Все команды Claude Code: чек-лист

ОТ НОВИЧКА ДО АВТОНОМНЫХ АГЕНТОВ

Чаще всего люди используют **Claude Code как обычный чат** — и теряют главную фишку: преднастроенные команды. Ниже — вся система команд лестницей: от трёх базовых, которые надо знать с первого дня, до субагентов, автономного режима и графа знаний. Многое работает не только в Claude Code, но и в Codex и Mimo Code.

Базовые команды и гигиена контекста

Базовые: знать с первого дня

→ **/help**

Справка по командам; нужна новичкам, доступна только в терминале

→ **/init**

Создаёт CLAUDE.md, но лучше сразу генерировать его промтами — выйдет правильнее

→ **/clear**

Полностью стирает диалог («чистый стол»). Это **НЕ** сжатие — модель не запомнит ничего

Гигиена контекста: три команды, которые экономят токены

/context

Показывает, на что потрачены токены

/compact

Сжимает диалог, сохраняя смысл и даёт продолжить в том же чате

ⓘ На практике: довожу до ~90% окна → делаю /compact → продолжаю. Контекстное окно бывает до **1 млн токенов**. По качеству после compact разницы почти нет — просто удобнее.

Выбор модели и контроль откатов

🔀 **/model**

Выбор модели (в расширении — кнопка **Switch model** при вводе /). Совет автора: **Sonnet с окном 1 млн** контекста; модель по умолчанию — Opus

⚡ Fast-режим

Быстрый режим (в примере Opus 4.6): когда нужно делать задачи быстро. Для быстрых задач автор часто уходит в Codex — там fast в ~1.5× быстрее, лимиты больше

📋 Plan Mode

`plan` в терминале / кнопка в расширении: модель сперва изучает базу, готовится и выдаёт понятный план → качество больших задач кратно выше. Оправдан на **больших задачах** (~500к токенов)

⏪ Rewind

Откат изменений в терминале; на практике удобнее держать историю в **git** — он не только для кода, годится и для контент-проектов

🔒 /permissions

Что Claude может делать без спроса, а что нет. По сути — **модель безопасности** твоей работы

Кастомизация под себя

CLAUDE.md и AGENTS.md

Описывает проект агенту. Для Codex/OpenCode дублируется в AGENTS.md, чтобы агенты работали вместе. Принцип: описал процесс один раз → дальше зовёшь командой

Skills и свои команды

Навыки под себя. Одной командой запускать цепочку: аудит оптимизации, безопасности, техдолга → потом автономный фикс. Вспомни любую повторяющуюся задачу и автоматизируй её

Реальный проект: YouTube-система

В пример берётся `CLAUDE.md` из **не-кодового** проекта — YouTube-системы (в GitHub, с ней работают и люди, и агенты)

Память проекта и субагенты

📂 /memory — память проекта

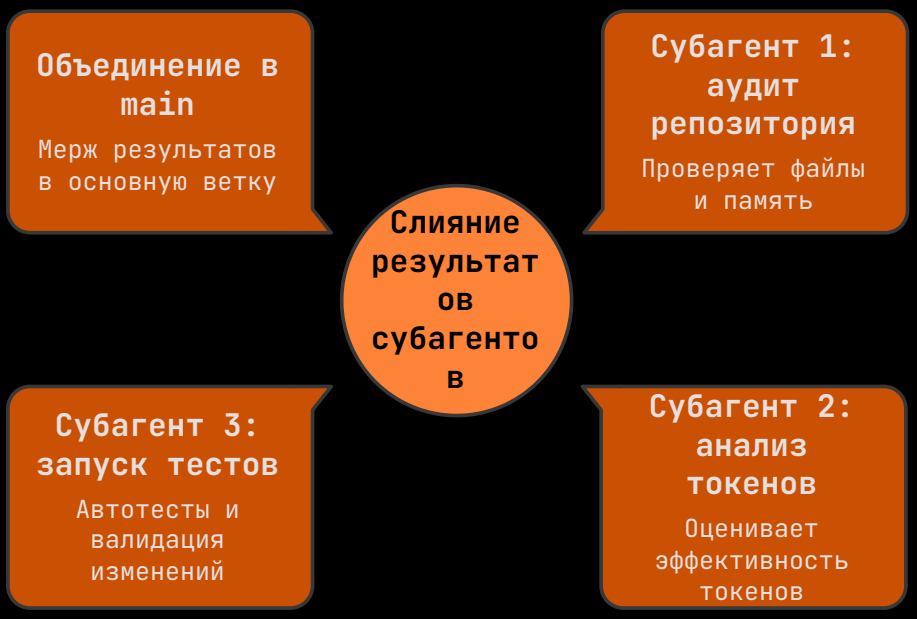
Выполняется в терминале (даже из расширения попросит открыть терминал). Смотрит репозиторий и **дописывает память проекта**, к которой потом обращаются агенты.

- В примере создал профиль (3 YouTube-канала, стиль), дату вывода, описание мастер-скиллов
- Все Skills можно вызывать через /, чтобы Claude точно их использовал
- **Skills храни не только глобально**, но и в папке `skills/` каждого репозитория — иначе OpenCode/Codex, вытянув репо из GitHub, не найдёт нужных навыков

🤖 /agents — параллельная работа

Субагенты делают подзадачи параллельно: один читает файлы, другой — другие, третий тестирует.

- Под каждого субагента создаётся отдельная документация; инструкции бери из AGENTS.md
- В расширении проговаривай прямо: «запусти параллельных субагентов, инструкции из agents, без skills: 1-й — аудит репо, 2-й — как тратить меньше токенов, 3-й — тестирование»
- С git субагенты могут заводить **себе отдельные ветки/копии** файлов, всё тестировать и только потом сливать в основную



Автоматизация: Hooks, MCP и автономность

Hooks

Субагенты/действия запускаются автоматически по событию. Пример: на коммит+push сначала срабатывает **аудит безопасности**, и push не уйдёт, пока аудит не пройден

MCP

Связь Claude Code с внешними инструментами (Notion, GitHub и т.д.). **MCP builder** — команда создания своего MCP: подключить свою систему задач, чтобы Claude давал ежедневные сводки и аналитику эффективности

Goal

Claude работает до самого финала без тебя; но нужно **детальное ТЗ**, иначе нейрослоп.

Loop

Задача по расписанию (например, каждую среду искать баги). Сильна для проверки, что **все файлы на местах и агенты не забили на инструкции**

Code Review

Маст-хэв: автоматически проводит аудит кода и безопасности, строит план доработок и автономно их реализует. **В режиме ultra** — несколько параллельных субагентов проверяют код перед отправкой в git

Уровень выше: граф знаний Graphify и что он умеет

→ **Open-source, ставится одной командой**

Прямо из диалога поставит решение, но **сам граф связей строится в обычном терминале** отдельной командой

→ **Разбирает любые файлы**

Код, PDF, PNG, видео (сам делает транскрипции) → строит систему связей, часто эффективнее RAG и Obsidian-вики

→ **Работает под любым агентом**

Claude Code, Codex, OpenCode (RMS), Mimo Code. Граф автор строит на M4 Max — дальше визуализация связей

Установка и использование

01

Установка

`pip install graphifyy` && `graphify install` (пакет **graphifyy** — двойная «y»); при ошибке _externally-managed_ / PATH → `pipx install graphifyy`
Ссылка на GitHub: github.com/safishamsi/graphify

02

Построить граф

В обычном терминале: `/graphify` · своя папка — `/graphify ./path` · только изменённые — `--update` · глубже — `--mode deep`

03

Любые файлы

Разбирает код, PDF, PNG, видео (сам делает транскрипции) → строит систему связей, часто эффективнее RAG и Obsidian-вики

04

Работает везде

Под любым агентом: Claude Code, Codex, OpenCode (RMS), Mimo Code. Граф строится на M4 Max — дальше визуализация связей

📺 **Канал**

Нейропросвещение · Рома Райт