

# Claude.ai + GitHub + Claude Code による知的生産環境構築ガイド

---

## はじめに

このガイドでは、Claude.aiを中心とした知的生産環境の構築方法を説明します。

### このガイドで何ができるようになるか

- 数週間～数ヶ月かかる長期プロジェクトを、AIと協働して進められる
- 作業の履歴を安全に保存し、いつでも過去の状態に戻れる
- セッションが変わっても、文脈を引き継いで作業を継続できる

### 所要時間の目安

初回セットアップに2～3時間

---

## 目次

1. [なぜこの環境が必要か](#)
  2. [事前準備](#)
  3. [Gitとは](#)
  4. [GitHubに保存場所を作る](#)
  5. [Claude Codeを使えるようにする](#)
  6. [プロジェクトを設定する](#)
  7. [日々の作業の進め方](#)
  8. [困ったときは](#)
  9. [付録](#)
- 

## 第1章：なぜこの環境が必要か

### 1.1 Claude.ai単体でできること

Claude.aiは強力なAIアシスタントです。質問に答える、文章を書く、コードを生成する、アイデアを整理する——これらは一回のセッション内で十分に機能します。

しかし、次のような作業をしようとする壁にぶつかります。

- 数週間かけて一冊の本を書く
- 論文を何度も推敲して完成させる
- 過去の議論を踏まえて理論を発展させる

### 1.2 Claude.ai単体の限界

#### 問題1：セッションの断絶

Claude.aiとの会話は、ブラウザを閉じたり時間が経つと途切れます。翌日「昨日の続きをやろう」と思っても、Claudeは昨日の文脈を覚えていません。

毎回「これまでの経緯は…」と説明し直す必要があります。プロジェクトが大きくなるほど、この説明コストが膨大になります。

問題2：成果物の散逸

Claude.aiで作った文章はどこに保存されるでしょうか。チャット履歴に埋もれます。

「先週書いたあの章はどこだっけ」と探し回ることになります。複数のセッションにまたがる成果物を一元管理する仕組みがありません。

問題3：変更履歴が残らない

文章を改訂したとき、「前のバージョンの方がよかった」と思うことがあります。しかしClaude.ai単体では、過去のバージョンに戻す方法がありません。

「3日前の版と比較したい」「なぜこの変更をしたか思い出したい」——これらができません。

問題4：作業の委任ができない

Claude.aiは会話相手であり、あなたのパソコン上でファイル进行操作することはできません。「このファイルをGitHubにアップロードして」と頼んでも、実行できません。

すべての作業を手動で行う必要があります。

これらの問題は、適切な環境を整えれば解決できます。

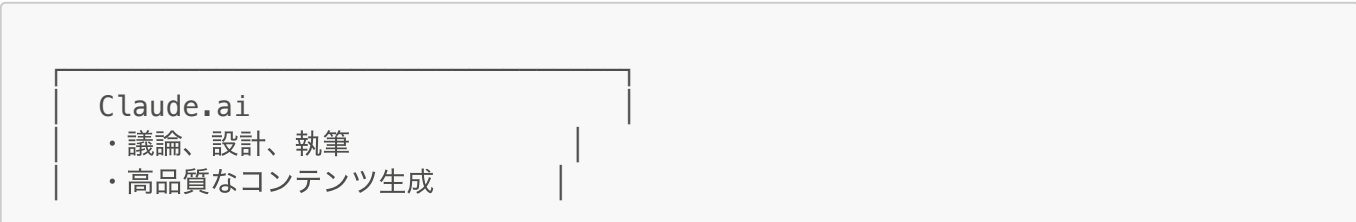
1.3 この環境で解決できること

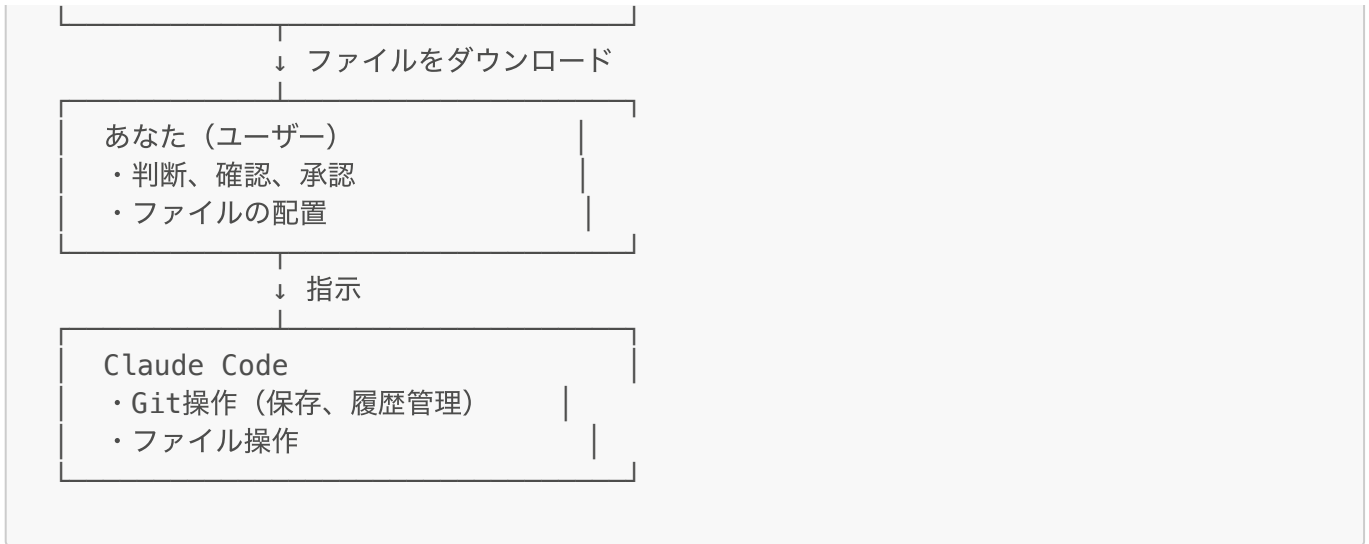
本ガイドで構築する環境は、上記の問題をすべて解決します。

問題	解決策
セッションの断絶	コンテキストファイルで文脈を引き継ぐ。新しい担当者への「引き継ぎメモ」のようなもの
成果物の散逸	GitHubリポジトリで一元管理。すべてのファイルが一箇所にまとまる
変更履歴が残らない	Gitで全履歴を保存。Wordの「変更履歴を記録」機能の強力版
作業の委任ができない	Claude Codeがファイル操作を代行。AIがあなたのパソコン上で作業できる

1.4 三者協働モデル

この環境では、3つの存在が役割分担して作業を進めます。





**Claude.ai**は「考える」担当です。議論を深め、文章を練り、構成を設計します。

**あなたは**「決める」担当です。どの方向に進むか判断し、成果物を確認し、承認します。

**Claude Code**は「動かす」担当です。これはあなたのパソコンにインストールするアプリで、ターミナル上で動きます。Claude.aiと同じAIですが、ファイル操作の実行権限を持っています。

## 1.5 具体的な作業の流れ

たとえば「論文の第2章を書く」という作業は、こう進みます。

1. **Claude.aiで執筆**：章の構成を議論し、本文を書く
2. **ダウンロード**：完成した文章をファイルとして受け取る
3. **配置**：ファイルをプロジェクトフォルダに置く
4. **Claude Codeで保存**：「この変更をGitにコミットして」と指示
5. **記録**：変更履歴がGitHubに残る

翌日、続きを書くときは：

1. **コンテキストファイルをアップロード**：昨日までの経緯をClaude.aiに伝える
2. **すぐに続きから開始**：説明し直す必要がない

実際に、このガイド自体もこの環境で執筆を進めています。

## 1.6 この環境が向いている人

- 数週間～数ヶ月かかるプロジェクトを進めたい
- 書いたものを何度も改訂したい
- 過去の作業に戻る安心感がほしい
- AIとの協働作業を効率化したい

逆に、一回きりの質問や短い文章作成なら、Claude.ai単体で十分です。

---

## 第2章：事前準備

まずClaude.aiを使えるようにします。そうすれば、この後の作業で分からないことがあっても、すべてClaude.aiに質問できます。

2.1 Claude.aiアカウントを作る

手順

- 1. <https://claude.ai> にアクセス
- 2. 「Sign up」 をクリック
- 3. メールアドレスまたはGoogleアカウントで登録
- 4. メール認証を完了

所要時間：約5分

プランを選ぶ（執筆時点）

プラン	月額	特徴
Free	無料	お試し用。メッセージ数に制限あり
Pro（推奨）	\$20	長期プロジェクト向け。Claude Codeも快適に使える
Max	\$100	Opus 4.5が使える。最高の応答品質

Freeではメッセージ数の制限にすぐ達します。長期プロジェクトにはPro以上を推奨します。Opus 4.5は現時点で最も高性能なモデルで、複雑な議論や長文の執筆で差が出ます。予算に余裕があればMaxを選ぶ価値があります。

最新の料金は <https://claude.ai> で確認してください。

2.2 このガイドをClaude.aiに読み込ませる

ここからは、Claude.aiと一緒に進めます。

このガイドのファイル（Markdownファイル、拡張子は.md。テキストファイルの一種です）を、Claude.aiに読み込ませてください。

手順

- 1. このガイドのファイルを保存する
- 2. Claude.aiを開く
- 3. チャット欄にファイルをドラッグ&ドロップする
- 4. 以下のメッセージを送る：

このガイドに沿って環境構築を進めたいです。  
ステップバイステップで教えてください。

- 5. Claude.aiの案内に従って、続きを進める

必要に応じて、Claude.aiがあなたの状況（使っているパソコン、経験など）を質問してきます。そのまま答えてください。

GitHubアカウントの作成、Gitのインストール、その他の準備は、すべてClaude.aiが案内します。

## 質問するコツ

分からないことがあったら、遠慮なく質問してください。

- 「ターミナルを開いたら、こういうエラーが出ました：（エラーメッセージを貼り付け）」
- 「『リポジトリ』という言葉の意味が分かりません」

エラーメッセージや画面の状態を具体的に伝えと、的確な回答が返ってきます。

## 補足：長期プロジェクトには「プロジェクト機能」

Claude.aiには「プロジェクト」という機能があり、ファイルを常に読み込んだ状態で対話できます。詳しくは第6章で説明します。

## 2.3 このガイドの読み方

第3章以降は「エッセンス」と「もっと知りたい人へ」に分かれています。

- **エッセンス**：必ず読む最小限の情報。Claude.aiへの指示文が含まれています
- **もっと知りたい人へ**：概念の詳しい説明や背景情報。読み飛ばしても進められます

急いでいる場合は「エッセンス」だけ読んで、Claude.aiの案内に従ってください。詳細はClaude.aiが必要に応じて教えてくれます。

## 2.4 この章のチェックリスト

- ☐ Claude.aiアカウントを作成した
- ☐ プランを決めた（Pro推奨）
- ☐ このガイドをClaude.aiに読み込ませた
- ☐ Claude.aiと対話できる状態になった

**準備完了です。** 分からないことがあっても大丈夫。Claude.aiがあなたの味方です。

この後の作業は、Claude.aiの案内に従って進めてください。

---

## 第3章：Gitとは

### エッセンス

Gitは、ファイルの変更履歴を記録するツールです。過去の状態にいつでも戻れる安心感が得られます。

GitHubは、その記録をインターネット上に保存するサービスです。

### やること

Claude.aiに伝えてください：

Gitをインストールしたいです。

もっと知りたい人へ

## なぜGitが必要か

長期プロジェクトを進めていると、こんなことが起きます：

- 月曜日：第1稿を作った
- 水曜日：大幅に作り直した
- 金曜日：やっぱり月曜日の版の方がよかった

Gitがあれば、月曜日の版に戻せます。いつ、何を変更したかがすべて記録されているからです。

Wordの「変更履歴を記録」機能に似ていますが、Gitはもっと強力です。文章でもコードでも、どんな種類のファイルでも管理できます。

## GitとGitHubの違い

この2つは混同しやすいので、整理します。

**Git**は、あなたのパソコンで動くツールです。変更履歴を記録します。

**GitHub**は、その記録をインターネット上に保存するサービスです。

なぜGitHubを使うのか。理由は2つあります：

- **バックアップ**：パソコンが壊れてもデータが残る
- **連携**：Claude.aiやClaude Codeと一緒に使える

第1章で説明した「三者協働モデル」を実現するために、GitHubは欠かせません。

## Gitのコマンド

Gitには多くのコマンドがありますが、今は覚えなくて大丈夫です。Claude.aiに「保存して」「GitHubに送って」と言えば、何をすべきか教えてくれます。

コマンド一覧は付録Bにまとめてあります。

## この章のチェックリスト

- ☐ Gitが何をするツールか理解した
- ☐ Gitがインストールされている

完了したら、第4章に進みます。

---

## 第4章：GitHubに保存場所を作る

### エッセンス

GitHubにリポジトリ（保存場所）を作り、あなたのパソコンと接続します。

### やること

Claude.aiに伝えてください：

1. 「GitHubアカウントを作りたいです。」（まだの場合）
2. 「GitHubに新しいリポジトリを作りたいです。プロジェクト名は『〇〇〇』です。」
3. 「リポジトリをローカルに接続したいです。」

「〇〇〇」には、あなたのプロジェクト名を入れてください（例：`my-writing-project`）。

もっと知りたい人へ

## リポジトリとは

リポジトリは、プロジェクトのファイルと変更履歴をまとめて保存する場所です。

「タイムマシン付きフォルダ」と考えてください。普通のフォルダは今あるファイルだけが入っていますが、リポジトリには過去のすべての状態も保存されています。いつでも過去に戻れます。

1つのプロジェクトにつき、1つのリポジトリを作るのが基本です。

## プロジェクト名のつけ方

- `my-writing-project`
- `research-notes`
- `kesson-project`

英数字とハイフン（-）を使うのがおすすめです。日本語は避けてください。

## 公開か非公開か

リポジトリを作るとき、公開（Public）か非公開（Private）かを選びます。

- **公開**：誰でも見られる。オープンソースやポートフォリオ向け
- **非公開**：自分だけが见られる。個人プロジェクト向け

迷ったら**非公開**を選んでください。未完成の作品を他人に見られる心配がありません。後から公開に変更することもできます。

## ローカルとの接続

GitHubにリポジトリを作っただけでは、まだあなたのパソコンとつながっていません。

パソコンのフォルダとGitHubのリポジトリをつなげる作業が必要です。一度つなげれば、作業内容をGitHubに送ったり、GitHubから取得したりできるようになります。

## この章のチェックリスト

- ☐ GitHubアカウントを持っている
- ☐ リポジトリを作成した
- ☐ ローカルとの接続を完了した

**これで、あなたの作業を安全に保存できる場所ができました。**

完了したら、第5章に進みます。

---

## 第5章：Claude Codeを使えるようにする

エッセンス

Claude Codeは、ファイル操作やGit操作ができるAIです。これが使えると、作業が劇的に楽になります。

やること

Claude.aiに伝えてください：

- 1. 「Claude Codeをインストールしたいです。」
- 2. 「Claude Codeの認証をしたいです。」
- 3. 「Claude Codeが正しく動くか確認したいです。」

もっと知りたい人へ

Claude.aiとClaude Codeの違い

Claude.aiとClaude Codeは、同じAI（Claude）ですが、できることが違います。

	Claude.ai	Claude Code
どこで動く	ブラウザ	あなたのパソコン
操作方法	チャット画面	ターミナル
ファイル操作	—	できる
Git操作	—	できる

一言でいうと、Claude.aiは「考える」専門、Claude Codeは「考える＋動かす」ができます。

ターミナルでの対話はこんな感じです。ターミナルに「このフォルダにREADME.mdを作って」と入力すると、Claude Codeが実際にファイルを作成してくれます。

なぜClaude Codeが必要か

Claude.aiで文章を書いた後、こんな作業が発生します：

- ファイルをダウンロードする
- 適切なフォルダに置く
- Gitで変更を記録する
- GitHubに送る

これらを毎回手動でやるのは面倒です。

Claude Codeがあれば、「この内容でファイルを作って、GitHubに送って」と言うだけで、すべて実行してくれます。あなたは指示を出すだけ。実際の作業はClaude Codeが担当します。

この章のチェックリスト

- ☐ Claude Codeをインストールした
- ☐ 認証を完了した
- ☐ Claude Codeに話しかけて、応答が返ってきた

これで、AIにファイル操作を任せられるようになりました。

完了したら、第6章に進みます。

---

## 第6章：プロジェクトを設定する

### エッセンス

プロジェクト機能で、ファイルを常に読み込んだ状態で対話できます。コンテキストファイルで、作業の経緯を引き継げます。

### やること

Claude.aiに伝えてください：

1. 「Claude.aiでプロジェクトを作りたいです。」
2. 「プロジェクトにナレッジを追加したいです。」
3. 「コンテキストファイルのテンプレートを作ってください。私のプロジェクトは『〇〇〇』です。」

「〇〇〇」には、あなたのプロジェクトの簡単な説明を入れてください。

### もっと知りたい人へ

#### プロジェクト機能とは

ファイルをチャットにドラッグ&ドロップする方法は手軽ですが、長期プロジェクトにはもっと便利な方法があります。

プロジェクト機能を使うと：

- ファイルを常に読み込んだ状態で対話できる
- 複数のファイルをまとめて管理できる
- チャットが変わっても、設定が引き継がれる

「このプロジェクト専用の作業スペース」を作るイメージです。

短期の作業ならドラッグ&ドロップで十分です。数週間以上続くプロジェクトには、プロジェクト機能をおすすめします。

#### ナレッジとは

プロジェクトには「ナレッジ」としてファイルを追加できます。Claude.aiでは、プロジェクトに追加するファイルをこう呼びます。

追加したファイルは、そのプロジェクト内のすべてのチャットで参照されます。

追加するとよいもの：

- プロジェクトの概要や方針
- 用語集や命名ルール
- 過去の作業履歴（コンテキストファイル）

#### コンテキストファイルとは

コンテキストファイルは、「引き継ぎメモ」のようなものです。これが長期プロジェクトの鍵になります。

新しいチャットを始めるとき、Claude.aiは過去の経緯を知りません。コンテキストファイルをナレッジに追加しておけば、Claude.aiは最初から状況を把握した状態で対話を始められます。

コンテキストファイルに書く内容：

- プロジェクトの目的
- これまでに完了した作業
- 現在の状態
- 次にやるべきこと

テンプレートは付録Cにあります。

### コンテキストファイルの更新

コンテキストファイルは、作業が進むたびに更新が必要です。自動では更新されません。更新のタイミングや方法は、第7章で詳しく説明します。

### この章のチェックリスト

- ☐ プロジェクトを作成した
- ☐ ナレッジにファイルを追加した
- ☐ コンテキストファイルを作成した

**これで、長期プロジェクトを効率的に進める準備が整いました。**

完了したら、第7章に進みます。

---

## 第7章：日々の作業の進め方

### エッセンス

日々の作業は、このサイクルで進みます：

1. 作業を始める  
↓
2. Claude.aiで対話・執筆  
↓
3. 成果物を保存  
↓
4. 作業を終える  
↓  
(翌日、1に戻る)

### やること

このサイクルを一度試してください。分からないことがあれば、Claude.aiに聞いてください。

もっと知りたい人へ

## 作業を始める

プロジェクトを開いて、作業を開始します。

コンテキストファイルがナレッジに入っていれば、Claude.aiは状況を把握しています：

作業を再開します。前回の続きから進めてください。

前回から状況が変わっている場合は、補足してください：

作業を再開します。  
前回の後、第3章の見出しを自分で考えました。  
今日は本文を書きたいです。

初めて作業を始めるときは：

今日から作業を始めます。まず何をすればよいですか？

## Claude.aiで対話・執筆

作業中は、自由にClaude.aiと対話してください。何を聞いても構いません。

例：

第4章の構成を一緒に考えてください。

この文章をレビューしてください。分かりにくい部分を指摘してください。

## 成果物を保存

Claude.aiで作成した文章やファイルを保存します。

### 方法1：Claude.aiからダウンロードして保存

1. Claude.aiが生成したファイルをダウンロード
2. 適切なフォルダに配置
3. Claude Codeに指示：

この変更をGitにコミットして、GitHubに送ってください。  
コミットメッセージは「第3章の導入部分を追加」です。

## 方法2：Claude Codeに直接作成させる

Claude.aiで内容を確定した後、Claude Codeに指示：

以下の内容で「chapter3.md」を作成して、GitHubに送ってください。  
コミットメッセージは「第3章の導入部分を追加」です。

(内容をペースト)

最初は方法1から始めてください。ファイルの流れが分かりやすいです。慣れてきたら方法2を試すと、作業が速くなります。

### コミットメッセージについて

「何を変更したか」を書いておくと、後で履歴を見たときに分かりやすくなります。短くて構いません。例は付録Cにあります。

### 作業を終える

作業を終えるとき、コンテキストファイルを更新します。

更新のタイミング（毎回更新する必要はありません）：

- 章やセクションが完成したとき
- 大きな方針変更があったとき
- 数日間作業を空けるとき

更新の方法：

今日の作業をコンテキストファイルに反映してください。

Claude.aiが更新版を生成します。内容を確認して、ナレッジのファイルを新しいものに入れ替えてください。入れ替え方が分からなければ、Claude.aiに聞いてください。

### 更新を忘れないために

プロジェクトには「カスタムインストラクション」という設定があります。ここに指示を書いておくと、Claude.aiが常にその指示に従います。

以下を設定しておくとう便利です：

作業の区切りで、コンテキストファイルの更新を提案してください。

設定方法はClaude.aiに聞いてください：

プロジェクトのカスタムインストラクションを設定したいです。

### この章のチェックリスト

- ☐ 作業を始めて、終える流れを一度試した
- ☐ 成果物をGitHubに保存できた
- ☐ コンテキストファイルを更新できた

これで、明日からプロジェクトを本格的に進められます。

完了したら、第8章に進みます。

## 第8章：困ったときは

この章では、つまずきやすいポイントと解決方法を紹介します。

### 8.1 基本の対処法

問題が起きたら、まずClaude.aiに状況を伝えてください。

こういう問題が起きています：（状況を説明）

エラーメッセージが出ている場合は、そのまま貼り付けてください：

このエラーが出ました：  
（エラーメッセージをコピー＆ペースト）

画面の状態を伝えたい場合は、スクリーンショットを添付してください。画像はチャット欄にドラッグ&ドロップできます。

### スクリーンショットのショートカット

OS	ショートカット	動作
Mac	Command + Shift + 4	範囲を選択して撮影
Mac	Command + Shift + 3	画面全体を撮影
Windows	Windows + Shift + S	範囲を選択して撮影

ほとんどの問題は、これで解決できます。

### 8.2 Claude.aiで解決しにくい問題

以下の問題は、Claude.aiに聞いても解決できません。

#### メッセージの上限に達した

プランの制限に達しています。時間を置くか、プランのアップグレードを検討してください。

Claude.aiやClaude Codeが動かない

サービス自体に障害が起きている可能性があります。しばらく待ってから再度試してください。公式の状況は以下で確認できます：

- <https://status.anthropic.com>

8.3 確認用コマンド（参考情報）

Claude Codeで状況を確認するときに便利なコマンドです。覚えなくても、Claude.aiに「確認して」と言えば教えてくれます。

確認したいこと	コマンド
Gitの状態	<code>git status</code>
変更履歴	<code>git log --oneline</code>
Claude Codeの診断	<code>claude doctor</code>

8.4 それでも解決しない場合

検索する

エラーメッセージをそのままWeb検索すると、解決策が見つかることがあります。

最初からやり直す

どうしても解決しない場合、その部分だけ最初からやり直すのも手です。Gitで履歴が残っているので、作業が失われることはありません。安心してやり直せます。

8.5 この章について

この章は、必要なときに見返すためのものです。チェックリストはありません。

問題は必ず解決できます。焦らず、一つずつ試してください。

付録

付録A：用語集

用語	意味
Git	ファイルの変更履歴を記録するツール
GitHub	Gitの記録をインターネット上に保存するサービス
リポジトリ	プロジェクトのファイルと変更履歴をまとめた保存場所
コミット	変更を確定して記録すること
プッシュ	ローカルの変更をGitHubに送ること
プル	GitHubの変更をローカルに取得すること

用語	意味
ローカル	あなたのパソコン上のこと
リモート	インターネット上（GitHub）のこと
ナレッジ	Claude.aiのプロジェクトに追加するファイル
コンテキストファイル	プロジェクトの状況をまとめた引き継ぎメモ

付録B：コマンド一覧

Git

コマンド	意味
<code>git status</code>	現在の状態を確認
<code>git add .</code>	すべての変更を記録対象にする
<code>git commit -m "メッセージ"</code>	変更を確定する
<code>git push</code>	GitHubに送る
<code>git pull</code>	GitHubから取得する
<code>git log --oneline</code>	変更履歴を一覧表示

ターミナル基本

コマンド	意味
<code>cd</code> フォルダ名	フォルダに移動
<code>ls</code> (Mac) / <code>dir</code> (Windows)	ファイル一覧を表示
<code>claude</code>	Claude Codeを起動
<code>claude doctor</code>	Claude Codeの問題を診断

付録C：テンプレート集

Markdownの基本：**#**は見出し、**-**は箇条書きです。

コンテキストファイル

<pre># プロジェクト名  ## 目的 （このプロジェクトで達成したいこと）  ## 完了した作業 - （完了した作業1） - （完了した作業2）</pre>
---

## ## 現在の状態

(今どこまで進んでいるか)

## ## 次にやること

- (次のタスク1)
- (次のタスク2)

## ## メモ

(覚えておきたいこと、決定事項など)

## カスタムインストラクション

作業の区切りで、コンテキストファイルの更新を提案してください。

## コミットメッセージの例

第1章を追加  
第2章の構成を修正  
誤字を修正  
図を追加

---

## おわりに

お疲れさまでした。

これで環境構築は完了です。あなたは以下を手に入れました：

- Claude.aiとの継続的な対話環境
- GitHubによる安全なバックアップ
- Claude Codeによる作業の自動化
- コンテキストファイルによる引き継ぎの仕組み

この環境があれば、長期プロジェクトを安心して進められます。困ったときはいつでもClaude.aiに相談してください。

良いプロジェクトを。