Claude Code는 괴물이다 💪

6개월 하드코어 사용에서 얻은 팁들

30만 LOC 리라이트 실전 경험기

📋 발표 내용

1. 배경 & 프로젝트 개요
2. **Skills Auto-Activation System** ⭐ Game Changer
3. **CLAUDE.md와 문서화 진화**
4. **Dev Docs System** - 컨텍스트 유지의 비밀
5. **PM2 프로세스 관리** - 백엔드 디버깅 혁신
6. **Hooks System** - #NoMessLeftBehind
7. 에이전트 & 슬래시 커맨드
8. 프롬프트 팁
9. 결론 & 핵심 정리

🎯 프로젝트 배경

미션

- 7년 된 레거시 웹앱의 **대규모 리디자인/리팩터**
- **혼자서, 몇 달 안에, 탑다운 전면 개편**

기술 스택 마이그레이션

Before	After
React 16 JS	React 19 TypeScript
React Query v2	TanStack Query v5
React Router v4	TanStack Router (파일 기반)
Material UI v4	MUI v7

📊 결과

숫자로 보는 성과

- **30~40만 LOC** 코드베이스
- **6개월** 하드코어 개발
- **테스트 커버리지 0% → 적절한 수준**
- **기술부채**: 감당 불가 → 관리 가능

부수효과

- 기대 수명 5년 감소 😅
- Claude의 능력과 한계 완벽 체득

🔧 Skills Auto-Activation System

문제점

- Anthropic이 Skills 기능 출시
- 수천 줄의 베스트 프랙티스, 패턴, 예시 작성
- **하지만... Claude가 안 씀! 😱**
- 키워드를 말해도, 관련 파일을 수정해도 무반응

💡 해결책: Hooks로 강제 활성화

"Aha!" Moment

> Claude가 스킬을 자동으로 안 쓰면,
> **무조건 먼저 확인하게 만들어 버리자!**

TypeScript 훅으로 **다층 자동 활성화 구조** 구축

🎣 핵심 훅 #1: UserPromptSubmit

Claude가 메시지를 보기 "전"에 동작

```
사용자: "레이아웃 시스템이 어떻게 작동하지?"
          ↓
훅: 키워드/의도 패턴 분석
          ↓
Claude가 보는 것:
🎯 SKILL ACTIVATION CHECK
Use project-catalog-developer skill
          ↓
사용자 메시지 전달
```

🎣 핵심 훅 #2: Stop Event

Claude가 답변을 끝낸 "후"에 동작

- 수정된 파일 분석
- 리스크 패턴 점검 (try-catch, DB 연산, async 등)
- 부드러운 셀프체크 리마인더 표시

```
❓ 에러 핸들링 추가했나요?
❓ Prisma 연산은 repository 패턴을 쓰나요?
```

블로킹 없이 인지 유지!

⚙️ skill-rules.json 설정

```
{
  "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 적용 전 vs 후

Before 😫

- 새 패턴 문서화해도 Claude는 옛 패턴 사용
- 매번 "가이드라인 확인하라" 수동 지시
- 30만+ LOC 전반의 불일치
- Claude의 "창의적 해석" 수정에 시간 낭비

After 🎉

- 일관된 패턴 자동 적용
- Claude가 먼저 자가 교정
- 가이드라인 준수 신뢰 확보
- 리뷰/수정 시간 대폭 절감

📚 Anthropic 베스트 프랙티스 적용

공식 권장사항

- 메인 SKILL.md는 **500줄 이하**
- 리소스 파일로 점진적 공개

제 실수

- `frontend-dev-guidelines` : **1,500줄 이상** 😱

개선 후

- **frontend-dev-guidelines**: 398줄 + 리소스 10개
- **backend-dev-guidelines**: 304줄 + 리소스 11개
- **토큰 효율 40~60% 향상!**

🗂️ 구축한 스킬 라인업

Guidelines & Best Practices

- `backend-dev-guidelines` - Routes → Controllers → Services → Repositories
- `frontend-dev-guidelines` - React 19, MUI v7, TanStack 패턴
- `skill-developer` - 스킬 제작용 메타 스킬

Domain-Specific

- `workflow-developer` - 워크플로우 엔진 패턴
- `notification-developer` - 이메일/알림 시스템
- `database-verification` - 컬럼명 오류 방지 (**가드레일!**)
- `project-catalog-developer` - DataGrid 레이아웃

📝 CLAUDE.md 진화

문제점

- CLAUDE.md가 **1,400줄+** 로 비대
- BEST_PRACTICES.md도 과부하
- Claude가 읽을 때도 있고 무시할 때도 있음

해결책: 관심사의 분리

역할	담당
Skills	코드를 어떻게 쓰는지
CLAUDE.md	이 프로젝트가 어떻게 동작하는지

🏗️ 새로운 문서 구조

```
Root CLAUDE.md (100줄)
├── 핵심 유니버설 룰
├── 레포별 claude.md 참조
└── 상세 가이드라인은 Skills 참조

Each Repo's claude.md (50-100줄)
├── Quick Start 섹션
│   ├── PROJECT_KNOWLEDGE.md
│   ├── TROUBLESHOOTING.md
│   └── Auto-generated API docs
└── 레포별 특이사항 & 커맨드
```

📖 Dev Docs System

핵심 문제

> Claude는 **"자신감 넘치는 기억상실증 주니어"** 같다
> 금방 맥락을 잃어버림!

해결책: 3개 파일 시스템

```
~/git/project/dev/active/[task-name]/
├── [task-name]-plan.md      # 승인된 플랜
├── [task-name]-context.md   # 핵심 파일/결정사항
└── [task-name]-tasks.md     # 작업 체크리스트
```

📋 Dev Docs 워크플로우

Starting Large Tasks

1. 플랜 모드에서 승인된 플랜 생성
2. 작업 디렉터리 생성
3. 3개 문서 생성
4. 완료 즉시 체크 표시

Continuing Tasks

- `/dev/active/` 에서 기존 작업 확인
- 진행 전 세 문서 모두 읽기
- "Last Updated" 타임스탬프 갱신

👑 플래닝이 왕이다

> 최소한 플래닝 모드로 계획을 세우지 않고 구현부터 시키면,
> 안 좋은 시간을 보내실 겁니다.

플래닝 프로세스

1. 플래닝 모드로 시작
2. `strategic-plan-architect` 서브에이전트 활용
3. 플랜 꼼꼼히 리뷰 (**매우 중요!**)
4. `/create-dev-docs` 로 문서화
5. 단계별 구현 + 주기적 코드 리뷰

🔄 PM2 프로세스 관리

문제점

- 백엔드 마이크로서비스 **7개 동시 구동**
- 서비스 실행 중 Claude가 로그를 볼 수 없음
- 제가 로그를 찾아서 복붙해야 함

PM2 도입 효과

- ✅ 각 서비스가 관리되는 프로세스로 실행
- ✅ **Claude가 실시간으로 로그 읽기 가능**
- ✅ 크래시 시 자동 재시작
- ✅ 메모리/CPU 모니터링

🔍 PM2 전후 비교

Before 😩

```
Me: "이메일 서비스에서 에러가 나"
Me: [로그를 찾아 복사]
Me: [채팅에 붙여넣기]
Claude: "분석해볼게..."
```

After 🚀

```
Me: "이메일 서비스에서 에러가 나"
Claude: [실행] pm2 logs email --lines 200
Claude: "원인 확인 - DB 연결 타임아웃..."
Claude: [실행] pm2 restart email
Claude: "서비스 재시작, 모니터링 중..."
```

🎣 Hooks System

#NoMessLeftBehind

Hook #1: File Edit Tracker

- Edit/Write/MultiEdit 후 수정 파일 기록

Hook #2: Build Checker

- 응답 종료 시 수정된 레포 빌드 체크
- 에러 5개 미만 → 바로 표시
- 에러 5개 이상 → Auto-error-resolver 에이전트 권고

Hook #3: Prettier Formatter

- 수정된 모든 파일 자동 포맷

📋 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

```
Claude가 응답 종료
        ↓
Hook 1: Prettier 포맷터 → 수정 파일 자동 포맷
        ↓
Hook 2: 빌드 체커 → TS 에러 즉시 포착
        ↓
Hook 3: 에러 리마인더 → 에러 핸들링 셀프체크
        ↓
에러가 있으면 → Claude가 즉시 수정
        ↓
결과: 깔끔하고, 포맷됐고, 에러 없는 코드
```

남기는 흔적: 0 🎯

🤖 전문화된 에이전트 군단

Quality Control

- `code-architecture-reviewer` - 아키텍처 관점 코드 리뷰
- `build-error-resolver` - TS 에러 체계적 해결
- `refactor-planner` - 포괄적 리팩터 플랜

Testing & Debugging

- `auth-route-tester` - 인증 라우트 테스트
- `auth-route-debugger` - 401/403 이슈 디버깅
- `frontend-error-fixer` - 프런트엔드 에러 수정

🤖 전문화된 에이전트 군단 (계속)

Planning & Strategy

- `strategic-plan-architect` - 상세 구현 플랜
- `plan-reviewer` - 구현 전 플랜 리뷰
- `documentation-architect` - 문서 생성/업데이트

Specialized

- `frontend-ux-designer` - 스타일/UX 개선
- `web-research-specialist` - 웹 리서치
- `reactour-walkthrough-designer` - UI 투어 제작

⚡ 슬래시 커맨드

Planning & Docs

- `/dev-docs` - 종합 전략 플랜 생성
- `/dev-docs-update` - 컴팩션 전 Dev Docs 갱신
- `/create-dev-docs` - 플랜 → Dev Doc 변환

Quality & Review

- `/code-review` - 아키텍처 관점 코드 리뷰
- `/build-and-fix` - 빌드 + 에러 수정

Testing

- `/route-research-for-testing` - 영향 라우트 탐색
- `/test-route` - 특정 인증 라우트 테스트

💬 프롬프트 팁

핵심 원칙

> "Claude가 너를 위해 뭘 해줄지 묻지 말고,
> Claude에게 어떤 컨텍스트를 줄 수 있을지 물어라."
> ~ 현자님

- **구체적으로 작성** - 원하는 결과를 명확히
- **리딩 질문 피하기** - 중립적으로 의견 요청
- **자주 재프롬프트** - Double ESC로 브랜치
- **직접 개입도 OK** - 2분이면 끝날 일에 30분 허비 금지

🛠️ 유용한 도구들

SuperWhisper on Mac

- 음성으로 프롬프트 입력
- 타자 피로 감소

Memory MCP

- 프로젝트별 결정/아키텍처 선택 추적

BetterTouchTool

- 상대 경로 URL 복사 자동화
- 더블 탭 핫키 (CMD+CMD = Claude Code)
- 앱 간 전환 시간 절약

✅ 결론: 필수 요소

순위	요소	설명
1	**플래닝**	플래닝 모드 또는 strategic-plan-architect
2	**Skills + Hooks**	자동 활성화가 사실상 필수
3	**Dev Docs**	Claude가 플롯을 잃지 않게 함
4	**코드 리뷰**	Claude에게 자기 코드 리뷰시키기
5	**PM2**	백엔드 디버깅이 견딜 만해짐

✨ 결론: 있으면 좋은 것

- 공통 작업용 **전문 에이전트**
- 반복 워크플로우용 **슬래시 커맨드**
- 포괄적 **문서화**
- 스킬에 첨부된 **유틸 스크립트**
- 결정 추적용 **Memory MCP**

🎯 TL;DR

> TypeScript 훅으로 **Claude Code 스킬 자동 활성화** 시스템을 만들고,
> 컨텍스트 유실을 막는 **Dev Docs 워크플로우**와
> **PM2 + 자동 에러 점검**을 도입했다.

결과

6개월 만에 혼자 30만 LOC를 일관된 품질로 리라이트 🎉

🙏 감사합니다

Q&A

원문: Reddit - Claude Code is a beast

번역: RosettaLens