Claude Code – это зверь: советы после 6 месяцев интенсивного использования

> **Перевод оригинального поста:** Claude Code is a Beast – Tips from 6 Months of Hardcore Use (r/ClaudeCode)
>
> **Автор:** u/JokeGold5455

Редактирование: Многие из вас спрашивают про репозиторий, поэтому я постараюсь выложить его в ближайшие пару дней. Всё это часть рабочего проекта, поэтому мне нужно время, чтобы скопировать всё в свежий проект и удалить идентифицирующую информацию. Я выложу ссылку здесь, когда будет готово. Вы также можете подписаться на меня, и я опубликую это в профиле, чтобы вы получили уведомление. Спасибо за добрые комментарии. Я рад поделиться этой информацией с другими, так как у меня не так много возможностей делать это в повседневной жизни.

Редактирование (финальное?): Я собрался с силами и потратил день на создание GitHub репозитория для вас. Только что сделал пост с дополнительной информацией здесь или можете перейти прямо к источнику:

🎯 **Репозиторий:** https://github.com/diet103/claude-code-infrastructure-showcase

Быстрый совет от ленивого человека: Вы можете загрузить этот огромный пост в один из многих AI сервисов преобразования текста в речь, таких как ElevenLabs Reader или Natural Reader и послушать пост 😃

Disclaimer

Я делал пост около шести месяцев назад, делясь своим опытом после недели интенсивного использования Claude Code. Прошло уже около шести месяцев интенсивного использования, и я хотел бы поделиться еще несколькими советами, трюками и потоком мыслей с вами всеми. Возможно, я немного перестарался, так что пристегнитесь, возьмите кофе, усядьтесь на унитаз или что там вы делаете, когда листаете Reddit.

Я хочу начать пост с disclaimer: всё содержимое этого поста — это просто я делюсь тем, какая настройка работает лучше всего для меня сейчас, и не должно восприниматься как истина в последней инстанции или единственно правильный способ делать вещи. Это призвано вдохновить вас на улучшение вашей настройки и рабочих процессов с AI agentic coding. Я просто парень, и это просто, ну знаете, моё мнение, чувак.

Также, я на тарифе 20x Max, так что ваш опыт может отличаться. И если вы ищете советы по vibe-coding, вам стоит поискать в другом месте. Если вы хотите получить максимум от CC, то вы должны работать вместе с ним: планирование, ревью, итерации, исследование разных подходов и т.д.

Quick Overview

После 6 месяцев проверки Claude Code на пределе возможностей (один переписывал 300k LOC), вот система, которую я построил:

- Skills, которые действительно активируются автоматически при необходимости
- Dev docs workflow, который предотвращает потерю фокуса Claude
- PM2 + hooks для zero-errors-left-behind
- Армия специализированных агентов для ревью, тестирования и планирования. Давайте углубимся.

Background

Я software engineer, который работает над production web apps последние семь лет или около того. И я полностью принял волну AI с распростертыми объятиями. Я не слишком беспокоюсь о том, что AI заберёт мою работу в ближайшее время, так как это инструмент, который я использую для расширения своих возможностей. Делая это, я создавал МНОЖЕСТВО новых фич и придумывал всевозможные новые презентации предложений, собранные вместе с Claude и GPT-5 Thinking, для интеграции новых AI систем в наши production приложения. Проекты, о которых я даже не мечтал иметь время подумать до интеграции AI в свой рабочий процесс. И со всем этим я даю себе хорошую долю job security и стал AI гуру на своей работе, так как все остальные примерно на год отстают в том, как они интегрируют AI в свою повседневную работу.

С моей новообретенной уверенностью я предложил довольно большой редизайн/рефакторинг одного из наших web apps, используемых как внутренний инструмент на работе. Это был довольно грубый проект, сделанный студентом колледжа, который был форкнут с другого проекта, разработанного мной как интерном (создан около 7 лет назад и форкнут 4 года назад). Возможно, это было немного слишком амбициозно с моей стороны, так как, чтобы продать это stakeholders, я согласился закончить top-down редизайн этого довольно приличного по размеру проекта (~100k LOC) за два-три месяца... полностью самостоятельно. Я знал, заходя в это, что мне придётся вложить дополнительные часы, чтобы это сделать, даже с помощью CC. Но глубоко внутри я знаю, что это будет хит, автоматизируя несколько ручных процессов и экономя много времени для многих людей в компании.

Прошло шесть месяцев... да, мне, вероятно, не следовало соглашаться на этот timeline. Я проверил пределы как Claude, так и моего собственного здравомыслия, пытаясь довести эту штуку до конца. Я полностью выбросил старый frontend, так как всё было серьёзно устаревшим, и я хотел поиграть с последними и величайшими достижениями. Я говорю React 16 JS → React 19 TypeScript, React Query v2 → TanStack Query v5, React Router v4 w/ hashrouter → TanStack Router w/ file-based routing, Material UI v4 → MUI v7, всё со строгим соблюдением best practices. Проект теперь на ~300-400k LOC, и моя продолжительность жизни на ~5 лет короче. Он наконец готов к тестированию, и я невероятно доволен тем, как всё получилось.

Раньше это был проект с непреодолимым tech debt, НУЛЕВЫМ test coverage, УЖАСНЫМ developer experience (тестирование было абсолютным кошмаром) и всякой jank происходящей. Я решил все эти проблемы с приличным test coverage, управляемым tech debt и реализовал command-line tool для генерации test data, а также dev mode для тестирования различных фич на frontend. За это время я хорошо узнал возможности CC и чего от него ожидать.

A Note on Quality and Consistency

Я заметил повторяющуюся тему на форумах и в дискуссиях — люди испытывают фрустрацию с лимитами использования и беспокойство о снижении качества output со временем. Я хочу быть ясным с самого начала: я не здесь, чтобы отклонять эти переживания или утверждать, что это просто вопрос "неправильного использования". У всех разные use cases и контексты, и обоснованные беспокойства заслуживают того, чтобы их услышали.

При этом, я хочу поделиться тем, что работает для меня. По моему опыту, output CC на самом деле значительно улучшился за последние пару месяцев, и я верю, что это в значительной степени благодаря workflow, который я постоянно совершенствую. Моя надежда в том, что если вы возьмёте даже небольшую часть вдохновения из моей системы и интегрируете её в свой CC workflow, вы дадите ей лучший шанс производить качественный output, которым вы будете довольны.

Теперь, давайте будем реалистами — абсолютно бывают моменты, когда Claude полностью промахивается мимо цели и производит suboptimal код. Это может происходить по различным причинам. Во-первых, AI модели stochastic, что означает, что вы можете получить сильно различающиеся outputs из одного и того же input. Иногда случайность просто не в вашу пользу, и вы получаете output, который законно плохого качества не по вашей вине. В другие разы это о том, как структурирован prompt. Могут быть значительные различия в outputs при слегка различной формулировке, потому что модель воспринимает вещи довольно буквально. Если вы неправильно сформулируете или выразите что-то неоднозначно, это может привести к значительно худшим результатам.

Sometimes You Just Need to Step In

Смотрите, AI невероятен, но это не магия. Есть определенные проблемы, где pattern recognition и человеческая интуиция просто побеждают. Если вы провели 30 минут, наблюдая, как Claude борется с чем-то, что вы могли бы исправить за 2 минуты, просто исправьте это сами. Нет стыда в этом. Думайте об этом как об обучении кого-то ездить на велосипеде — иногда вам просто нужно придержать руль на секунду, прежде чем снова отпустить.

Я видел это особенно с logic puzzles или проблемами, которые требуют real-world common sense. AI может brute-force многие вещи, но иногда человек просто "понимает" быстрее. Не позволяйте упрямству или какому-то misguided чувству "но AI должен делать всё" тратить ваше время. Вмешайтесь, исправьте проблему и двигайтесь дальше.

У меня была своя доля ужасного prompting, что обычно происходит ближе к концу дня, где я становлюсь ленивым и не вкладываю столько усилий в свои prompts. И результаты действительно показывают это. Так что в следующий раз, когда у вас будут такие проблемы, где вы думаете, что output намного хуже в эти дни, потому что вы думаете, что Anthropic shadow-nerfed Claude, я призываю вас сделать шаг назад и подумать о том, как вы делаете prompting.

Re-prompt часто. Вы можете нажать double-esc, чтобы вызвать ваши предыдущие prompts и выбрать один для branch. Вы будете удивлены, как часто вы можете получить гораздо лучшие результаты, вооруженные знанием того, чего вы не хотите, при даче того же prompt. Всё это для того, чтобы сказать, что может быть много причин, почему качество output кажется хуже, и хорошо делать самоанализ и рассматривать, что вы можете сделать, чтобы дать ему наилучший возможный шанс получить output, который вы хотите.

Как, вероятно, сказал какой-то мудрый чувак где-то: "Не спрашивай, что Claude может сделать для тебя, спроси, какой контекст ты можешь дать Claude" ~ Wise Dude

Хорошо, я собираюсь спуститься со своей soapbox сейчас и перейти к хорошим вещам.

My System

Я внедрил много изменений в свой workflow, связанный с CC, за последние 6 месяцев, и результаты были довольно великолепными, IMO.

Skills Auto-Activation System (Game Changer!)

Это заслуживает своего собственного раздела, потому что полностью трансформировало то, как я работаю с Claude Code.

The Problem

Итак, Anthropic выпускает эту фичу Skills, и я думаю "это выглядит потрясающе!" Идея иметь эти портативные, переиспользуемые guidelines, на которые Claude может ссылаться, звучала идеально для поддержания консистентности в моей массивной codebase. Я потратил приличный кусок времени с Claude, написав comprehensive skills для frontend development, backend development, database operations, workflow management и т.д. Мы говорим о тысячах строк best practices, паттернов и примеров.

А потом... ничего. Claude просто не хотел их использовать. Я буквально использовал точные keywords из skill descriptions. Ничего. Я работал над файлами, которые должны были triggering skills. Ничего. Это было невероятно фрустрирующе, потому что я мог видеть потенциал, но skills просто сидели там как дорогие украшения.

The "Aha!" Moment

Вот тогда у меня появилась идея использовать **hooks**. Если Claude не будет автоматически использовать skills, что если я построю систему, которая ЗАСТАВИТ его проверять релевантные skills перед тем, как что-либо делать?

Итак, я углубился в hook систему Claude Code и построил multi-layered auto-activation архитектуру с TypeScript hooks. И это действительно работает!

How It Works

Я создал два основных hooks:

1\. UserPromptSubmit Hook (запускается ДО того, как Claude видит ваше сообщение):

- Анализирует ваш prompt на keywords и intent patterns
- Проверяет, какие skills могут быть релевантны

- Inject formatted reminder в контекст Claude
- Теперь, когда я спрашиваю "как работает layout система?" Claude видит большое "🎯 SKILL ACTIVATION CHECK - Use project-catalog-developer skill" (project catalog это большая сложная data grid based фича на моём front end) перед тем, как даже прочитать мой вопрос

2\. Stop Event Hook (запускается ПОСЛЕ того, как Claude заканчивает отвечать):

- Анализирует, какие файлы были отредактированы
- Проверяет на risky patterns (try-catch blocks, database operations, async functions)
- Отображает gentle self-check reminder
- "Вы добавили error handling? Prisma operations используют repository pattern?"
- Non-blocking, просто держит Claude aware без раздражения

skill-rules.json Configuration

Я создал центральный configuration файл, который определяет каждый skill с:

- **Keywords**: Explicit topic matches ("layout", "workflow", "database")
- **Intent patterns**: Regex для catch actions ("(create|add).*?(feature|route)")
- **File path triggers**: Активируется на основе того, какой файл вы редактируете
- **Content triggers**: Активируется, если файл содержит specific patterns (Prisma imports, controllers и т.д.)

Example snippet:

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

The Results

Теперь, когда я работаю над backend кодом, Claude автоматически:

1. Видит skill suggestion перед чтением моего prompt
2. Загружает релевантные guidelines
3. На самом деле следует паттернам консистентно
4. Self-checks в конце через gentle reminders

Разница как день и ночь. Больше нет inconsistent кода. Больше нет "подожди, Claude снова использовал старый паттерн." Больше нет manual сообщения ему проверять guidelines каждый раз.

Following Anthropic's Best Practices (The Hard Way)

После того, как auto-activation заработала, я углубился дальше и нашёл официальные best practices docs Anthropic. Оказывается, я делал это неправильно, потому что они рекомендуют держать основной SKILL.md файл **под 500 строк** и использовать progressive disclosure с resource files.

Упс. Мой frontend-dev-guidelines skill был 1,500+ строк. И у меня было пару других skills свыше 1,000 строк. Эти monolithic файлы подрывали всю цель skills (загружать только то, что вам нужно).

Итак, я реструктурировал всё:

- **frontend-dev-guidelines**: 398-строчный main file + 10 resource files
- **backend-dev-guidelines**: 304-строчный main file + 11 resource files

Теперь Claude загружает lightweight main file изначально и только подтягивает детальные resource files, когда действительно нужно. Token efficiency улучшилась на 40-60% для большинства запросов.

Skills I've Created

Вот мой текущий skill lineup:

Guidelines & Best Practices:

- `backend-dev-guidelines` - Routes → Controllers → Services → Repositories
- `frontend-dev-guidelines` - React 19, MUI v7, TanStack Query/Router patterns
- `skill-developer` - Meta-skill для создания больше skills

Domain-Specific:

- `workflow-developer` - Complex workflow engine patterns
- `notification-developer` - Email/notification система
- `database-verification` - Предотвращает column name errors (это guardrail, который на самом деле blocks edits!)

- `project-catalog-developer` - DataGrid layout система

Все они автоматически активируются на основе того, над чем я работаю. Это как иметь senior dev, который на самом деле помнит все паттерны, смотрящего через плечо Claude.

Why This Matters

До skills + hooks:

- Claude использовал бы старые паттерны, даже если я документировал новые
- Приходилось manually сообщать Claude проверять BEST_PRACTICES.md каждый раз
- Inconsistent код по всей 300k+ LOC codebase
- Тратил слишком много времени на fixing "креативных интерпретаций" Claude

После skills + hooks:

- Consistent паттерны автоматически enforced
- Claude self-corrects до того, как я даже вижу код
- Могу доверять, что guidelines соблюдаются
- Намного меньше времени тратится на reviews и fixes

Если вы работаете над большой codebase с установленными паттернами, я не могу достаточно рекомендовать эту систему. Начальная setup заняла пару дней, чтобы сделать правильно, но окупилась в десять раз.

CLAUDE.md and Documentation Evolution

В посте, который я написал 6 месяцев назад, у меня была секция о том, что rules — ваш лучший друг, с чем я всё ещё согласен. Но мой CLAUDE.md файл быстро выходил из-под контроля и пытался делать слишком много. У меня также был этот массивный BEST_PRACTICES.md файл (1,400+ строк), который Claude иногда читал, а иногда полностью игнорировал.

Итак, я потратил день с Claude, чтобы consolidate и reorganize всё в новую систему. Вот что изменилось:

What Moved to Skills

Ранее, BEST_PRACTICES.md содержал:

- TypeScript standards
- React patterns (hooks, components, suspense)
- Backend API patterns (routes, controllers, services)
- Error handling (Sentry integration)

- Database patterns (Prisma usage)
- Testing guidelines
- Performance optimization

Всё это теперь в skills с auto-activation hook, обеспечивающим, что Claude на самом деле их использует. Больше нет надежды, что Claude вспомнит проверить BEST_PRACTICES.md.

What Stayed in CLAUDE.md

Теперь CLAUDE.md laser-focused на **project-specific info** (только ~200 строк):

- Quick commands (`pnpm pm2:start`, `pnpm build` и т.д.)
- Service-specific configuration
- Task management workflow (dev docs система)
- Testing authenticated routes
- Workflow dry-run mode
- Browser tools configuration

The New Structure

```
Root CLAUDE.md (100 lines)
├── Critical universal rules
├── Points to repo-specific claude.md files
└── References skills for detailed guidelines

Each Repo's claude.md (50-100 lines)
├── Quick Start section pointing to:
│   ├── PROJECT_KNOWLEDGE.md - Architecture & integration
│   ├── TROUBLESHOOTING.md - Common issues
│   └── Auto-generated API docs
└── Repo-specific quirks and commands
```

Магия: Skills обрабатывают все "как писать код" guidelines, а CLAUDE.md обрабатывает "как работает этот specific проект." Separation of concerns для победы.

Dev Docs System

Эта система, из всего (кроме skills), я думаю, оказала наибольшее влияние на результаты, которые я получаю из CC. Claude как extremely confident junior dev с extreme amnesia, легко теряющий track того, что он делает. Эта система нацелена на решение этих недостатков.

Dev docs секция из моего CLAUDE.md:

```
### Starting Large Tasks

When exiting plan mode with an accepted plan: 1.**Create Task Directory**:
mkdir -p ~/git/project/dev/active/[task-name]/

2.**Create Documents**:

- `[task-name]-plan.md` - The accepted plan
- `[task-name]-context.md` - Key files, decisions
- `[task-name]-tasks.md` - Checklist of work

3.**Update Regularly**: Mark tasks complete immediately

### Continuing Tasks

- Check `/dev/active/` for existing tasks
- Read all three files before proceeding
- Update "Last Updated" timestamps
```

Это документы, которые всегда создаются для каждой фичи или большой задачи. До использования этой системы у меня было много раз, когда я вдруг осознавал, что Claude потерял plot, и мы больше не реализовывали то, что мы планировали 30 минут назад, потому что мы ушли в какой-то tangent по какой-то причине.

My Planning Process

Мой process начинается с планирования. **Планирование — король**. Если вы не используете как минимум planning mode перед тем, как просить Claude что-то реализовать, у вас будет плохое время, ммм'кей. Вы бы не позволили строителю прийти к вашему дому и начать лепить пристройку без того, чтобы он сначала нарисовал планы.

Когда я начинаю планировать фичу, я ставлю это в planning mode, даже если в конечном итоге попрошу Claude записать план в markdown файл. Я не уверен, что putting в planning mode необходимо, но для меня кажется, что planning mode даёт лучшие результаты при research вашей codebase и получении всего правильного контекста, чтобы иметь возможность составить план.

Я создал `strategic-plan-architect` subagent, который basically planning beast. Он:

- Собирает контекст эффективно
- Анализирует project structure
- Создаёт comprehensive structured plans с executive summary, phases, tasks, risks, success metrics, timelines
- Генерирует три файла автоматически: plan, context и tasks checklist

Но я нахожу действительно раздражающим, что вы не можете видеть output агента, и ещё более раздражающим является то, что если вы говорите нет плану, он просто убивает агента вместо продолжения planning. Поэтому я также создал custom slash command (`/dev-docs`) с тем же prompt для использования на main CC instance.

Как только Claude выдаёт этот beautiful план, я трачу время на тщательный review. **Этот шаг действительно важен.** Потратьте время, чтобы понять его, и вы будете удивлены, как часто вы ловите silly mistakes или Claude неправильно понимает очень vital часть запроса или задачи.

Чаще всего я буду на 15% context left или меньше после выхода из plan mode. Но это нормально, потому что мы собираемся положить всё, что нам нужно, чтобы начать fresh в наши dev docs. Claude обычно любит просто jump in guns blazing, поэтому я немедленно slap ESC key, чтобы interrupt и запустить мой `/dev-docs` slash command. Команда берёт approved план и создаёт все три файла, иногда делая немного больше research, чтобы fill in gaps, если осталось достаточно контекста.

И как только я закончил с этим, я в принципе готов заставить Claude полностью реализовать фичу без потери или losing track того, что он делал, даже через auto-compaction. Я просто убеждаюсь, что напоминаю Claude время от времени обновлять tasks, а также context файл с любым релевантным контекстом. И как только у меня running low на контекст в текущей сессии, я просто запускаю мой slash command `/update-dev-docs`. Claude отметит любой релевантный контекст (с next steps), а также mark любые completed tasks или add новые tasks перед тем, как я compact conversation. И всё, что мне нужно сказать, это "continue" в новой сессии.

Во время implementation, в зависимости от размера фичи или задачи, я специально скажу Claude реализовать только одну или две секции за раз. Таким образом, я получаю шанс войти и review код между каждым set of tasks. И периодически у меня subagent также reviewing изменения, чтобы я мог catch большие mistakes рано. Если вы не заставляете Claude review свой собственный код, то я очень рекомендую это, потому что это сохранило мне много headaches, catching critical errors, missing implementations, inconsistent код и security flaws.

PM2 Process Management (Backend Debugging Game Changer)

Это относительно недавнее добавление, но сделало debugging backend issues намного легче.

The Problem

Мой проект имеет семь backend microservices, работающих одновременно. Проблема была в том, что Claude не имел access к просмотру логов, пока services работали. Я не мог просто спросить "что не так с email service?" - Claude не мог видеть логи без того, чтобы я manually copy и paste их в chat.

The Intermediate Solution

Какое-то время у меня каждый service писал свой output в timestamped log файл, используя `devLog` script. Это работало... нормально. Claude мог читать log файлы, но это было clunky. Логи не были real-time, services не auto-restart on crashes, и managing всего было pain.

The Real Solution: PM2

Затем я обнаружил PM2, и это был game changer. Я настроил все мои backend services для запуска через PM2 одной командой: `pnpm pm2:start`

Что это даёт мне:

- Каждый service запускается как managed process со своим own log файлом
- **Claude может легко читать individual service logs в real-time**
- Automatic restarts on crashes
- Real-time monitoring с `pm2 logs`
- Memory/CPU monitoring с `pm2 monit`
- Easy service management (`pm2 restart email`, `pm2 stop all` и т.д.)

PM2 Configuration:

```
// ecosystem.config.jsmodule.exports = {
  apps: [
    {
      name: 'form-service',
      script: 'npm',
      args: 'start',
      cwd: './form',
      error_file: './form/logs/error.log',
      out_file: './form/logs/out.log',
    },
// ... 6 more services
  ]
};
```

До PM2:

```
Я: "Email service выдаёт errors"
Я: [Manually находит и копирует логи]
Я: [Вставляет в chat]
Claude: "Дайте мне проанализировать это..."
```

Debugging workflow теперь:

```
Я: "Email service выдаёт errors"
Claude: [Запускает] pm2 logs email --lines 200
Claude: [Читает логи] "Я вижу проблему - database connection timeout..."
Claude: [Запускает] pm2 restart email
Claude: "Перезапустил service, monitoring для errors..."
```

Разница как день и ночь. Claude теперь может autonomously debug issues без того, чтобы я был human log-fetching service.

Одна caveat: Hot reload не работает с PM2, поэтому я всё ещё запускаю frontend отдельно с `pnpm dev`. Но для backend services, которые не нуждаются в hot reload так часто, PM2 incredible.

Hooks System (#NoMessLeftBehind)

Проект, над которым я работаю, multi-root и имеет около восьми различных repos в root project directory. Один для frontend и семь microservices и utilities для backend. Я постоянно bounce around, делая изменения в паре repos одновременно, в зависимости от фичи.

И одна вещь, которая раздражала меня до бесконечности, это когда Claude забывает запустить build command в каком бы то ни было repo, который он редактирует, чтобы catch errors. И он просто оставит дюжину или около того TypeScript errors без того, чтобы я это заметил. Затем пару часов спустя я вижу, что Claude запускает build script как хороший boy, и я вижу output: "Есть несколько TypeScript errors, но они unrelated, так что мы здесь all good!"

Нет, мы не good, Claude.

Hook #1: File Edit Tracker

Во-первых, я создал **post-tool-use hook**, который запускается после каждой Edit/Write/MultiEdit операции. Он логирует:

- Какие файлы были отредактированы
- К какому repo они принадлежат
- Timestamps

Изначально я сделал его запускающим builds немедленно после каждого edit, но это было stupidly inefficient. Claude делает edits, которые break вещи всё время перед тем, как quickly fix их.

Hook #2: Build Checker

Затем я добавил **Stop hook**, который запускается, когда Claude заканчивает отвечать. Он:

1. Читает edit логи, чтобы найти, какие repos были modified
2. Запускает build scripts на каждом affected repo
3. Проверяет на TypeScript errors
4. Если < 5 errors: Показывает их Claude
5. Если ≥ 5 errors: Рекомендует launching auto-error-resolver agent
6. Логирует всё для debugging

С момента implementation этой системы у меня не было ни одного instance, где Claude оставил errors в коде для меня, чтобы найти позже. Hook catches их немедленно, и Claude fixes их перед тем, как двигаться дальше.

Hook #3: Prettier Formatter

Этот простой, но effective. После того, как Claude заканчивает отвечать, автоматически format все edited файлы с Prettier, используя appropriate `.prettierrc` config для того repo.

Больше нет going into для manually edit файла просто чтобы prettier запустился и произвёл 20 changes, потому что Claude решил leave off trailing commas на прошлой неделе, когда мы создавали этот файл.

⚠ Update: Я больше не рекомендую этот Hook

После publishing, reader поделился detailed data, показывающими, что file modifications trigger `<system-reminder>` notifications, которые могут consume significant context tokens. В их случае Prettier formatting привёл к 160k tokens consumed всего за 3 rounds из-за system-reminders, показывающих file diffs.

Хотя impact варьируется по проекту (большие файлы и strict formatting rules являются worst-case scenarios), я удаляю этот hook из своей setup. Это не big deal позволить formatting происходить, когда вы manually edit файлы в любом случае, и potential token cost не стоит convenience.

Если вы хотите automatic formatting, рассмотрите запуск Prettier manually между sessions вместо during Claude conversations.

Hook #4: Error Handling Reminder

Это gentle philosophy hook, который я упоминал ранее:

- Анализирует edited файлы после того, как Claude заканчивает
- Detects risky patterns (try-catch, async operations, database calls, controllers)
- Показывает gentle reminder, если risky код был написан
- Claude self-assesses, нужен ли error handling
- No blocking, no friction, просто awareness

Example output:

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

⚠️  Backend Changes Detected
   2 file(s) edited

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

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

The Complete Hook Pipeline

Вот что происходит на каждом Claude response теперь:

```
Claude finishes responding
  ↓
Hook 1: Prettier formatter runs → All edited files auto-formatted
  ↓
Hook 2: Build checker runs → TypeScript errors caught immediately
  ↓
Hook 3: Error reminder runs → Gentle self-check for error handling
  ↓
If errors found → Claude sees them and fixes
  ↓
If too many errors → Auto-error-resolver agent recommended
  ↓
Result: Clean, formatted, error-free code
```

И UserPromptSubmit hook обеспечивает, что Claude загружает relevant skills ДО даже начала работы.

No mess left behind. Это beautiful.

Scripts Attached to Skills

Один действительно cool паттерн, который я подхватил из официальных skill примеров Anthropic на GitHub: **attach utility scripts к skills**.

Например, мой `backend-dev-guidelines` skill имеет секцию о testing authenticated routes. Вместо простого объяснения, как работает authentication, skill ссылается на actual script:

```
### Testing Authenticated Routes

Use the provided test-auth-route.js script:

`node scripts/test-auth-route.js http://localhost:3002/api/endpoint`
```

Script обрабатывает все сложные authentication steps для вас:

1. Получает refresh token от Keycloak
2. Подписывает token с JWT secret
3. Создаёт cookie header
4. Делает authenticated request

Когда Claude нужно протестировать route, он знает точно, какой script использовать и как его использовать. Больше нет "позвольте мне создать test script" и reinventing wheel каждый раз.

Я планирую расширить этот паттерн - attach больше utility scripts к relevant skills, чтобы у Claude были ready-to-use инструменты вместо generating их с нуля.

Tools and Other Things

SuperWhisper on Mac

Voice-to-text для prompting, когда мои руки устали от typing. Работает surprisingly well, и Claude понимает мой rambling voice-to-text surprisingly well.

Memory MCP

Я использую это меньше со временем теперь, когда skills handle большую часть "remembering patterns" работы. Но это всё ещё полезно для tracking project-specific decisions и architectural choices, которые не принадлежат skills.

BetterTouchTool

- Relative URL copy из Cursor (для sharing code references)
    - У меня VSCode открыт, чтобы легче находить файлы, которые я ищу, и я могу double tap CAPS-LOCK, затем BTT вводит shortcut для copy relative URL, transforms clipboard contents, prepending '@' symbol, focuses terminal, и затем pastes file path. Всё в одно.
- Double-tap hotkeys для быстрого focus apps (CMD+CMD = Claude Code, OPT+OPT = Browser)
- Custom gestures для common actions

Честно говоря, экономия времени на просто не fumbling между apps стоит BTT purchase alone.

Scripts for Everything

Если есть любая annoying tedious задача, chances are есть script для этого:

- Command-line tool для генерации mock test data. До использования Claude code было extremely annoying генерировать mock data, потому что мне приходилось делать submission в форму, которая имела около 120 вопросов, просто чтобы сгенерировать одну single test submission.
- Authentication testing scripts (get tokens, test routes)
- Database resetting и seeding
- Schema diff checker перед migrations
- Automated backup и restore для dev database

Pro tip: Когда Claude помогает вам написать useful script, немедленно document его в CLAUDE.md или attach к relevant skill. Future you поблагодарит past you.

Documentation (Still Important, But Evolved)

Я думаю, что рядом с planning, documentation почти так же important. Я document всё, как я иду, в дополнение к dev docs, которые создаются для каждой задачи или фичи. От system architecture до data flow diagrams до actual developer docs и APIs, just to name a few.

Но вот что изменилось: Documentation теперь работает WITH skills, не вместо них.

Skills содержат: Reusable patterns, best practices, how-to guides **Documentation содержит:** System architecture, data flows, API references, integration points

Например:

- "Как создать controller" → **backend-dev-guidelines skill**
- "Как работает наш workflow engine" → **Architecture documentation**
- "Как писать React components" → **frontend-dev-guidelines skill**
- "Как notifications текут через систему" → **Data flow diagram + notification skill**

У меня всё ещё МНОГО docs (850+ markdown файлов), но теперь они laser-focused на project-specific architecture, а не repeating general best practices, которые лучше обслуживаются skills.

Вам не обязательно идти так crazy, но я очень рекомендую setting up multiple levels of documentation. Ones для broad architectural overview specific services, wherein вы будете include paths к другой documentation, которая goes into more specifics различных частей architecture. Это сделает major difference на способности Claude легко navigate вашу codebase.

Prompt Tips

Когда вы пишете свой prompt, вы должны попытаться быть as specific as possible о том, что вы хотите в результате. Ещё раз, вы бы не попросили строителя прийти и построить вам новую ванную без по крайней мере discussing plans, верно?

"Вы абсолютно правы! Shag carpet вероятно не лучшая идея иметь в ванной."

Иногда вы можете не знать specifics, и это okay. Если вы не задаёте вопросы, скажите Claude исследовать и вернуться с несколькими potential solutions. Вы даже можете использовать specialized subagent или использовать любой другой AI chat interface для вашего research. Мир — ваша устрица. Я обещаю вам, это pay dividends, потому что вы сможете посмотреть на план, который Claude произвёл, и иметь better idea, если он good, bad или needs adjustments. Иначе вы просто flying blind, pure vibe-coding. Затем вы окажетесь в ситуации, где вы даже не знаете, какой контекст include, потому что вы не знаете, какие файлы related к тому, что вы пытаетесь fix.

Постарайтесь не lead в ваших prompts, если вы хотите honest, unbiased feedback. Если вы unsure о чём-то, что Claude сделал, спросите об этом neutral way вместо того, чтобы говорить "Это good или bad?" Claude tends to tell you, что он thinks you want to hear, поэтому leading questions могут skew response. Лучше просто describe ситуацию и ask for thoughts или alternatives. Таким образом, вы получите more balanced answer.

Agents, Hooks, and Slash Commands (The Holy Trinity)

Agents

Я построил небольшую армию specialized agents:

Quality Control:

- `code-architecture-reviewer` - Reviews код для best practices adherence
- `build-error-resolver` - Systematically fixes TypeScript errors
- `refactor-planner` - Creates comprehensive refactoring plans

Testing & Debugging:

- `auth-route-tester` - Tests backend routes с authentication
- `auth-route-debugger` - Debugs 401/403 errors и route issues
- `frontend-error-fixer` - Diagnoses и fixes frontend errors

Planning & Strategy:

- `strategic-plan-architect` - Creates detailed implementation plans
- `plan-reviewer` - Reviews plans перед implementation
- `documentation-architect` - Creates/updates documentation

Specialized:

- `frontend-ux-designer` - Fixes styling и UX issues
- `web-research-specialist` - Researches issues along с many other things на web
- `reactour-walkthrough-designer` - Creates UI tours

Key с agents это давать им **very specific roles** и clear instructions на то, что return. Я learned это hard way после creating agents, которые would go off и do who-knows-what и come back с "I fixed it!" без telling мне, что они fixed.

Hooks (Covered Above)

Hook система honestly то, что ties всё вместе. Без hooks:

- Skills sit unused
- Errors slip through
- Код inconsistently formatted
- No automatic quality checks

C hooks:

- Skills auto-activate
- Zero errors left behind
- Automatic formatting
- Quality awareness built-in

Slash Commands

У меня quite a few custom slash commands, но это те, которые я use most:

Planning & Docs:

- `/dev-docs` - Create comprehensive strategic plan
- `/dev-docs-update` - Update dev docs перед compaction
- `/create-dev-docs` - Convert approved plan в dev doc files

Quality & Review:

- `/code-review` - Architectural code review
- `/build-and-fix` - Run builds и fix all errors

Testing:

- `/route-research-for-testing` - Find affected routes и launch tests
- `/test-route` - Test specific authenticated routes

Beauty of slash commands это то, что они expand в full prompts, поэтому вы можете pack тонну контекста и instructions в simple command. Way better, чем typing out те же instructions каждый раз.

Conclusion

После шести месяцев hardcore use, вот что я learned:

Essentials:

1. **Plan everything** - Use planning mode или strategic-plan-architect
2. **Skills + Hooks** - Auto-activation это единственный way skills actually work reliably
3. **Dev docs система** - Prevents Claude от losing the plot
4. **Code reviews** - Have Claude review его own work
5. **PM2 для backend** - Makes debugging actually bearable

Nice-to-Haves:

- Specialized agents для common tasks
- Slash commands для repeated workflows
- Comprehensive documentation
- Utility scripts attached к skills
- Memory MCP для decisions

И это about all I can think of на сейчас. Как я сказал, я просто some guy, и я would love услышать tips и tricks от всех остальных, а также any criticisms. Потому что я always up для improving upon мой workflow. Я honestly просто хотел share то, что работает для меня, с другими людьми, since я don't really have anybody else to share this with IRL (моя team очень small, и они все very slow getting on AI train).

Если вы сделали это до сюда, спасибо за то, что took time to read. Если у вас questions о любом из этого stuff или want more details на implementation, happy to share. Hooks и skills система especially took some trial and error to get right, но теперь, когда это working, я can't imagine going back.

TL;DR: Построил auto-activation систему для Claude Code skills, используя TypeScript hooks, создал dev docs workflow для prevent context loss и implemented PM2 + automated error checking. Результат: Solo переписал 300k LOC за 6 месяцев с consistent quality.