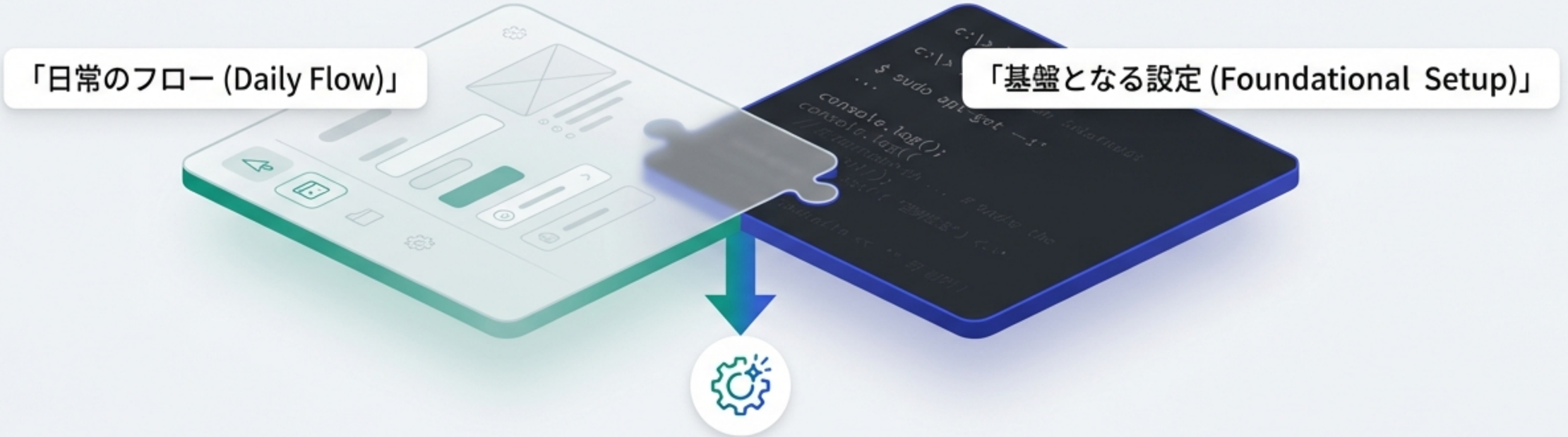
Claude Codeを使いこなす鍵：GUIとCLIの「2層戦略」



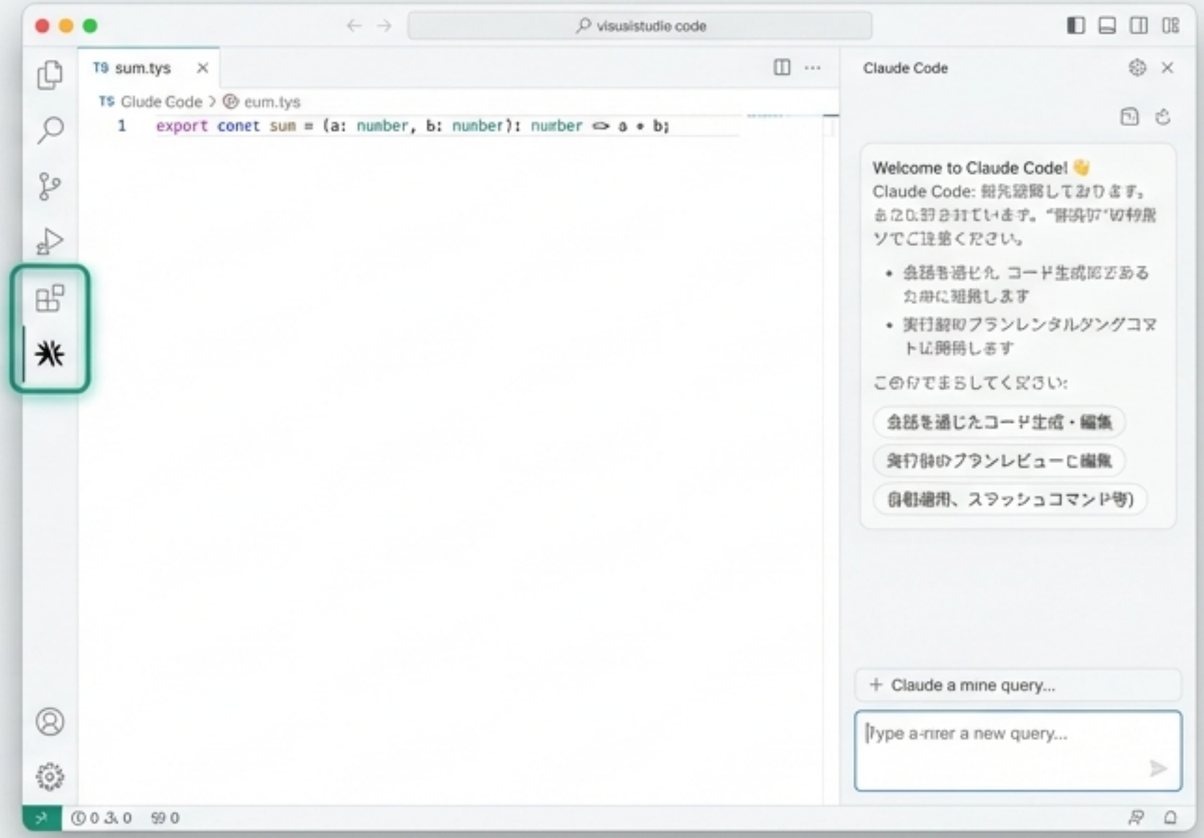
Claude Codeの真価は、単一のツールとしてではなく、2つのレイヤーを組み合わせることで引き出される。
本資料では、VS Code中心の開発者がAIをワークフローに最適に統合するための、最も効率的な運用モデルを解説する。

- **レイヤー1: 拡張機能**（GUI）：普段使いのタスクを高速で回す「日常のドライバー」。
- **レイヤー2: CLI**：初回セットアップと高度な設定を担う「強力な基盤」。

この2層構造を理解することが、Claude Code**習熟への最短ルートである。**

レイヤー1：日常のフローはVS Code拡張で完結させる

日常的な開発タスクのほとんどは、VS CodeのSparkサイドバー内で完結するように設計されている。ターミナルへの切り替えは不要。



運用思想

“普段使い”の利便性を最大化し、思考の分断を最小限に抑える。

このレイヤーでマスターブを機能しよう：

- 会話を通じたコード生成・編集
- 実行前のプランレビューと編集
- 効率化のための各種機能（自動適用、スラッシュコマンド等）

会話からコード反映まで、シームレスな開発サイクル

1. 変更をデリスク化：実行前にAIの計画をレビュー＆編集（Plan mode）

- Claudeが提案する変更計画（どのファイルをどう変更するか）を、実行前に確認・編集できる。
- 意図しない変更を防ぎ、作業の透明性を確保する。v2.0.68で終了UXも改善。

2. 変更内容を確実に把握：インライン差分でレビュー＆承認

- サイドバー内で差分が分かりやすく表示され、クリックで詳細を展開。
- 安心して変更をコードベースに適用できる。

3. 承認作業を高速化：自動適用モード (Auto-accept edits)

- 信頼できるタスクでは、“提案→承認”のステップを省略し、変更を自動反映。
- 注意：IDEの設定ファイルなども変更しうるため、信頼できないワークスペースでは手動承認が推奨される (Restricted Mode)。

開発の効率を高めるVS Code拡張機能のパワーアップツール

コンテキストの簡単な投入

`@`メンションやファイルピッカーで、会話に必要なファイルや画像を迅速に添付。

思考の拡張

入力欄右下のボタン一つで拡張推論（Extended Thinking）のON/OFFを切り替え。

作業の継続と分岐

過去の会話履歴へのアクセスと、複数セッションの同時並行が可能。

定型作業の自動化

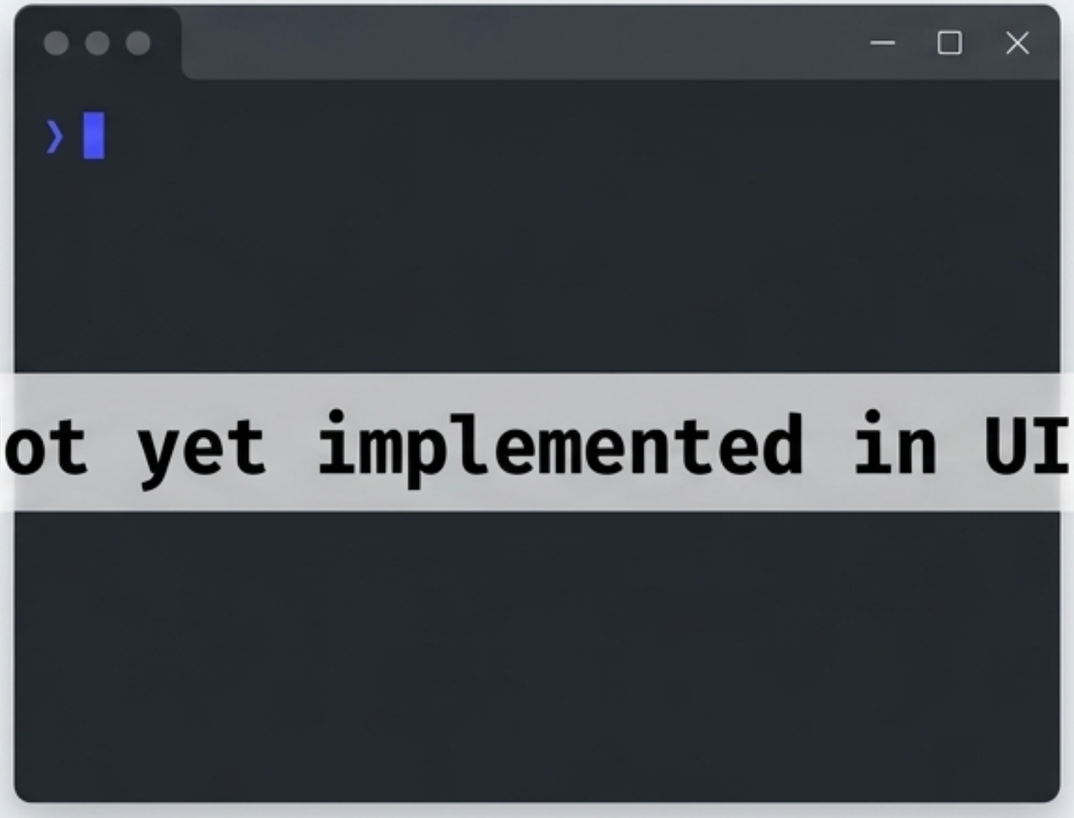
多くのCLIスラッシュコマンドが拡張機能内で直接利用可能。

- `/review`：レビュー依頼
- `/security-review`：セキュリティ観点のレビュー
- `/stats`：利用状況の可視化
- `/resume`：会話の再開

レイヤー2：CLIは「設定」と「高度な操作」の基盤となる

VS Code拡張は“使う”には十分だが、「設定する」系の機能の一部はUIが未実装。

CLIの役割は、この基盤設定を一度行い、拡張機能の能力を最大限に引き出すことにある。



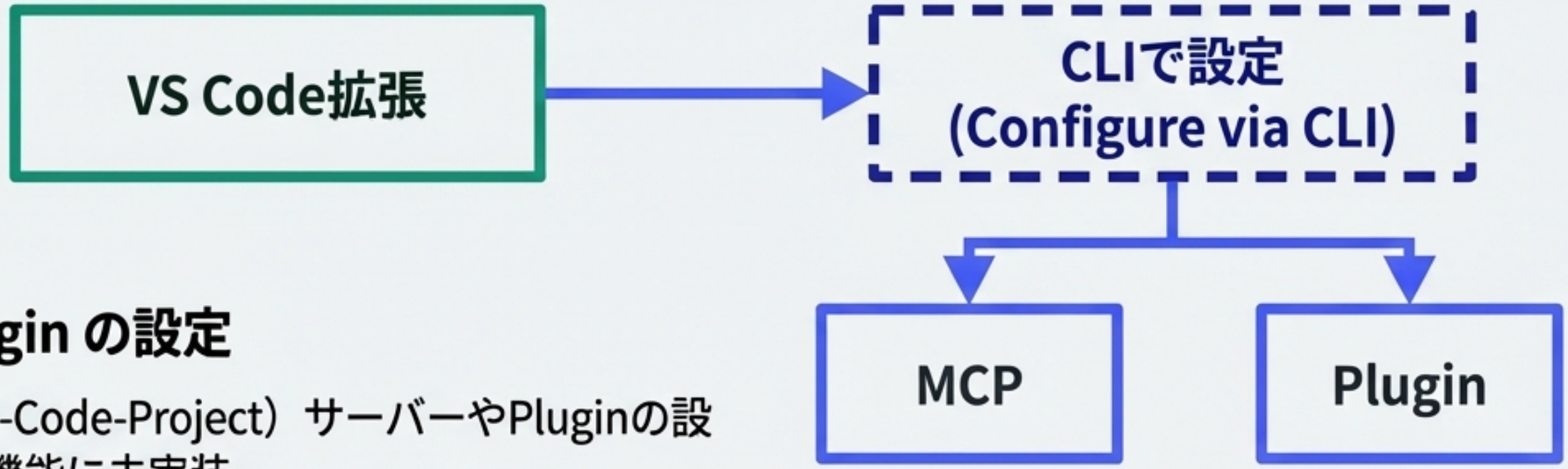
運用思想

“初回セットアップと例外対応”に特化させる。

CLIに寄せるべき範囲は明確に定義されている：

- MCP / Plugin の設定
- サブエージェントの設定
- 拡張機能で未実装の高度なショートカットや機能

外部ツール連携のセットアップはCLIで一度だけ行う



1) MCP / Plugin の設定

- MCP（Multi-Code-Project）サーバーやPluginの設定UIは拡張機能に未実装。
- **最短ルート:** 拡張機能のチャットで `/mcp` や `/plugin` を実行すると、ターミナルベースの設定画面が起動する。
- CLIで直接設定する場合は `claude mcp` コマンドを使用。
- 一度設定すれば、利用自体は拡張機能からシームレスに行える。

2) Subagents（サブエージェント）の設定

- サブエージェントも同様に、設定はCLI側で行う必要がある。

CLI限定の高度な機能とショートカット

```
> claude --show-exclusive-features

以下の機能は、現状VS Code拡張では利用できず、CLI（統合ターミナル）での対話モードに特化している。

**状態の保存・復元**:
  Checkpoints: 会話の状態を保存し、後から復元する。

**会話の巻き戻し**:
  /rewind: 拡張機能では“coming soon”の機能。


**高度なショートカット**:
  #: メモリ（長期記憶）への追加
  !: bashコマンドの直接実行

**その他**:
  Tabキーによるファイルパス補完
  旧バージョンなど特定のモデルを選択するためのUI
```

統合ワークフロー：GUIとCLIを組み合わせた最短セットアップ

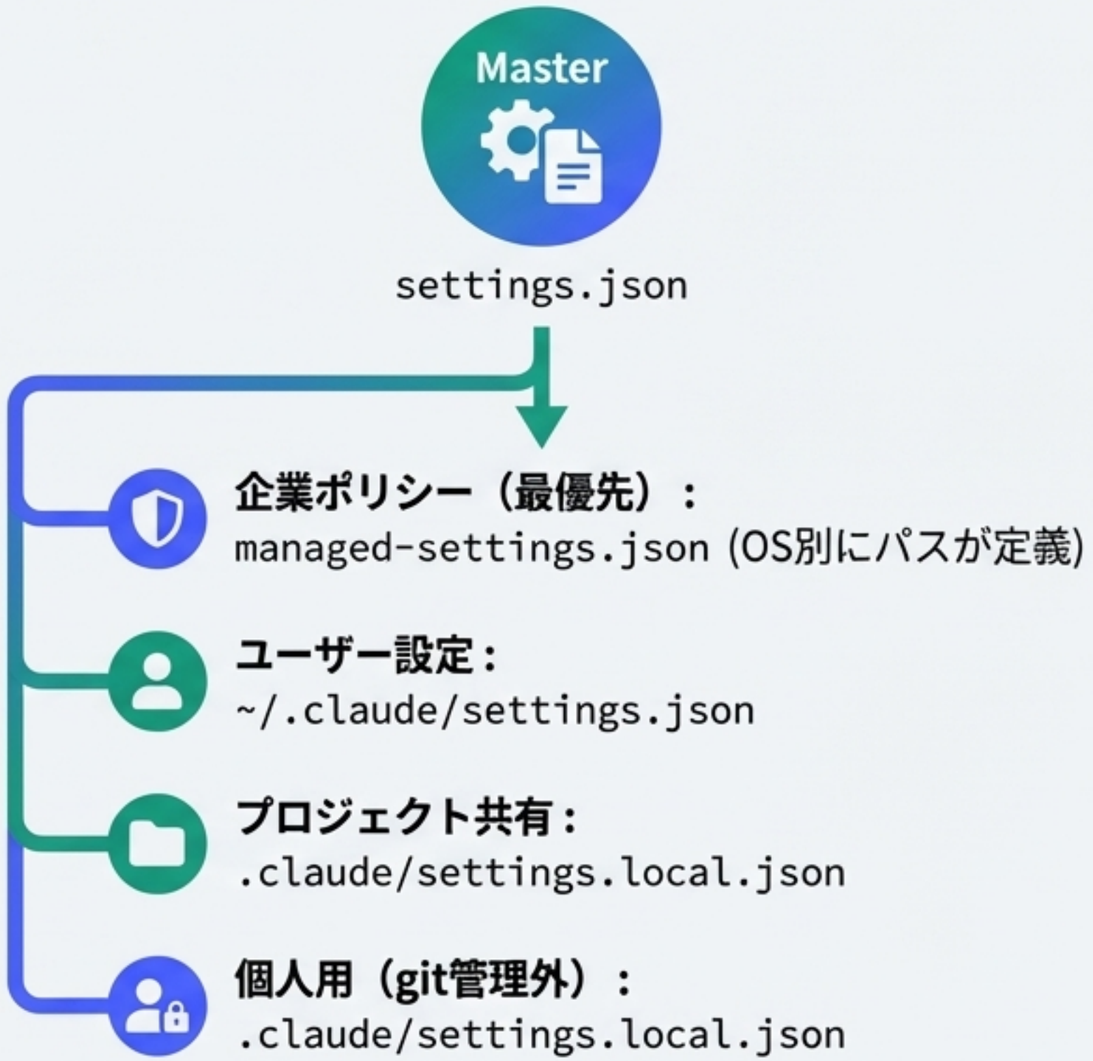
「最適化されたワークフロー」

真の生産性は、GUIの速度とCLIのパワーを意図的に使い分けることで生まれる。

ここからは、VS Code拡張をメインに据えた運用における、最も効率的な初期設定手順を解説する。

ゴール：
一度のセットアップで、日々の作業を可能な限りGUI内で完結させる環境を構築する。

設定の心臓部：両レイヤーが参照する`settings.json`を制する



VS Code拡張とCLIは、共通の設定ファイル（settings.json）を読み込む。ここが設定のマスターとなる。

3rd Partyプロバイダ利用時の設定 (Bedrock/Vertex等)：

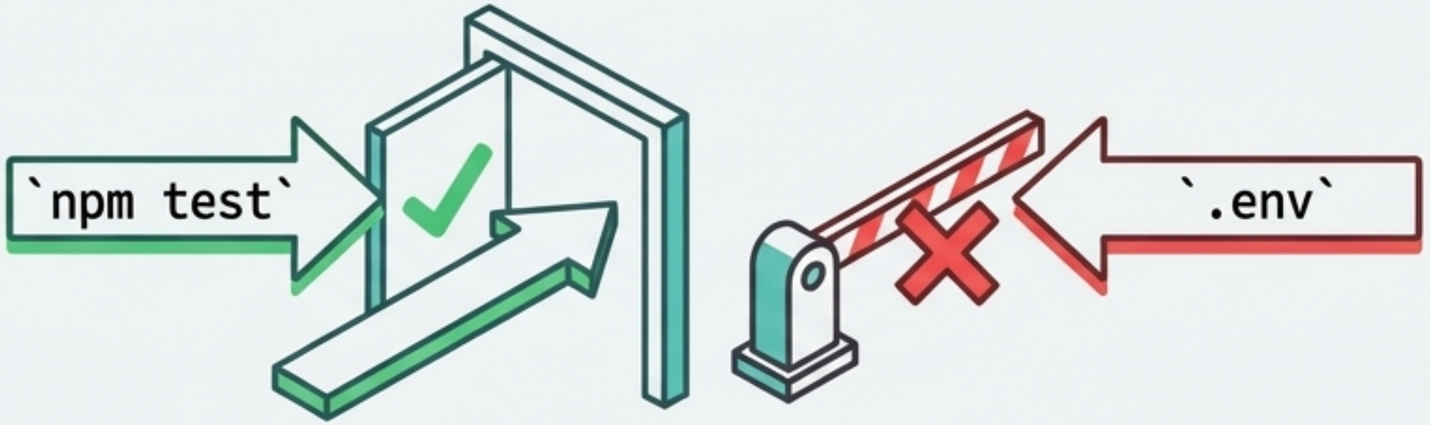
```
{
  "env": {
    "AWS_PROFILE": "your-profile",
    "ANTHROPIC_API_KEY": "sk-ant-..."
  }
}
```

VS Code設定で `claudeCode.disableLoginPrompt: true` を設定し、拡張機能のログイン画面を無効化する。

安全な自動化のために：`permissions`を最初に設定する

Claude Codeの自動化で最も詰まりやすいのが「許可プロンプトの往復」。最初にルールを定義することで、この手間を削減する。

`settings.json`の`permissions.allow/deny`で挙動を管理。

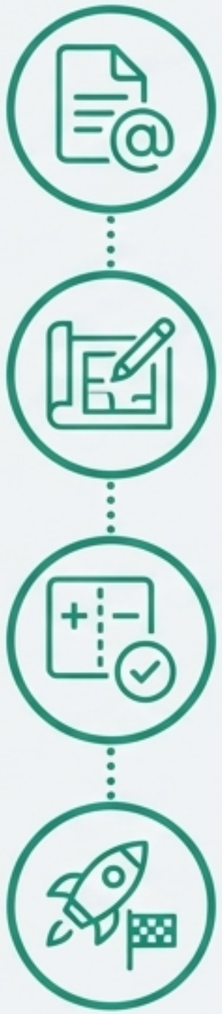


最短セットアップ方針:

- **Allow (許可)**：`lint`や`test`など、安全で定型的なコマンドだけを最初に許可する。
- **Deny (拒否)**：`.env`ファイルや`secrets/**`ディレクトリなど、機密情報への読み取りアクセスを先にブロックする。

`/permissions`スラッシュコマンドで、現在の権限設定をいつでも確認・検索できる（v2.0.67で検索機能が強化）。

日々の最短ワークフロー：Sparkサイドバーで思考を止めない



1. **コンテキスト投入:** 該当ファイルを開き、サイドバーで@メンションまたはファイルピッカーで添付。

2. **Plan Modeで作業の形を確定:** 「まず計画を出して」と依頼し、出てきたPlanを必要に応じて**拡張機能上で編集してから承認**する。

3. **差分レビューと承認:** 差分を確認して承認。慣れたらAuto-acceptを使い、承認回数を減らす。

4. **テストと品質ゲート:** 「**`npm test`**と**`npm run lint`**を実行して、失敗したら直して」のように依頼。よく使うコマンドは**`permissions`**で**`allow`**し、許可の往復をなくす。

ショートカット & コマンド早見表

VS Code拡張（日常使い）

- **UI:** Sparkアイコン → Claude Codeパネル
- **コア機能:** Plan mode, Extended Thinkingボタン
- **コンテキスト:** `@`メンション, ファイル添付
- **主要スラッシュコマンド:**
  `/config`, `/model`, `/permissions`, `/review`, `/stats`, `/resume`

統合ターミナル（高度な操作）

- **モデル切替:** `Option+P` (macOS) / `Alt+P` (Win/Linux)
  ※入力中のプロンプトを消さずに切り替え可能
- **権限モード切替:** `Shift+Tab` or `Alt+M`
- **改行入力設定:** `/terminal-setup`
  (Shift+Enterでの改行を自動設定)
- **未対応ショートカット:** `#` (メモリ追加) / `!` (bash実行) はCLI限定

上級者向け：Cursorエディタとの最適な併用パターン

Claude Codeは公式にCursorでの動作をサポートしている。

パターンA（最短）

Cursorの統合ターミナルで`claude`を起動。Legacy IDE連携が自動で有効化される。

パターンB（UI中心）

CursorにClaude CodeのVS Code拡張をインストール。

衝突を避けるための推奨役割分担:

Claude Code: 「実行エンジン」役。複数ファイル変更、コマンド実行、テスト、コミットまでを担当。

Cursor: 「レビュー＆仕上げ」役。人間による編集、差分レビュー、リファクタリング、最終調整を担当。

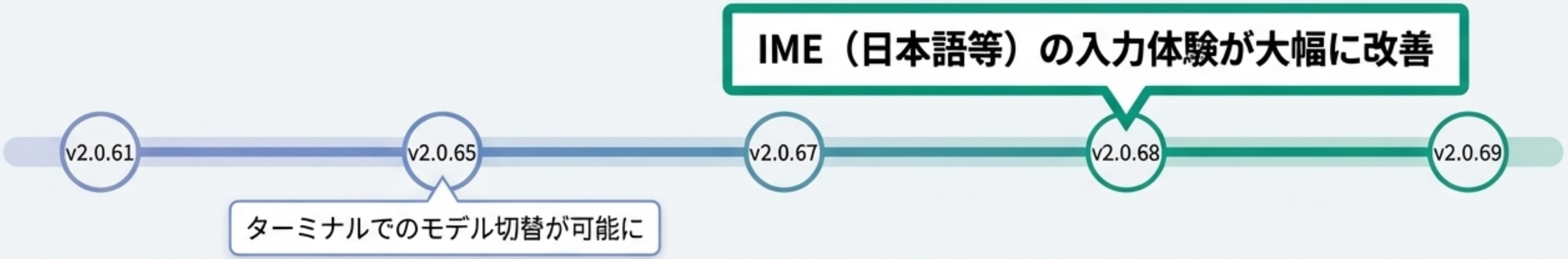
この分担により、両ツールの長所を最大限に活かし、安定したワークフローを構築できる。

最新動向：配布チャンネルとVS Code運用に効く重要アップデート

stable: 安定運用 (例: `2.0.65`)

latest/next: 最新機能 (例: `2.0.69`)

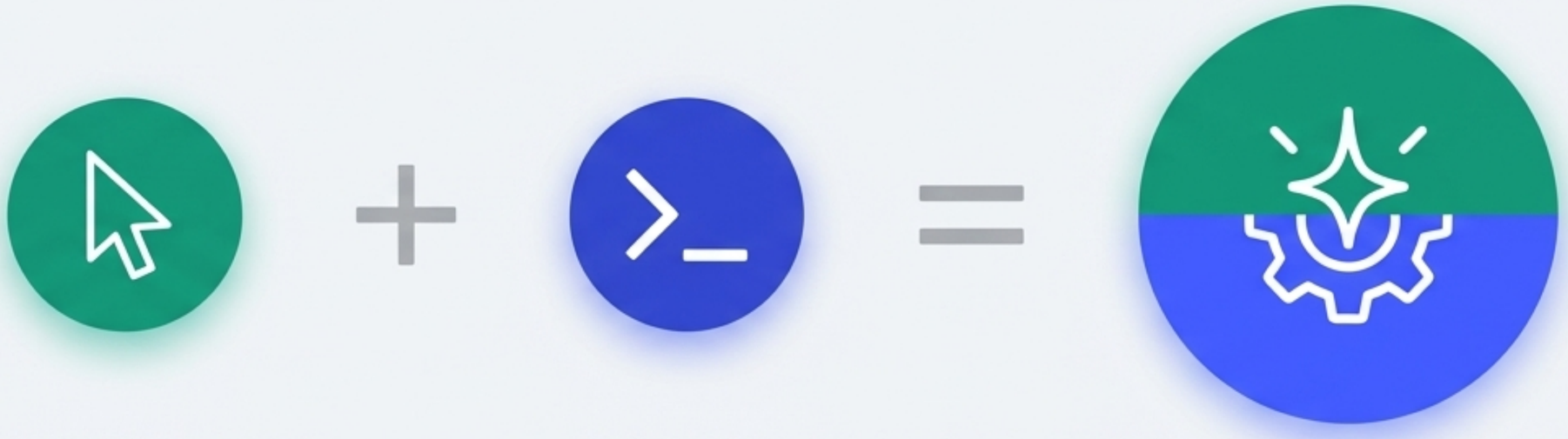
配布チャンネルを理解する: 組織で導入する場合は`stable`を基準とした段階的な展開が安全。



直近の重要アップデート（v2.0.61-v2.0.69）:

- **v2.0.68: IME（日本語等）の入力体験が大幅に改善。**カーソル追従や単語移動が自然に。
- **v2.0.67:** Opus 4.5でThinking ModeがデフォルトONに。設定は`/config`に集約。
- **v2.0.65:** ターミナルでの入力中にモデル切替（`Alt/Option+P`）が可能に。
- **v2.0.61:** VS Codeでの「複数ターミナル同時接続」はレスポンス問題で一時撤回。並列作業は拡張機能の「複数セッション」利用が推奨。

習熟への道：GUIの「速度」とCLIの「パワー」を統合する



Claude Codeの習熟とは、ツールを一つ選ぶことではない。

- 日常のコーディングサイクルは、VS Code拡張の**速度**と**利便性**を最大限に活用する。
- ツールの連携や高度な制御が必要な場面では、CLIの**パワー**と**確実性**に頼る。

この「2層戦略」を意識的に実践することで、AIコーディングアシスタントの能力を真真に引き出し、開発ワークフローを次のレベルへと進化させることができる。