

Claude Code 実践導入ガイド 2026

AIコーディングで開発を速く・安全に

— 「AIに書かせる」から「AIと開発する仕組み」へ —

- ✓ Claude Codeで何ができて、何が人間の仕事として残るのか
- ✓ 成果を左右する「指示の出し方」と「コンテキスト管理」
- ✓ 権限・秘密情報・レビューで守る、安全な運用ルール
- ✓ チーム導入のロードマップと、チェックリスト20・CLAUDE.mdテンプレート付き

発行：Harmonic Society株式会社

（千葉市 | ホームページ制作・AI導入支援・IT顧問）

はじめに — 「AIが書いたコード」を、会社の資産にするために

「AIにコードを書かせてみたら、それらしく動いた。でも、このまま本番に出してよいのか分からない」
——AIコーディングを試した開発者や経営者の方から、最もよく聞く声です。

Claude Codeは、Anthropic社が提供する**エージェント型のAIコーディングツール**です。チャット画面にコードを貼り付けて質問する使い方とは違い、Claude Code自身がプロジェクトのファイルを読み、コードを書き換え、テストやコマンドを実行し、結果を確かめながら作業を進めます。いわば「手を動かせるAI」です。

手を動かせるからこそ、速さと同時にリスクも生まれます。意図しないファイルの変更、秘密情報の読み込み、検証されないまま積み上がるコード。同じツールを使っている、成果を出すチームと混乱するチームに分かれる理由は、AIの性能ではなく**使い方と仕組み**にあります。

本書は、当社ブログのClaude Code関連記事（導入・指示の出し方・デバッグ・テスト・Git・セキュリティ・チーム運用など）を、導入を検討する方が順に読めるよう一冊に再構成したものです。当社自身もClaude Codeでこのコーポレートサイトの開発・記事制作を日常的に行っており、その運用で得た知見も反映しています。

読み終えたとき、「自社で使うべきか」「最初に何を整えるか」「どこまでAIに任せ、どこを人が見るか」を説明できることがゴールです。

本書の情報について：本書は2026年9月時点の情報に基づきます。Claude Codeは更新頻度が非常に高いツールで、料金プラン、利用できるモデル、コマンド名、設定項目、権限モードの名称などは変更される可能性があります。導入・契約の判断にあたっては、必ずAnthropic社の公式ドキュメントで最新情報をご確認ください。本書に掲載した手順・プロンプト例は考え方を示すためのもので、特定の成果を保証するものではありません。

第1章 Claude Codeとは何か

1-1. 「補完」ではなく「作業を任せる」AI

AIによる開発支援ツールは、大きく2つのタイプに分けられます。ひとつは、エディタの中で次を書くコードを予測して提案する**補完型**。もうひとつが、目的を伝えると自分でファイルを調べ、編集し、コマンドを実行して結果を確認する**エージェント型**です。Claude Codeは後者の代表例です。

もともとはターミナル（コマンドを打ち込む黒い画面）で動くツールとして登場しましたが、2026年9月時点ではデスクトップアプリ、Web版、VS CodeやJetBrainsの拡張機能としても提供されています。どの形態でも、基本的な考え方は共通です。

図表1：AI開発支援ツールの2つのタイプ

比較項目	補完型（エディタ内の提案）	エージェント型（Claude Code等）
人の役割	1行ずつ書きながら提案を採否	目的と条件を伝え、結果を確認
扱う範囲	今開いているファイル中心	プロジェクト全体・複数ファイル
コマンド実行	基本的にしない	テスト・ビルド・Git操作まで実行
向く作業	手元の実装を速く書く	調査・修正・リファクタリング・定型作業
必要な管理	少ない	権限・ルール・検証の設計が必要

1-2. 成果を分けるのは「ハーネス」

AIモデルの周囲に組まれた実行の枠組みを**ハーネス**と呼びます。車にたとえるなら、AIモデルがエンジンで、ハーネスは車体・ハンドル・ブレーキです。エンジンが強力でも、ハンドルとブレーキがなければ安全には走れません。

Claude Codeのハーネスは、次の4つの役割を担っています。

- **ツール実行**：ファイルの読み書き、検索、シェルコマンド実行などの「手足」
- **権限管理**：どの操作を自動で許可し、どの操作で人の確認を求めるか
- **コンテキスト管理**：長い作業でも文脈を失わないための要約・記憶の仕組み
- **拡張機構**：スキル、フック、MCP、サブエージェントなど、チーム独自の使い方に育てる仕組み

同じモデルを使っている、このハーネスをどう設定するかで働きぶりは大きく変わります。本書の後半（第5章・第6章）は、このハーネスを自社向けに整える方法の解説です。

第1章 Claude Codeとは何か（続き）

1-3. AIの得意と、人間に残る仕事

Claude Codeは万能ではありません。導入の最初に「任せる仕事」と「人が担う仕事」の線を引いておく
と、期待外れや事故を防げます。

図表2：AIに任せやすい仕事と、人が担う仕事

AIに任せやすい	人が担うべき
エラーメッセージやログの読み解き、原因候補の洗い出し	何を作るか・作らないかの判断（要件・優先順位）
テストコードの下書き、既存コードの説明	ビジネスロジックが正しいかの最終確認
命名の統一、重複の整理などの機械的なリファクタリング	設計方針・アーキテクチャの決定
コミットメッセージ、PR本文、README、変更履歴の作成	本番環境への反映の可否（デプロイ判断）
規約違反・典型的な脆弱性パターンの一次チェック	セキュリティの最終判断、専門家による監査

ブログの各記事で繰り返し強調しているのは、**AIのレビューや監査は「第一段階のスクリーニング」であり、人間の判断の代替ではない**という点です。AIは既知のパターンを網羅的・一貫的に確認するのが得意な一方、「この仕様はお客様にとって正しいか」という文脈の判断は苦手です。

1-4. どんな組織に向いているか

- ・ **少人数で多くの案件を抱える開発チーム**：調査・テスト・ドキュメントなどの周辺作業を任せ、人は設計とレビューに集中できる
- ・ **エンジニアが1～2人の中小企業**：レビュー相手がいない環境で、「即座に意見をくれる相手」として品質の底上げに使える
- ・ **自社サイトや業務ツールを内製したい会社**：日本語で指示できるため、非エンジニアでも小規模なWebサイト制作・修正から始められる

一方で、変更の影響を確認する手段（テストやステージング環境）が何もない状態で本番システムを直接触らせるのは危険です。導入は「失敗しても困らない場所」から始めるのが鉄則です。

第2章 はじめの一步 — 導入と初期設定

2-1. 導入の流れ

Claude Codeの導入自体は、長くても半日で終わります。大まかな流れは次のとおりです。

- 動作環境を確認する**：macOS・Linux・Windows（WSL2などの環境経由）で動作します。対応OSや前提ソフトウェアの条件は更新されるため、公式ドキュメントで確認します
- インストールする**：公式が案内するインストール方法（npm経由、Homebrew等のパッケージ管理ツール経由など）から、自社の環境に合うものを選びます
- 認証する**：初回起動時にブラウザが開き、Anthropicアカウントでログインします。CI（自動テスト環境）などで使う場合は、APIキーを環境変数で渡す方法が適しています
- プロジェクトのフォルダで起動する**：対象のフォルダに移動して起動すると対話が始まります。まずは「このプロジェクトの構造を教えて」と尋ねてみるのがおすすめです

APIキーは、ソースコードや設定ファイルに直接書かず、環境変数や.envファイルで管理し、.envは必ず.gitignore（Gitの管理対象から除外する設定ファイル）に入れてください。

2-2. 料金の考え方（2026年9月時点）

Claude Codeの利用形態は、大きく分けて3つあります。具体的な金額・含まれる利用量・対象プランは頻繁に改定されるため、ここでは考え方だけを示します。

図表3：利用形態と向いているケース

利用形態	課金の仕組み	向いているケース
個人向けサブスクリプション（Pro・Max等）	月額固定（利用量に上限あり）	個人・少人数での日常的な開発。費用が予測しやすい
チーム・企業向けプラン	人数×月額	管理機能や請求の一元化が必要な組織
API従量課金	使ったトークン量に応じて課金	CIへの組み込み、利用頻度が低い場合

トークンとは、AIが文章を処理するときの単位です。後述する拡張思考を多用したり、大きなファイルを大量に読ませたりすると消費が増えます。セッション中に利用状況を確認するコマンドも用意されているので、最初の1か月は意識して見ておくと、自社に合うプランが判断しやすくなります。

第2章 はじめの一步（続き）

2-3. 最初に書くべきファイル「CLAUDE.md」

導入直後に最も費用対効果が高いのが、プロジェクトのフォルダ直下に置く**CLAUDE.md**です。Claude Codeは起動するたびにこのファイルを読み込み、書かれた内容を前提に作業します。新しく入ったメンバーに渡すオンボーディング資料を、AIにも渡すイメージです。

CLAUDE.mdが効く理由は、会話の履歴が要約されても、セッションを作り直しても、**常に読み込まれ続ける**点にあります。毎回同じ説明を繰り返す必要がなくなり、メンバー全員が同じ前提でAIを使えるようになります。

2-4. 書く内容は「新人に最初に伝えること」

- ・ **プロジェクトの概要**：何のためのシステムか（1～2行）
- ・ **技術スタック**：言語・フレームワーク・データベースとそのバージョン
- ・ **よく使うコマンド**：ビルド、テスト、リント（コードの書き方チェック）、型チェックの実行方法
- ・ **コーディング規約**：命名規則、ファイルの置き場所、使ってはいけない書き方
- ・ **やってはいけないこと**：.envを読まない・コミットしない、本番DBに接続しない、マイグレーションは必ずレビューを受ける、など

2-5. 最初は10行、使いながら育てる

最初から完璧なCLAUDE.mdを書こうとする必要はありません。ビルドとテストのコマンド、守ってほしいルールを箇条書きで10行程度書き、「AIが同じ間違いを2回したら1行追記する」運用で十分です。

ただし、CLAUDE.mdもAIの「作業机」の広さ（次章で説明するコンテキスト）を消費します。冗長な説明は避けて箇条書きにし、特定のフォルダにしか関係しないルールは、そのフォルダに置くCLAUDE.mdへ分けると効率的です。配置場所ごとの役割は次のとおりです。

図表4：CLAUDE.mdの配置場所と役割

配置場所	適用範囲	Gitで共有
プロジェクト直下	プロジェクト全体の共通ルール	する（チームの資産）
サブフォルダ	そのモジュール・パッケージ固有のルール	する
ホームフォルダ（~/ .claude/）	個人の好み・環境固有の設定	しない

巻末付録2に、そのまま書き換えて使えるテンプレート例を掲載しています。

第3章 成果を左右する「指示の出し方」

3-1. AIは行間を読まない

指示（プロンプト）は日本語で構いません。ただしAIは行間を読むのが得意ではなく、曖昧な指示では意図と違うコードを書いたり、頼んでいない作業まで進めたりします。

図表5：悪い指示と良い指示

悪い例	良い例
「ログイン機能を作 って」	「JWT認証のログインAPIを作って。エンドポイントはPOST /api/auth/login、email とpasswordを受け取る」
「バグを直して」	「src/utils/date.tsのformatDate関数で、1月が0と表示されるバグを修正して」
「テストを書いて」	「userService.tsのcreateUser関数のユニットテストを書いて。正常系・入力エラー・ 重複エラーの3パターン」

3-2. 良い指示の5原則

- 対象を具体的に指定する**：ファイルパスや関数名を明示する（「全体を見て」は不要な読み込みの原因）
- 制約条件を伝える**：使うライブラリ、バージョン、規約、変更してはいけない範囲
- 段階的に進める**：「方針を提案して」→確認→「実装して」→「テストを書いて実行して」（出力の検証まで必ず頼む）
- 出力の形を指定する**：ファイルの分け方、コメントの有無、報告のフォーマット
- 背景を共有する**：「レポート生成に30秒以上かかる」など、なぜ必要かを伝えると判断の質が上がる

3-3. 「確認してから実行」の型

大きな変更ほど、**計画だけを先に知らせる**のが安全です。ファイルを変更せずに計画を立てる「プランモード」を使うか、次のように頼みます。

以下の変更を、実際には行わずに計画だけ出してください。

変更対象のファイル一覧／各ファイルでの具体的な変更内容／潜在的なリスクや注意点

計画の段階で方向性を直せば、実装後に巻き戻すよりはるかに安く済みます。

第3章 成果を左右する「指示の出し方」（続き）

3-5. コンテキスト — AIの「作業机」の広さ

コンテキストウィンドウとは、AIが一度に扱える情報量の上限です。会話の履歴、読み込んだファイル、コマンドの実行結果、CLAUDE.mdの内容——これらすべてが同じ机の上に並びます。上限はモデルや契約によって異なりますが、大きなプロジェクトのコードを片端から読ませれば、どの環境でもすぐに埋まります。

机があふれてくると、AIは初めのほうの指示や決定事項を「忘れ」、矛盾した回答や、関係ないファイルの変更が起き始めます。精度が落ちたと感じたら、まずコンテキストを疑ってください。

3-6. 机を片付ける3つの習慣

- **1セッション1タスク**：認証機能の実装と画面の修正を同じ会話で行わない。タスクが終わったら新しいセッションを始める（CLAUDE.mdは引き継がれるので基本情報は失われない）
- **区切りで要約する（/compact）**：会話の履歴を要約して机を空けるコマンド。「APIの認証方式はJWT、DBはPostgreSQL」のように、残すべき情報を添えて実行すると要点が落ちにくい。作業の途中ではなく、区切り目で使うのがコツ
- **まったく別の作業ならリセット（/clear）**：前のタスクの情報が残っていると、新しいタスクの判断を狂わせることがある

図表6：精度が落ちたときの症状と対処

症状	考えられる原因	対処
以前の指示を忘れる	机があふれ、古い情報が押し出された	要約して重要事項を再度伝える
同じ質問を繰り返す	過去の回答が机から消えている	決定事項はCLAUDE.mdに書く
関係ないファイルを変更する	不要な情報が多く混乱している	作業対象のファイルを明示する
変更の整合性が崩れる	関連ファイルの一部しか読めていない	関連ファイルを段階的に読ませる

3-7. 「じっくり考えさせる」機能

設計判断、複雑なバグの原因調査、セキュリティの洗い出しなど、難しい課題では**拡張思考**（回答前に考える時間を増やす機能）が有効です。精度が上がる代わりに、時間とトークン消費は増えます。すべての作業で使うのではなく、「難しく、間違えると高くつく」場面に限定するのが基本です。なお、拡張思考を使ってもAIの誤り（ハルシネーション）はゼロにはなりません。有効化の方法やキーワードはバージョンによって変わるため、公式ドキュメントで確認してください。

第4章 開発の現場での使い方

4-1. デバッグ — 「再現するテスト」から始める

Claude Codeは、エラーの解釈と関連ファイルを横断した原因の追跡を同時にこなせます。コツは情報を惜しまず渡すことです。

- **スタックトレース（エラー発生までの呼び出し履歴）を省略せずに貼る**
- **再現手順・期待する動作・実際の動作をセットで伝える**
- **ログが大量なら、該當時刻の前後を渡して「原因候補を3つ挙げて」と頼む**

最も確実なのは、**テスト駆動の修正**です。

1. 「このバグを再現するテストを書いて。まだ修正はしないで」と頼み、テストが失敗することを確認する
2. 「テストが通るように修正して」と頼む
3. 「テスト全体を実行して、他が壊れていないか確認して」で締める

こうすると修正の正しさがテストで担保され、そのテストが将来の再発防止にもなります。

4-2. テストの自動生成 — 生成されたテストも「レビュー対象」

テスト作成は、AIに任せる効果が分かりやすい作業です。正常系・異常系・境界値（ぎりぎりの入力値）を漏れなく洗い出せます。テストの書き方（使用するフレームワーク、ファイルの置き場所、命名規則）をCLAUDE.mdに書いておけば、毎回同じスタイルで生成されます。

注意点は、**AIが書いたテストも人がレビューすること**です。何を検証しているかを理解せずに採用すると、実装の誤りを正解として固定してしまうおそれがあります。モック（本物の代わりにの仮の部品）の使いすぎにも注意します。

4-3. リファクタリング — 振る舞いを変えない変更を、テストで守る

リファクタリング（動作を変えずにコードの構造を改善すること）は、AIの「一貫して漏れなく変換する」力が活きる領域です。安全に進める手順は次の4ステップです。

1. 現状の振る舞いをテストで固める
2. 方針を提案させて確認する（関数の分割、型の強化、重複の統合など）
3. 小さな単位で実行する
4. テストを実行して、振る舞いが変わっていないことを確かめる（仕様の変更は別の作業として分けて依頼する）

第4章 開発の現場での使い方（続き）

4-4. Git操作 — 面倒な作業ほど任せやすい

Git（ソースコードの変更履歴を管理する仕組み）の操作は、手間の割に後回しにされがちな作業の代表です。Claude Codeは自然な日本語の指示をGitコマンドに変換して実行できます。

- **コミットメッセージの作成**：変更差分を読んで、何を・なぜ変えたかを書く。「変更を論理的な単位に分割して、それぞれコミットして」という依頼も可能
- **PR（プルリクエスト：変更の取り込み依頼）の作成**：概要・変更ファイル・影響範囲・テスト方法を含む本文を下書きする。GitHub CLIと組み合わせればPRの作成まで行える
- **コンフリクト（同じ箇所の変更の衝突）の解消**：両方の変更意図を読み取ったうえで統合案を出す。ただし結果は必ず人が確認する
- **リリースノート**：コミット履歴から「新機能・修正・改善」に分類して作成する

ブランチの命名規則、コミットメッセージの形式、PRに書く項目をCLAUDE.mdに定義しておけば、誰が指示しても同じ形式で履歴が残ります。

4-5. 複数ファイルにまたがる大きな変更

名前の一括変更やライブラリの移行など、多数のファイルにまたがる変更こそエージェント型の本領ですが、失敗時の影響も大きくなります。次の3つを「安全装置」としてセットにし、AIの作業を「いつでも巻き戻せる」状態に保ってください。

- **作業用ブランチを切ってから始める**：本番用のブランチで直接作業させない
- **段階的にコミットする**：「新しい定義を追加」→「実装を追加」→「既存コードを移行」→「古い定義を削除」のように分け、各段階でビルドとテストが通ることを確認させる
- **ドライランを先に行う**：変更対象の一覧と変更内容の計画だけを先に出させる（3-3の型）

図表7：変更規模別の巻き戻し方

変更規模	巻き戻し方
小規模（数ファイル・未コミット）	該当ファイルの変更を取り消す
中規模（1コミット単位）	そのコミットを打ち消すコミット（revert）を作る
大規模（複数コミット）	作業ブランチごと破棄し、元のブランチに戻る

第5章 安全に使うためのルール

5-1. 権限設定 — 「何を自動で許すか」を決める

Claude Codeは標準では、ファイルの書き換えやコマンド実行の前に承認を求めます。この確認こそが安全の要です。権限には次のような段階があります（名称や仕様はバージョンにより異なります）。

図表8：権限の段階と推奨される使い方

段階	動作	推奨する場面
計画のみ（プランモード）	ファイルを変更せず計画を立てる	大きな変更の前、初めて触るコード
都度確認（標準）	編集・コマンド実行の前に承認を求める	導入初期、本番に影響するリポジトリ
編集は自動承認	ファイル編集は自動、コマンドは確認	慣れたメンバーの日常的な開発
すべて自動	確認なしで実行	隔離された環境・CIなど限定的な用途

毎回承認している安全な操作（テスト実行など）は、設定ファイル（.claude/settings.json）の**許可ルール**に移すと確認を省略でき、触らせたくないファイルやコマンドは**拒否ルール**で防げます。数日使ってから、毎回承認しているものを許可リストへ移す順番が安全です。

5-2. 秘密情報を「AIの手の届く場所」に置かない

- APIキー等はコードに書かず環境変数や.envで管理し、.envや秘密鍵は拒否ルールで読み込みを禁止する（CLAUDE.mdにも明記）
- スキル等の定義ファイルはGitで共有されるため、秘密の値は書かず「.envから読む」手順を書く

5-3. 外部ツール連携（MCP）は最小権限で

MCP（Model Context Protocol）は、GitHub、データベース、社内システムなどを接続する標準規格です。接続先の権限をAIが使えるようになる点に注意します。

- 接続用の認証情報（トークン）の権限は最小限に。本番DBは読み取り専用で
- MCPサーバーは信頼できる提供元のものに限り、使わないものは外す
- 認証情報は環境変数で渡し、設定ファイルをコミットする前に混入がないか確認する

第5章 安全に使うためのルール（続き）

5-4. AIによるコードレビューの位置づけ

Claude Codeは、規約違反、典型的なバグ、例外処理の漏れ、型の不整合などを一貫した基準で指摘します。細かな問題を先に拾うことで、人のレビューアーは**ビジネスロジックの妥当性や設計判断に集中できます**。

依頼するときは観点と出力形式を指定すると、結果が扱いやすくなります。

直近のコミットの変更をレビューしてください。観点はバグ、性能、セキュリティ、可読性、テストの網羅性。
各指摘に深刻度（Critical／Warning／Info）と修正案を付けてください。

5-5. セキュリティの一次チェックを定期的に

専任の担当者がいない中小企業でも、基本的なチェックを定期的に回せます。

1. **全体スキャン**：コード全体を把握し、リスクの高い箇所を特定する
2. **OWASP Top 10のチェック**：Webアプリの代表的な脆弱性（インジェクション、アクセス制御の不備、認証の不備など）を体系的に確認する
3. **依存パッケージの監査**：既知の脆弱性、メンテナンスが止まったパッケージ、名前の紛らわしい偽パッケージがないか
4. **シークレット検出**：APIキーやパスワードがコードや履歴に紛れていないか
5. **対策の実装**：見つかった問題の修正案を作り、優先度順に対応する

ただし、**AIによる監査は専門家の監査や侵入テストの代替ではありません**。既知のパターンの検出には強い一方、ビジネスロジックに起因する脆弱性や、複数の要素が組み合わさって現れる問題の発見には限界があります。「専門家に見てもらおう前の下ごしらえ」と位置づけてください。

5-6. 人が必ず確認する場面を決めておく

次の操作はCLAUDE.mdに「実行前に必ず確認を取る」と明記し、権限設定でも確認が挟まる状態を残します。

- 本番環境へのデプロイ、データベースの構造変更（マイグレーション）、認証・認可に関わる変更
- 外部への送信・公開を伴う操作（メール送信、公開中サイトの変更など）
- 大量のファイル削除、Gitの履歴を書き換える操作

第6章 チームの仕組みにする

6-1. 「個人のプロンプト術」を「会社の資産」に変える

同じ長文の指示を毎回打ち込む、メンバーによって頼み方が違い品質がばらつく——使い込んだチームが次に直面するのが**再現性**の問題です。Claude Codeには、指示や手順をファイルとして保存・共有する仕組みが複数あります。

図表9：仕組み化の道具と使い分け

仕組み	ひとことで言うと	使う場面
CLAUDE.md	常に読まれる「プロジェクトの憲法」	規約・禁止事項など全タスク共通の前提
スラッシュコマンド／Skills	呼び出して使う「タスク別の手順書」	レビュー、デプロイ前チェックなどの定型作業
Hooks（フック）	必ず実行される「自動チェック」	編集後の整形、危険なコマンドの遮断
サブエージェント	別の机で働く「専門の部下」	大量の調査、レビュー専任など文脈を分けたい作業
MCP	外部ツールへの「接続口」	GitHub、DB、社内システムとの連携

6-2. 定型作業はコマンド・スキルにする

プロジェクトの.claude/フォルダにMarkdownで手順を書いておくと、「/review」「/deploy-check」のような短い名前呼び出せます（引数も渡せます）。テンプレートやスクリプトを同梱できるSkillsは、カスタムスラッシュコマンドの発展形という位置づけです。デプロイ前チェック、レビュー観点、テストやコミットメッセージの規約など、「同じ指示を3回書いたら、コマンドにする合図」です。定義ファイルはGitで管理でき、担当者が替わってもノウハウが残ります。

6-3. フックとサブエージェント

フックは、ファイル編集の後など決まったタイミングでコマンドを自動実行する仕組みです。「編集のたびにフォーマッタをかける」「特定のコマンドをブロックする」など、確実に効かせたい統制に使います。実行されるコマンドは利用者の権限で動くため、中身をレビューしてから導入してください。**サブエージェント**は別のコンテキストで働く補助役で、大規模な調査を任せてもメインの机が散らかりません。役割ごとに権限を絞る（レビュー専任は読み取りのみ等）こともできます。

第6章 チームの仕組みにする（続き）

6-4. チーム導入は4週間で段階的に

全員に一齐に配って「自由に使って」では、習熟度の差がそのまま品質の差になります。ブログでは、次のような4週間の段階導入を例として紹介しています。

図表10：4週間のオンボーディング例

週	目標	主な活動
第1週	基本操作の習得	都度確認の権限で、簡単な質問や小さな修正を依頼する
第2週	日常業務への適用	バグ修正・テスト作成に使い始める。安全な操作を許可リストへ
第3週	応用の習得	拡張思考、サブエージェント、複数ファイルのリファクタリング
第4週	チーム運用の定着	共有CLAUDE.mdの改善に参加、活用ノウハウの共有会

小さなパイロットチームで始め、ルールと権限を固めてから広げるのが安全です。

6-5. CI/CDに組み込む

CI/CDとは、コードの変更をきっかけにテストやデプロイを自動で行う仕組みです。Claude Codeは、対話せずにプロンプトを渡して結果を受け取る**ヘッドレスモード**（-pオプションによる非対話実行）でも動かせるため、GitHub Actionsなどに組み込めます。Anthropic公式のGitHub Actions連携も提供されています。

- **PR作成時の自動レビュー**：変更内容を分析し、改善提案をコメントする
- **セキュリティゲート**：変更にしークレットが含まれていないかを確認する
- **ドキュメント更新**：コードの変更に合わせてREADMEやAPI仕様書の更新案を作る

自動化する場合も、「AIの指摘を参考に人がマージを判断する」形を基本とし、AIに本番反映の最終判断を委ねないようにします。まずローカルで出力を検証し、小さなCIから始め、安定してから範囲を広げる3ステップが推奨です。

6-6. コストを見える化する

メンバー・プロジェクトごとの利用状況を定期的に共有します。無駄な消費の多くは曖昧な指示による試行錯誤と不要なファイルの読み込みから生まれるため、第3章の習慣がそのままコスト対策になります。

第6章 チームの仕組みにする（続き） — 導入ロードマップ

ここまでの内容を、導入の順番に並べ直します。ゼロから完璧を目指す必要はありません。次の4段階で、少しずつ「自社のハーネス」を育てるのが現実的です。

図表11：Claude Code導入の4段階

段階1 試す → 段階2 ルールを書く → 段階3 定型化する → 段階4 自動化・接続する

段階1：失敗しても困らない場所で試す（1～2週間）

社内ツールや検証用のリポジトリなど、本番に影響しない場所で、都度確認の権限のまま使います。「このコードを説明して」「このエラーの原因は？」「このテストを書いて」から始め、自社のコードでどこまで役立つかを確かめます。

段階2：CLAUDE.mdと権限設定を整える（2～4週間）

使ってみて気づいたルールをCLAUDE.mdに書き足し、毎回承認している安全な操作を許可リストへ、触れたくないファイルを拒否リストへ移します。人が必ず確認する操作（第5章5-6）もここで明文化します。

段階3：定型作業をコマンド・スキルにする（1～2か月）

「同じ指示を3回書いた」作業から順に、スラッシュコマンドやスキルにします。レビュー観点、デプロイ前チェック、コミット規約など、チームの暗黙知がファイルとして蓄積されていきます。

段階4：フック・MCP・CIで自動化と接続を進める（必要になってから）

チーム利用が本格化したら、フォーマッタの強制実行、GitHubや社内システムとの接続、PRの自動レビューなどを検討します。権限を広げる操作が多いため、段階2のルールが固まっていることが前提です。

非エンジニアの方へ — 小さなWebサイトから始める選択肢

Claude Codeは日本語で指示できるため、プログラミング経験のない方でも、自社サイトの文言修正や小規模なページ制作から使い始めることができます。その場合も、「変更前の状態に戻せる（Gitで記録する）」「公開前に必ず自分の目で確認する」の2点だけは守ってください。

巻末付録1 Claude Code導入チェックリスト20

【導入判断・準備】

- ☐ 1. AIに任せたい作業（調査・テスト・ドキュメント等）を具体的に挙げた
- ☐ 2. 「人が必ず判断する作業」（要件・設計・本番反映）を決めた
- ☐ 3. 最初に試す、本番に影響しないリポジトリを選んだ
- ☐ 4. 利用形態（サブスクリプション・チーム向け・API従量）を最新の公式情報で比較した
- ☐ 5. 変更をいつでも巻き戻せるよう、Gitで管理している

【初期設定】

- ☐ 6. CLAUDE.mdに、ビルド・テスト・リントのコマンドを書いた
- ☐ 7. CLAUDE.mdに、コーディング規約と「やってはいけないこと」を書いた
- ☐ 8. APIキー等は環境変数・.envで管理し、.envは.gitignoreに入れた
- ☐ 9. .envや秘密鍵を、設定ファイルの拒否ルールで読めないようにした
- ☐ 10. 権限は「都度確認」から始め、安全な操作だけを許可リストに移している

【日々の使い方】

- ☐ 11. 指示には対象ファイル・制約・期待する結果を書いている
- ☐ 12. 大きな変更は、先に計画（プランモード／ドライラン）を確認している
- ☐ 13. 1セッション1タスクを守り、区切りで要約・リセットしている
- ☐ 14. 変更には「テストも書いて実行して」を添えている
- ☐ 15. 大きな変更は作業ブランチで行い、段階的にコミットしている

【チーム運用・安全】

- ☐ 16. 共有CLAUDE.mdをGitで管理し、改善をPRでレビューしている
- ☐ 17. 繰り返す指示をスラッシュコマンド・スキルにしている
- ☐ 18. デプロイ・DB変更・認証関連の変更は、人が確認してから実行している
- ☐ 19. MCPやCIに渡す認証情報は最小権限にしている
- ☐ 20. 利用状況（コスト）を定期的に確認し、チームで共有している

8個以下の方は、まず第2章（CLAUDE.md）と第5章（権限・秘密情報）から。16個以上の方は、第6章の段階3～4（定型化・自動化）に進む準備ができています。

巻末付録2 CLAUDE.mdテンプレート例と指示の型

CLAUDE.mdテンプレート例（【 】は実際のファイルでは見出しにする）

【プロジェクト概要】

中小企業向けの予約管理システム。店舗スタッフが管理画面から予約を登録・変更する。

【技術スタック】

フロントエンド：React+TypeScript／バックエンド：Node.js／DB：PostgreSQL

【よく使うコマンド】

開発サーバー：npm run dev／テスト：npm run test／リント：npm run lint／型チェック：npm run type-check

【コーディング規約】

変数名・関数名はキャメルケース。any型は使わない。新しい関数にはユニットテストを追加する。

【Git】

ブランチ名は feature/チケット番号-概要。コミットメッセージは日本語で、種類（feat・fix等）を先頭に付ける。

【やってはいけないこと】

.envや秘密鍵ファイルを読まない・コミットしない。本番DBに接続しない。

【必ず確認を取ること】

デプロイ、DBマイグレーション、認証・権限まわりの変更、ファイルの一括削除。

【作業完了の条件】

リント・型チェック・テストがすべて通り、変更点と確認方法を要約していること。

指示の型（コピーして使う）

新機能の実装

【目的】 ○○できるようにしたい（背景：△△で困っている）

【対象・制約】 src/○○/ 配下。既存の□□を利用。変更してはいけない範囲は××

【進め方】 まず実装方針を提案し、確認後に実装。最後にテストを書いて実行

バグ修正

【症状と期待する動作】 ○○すると△△になる。本来は□□になるべき

【再現手順】 1. … 2. … 3. …

【進め方】 まず再現するテストを書いて失敗を確認し、その後に修正。テスト全体を実行して報告

おわりに — AIに任せるほど、人の判断が大切になる

Claude Codeを使いこなしているチームに共通するのは、AIに「全部任せる」ことではなく、**任せる範囲と確認する場所をはっきり決めている**ことです。指示を具体的に書き、机（コンテキスト）を片付け、変更はGitで巻き戻せるようにし、ルールと手順をファイルとして育てていく。どれも地味な習慣ですが、この積み重ねが「速い」と「安全」を両立させます。まずは本書のチェックリストで、今の状態を確かめるところから始めてください。

発行元：Harmonic Society株式会社

千葉市を拠点に、中小企業のホームページ制作・AI導入支援・IT顧問を提供しています。当社はClaude Codeを日常の開発・記事制作に使っており、本書の内容はその運用経験とブログでの解説をもとにまとめたものです。

関連するサービス

サービス	内容
AI導入支援・IT顧問	開発チームへのClaude Code導入設計、CLAUDE.md・権限ルールづくりの伴走
AIトレーニング（マンツーマン）	貴社の実務を教材にした生成AI研修
Claude Codeでホームページを作る講座	日本語で指示して自社サイトを自分で作る対面マンツーマン講座 (https://harmonic-society.co.jp/hp-course/)

まずは30分の無料相談から

「自社の開発にどう組み込めばよいか」「最初のCLAUDE.mdを一緒に作りたい」——本書を読んで浮かんだ疑問を、そのままお持ちください。売り込みはしません。

- Web：<https://harmonic-society.co.jp/contact/>
- お役立ち資料一覧：<https://harmonic-society.co.jp/whitepaper/>

本書の内容は2026年9月時点の情報に基づきます。Claude Codeの料金・機能・コマンド・設定項目は変更される場合があります。最新情報はAnthropic社の公式ドキュメントをご確認ください。掲載した手順・プロンプト例は効果を保証するものではありません。Claude、Claude CodeはAnthropic社の商標です。無断転載を禁じます。