

Clineの基本と プロンプトエンジニアリング

開発効率と品質を高める実践ガイド

目次

第1章 Clineの基本的な利用方法

P3-P13

- 1 Clineとは（概要導入）
- 2 ActモードとPlanモード
- 3 仕組みと動作フロー
- 4 実行できる操作
- 5 権限管理とAuto-approve
- 6 ファイル変更の承認フロー
- 7 変更を元に戻す方法
- 8 コンテキストの追加方法
- 9 ワークフロー機能の活用
- 10 .clinerules設定

第2章 プロンプトエンジニアリングの要諦

P14-P19

- 1 抽象度とトレードオフ
- 2 Simple/Just/Much分類
- 3 レビューとの兼ね合い
- 4 ユースケース（前半）
- 5 ユースケース（後半）

ご参考：プライベート環境での学習継続について

P20

セットアップ手順とリポジトリ情報

第1章

Clineの基本的な利用方法

VS Code上のAIコーディングエージェントの活用と安全な運用



対象

VS Code上のClineの基本操作を体系的に理解したい開発者



主な内容

Act/Planの使い分け、操作体系、権限管理、承認フロー、変更復元、コンテキスト追加、ワークフロー機能、設定ファイル

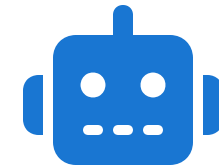


到達目標

実務環境で安全かつ効率的にClineを運用できるようになる

Clineとは（概要導入）

- VS Codeで動作するAIコーディングエージェント
- LLMに基づき、適切なツールで操作を自動化
- ファイル/ターミナル/ブラウザを横断して実行
- Act（即実行）とPlan（計画→承認）で安全・迅速を両立
- 現場運用を想定した権限管理と承認フローを提供



ActモードとPlanモードの使い分け



Actモード（実行）

- すぐに変更・実行を開始
- 明確なタスクや短時間作業に最適
- 実装方針が固まっている場合



Planモード（計画）

- 実行前に計画を提示→確認→修正
- 複雑・不明点あり・大規模変更向け
- 安全性・レビュー重視の場面で推奨

Clineの仕組みと動作フロー

- **1. LLMへの問い合わせ:** プロンプトがLLMに送信される
- **2. ツールの選択:** LLMが必要な操作を判断し、適切なツールを選択
- **3. 操作の実行:** 選択されたツールを使って実際の操作を実行
- 実行結果はコンテキストに反映され、次の判断材料に活用



LLMへの問い合わせ



ツールの選択



操作の実行

Clineが実行できる操作

ファイル操作

- 読み込み/編集/作成/削除、ディレクトリ検索

ターミナル操作

- npm・pip・git等の実行、ビルド/テスト/サーバー起動
- 出力をClineにフィードバック

ブラウザ操作

- WebドキュメントやAPI仕様の閲覧
- スクリーンショット取得・Web動作確認



ファイル操作



ターミナル操作



ブラウザ操作

権限管理とAuto-approve

- 各操作はユーザーの承認が必要（初回/重要操作）
- Auto-approve: Act中の変更を自動承認に切替可能
- 単純・安全な変更では有効、重要変更は手動承認推奨
- セキュリティとコントロールのバランスが要点

Auto-approve: Read, All Commands, Browser ▼

Let Cline take these actions without asking for approval. [Configure notification settings](#)

- ☒ 🔍 Read project files
- ☐ 📁 Read all files
- ☐ ✎ Edit project files
- ☒ ▶️ Execute safe commands
- ☒ 🏠 Execute all commands
- ☒ 🌐 Use the browser
- ☐ 🖨️ Use MCP servers

⚠️ 重要な変更時は手動承認を推奨します

ファイル変更の承認フロー

✓ Save

✗ Reject

- **Save:** 変更を承認・適用
- **Reject:** 変更を拒否・再提案を依頼

■ 追加された行 ■ 削除された行

確認ポイント

- コードの意図が正しいか
- 副作用やセキュリティ影響がないか
- 既存機能や依存関係への影響を確認

142 143 144 145 146 147 148 149	<pre>// 現在の在庫数を取得する Stock currentStock = stockDao.fin</pre>	142 143 144+ 145+ 146+ 147+ 148+ 149+ 150+ 151+ 152+ 153+ 154+ 155+ 156+ 157 158 159 160 161+ 162	<pre>// 現在の在庫数を取得する Stock currentStock = stockDao.findById(cartIt // 在庫チェック：在庫数が注文数に満たない場合 if (currentStock.getQuantity() < cartItem.get // 書籍情報を取得 Book book = bookDao.findById(cartItem.getI String bookName = book.getBookName(); // 在庫不足例外をスロー throw new OutOfStockException(cartItem.getBookId(), bookName, "在庫不足: " + bookName); } // 楽観的ロックのために現在の最新VERSIONをセッ stock.setVersion(currentStock.getVersion()); // 在庫を減らす int remaining = currentStock.getQuantity() - c</pre>
--	---	---	--

変更を元に戻す方法

- **方法1：Clineの履歴機能を使う**

- 「元に戻して」「Undo」などと指示すると復元
- セッション内であればGitを使わずに戻せる

- **方法2：Gitで復元（推奨）**

- `git diff`でClineの変更を確認
- `git checkout <ファイル>`や`git reset --hard`で元に戻す
- 作業前に`git commit`でスナップショットを残す
- 実験的な作業は`git checkout -b <ブランチ名>`で実施

- **安全な運用のポイント**

- Cline履歴＋Git履歴の二重管理で安心



コンテキストの追加方法

- **Shift** + ドラッグ & ドロップでファイル/フォルダ追加
- チャット欄で **@** を入力→候補一覧が表示
 - **@file**：特定ファイルを追加
 - **@folder**：フォルダ全体を追加
 - **@terminal**：ターミナル出力を追加
- 必要情報を都度追加して精度向上



コンテキスト追加のイメージ

@ file: src/main/java/App.java



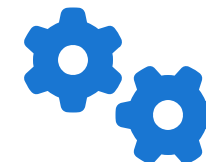
ドラッグ & ドロップまたは@コマンドで追加

ワークフロー機能の活用






- 定型タスクをショートカットで実行できる機能
- 「/」を入力すると利用可能なワークフロー一覧が表示される
- 配置場所: `.clinerules/workflows/` フォルダ
- YAML または Markdown で定義可能

活用例

- コードレビューリクエスト
- テストコード生成
- ドキュメント作成
- デプロイチェックリスト



ワークフロー一覧

-  /review-code
-  /generate-tests
-  /create-docs
-  /deploy-check
-  /fix-bugs

.clinerules設定

プロジェクトごとのルール・ガイドラインを定義し、Clineの提案基準を統一して品質を安定化します。

設定例

- コーディング規約/命名ルール
- 禁止ライブラリ/関数
- 推奨デザインパターン
- テスト要件/カバレッジ基準

メリット

- 一貫したAI提案・コード生成
- チーム開発でのルール統一
- 新メンバーのオンボーディング容易化

ファイル配置

- プロジェクトルートに.clinerulesディレクトリを作成
- 設定ファイル：config.ymlなど
- ワークフロー：workflows/ディレクトリ

2 プロンプトエンジニアリングの要諦



対象：プロンプト設計で精度と生産性を高める



目次：抽象度とトレードオフ、Simple/Just/Muchの分類、レビュー観点との整合



到達目標：タスク特性に応じて最適な具体度・観点を設計できる

抽象度とトレードオフ（プロンプト）

抽象的プロンプト

- 作成負荷: **小さい**（短く簡単）
- AI委譲範囲: **広い**（AIが多くを判断）
- 精度: ばらつきが生じる
- 探索的な作業、方向性を決める段階に有効

具体的プロンプト

- 作成負荷: **大きい**（詳細な記述が必要）
- AI委譲範囲: **小さい**（人間が多くを指定）
- 精度: 向上する
- 本番コード、重要な修正に最適

最適化の観点

- コストパフォーマンス（プロンプト作成時間 vs 出力精度）
- タスクの性質・重要度・時間制約で判断
- 万能なレベルは存在しない

プロンプトの分類：Simple / Just / Much



Simple（最も抽象）

- ~20行程度の短いプロンプト
- 迅速に指示可能
- 出力のばらつきが大きい
- 探索・試作・アイデア出しに最適



Just（バランス）

- 20～100行程度
- 作成負荷と精度のバランスが良い
- 状況判断が必要
- 通常の開発タスクの大半に適用



Much（最も具体）

- 数百行の詳細な記述
- 高精度な出力が可能
- プロンプト作成に時間がかかる
- 本番実装・複雑な要件に最適

レビューとの兼ね合い

- AI駆動開発ではレビューアの負担が重くなる課題が存在
- レビューには必ず「観点」があるため、それをプロンプトに反映させることが重要
- Clineの場合、.clinerulesでレビュー観点（設計標準やコーディング規約）を定義可能
- 事前に観点を提示すると再修正コストを低減できる

AI開発サイクル

プロンプト → コード生成 → レビュー → 修正

.clinerules観点の事前定義

プロンプト → 高品質コード → 軽量レビュー



clinerules.yaml

```
coding_standards: true
design_patterns: ["Factory", "Observer"]
test_coverage: 80%
```

ユースケース（前半）

berry-books（Jakarta EE）

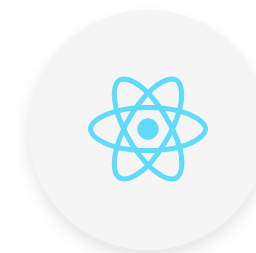
- L1: 小規模改善
- L2: ガイドラインへの準拠チェック
- L3: 不具合修正
- L4: 機能拡張
- L5: 単体テスト生成

berry-books-frontend（React）

- L6: 仕様書からのスクラッチ開発（SPA管理者画面）



Jakarta EE



React

ユースケース（後半）

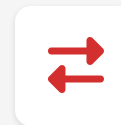
struts-to-jsf-person

- L7: Struts→JSFのマイグレーション（リライト）
- レガシーフレームワークから現代的フレームワークへの移行
- コンポーネント指向設計への変換

accounting_glue（PySpark/Glue）

- L8: 仕様書からのスクラッチ開発（ETL／会計EPR連携）
- 大規模データ処理パイプラインの構築
- AWS環境でのバッチ処理最適化

技術スタック



Struts



JSF



PySpark



AWS Glue



ETL

ご参考：プライベート環境での学習継続について



セットアップ①：ツールおよび言語のインストール

- VSCode
- Cline (VSCodeプラグイン)
- Git
- JDK：JDK 21
- Python：Python 3.11



セットアップ②：Gitリポジトリからのクローン

- リポジトリ名：KenyaSaitoh/cline_training_work
- Git (Git Bashなど) を起動し、以下のコマンドで任意のディレクトリにクローン

```
git clone  
https://github.com/KenyaSaitoh/cline_training_work.git
```

- リポジトリ内には各種ミドルウェア (Payara、Hsqldb、TomEEなど) が最初から配置されているためすぐに利用可能



セットアップ③：API KEYの取得

- AnthropicのAPI KEYを以下のサイトより取得

```
https://console.anthropic.com/settings/keys
```

- 5ドルの無料枠あり
- 取得したAPI KEYをClineに設定する