Claude Codeは怪物だ 💪

6ヶ月のハードコア使用から得たTips

30万LOCリライト実戦経験記

📋 発表内容

1. **背景 & プロジェクト概要**
2. **Skills Auto-Activation System** ⭐ Game Changer
3. **CLAUDE.mdとドキュメント化の進化**
4. **Dev Docs System** - コンテキスト維持の秘密
5. **PM2プロセス管理** - バックエンドデバッグ革新
6. **Hooks System** - #NoMessLeftBehind
7. **エージェント & スラッシュコマンド**
8. **プロンプトTips**
9. **結論 & 核心整理**

🎯 プロジェクト背景

ミッション

- 7年経ったレガシーウェブアプリの**大規模リデザイン/リファクタ**
- **一人で、数ヶ月で、トップダウン全面改編**

技術スタックマイグレーション

Before	After
React 16 JS	React 19 TypeScript
React Query v2	TanStack Query v5
React Router v4	TanStack Router（ファイルベース）
Material UI v4	MUI v7

📊 結果

数字で見る成果

- **30〜40万LOC** コードベース
- **6ヶ月** ハードコア開発
- テストカバレッジ **0% → 適切なレベル**
- **技術負債**: 手に負えない → 管理可能

副作用

- 予想寿命5年減少 😅
- Claudeの能力と限界を完全に体得

🔧 Skills Auto-Activation System

問題点

- AnthropicがSkills機能をリリース
- 数千行のベストプラクティス、パターン、例を作成
- しかし... **Claudeが使わない！** 😱
- キーワードを言っても、関連ファイルを修正しても無反応

💡 解決策: Hooksで強制アクティベーション

"Aha!" Moment

> Claudeがスキルを自動で使わないなら、
> **必ず先に確認させてしまおう！**

TypeScript hookで**多層自動アクティベーション構造**を構築

🎣 核心Hook #1: UserPromptSubmit

Claudeがメッセージを見る"前"に動作

```
ユーザー: "レイアウトシステムはどう動くの？"
          ↓
Hook: キーワード/意図パターン分析
          ↓
Claudeが見るもの:
🎯 SKILL ACTIVATION CHECK
Use project-catalog-developer skill
          ↓
ユーザーメッセージ伝達
```

🎣 核心Hook #2: Stop Event

Claudeが回答を終えた"後"に動作

- 修正されたファイル分析
- リスクパターン点検 (try-catch、DB操作、asyncなど)
- 柔らかいセルフチェックリマインダー表示

```
❓ エラーハンドリング追加した？
❓ Prisma操作はrepositoryパターンを使ってる？
```

ブロッキングなしで認識維持！

⚙️ skill-rules.json設定

```
{
  "backend-dev-guidelines": {
    "type": "domain",
    "enforcement": "suggest",
    "priority": "high",
    "promptTriggers": {
      "keywords": ["backend", "controller", "service"],
      "intentPatterns": [
        "(create|add).*?(route|endpoint|controller)"
      ]
    },
    "fileTriggers": {
      "pathPatterns": ["backend/src/**/*.ts"],
      "contentPatterns": ["router\\.", "export.*Controller"]
    }
  }
}
```

📈 Skills適用前 vs 後

Before 😫

- 新パターンをドキュメント化してもClaudeは古いパターン使用
- 毎回「ガイドライン確認しろ」と手動指示
- 30万+LOC全般の不一致
- Claudeの「創造的解釈」修正に時間浪費

After 🎉

- 一貫したパターン自動適用
- Claudeが先に自己修正
- ガイドライン遵守への信頼確保
- レビュー/修正時間大幅削減

📚 Anthropicベストプラクティス適用

公式推奨事項

- メインSKILL.mdは**500行以下**
- リソースファイルで段階的公開

私の失敗

- `frontend-dev-guidelines`: **1,500行以上** 😱

改善後

- **frontend-dev-guidelines**: 398行 + リソース10個
- **backend-dev-guidelines**: 304行 + リソース11個
- **トークン効率40〜60%向上！**

🗂️ 構築したスキルラインナップ

Guidelines & Best Practices

- `backend-dev-guidelines` - Routes → Controllers → Services → Repositories
- `frontend-dev-guidelines` - React 19, MUI v7, TanStackパターン
- `skill-developer` - スキル制作用メタスキル

Domain-Specific

- `workflow-developer` - ワークフローエンジンパターン
- `notification-developer` - メール/通知システム
- `database-verification` - カラム名エラー防止（ガードレール！）
- `project-catalog-developer` - DataGridレイアウト

📝 CLAUDE.md進化

問題点

- CLAUDE.mdが**1,400行+**に肥大化
- BEST_PRACTICES.mdも過負荷
- Claudeが読むときもあれば無視することも

解決策: 関心の分離

役割	担当
Skills	コードをどう書くか
CLAUDE.md	このプロジェクトがどう動くか

🏗️ 新しいドキュメント構造

```
Root CLAUDE.md (100行)
├── 核心ユニバーサルルール
├── レポ別claude.md参照
└── 詳細ガイドラインはSkills参照

Each Repo's claude.md (50-100行)
├── Quick Startセクション
│   ├── PROJECT_KNOWLEDGE.md
│   ├── TROUBLESHOOTING.md
│   └── Auto-generated API docs
└── レポ別特異事項 & コマンド
```

📖 Dev Docs System

核心問題

> Claudeは**「自信満々の記憶喪失ジュニア」**みたい
> すぐにコンテキストを失ってしまう！

解決策: 3ファイルシステム

```
~/git/project/dev/active/[task-name]/
├── [task-name]-plan.md      # 承認されたプラン
├── [task-name]-context.md   # 核心ファイル/決定事項
└── [task-name]-tasks.md     # 作業チェックリスト
```

📋 Dev Docsワークフロー

Starting Large Tasks

1. プランモードで承認されたプラン生成
2. 作業ディレクトリ生成
3. 3つのドキュメント生成
4. 完了したらすぐチェック

Continuing Tasks

- `/dev/active/` で既存作業確認
- 進行前に3つのドキュメントすべて読む
- "Last Updated"タイムスタンプ更新

👑 プランニングが王様だ

> 最低限プランニングモードで計画を立てずに実装から始めさせると、
> **良くない時間を過ごすことになります。**

プランニングプロセス

1. プランニングモードで開始
2. `strategic-plan-architect` サブエージェント活用
3. プランを細かくレビュー（**超重要！**）
4. `/create-dev-docs` でドキュメント化
5. 段階的実装 + 定期的コードレビュー

🔄 PM2プロセス管理

問題点

- バックエンドマイクロサービス**7つ同時稼働**
- サービス実行中Claudeがログを見れない
- 私がログを探してコピペしなければならない

PM2導入効果

- ✅ 各サービスが管理されるプロセスとして実行
- ✅ **Claudeがリアルタイムでログ読み取り可能**
- ✅ クラッシュ時自動再起動
- ✅ メモリ/CPUモニタリング

🔍 PM2前後比較

Before 😩

```
Me: "メールサービスでエラーが出てる"
Me: [ログを探してコピー]
Me: [チャットに貼り付け]
Claude: "分析してみる..."
```

After 🚀

```
Me: "メールサービスでエラーが出てる"
Claude: [実行] pm2 logs email --lines 200
Claude: "原因確認 - DB接続タイムアウト..."
Claude: [実行] pm2 restart email
Claude: "サービス再起動、モニタリング中..."
```

🎣 Hooks System

#NoMessLeftBehind

Hook #1: File Edit Tracker

- Edit/Write/MultiEdit後に修正ファイル記録

Hook #2: Build Checker

- 応答終了時に修正されたレポのビルドチェック
- エラー5個未満 → すぐ表示
- エラー5個以上 → Auto-error-resolverエージェント推奨

Hook #3: Prettier Formatter

- 修正されたすべてのファイル自動フォーマット

📋 Error Handling Reminder

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
📋 ERROR HANDLING SELF-CHECK
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

⚠️  Backend Changes Detected
   2 file(s) edited

   ❓ Did you add Sentry.captureException()?
   ❓ Are Prisma operations wrapped in error handling?

   💡 Backend Best Practice:
      - All errors should be captured to Sentry
      - Controllers should extend BaseController
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
```

⚡ Complete Hook Pipeline

```
Claudeが応答終了
        ↓
Hook 1: Prettierフォーマッター → 修正ファイル自動フォーマット
        ↓
Hook 2: ビルドチェッカー → TSエラー即座にキャッチ
        ↓
Hook 3: エラーリマインダー → エラーハンドリングセルフチェック
        ↓
エラーがあれば → Claudeが即修正
        ↓
結果: きれいで、フォーマットされて、エラーのないコード
```

残す痕跡: 0 🎯

🤖 専門化されたエージェント軍団

Quality Control

- `code-architecture-reviewer` - アーキテクチャ観点コードレビュー
- `build-error-resolver` - TSエラー体系的解決
- `refactor-planner` - 包括的リファクタプラン

Testing & Debugging

- `auth-route-tester` - 認証ルートテスト
- `auth-route-debugger` - 401/403問題デバッグ
- `frontend-error-fixer` - フロントエンドエラー修正

🤖 専門化されたエージェント軍団（続き）

Planning & Strategy

- `strategic-plan-architect` - 詳細実装プラン
- `plan-reviewer` - 実装前プランレビュー
- `documentation-architect` - ドキュメント生成/更新

Specialized

- `frontend-ux-designer` - スタイル/UX改善
- `web-research-specialist` - ウェブリサーチ
- `reactour-walkthrough-designer` - UIツアー制作

⚡ スラッシュコマンド

Planning & Docs

- `/dev-docs` - 総合戦略プラン生成
- `/dev-docs-update` - コンパクション前にDev Docs更新
- `/create-dev-docs` - プラン → Dev Doc変換

Quality & Review

- `/code-review` - アーキテクチャ観点コードレビュー
- `/build-and-fix` - ビルド + エラー修正

Testing

- `/route-research-for-testing` - 影響ルート探索
- `/test-route` - 特定認証ルートテスト

💬 プロンプトTips

核心原則

> "Claudeが君のために何をしてくれるか問うな、
> Claudeにどんなコンテキストを与えられるか問え。"
> ~ 賢者

- **具体的に書く** - 欲しい結果を明確に
- **誘導質問を避ける** - 中立的に意見を求める
- **頻繁に再プロンプト** - Double ESCでブランチ
- **直接介入もOK** - 2分で終わることに30分浪費禁止

🛠️ 便利なツール

SuperWhisper on Mac

- 音声でプロンプト入力
- タイピング疲労軽減

Memory MCP

- プロジェクト別決定/アーキテクチャ選択追跡

BetterTouchTool

- 相対パスURLコピー自動化
- ダブルタップホットキー（CMD+CMD = Claude Code）
- アプリ間切り替え時間節約

✅ 結論: 必須要素

順位	要素	説明
1	プランニング	プランニングモードまたはstrategic-plan-architect
2	**Skills + Hooks**	自動アクティベーションが事実上必須
3	**Dev Docs**	Claudeがプロットを失わないように
4	コードレビュー	Claudeに自分のコードレビューさせる
5	**PM2**	バックエンドデバッグが耐えられるようになる

✨ 結論: あると良いもの

- 共通作業用**専門エージェント**
- 繰り返しワークフロー用スラッシュコマンド
- 包括的ドキュメンテーション
- スキルに添付したユーティリティスクリプト
- 決定追跡用**Memory MCP**

🎯 TL;DR

> TypeScript hookで**Claude Codeスキル自動アクティベーションシステム**を作り、
> コンテキスト喪失を防ぐ**Dev Docsワークフロー**と
> **PM2 + 自動エラー点検**を導入した。

結果

6ヶ月で一人30万LOCを一貫した品質でリライト 🎉

🙏 ありがとうございます

Q&A

原文: Reddit - Claude Code is a beast