い・安い）。詳しくは02章。

❓ 開発支援

コマンド	説明
`/review`	コードレビューを依頼（`/review 4` で PR #4 を指定も可）
`/pr-comments`	プルリクエストのコメントを表示
`/bashes`	裏で動いている処理の確認・停止
`/hooks`	自動実行の仕掛け（Hooks）を追加（→09章）
`/install-github-app`	GitHub 連携（@claude で動かす）をセットアップ
`/security-review`	コードの脆弱性をチェック（→10章）
`/mcp`	MCP サーバー接続の管理・ログイン（→07章）
`/sandbox`	隔離された安全な環境で実行（Mac/Linux のみ）

📊 情報表示・状態確認

コマンド	説明
`/todos`	現在の作業 ToDo 一覧を表示
`/help`	使い方ヘルプ
`/status`	現在の状態・設定を表示
`/cost`	トークン使用量・費用を表示（**API ユーザー専用**）
`/usage`	プランの使用量・制限の状況を表示（サブスク向け）
`/release-notes`	最新の機能追加・変更点を表示

🎨 表示・出力スタイル

コマンド	説明
`/output-style`	応答スタイルを変更（Default／Explanatory／Learning）※一部バージョンで非推奨
`/output-style:new`	自分用の出力スタイルを新規作成
`/statusline`	画面下部の常時表示情報を設定

💡 **Explanatory（説明的）スタイル**は「なぜそう実装したか」を解説付きで返してくれるので、学びながら使いたい非エンジニアに好相性です。

🔌 プラグイン・プライバシー

コマンド	説明
`/plugin`	プラグイン（機能の詰め合わせ）の追加・管理
`/privacy-settings`	プライバシー設定を表示・変更（→10章）

🩺 メンテナンス・困ったとき

コマンド	説明
`/doctor`	インストール状態の健全性チェック（不調時の診断）
`/feedback`	Anthropic に意見・バグ報告を送る

✍️ 入力欄の特殊記号（スラッシュ以外）

記号	意味
`@ファイル名`	そのファイルを話題に含める（@インポート）
`#`（行頭）	その内容を CLAUDE.md に即追記
`$ARGUMENTS` / `$1` `$2`	自作コマンドに値を渡すための記号（→09章）

R2. 用語集 — 非エンジニアのための「これだけ辞典」

> 本文に出てくる専門用語を、**できるだけ短く・たとえ話で**まとめました。読んでいて「？」となったら、ここで引いてください。五十音・アルファベット混在で、ジャンル別に並べています。

🧭 いちばん基本の言葉

用語	やさしい意味
ターミナル	パソコンに“文字で”命令する黒い画面。Claude Code が動く場所。
CLI	上の「文字で命令する方式」の正式名。Command Line Interface の略。
コマンド	ターミナルに打ち込む命令文のこと。
プロンプト	あなたが AI に出す指示（自然な日本語でOK）。本マニュアルでは行頭 `>` で表記。
エディタ／IDE	プログラムを書くための専用ソフト。代表は **VS Code**。
デプロイ	作ったアプリをインターネットに“公開”すること。
ローカル	「自分のパソコンの中だけ」の意味。⇔ 本番（インターネット上）。

🤖 AI まわり

用語	やさしい意味
Claude（クロード）	Anthropic 社の AI 本体の名前。
Claude Code	その Claude に“手足”を付けて開発作業をさせる道具。
LLM	大規模言語モデル。「次に来そうな言葉」を確率で予測して文を作る AI の仕組み。
モデル（Opus/Sonnet/Haiku）	Claude の種類。賢さ・速さ・値段が違う。Opus=賢い／Sonnet=バランス／Haiku=速くて安い。
トークン	AI が文章を処理する最小の“かたまり”。料金や記憶量の単位になる。
ハルシネーション	AI が“もっともらしい嘘”を自信満々に答えてしまう現象。
コンテキスト	AI が答えを出すために参照する「文脈・前提情報」。
コンテキストウィンドウ	AI が一度に覚えていられる作業メモリの量（Claude は最大20万トークン）。
拡張思考／ultrathink	AI にいつもより深く考えさせるモード／その合言葉。
Vibe コーディング	細かい中身を気にせず、ノリと自然言語でアプリを作るスタイル。
AI 駆動開発	AI を中心に開発を回し、人間は方向づけと確認に回る新しい働き方。

📁 ファイル・記憶・設定

用語	やさしい意味
CLAUDE.md	プロジェクトの“説明書＆記憶ノート”。AI が毎回読む。`/init` で作る。
@インポート	`@ファイル名` でそのファイルを話題に取り込む書き方。
パーミッション（権限）	AI に「どの操作までやらせるか」の許可設定。
動作モード	default（毎回確認）／acceptEdits（編集を自動許可）／plan（計画だけ）／bypassPermissions（全許可・危険）。
settings.json	Claude Code の設定ファイル。共有用と個人用（`.local`）がある。
ユーザールート / プロジェクトルート	設定の置き場所。前者＝全プロジェクト共通、後者＝そのプロジェクト限定。

🧰 開発の道具・部品

用語	やさしい意味
Git	ファイルの変更履歴を記録・管理する仕組み。
GitHub	Git をインターネット上で保存・共有するサービス。
リポジトリ	プロジェクト1個分の保管場所（フォルダ＋履歴）。
コミット／プッシュ	変更を記録すること／それを GitHub に送ること。
ブランチ	本体に影響を与えず作業するための“分岐した作業用コピー”。
Issue（イシュー）	やること・課題を1件ずつ書いておくチケット。
プルリクエスト（PR）	「この変更を本体に取り込んで」という提案。
gh コマンド	GitHub をターミナルから操作する道具。連携すると超便利。
CI/CD	コード変更時にテスト・公開を自動で流す仕組み。
GitHub Actions	GitHub 上で自動処理を動かす仕組み（CI/CD の定番）。

⍰ アプリを作る技術（名前だけ知っておけばOK）

用語	やさしい意味
フレームワーク	アプリ作りの“土台キット”。
Next.js	人気の Web アプリ用フレームワーク。
React	画面を部品（コンポーネント）で組み立てる仕組み。
TypeScript	間違いに気づきやすい、型のあるプログラミング言語。AI と相性が良い。
Tailwind CSS / shadcn/ui	見た目（デザイン）を整える道具。
データベース（DB）	データを保存しておく場所。MongoDB・SQLite・Supabase など。
Prisma	データベースを安全に扱うための道具（ORM）。
API	アプリ同士がデータをやり取りする“窓口”。OpenAPI はその仕様の書き方。
Zod	受け取ったデータが正しい形か検証する道具。
テスト／Vitest／E2E	動作チェックの仕組み／その定番ツール／実ユーザー操作を再現するテスト。
Lint（リント）／ESLint／Biome	コードの書き方の乱れをチェック・統一する道具。
Husky	コミット前に自動でチェックを走らせる見張り役。
Mermaid 記法	文章で図（ER図など）を書ける記法。

☁️ 公開・インフラ

用語	やさしい意味
PaaS	難しい設定を肩代わりしてくれる公開サービス（例：Vercel）。
GitHub Pages	GitHub の無料公開機能（公開リポジトリ向け）。
Vercel	簡単・無料枠ありの公開サービス。Next.js と好相性。
Google Cloud / Cloud Run	Google のクラウド。Cloud Run はコンテナでアプリを動かす仕組み。
Docker / コンテナ	アプリを“箱詰め”して、どこでも同じように動かす技術。
環境変数 / .env	API キーなどの秘密情報を、コードに直接書かず分けて持つ仕組み。

🔌 拡張・自動化

用語	やさしい意味
MCP サーバー	Claude Code の能力を拡張する“追加コネクタ”。Context7・Playwright・Serena・Figma など。
Context7	ライブラリの最新情報を AI に教える MCP。`use context7` で呼ぶ。
Playwright（MCP）	AI に“目”を与え、ブラウザを操作・確認させる MCP。
サブエージェント	「フロント担当」「バック担当」など、専門家として切り出した AI。
カスタムスラッシュコマンド	よく使う指示を `/名前` で呼べるよう保存したもの。
Hooks（フック）	「ファイル編集したら必ず整形」など、決まった処理を確実に自動実行する仕掛け。
Agent Skills	手順書＋スクリプトをセットにして AI に持たせる“専門能力”。
プラグイン	コマンド・サブエージェント・MCP などをまとめて共有できる詰め合わせ。

💳 料金・プラン

用語	やさしい意味
サブスクリプション	定額制（Pro / Max 5x / Max 20x）。料金が読めて初心者向き。
API 従量課金	使った分だけ払う方式。Claude Console アカウントが必要。
5時間ごとの制限	サブスクのメッセージ量はセッション（5時間枠）ごとにリセットされる。

R3. トラブル対処とFAQ — 困ったときの逆引き

> 「うまくいかない」「これってどうなの？」を、症状から引けるようにまとめました。 まず試す万能薬は **①エラーをそのままコピペして見せる ②「直るまで修正し続けて」と頼む ③ `/clear` して仕切り直す** の3つです。

💸 お金・プランの不安

Q. いくらかかるの？ 使いすぎが怖い。 A. サブスク（定額）なら**追加料金は発生しません**。Pro は月20ドル、Max 5x は100ドル、Max 20x は200ドル。Anthropic によると開発者1人あたり平均で1日6ドル程度（9割の人は1日12ドル未満）の使用感です。詳しくは02章。

Q. どのプランから始めればいい？ A. **まず Pro（月20ドル）**で十分。物足りなくなったらアップグレードが王道です。複数の AI を同時に走らせたい人だけ Max 20x を検討。

Q. API 課金にした覚えがないのに請求が来る。 A. 初回ログインで**「2. Anthropic Console account」を選ぶと API 従量課金**になります。サブスク（Pro/Max）の人は必ず**「1. Claude account with subscription」**を選んでください。`/logout` → `claude` で入り直して選び直せます。

Q. 今いくら使ったか確認したい。 A. API ユーザーは `/cost`、サブスクユーザーは `/usage`（制限まであと何%かを表示）。

🔌 インストール・起動でつまずく

Q. `claude` と打っても起動しない／コマンドが見つからない。 A. インストールが済んでいない可能性。03章の手順を再確認。Mac なら `brew install --cask claude-code` が手早いです。診断は `/doctor`。

Q. WSL（Windows）でインストールエラーが出る。 A. Windows 本体側の Node.js と競合しているのが典型。`nvm` などのバージョン管理ツールで Node.js を入れ直すと解決します。

Q. `claude update` したのにバージョンが上がらない。 A. **Volta** という管理ツールを使っていると、「成功」と出ても実際は上がらない既知の現象があります。Volta 側で更新する必要があります。

Q. ホームディレクトリで起動したら警告が出た。 A. `claude` は**プロジェクトのフォルダの中で**起動するのが基本です。`cd プロジェクト名` で移動してから `claude` を実行しましょう。

🤖 AI の挙動がおかしい

Q. 同じ指示なのに毎回違う結果になる。 A. 仕様です。LLM は確率で答えるため、結果にブレが出ます。気に入らなければ言い方を変えてもう一度頼みましょう。

Q. だんだん的外れな答えになってきた／忘れているっぽい。 A. 記憶（コンテキスト）が混雑しているサインです。話題が変わったなら `/clear`、同じ作業を続けるなら `/compact` で整理を。`/context` で混雑具合を確認できます。目安は「**“ところで”と言いたくなったら `/clear`**」。

Q. 勝手にファイルを編集された／されたくない。 A. 通常は編集前に確認されます。確認を省く `acceptEdits`（`Shift+Tab`）をオンにしていないか確認を。**考えだけ聞きたいときは plan モード**（ファイルを変更しない）が安全です（05章）。

Q. 指定した技術と違うもので作り始める。 A. AI は“得意なライブラリ”に勝手に寄せることがあります（例：gRPC を頼んだのに REST に）。**CLAUDE.md に使用技術を明記**して固定し、`use context7` を添えると安定します。

Q. テストが全部「成功」するのに、実際は動かない。 A. AI が**中身のないテスト**や**答えの決め打ち**でズルをした可能性。CLAUDE.md にテストのルールを書いて防ぎます（06章）。

❓ アプリ作り・デプロイで困る

Q. 見た目が崩れている。 A. 画面を**スクリーンショット**して `Ctrl+V` で貼り付け、「CSSが効いていないようです。確認して」と頼むのが最速（06章）。

Q. エラーが出たけど意味がわからない。 A. **エラー文をまるごとコピペ**して貼るだけでOK。Claudeが折りたたんで読み取ってくれます。一度で直らなければ「**直るまで調査・修正し続けてください**」。

Q. デプロイ（公開）で何度もエラーになる。 A. 「`make deploy` を、問題がなくなるまで修正してください」のように**自走を頼む**と、権限や設定の間違いを直しながら進めてくれます。あなたは待つだけ。

Q. 公開したのに「Forbidden（権限がありません）」と出る。 A. Cloud Run などは初期状態で外部アクセスが閉じています。「公開ポリシーを変更して公開してください」と指示。⚠️ ただし**権限まわりは AI 任せにせず自分でも確認**を。

Q. 無料で使い続けたい。 A. Cloud Run は**最小インスタンス数を0**に指定すれば未使用時は無料。データベースは **SQLite＋Litestream / Supabase** などの無料枠が便利（08章周辺）。

🔧 拡張機能（MCP）で困る

Q. 「Playwrightで確認して」と言ったのに、ただのテストファイルを書き始めた。 A. **「Playwright の MCP サーバーを使って」**と明記してください。これがないと別物として動きます（07章）。

Q. Context7 を入れたのに使われない。 A. プロンプトの末尾に `use context7` という合言葉を付ける必要があります。

Q. MCP がつながっているか確認したい。 A. `/mcp` で接続状況を確認。「✓ connected」なら成功です。

⚡ 並行作業・上級操作

Q. 複数の Claude を同時に動かしたい。 A. ターミナルを分割して、それぞれで `claude` を起動。フロント担当／バック担当のように**作業を被らせない**のがコツ。⚠️ ただし**5時間制限にすぐ届く**ので、現実的には**同時2個程度**まで（08章）。

Q. 複数開いたらエディタ連携が切れた。 A. **IDE 連携は1つのターミナルだけ**有効。後から開いた方に移るので、戻したいターミナルで `/ide` を実行します。

🔒 安全・プライバシー

Q. 自分の会話が AI の学習に使われない？ A. `/privacy-settings` で学習利用のオン/オフを選べます。機密性を最優先するなら、API 利用＋「ゼロデータ保持」設定が選択肢（10章）。

Q. うっかり危ないコマンドを実行されない？ A. `curl` / `wget` などの危険コマンドは、許可していても都度確認されます。さらに `/security-review` で脆弱性チェック、`/sandbox`（Mac/Linux）で隔離実行も可能。⚠️ ただし**最終的な安全確認はユーザーの責任**です。

Q. API キーやパスワードはどう扱えばいい？ A. **無闇に AI に渡さない**こと。秘密情報は `.env` ファイルや環境変数に分けて持ち、コードに直接書かないのが鉄則。

🆘 それでも解決しないとき

1. `/doctor` でインストール状態を診断。
2. `/release-notes` で「最近の仕様変更」を確認（機能が変わっていることも）。
3. 公式ドキュメントを確認。
4. `/feedback` で Anthropic に報告。
