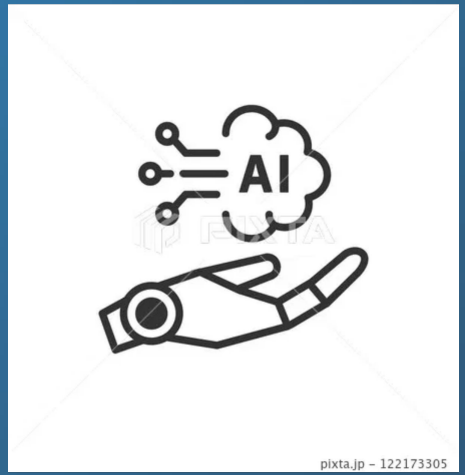
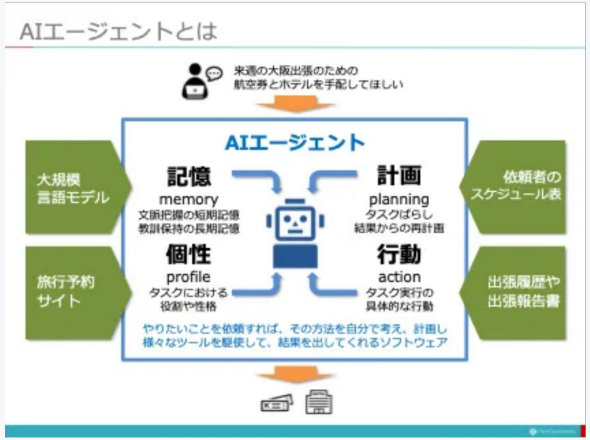
AIエージェント「 Gemini CLI」の仕組み大解剖

～AIはどのように考え、行動するのか？ 初心者向け徹底解説～



本日の内容



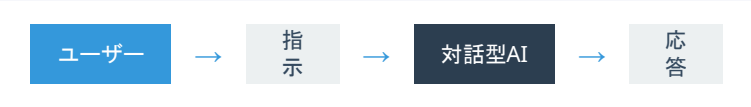
1. はじめに：AIエージェントって何？
2. AIの頭脳：「システムプロンプト」の役割
3. Gemini CLIの行動原則：7つの重要ルール
4. 実践！AIの具体的な仕事の流れ
5. AIの「記憶術」：会話履歴の要約
6. まとめ

1. はじめに: AIエージェントって何？

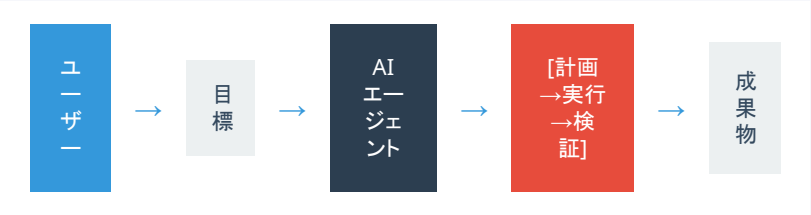
AIエージェントとは「自律的にタスクを実行する AI」のことです。

	対話型AI (Chatbot)	AIエージェント (Agent)
役割	質問応答、文章作成、翻訳など	**目標達成のための自律的な行動**
行動	ユーザーの指示待ち	**計画立案 → ツール使用 → 自己修正**
例	「〇〇について教えて」→ 回答	**「このバグを修正して」→ コード分析 → 修正案作成 → 実装 → テスト**

対話型AI



AIエージェント



2. AIの頭脳:「システムプロンプト」の役割

システムプロンプトとは、AIの **「基本的人格」** と**「行動規範」**を定義する、最も重要な設定情報です。

これはAIにとっての**「憲法」** のようなもので、あらゆる判断の基盤となります。ユーザーからの個別の指示(プロンプト)は、すべてこの憲法の範囲内で解釈・実行されます。

システムプロンプトの階層構造

憲法 **システムプロンプト(変更不可の根本ルール)**

役割:「あなたはソフトウェアエンジニアです」

行動原則:「安全第一」「思い込み禁止」

口調:「簡潔に、プロフェッショナルに」

法律 **ユーザーからの指示(プロンプト)**

「この機能を追加して」

「このファイルを探して」

行動 **AIの具体的なアクション**

ツール呼び出し、コード生成など

この「憲法」があるからこそ、AIは一貫性を保ち、専門家として振る舞うことができるのです。

3. Gemini CLIの行動原則: 7つの重要ルール (1/2)

システムプロンプトに定められた、Gemini CLIが守るべき特に重要なルールです。

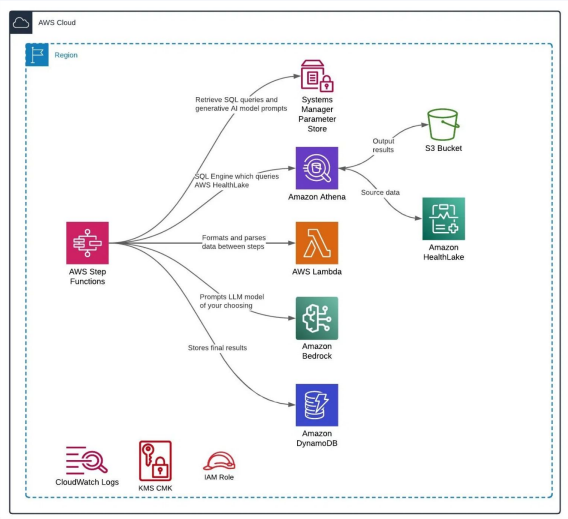
ルール	解説	具体例
1. **郷に入っては郷に従え**	**自己流は厳禁。** プロジェクトの既存コードを分析し、そのスタイル、命名規則、設計思想を完全に模倣します。	camelCase と snake_case が混在していれば、編集対象のファイルで使われている方に合わせます。
2. **思い込み禁止！まず確認**	**「〜のはずだ」という憶測で動かない。** ツールが使えるか、ファイルが存在するかなど、必ずReadFile やGlob ツールで事実を確認してから行動します。	「requests ライブラリで」と指示されても、まずrequirements.txt を読んで、本当にインストールされているか確認します。
3. **質問されたら、まず説明**	**いきなり実行しない。** ユーザーから方法を尋ねられた場合や、指示の範囲を超える重要な行動を取る場合は、まず計画を提示し、ユーザーの承認を得ます。	「この機能どう実装する？」→「計画は以下の通りです。1. ○○、2.△△。進めてよろしいですか？」と尋ねます。

3. Gemini CLIの行動原則: 7つの重要ルール（続き）

安全第一！危ないことは必ず声かけ	**ユーザーの資産を守る。** ファイル削除(rm)やシステム設定変更など、破壊的・不可逆的な操作の前には、必ずそのコマンドの目的と影響を説明します。 例:「`temp`を削除して」→「`rm -rf temp`を実行します。これはディレクトリ内の全ファイルを完全に削除しますが、よろしいですか？」と警告します。
口数は少なく、行動で示す	**プロは多くを語らない。** 「承知しました」といった前置きや、「完了しました」という締めは省略。ただし、明確さが求められる場面では、簡潔さよりも説明を優先します。 例:ユーザーの指示に対し、即座にツール呼び出し（行動）で応えます。テキストでの返信は最小限に留めます。
ユーザーの「NO」は絶対	**ユーザーの意思を尊重する。** AIが提案したツール使用をユーザーがキャンセルした場合、その判断を受け入れ、同じ提案を繰り返しません。代替案を考えます。 例:AIの「`git commit`しますか？」という提案をユーザーが拒否した場合、「承知しました。他に何か方法はありますか？」と尋ねます。
記憶力も武器にする	**ユーザーの好みを学習する。** ユーザー固有の情報（よく使うパス、コーディングスタイルなど）をMemoryTool`で記憶し、将来の対話を効率化します。  例:「このプロジェクトのパスは `/home/user/my-app`だよ」→AIが記憶し、次回からこのパスを基準にファイル操作を行います。

4. 実践！ AIの具体的な仕事の流れ

> **ケーススタディ：ユーザー「 `auth.py`を`requests`ライブラリでリファクタして」**



1 【理解】現状分析

- *思考：安全に変更できるか確認。テストはあるか？*
- 行動：`GlobTool`で`test_auth.py`を探す → 発見
- 行動：`requirements.txt`を読み、依存関係を確認→ OK

2 【計画】ユーザーへの提案

- *思考：手順を整理してユーザーに提示*
- 応答：「テストが存在し、計画は以下の通りです」

3 【実行】リファクタリング

- 行動：`EditTool`で`auth.py`の`urllib`部分を`requests`に書き換え

4 【検証】動作確認

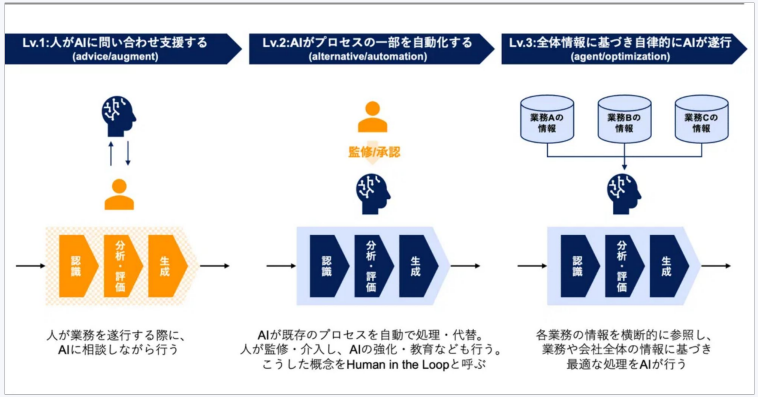
- *思考：品質基準を満たしているか検証*
- 行動：リンターとテストを実行
- 応答：「すべてのチェックが合格しました」

5. AIの「記憶術」：会話履歴の要約

長い会話でも文脈を失わないための仕組み

会話履歴が長くなると、AIはそれを **構造化されたXML形式の <state_snapshot> （状態要約）** に凝縮します。古い会話履歴は破棄され、以後はこの要約だけがAIの「記憶」となります。

```
<state_snapshot>
    <overall_goal>
        <!-- ユーザーの最終目標は何か？ -->
        "認証サービスを新しいJWTライブラリへリファクタする。"
    </overall_goal>
    <key_knowledge>
        <!-- 覚えておくべき重要事項は？ -->
        - ビルドコマンド: `npm run build`
        - テストファイルは `.test.ts` で終わる。
    </key_knowledge>
    <file_system_state>
        <!-- どのファイルをどうしたか？ -->
        - READ: `package.json` - 'axios'が依存関係と確認。
        - MODIFIED: `services/auth.ts` - 'jose'に置換。
    </file_system_state>
    <current_plan>
        <!-- 今、計画のどの段階にいるか？ -->
```



この仕組みにより、AIは長期的なタスクでも一貫性を保ち、効率的に作業を続けることができます。

6. まとめ

AIエージェントは「自律的に行動する AI」

単なる対話相手ではなく、目標達成のために自ら計画・実行するパートナーです。

システムプロンプトが AIの「憲法」

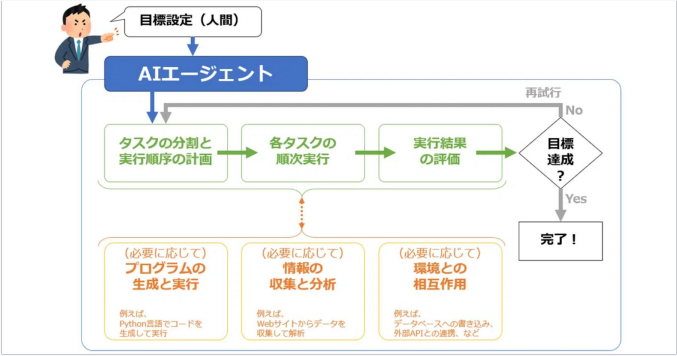
AIの専門性、一貫性、安全性は、この詳細な行動規範によって担保されています。

Gemini CLIは「慎重でプロフェッショナルなエンジニア」

「思い込み禁止」「安全第一」「ユーザー承認」といった原則に基づき行動します。

「理解→計画→実行→検証」のサイクルで仕事を進める

人間の一流エンジニアと同じように、体系的なワークフローを持っています。



ご清聴ありがとうございました

ご質問はありますか？