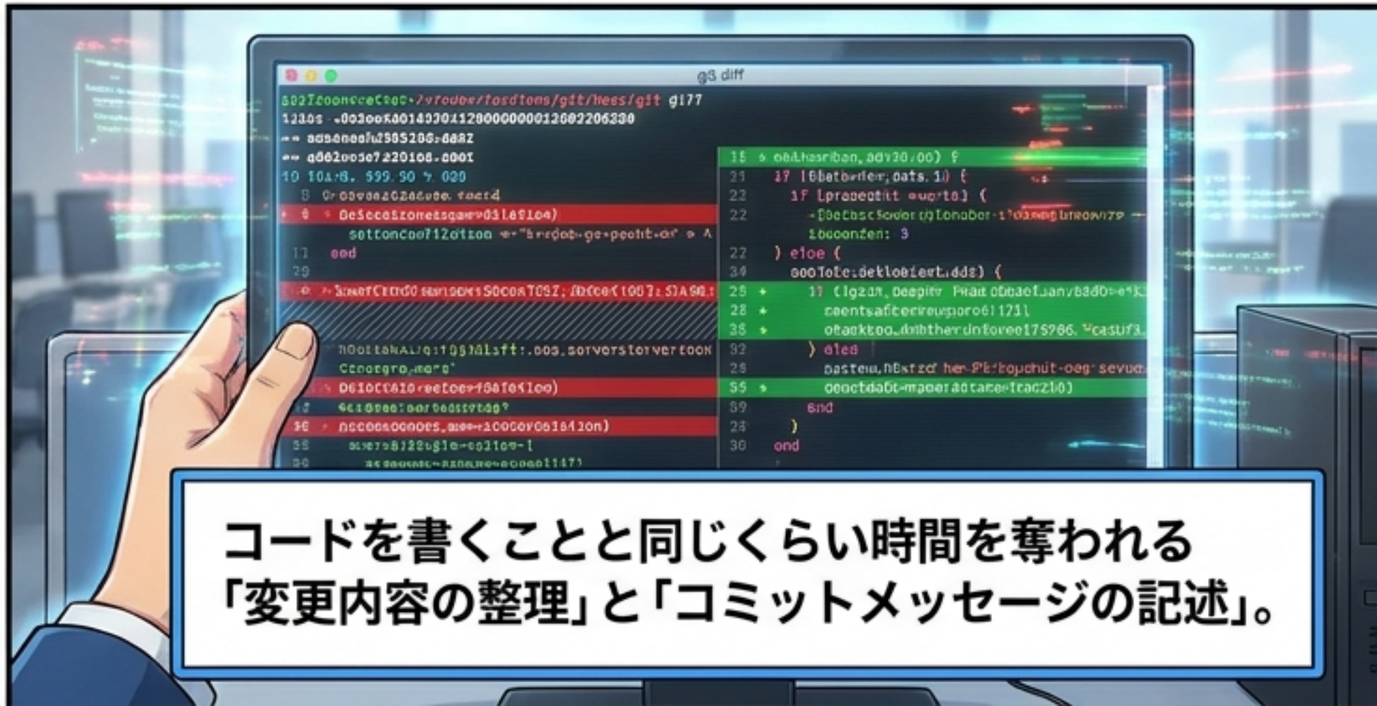
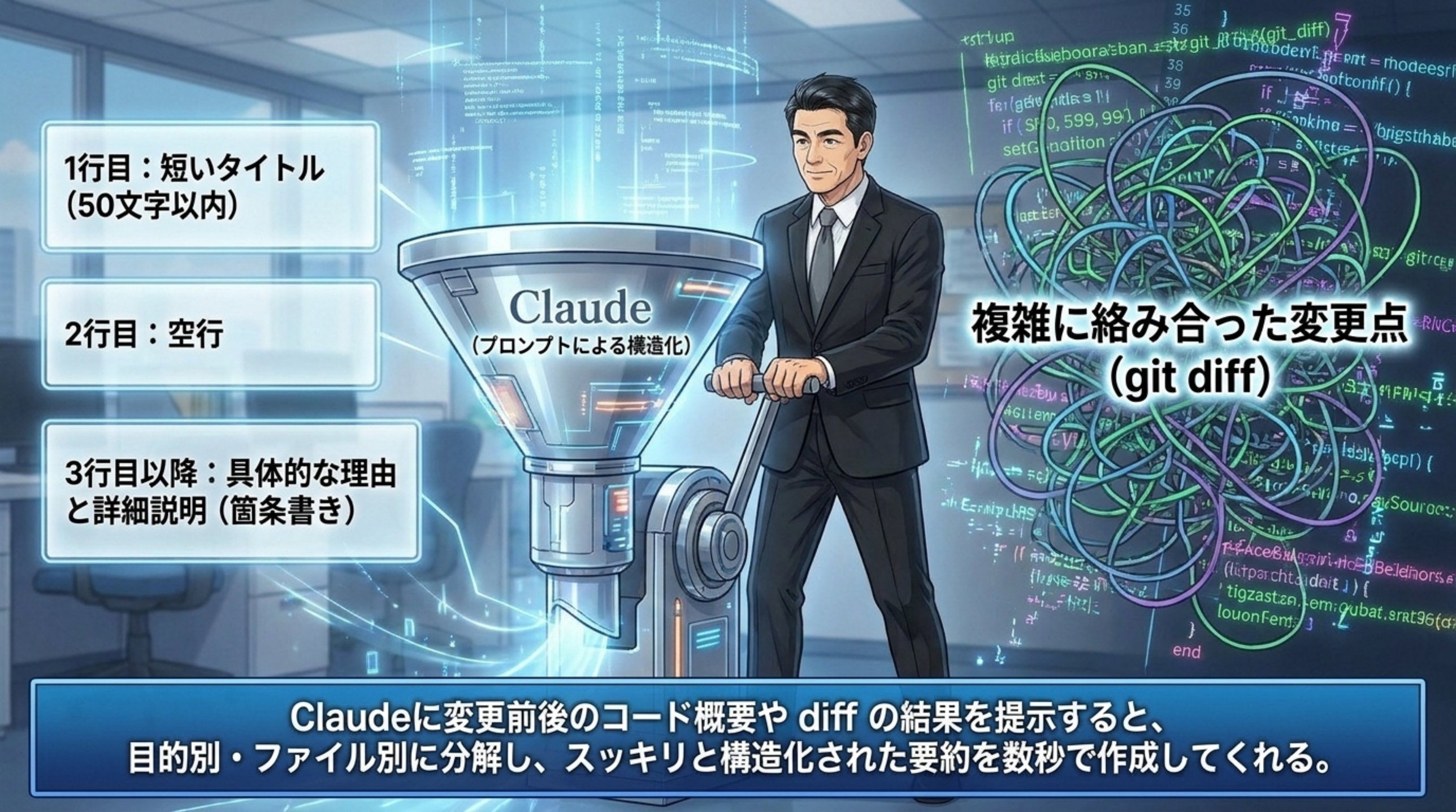

※本資料は右から左へお読みください
よし、完璧な
コミット戦略だ。
Claude
Claudeのおかげで、
こんなにスムーズに
整理できるなんて！
初心者歓迎！Claudeと
極めるGitコミット
変更内容の整理から読みやすい
メッセージ作成までの完全ガイド


どこをどう直したか、
自分でも整理がつかない！
後から見て分かるメッセージなんて書けないよ…


機能追加にバグ修正、
リファクタリング…
一気に作業しすぎた…


コードを書くことと同じくらい時間を奪われる
「変更内容の整理」と「コミットメッセージの記述」。

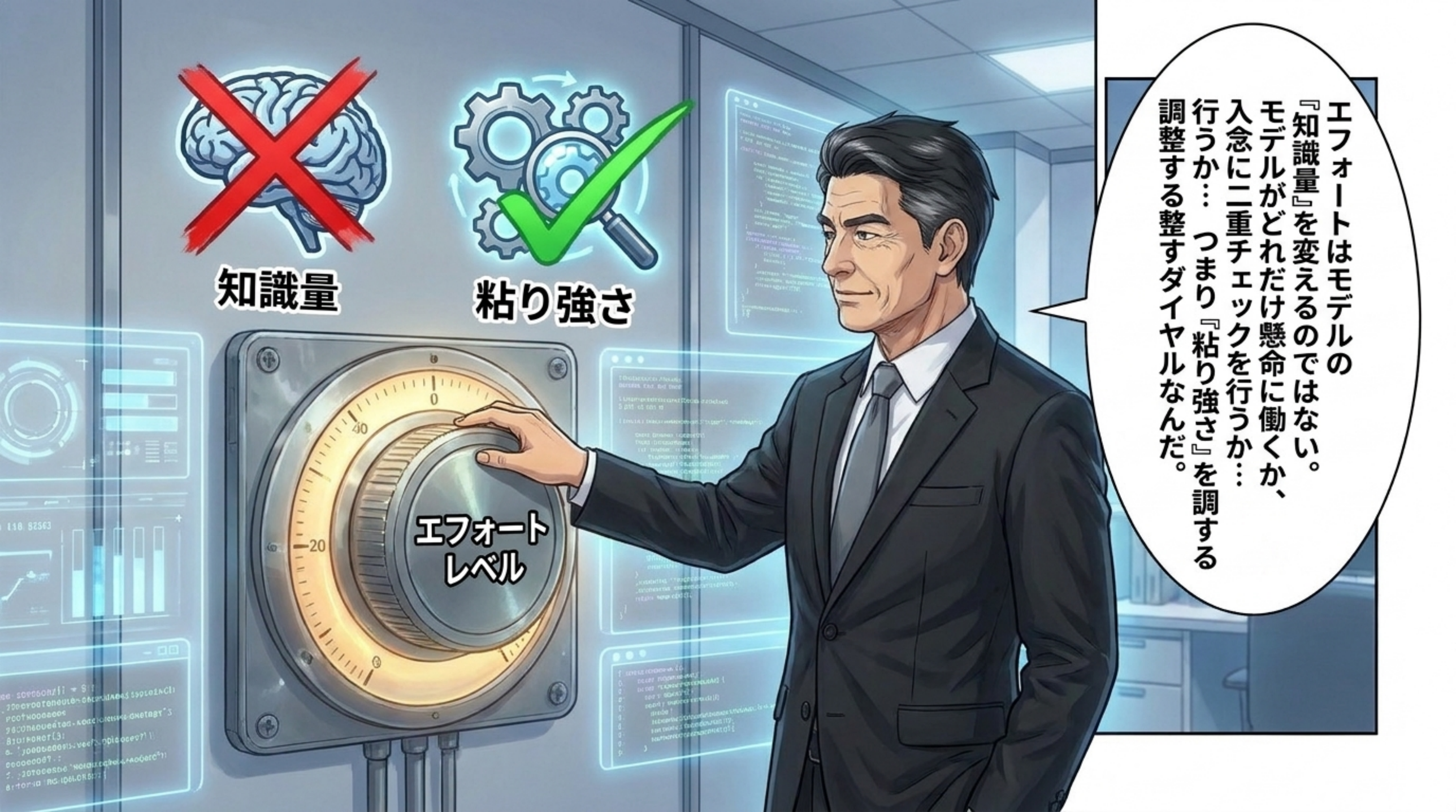

AIアシスタント「Claude」。
単なる自動化ツールではなく、
開発作業の頼れる相棒として迎えることで、
面倒な文章化から解放される！
Claude
一人で頭を抱える必要はない。
定型作業の負担を激減させる
相棒（コラボレーター）を
紹介しよう。


1行目：短いタイトル
（50文字以内）
2行目：空行
3行目以降：具体的な理由
と詳細説明（箇条書き）
Claude
（プロンプトによる構造化）
複雑に絡み合った変更点
（git diff）
Claudeに変更前後のコード概要や diff の結果を提示すると、
目的別・ファイル別に分解し、スッキリと構造化された要約を数秒で作成してくれる。


出力構造を明示的に指定するのがコツだ。AIのAIの自由奔放出力を防ぎ、規約に沿った一貫性のある結果を得ためだよ。
ステップ1：プロンプトの準備と入力。
【変更の目的】ユーザー登録機能のバリデーション処理追加とタイポ修正
【変更内容/コード差分】
auth.py / user_model.py
【出力フォーマット】
1行目タイトル、空行、箇条書き

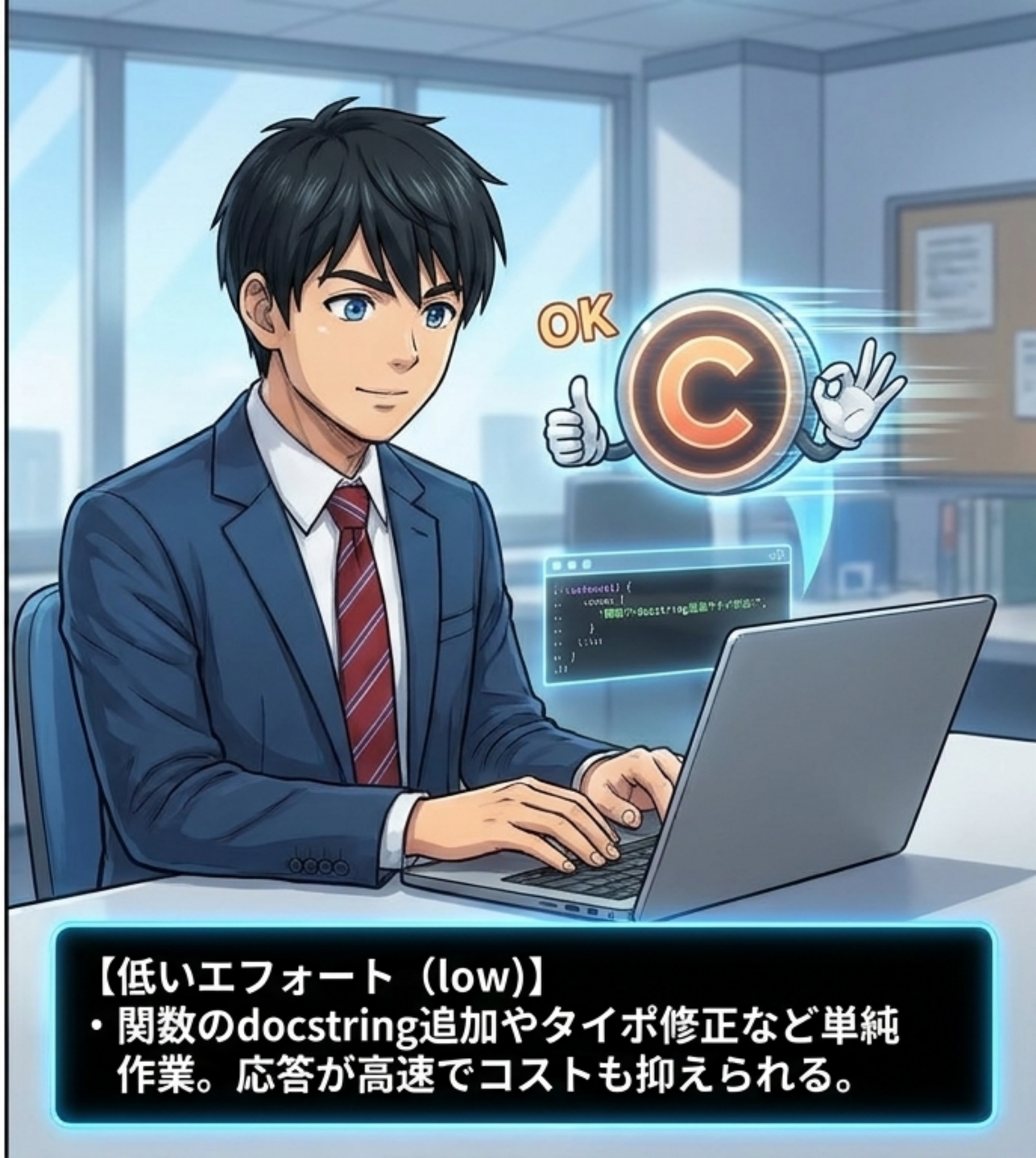

Claudeへの依頼・つまずき解消チートシート
[現象]
入力内容が雑で
意図を誤解される
➡[対策] コードを変えただけでなく、
git diffや「機能追加」等の文脈
(コンテキスト)を添える。
[現象]
メッセージが
「バグ修正」など抽象的
➡[対策] フォーマット指示を強化。
「どの処理をどう変更したか第三
者に伝わる具体性」を要求する。
[現象]
チームの命名規約に
合わない
➡[対策]「Conventional Commits
形式(fix:, feat:)で」とプロンプトに
事前定義する。
最初から完璧な答えが
出なくても焦るな。
指示を微調整して
再生成を依頼すればいい。


Web画面での使い方は
完璧です！
でも、いちいちブラウザを
開くのは少し手間で…
そこで登場するのが
『Claude Code』だ。
ターミナル環境で直接動作し、
コマンド操作だけ コミット前の
セルフチェックが完結するぞ。
トピック2：Claude Codeを活用した
ターミナルでのコミット前レビュー。

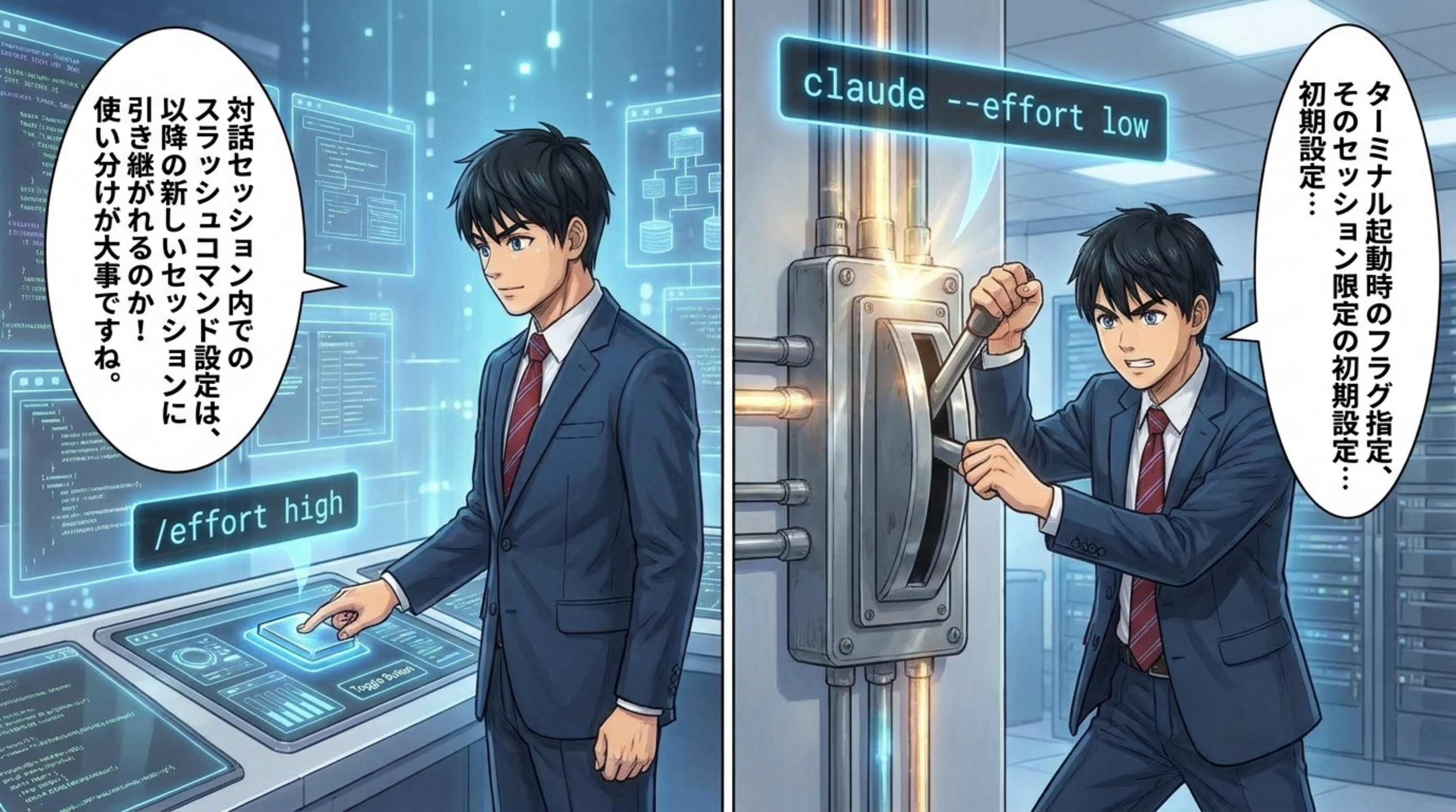

知識量
粘り強さ
エフォート
レベル
0
20
40
エフォートはモデルの
『知識量』を変えるのではない。
モデルがどれだけ懸命に働くか、
入念に二重チェックを行うか…
行うか…　つまり『粘り強さ』を調する
調整する整すダイヤルなんだ。



【高いエフォート（high / max)】
・複雑なタイムゾーン処理やロジック変更。
モデルにより多くの思考と入念な自己検証
（二重チェック）を行わせ、バグを防ぐ。

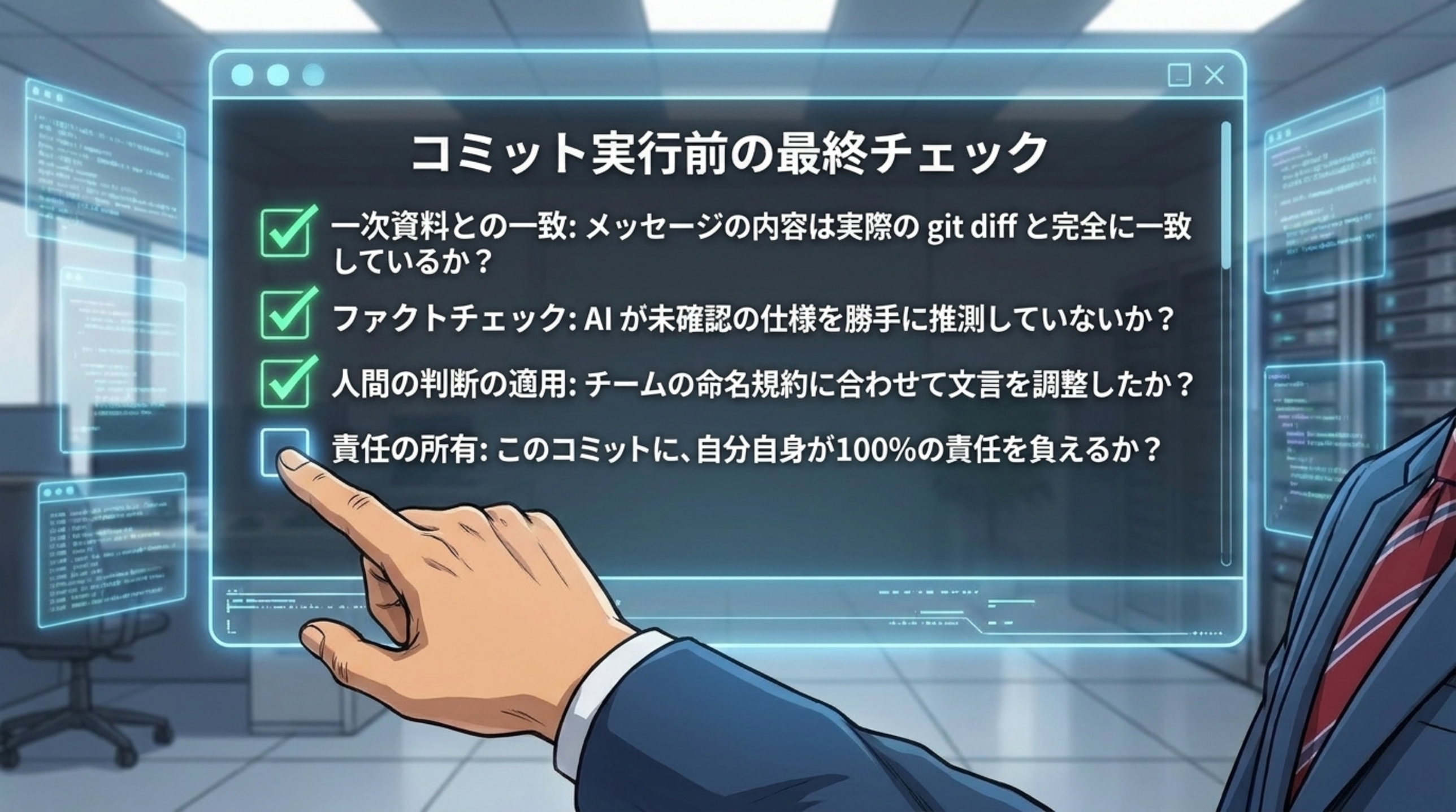


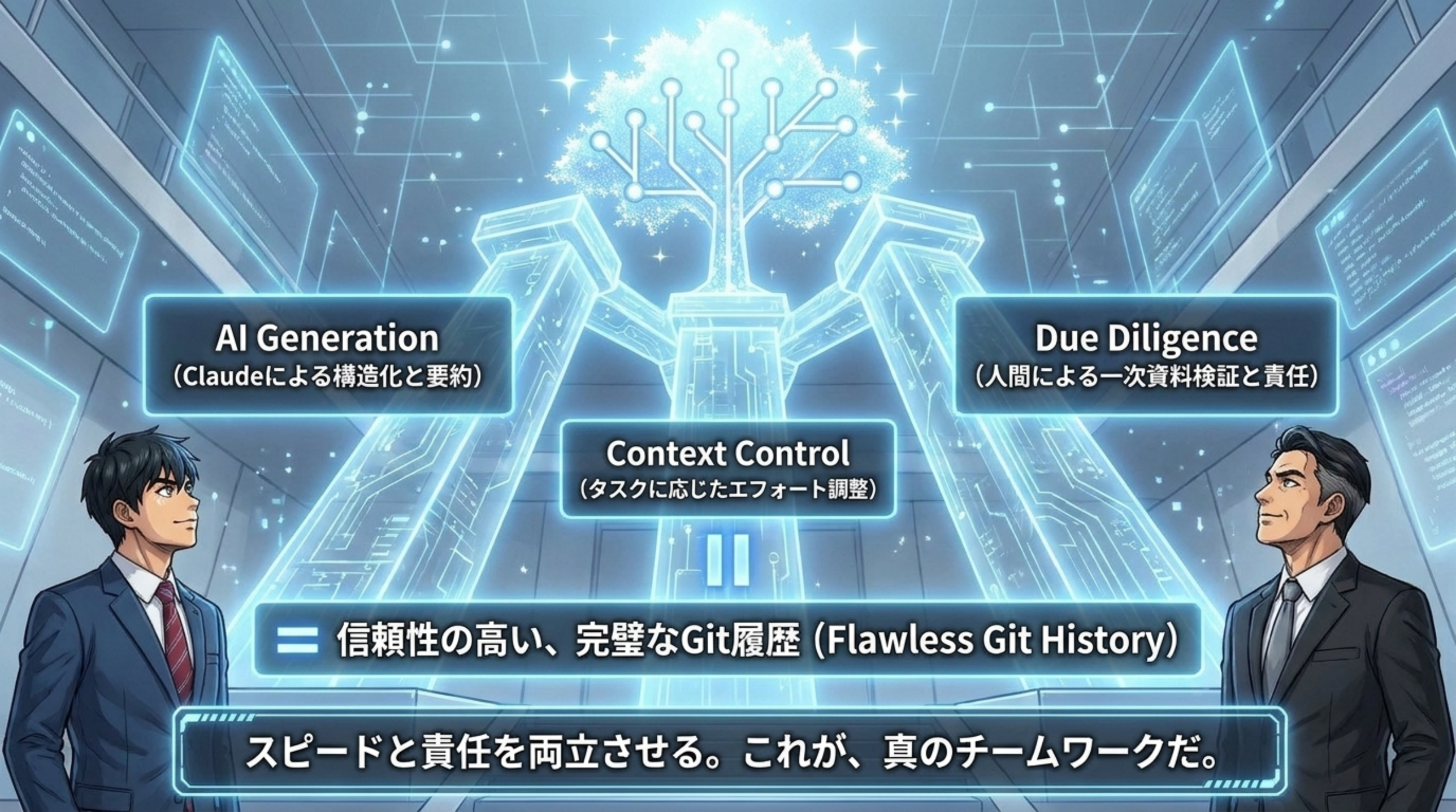
【低いエフォート（low)】
・関数のdocstring追加やタイポ修正など単純
作業。応答が高速でコストも抑えられる。

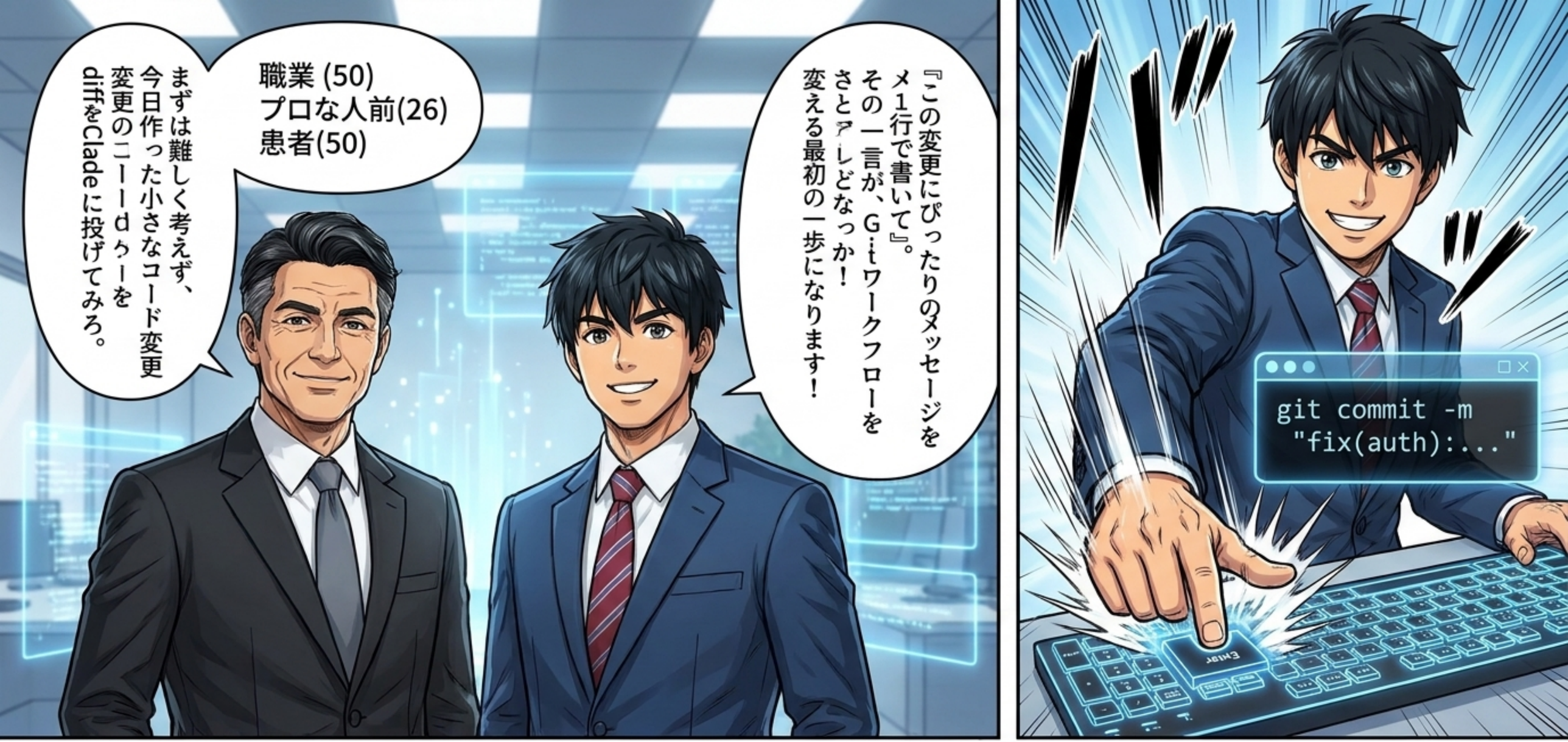

ターミナル起動時のフラグ指定、
そのセッション限定の初期設定…
初期設定…
claude --effort low
対話セッション内での
スラッシュコマンド設定は、
以降の新しいセッションに
引き継がれるのか！
使い分けが大事ですね。
/effort high


AIは優秀だが、
無条件で鵜呑みに
してはいけない。
最終的な検証と
最終的な齟検証を』
責任を担うのは、
Iではなく
AIではなく『人だ。
AIデューデリジェンスの原則。
AIが下書きを作成し、人間が一次
資料と照らし合わせて検証する。
最終成果物に対する全責任は
開発者自身が持つ。


ステップ1：
一次資料（git diff）
との目視照合。
git diff
ん？Claudeの出力には
『データベース接続の
最適化』とあるけど…
実際はただの
『バリデーション関数の
呼び出し漏れ修正』だ。
事実と異なる理由が履歴に
未来のチームが混乱する。
修正指示を出そう！
ガッ


コミット実行前の最終チェック
一次資料との一致: メッセージの内容は実際の git diff と完全に一致しているか？
ファクトチェック: AI が未確認の仕様を勝手に推測していないか？
人間の判断の適用: チームの命名規約に合わせて文言を調整したか？
責任の所有: このコミットに、自分自身が100%の責任を負えるか？


AI Generation
（Claudeによる構造化と要約）
Due Diligence
（人間による一次資料検証と責任）
Context Control
（タスクに応じたエフォート調整）
＝ 信頼性の高い、完璧なGit履歴（Flawless Git History）
スピードと責任を両立させる。これが、真のチームワークだ。



さあ、今日から新しい相棒との開発を始めよう。