The Claude Code Ecosystem.

A systematic guide to everything that connects to Claude Code — IDEs, GitHub, MCPs, Skills, Hooks, and more. The complete picture that no one else has put in one place.

HOW EVERYTHING CONNECTS

Your IDE (VS Code / JetBrains / Cursor) → Claude Code CLI → MCP Servers (Notion, Gmail, Stripe...)

GitHub (repo + Actions + PR review) → Claude Code CLI → Skills (.claude/skills/ or commands/)

CLAUDE.md (project memory) → Claude Code CLI → Hooks (pre/post tool automation)

CLAUDE.md

FOUNDATION

Your project's permanent memory — the first thing Claude reads every session

WHAT IT IS

A markdown file that sits in your project folder. Claude reads it automatically at the start of every session. It's your standing instructions, project context, and operating rules — all in one place. No need to re-explain yourself every conversation.

WHAT TO PUT IN IT

- → Project overview and architecture
- → Operating rules (how Claude should behave)
- → Tool and credential locations
- → Banned behaviors and approval gates
- → Where to find key files and configs

SCOPE LEVELS

- → **Global:** ~/.claude/CLAUDE.md — applies everywhere
- → **Project:** ./CLAUDE.md — applies to this repo
- → **Subdirectory:** ./src/CLAUDE.md — applies in that folder

QUICK START

Run `/init` in any project to generate a starter CLAUDE.md automatically. Then edit it to match your real project structure and rules.

✓ Bottom line: **The single most important setup step. Without it, Claude starts from scratch every session. With it, it already knows your project.**

IDE Integrations

PRODUCTIVITY

VS Code · JetBrains · Cursor · Zed — Claude inside your editor

WHAT THEY DO

Claude Code IDE extensions put the full Claude Code experience inside your editor. Claude sees your open files, error messages, and terminal output — all in context. No copy-pasting code between a terminal and your editor.

`VS Code Extension` `JetBrains Plugin` `Cursor (built-in)` `Zed (built-in)`

WHY IT MATTERS

- → Claude sees your exact file, line number, and error without you describing it
- → Inline suggestions appear as you type
- → Terminal + editor share the same Claude session
- → Run /terminal-setup once to enable Shift+Enter shortcuts

✓ Bottom line: **Install the extension for whichever IDE you already use. VS Code: search "Claude Code" in Extensions. JetBrains: search in Plugin Marketplace.**

The Claude Code Ecosystem.

GitHub

Version control for everything — your code, your agents, your CLAUDE.md

CRITICAL

WHY CLAUDE NEEDS GITHUB

GitHub is the version history for your entire Claude setup — not just your code. Your CLAUDE.md, skills, agent configs, SOPs, and decision logs all live in git. If something breaks or Claude goes in the wrong direction, you roll back. Without git, you have no safety net.

WHAT CLAUDE CAN DO WITH GITHUB

- → Review PRs with /review or /code-review
- → Auto-fix CI failures with /autofix-pr
- → Create branches, commits, and PRs via bash tools
- → Read issues and PR comments for context
- → Run on push/PR via GitHub Actions

GITHUB APP

Run `/install-github-app` to connect Claude to a repo. Enables automatic PR reviews when code is pushed and /autofix-pr to watch and fix CI failures without you touching anything.

GITHUB ACTIONS

Claude can run in CI/CD pipelines via GitHub Actions — review every PR, run security scans, and post inline comments automatically. Setup is handled by /install-github-app.

✓ Bottom line: **Not optional if you're using Claude Code seriously. Run /install-github-app once in any project where you want automated PR review.**

MCPs — Model Context Protocol

The bridge between Claude and every external tool or data source you use

EXTENDS CLAUDE

WHAT MCPS ARE

MCP is Anthropic's open standard for connecting Claude to external tools. An MCP server sits between Claude and a service (Notion, Gmail, Stripe, etc.) and exposes that service's actions as tools Claude can call — read a Notion page, send a Gmail, charge a card — without you writing API code.

EXAMPLES IN THIS ENGINE

`Notion (content calendar)` `Gmail (briefing emails)` `iMessage (notifications)` `Google Drive` `Google Calendar` `Stripe (payments)` `Slack` `GitHub`

HOW TO SET ONE UP

Run `/mcp` inside a session to see connected servers and add new ones. MCP servers are defined in `.mcp.json` or via the settings UI. Most popular services have pre-built MCP servers available at mcp.so.

MCP AS COMMANDS

MCP servers can also expose prompts that appear as slash commands in the format `/mcp__server__prompt`. These show up automatically once the server is connected — no extra setup needed.

✓ Bottom line: **MCPs are how Claude Code becomes an agent that takes real actions in real tools — not just a coding assistant. Connect Notion and Gmail first; they unlock the most workflow.**

Skills

Custom slash commands that turn Claude into a specialist for any repeatable task

CUSTOM COMMANDS

WHAT SKILLS ARE

A skill is a SKILL.md file that contains instructions, checklists, or multi-step procedures. Drop it in `.claude/skills/my-skill/SKILL.md` and it becomes `/my-skill`. The instructions only load into context when the skill is actually used — zero cost when idle.

WHEN TO CREATE A SKILL

- → You keep pasting the same instructions into chat
- → A section of CLAUDE.md has grown into a procedure
- → You want Claude to invoke the behavior automatically when relevant
- → You want to share a workflow with a teammate or community

IN THIS ENGINE (EXAMPLES)

`/full-engine` `/research` `/create-content` `/newsletter` `/edit` `/monday` `/ceo`

SKILL VS. CLAUDE.MD

CLAUDE.md = permanent facts Claude always knows. Skill = procedure that loads only when triggered. Long procedures belong in skills, not CLAUDE.md — they don't cost context until you need them.

✓ Bottom line: **Skills are how you systematize your workflow. Every repeatable process becomes a command. The team runs on commands, not memory.**

The Claude Code Ecosystem.

⚡ Hooks

AUTOMATION

Shell commands that run automatically before or after Claude takes any action

WHAT HOOKS ARE

Hooks are shell commands you configure in settings. They fire at specific events — before Claude edits a file, after a bash command runs, before a response is sent. They let you enforce rules, log activity, or trigger side effects automatically — without Claude having to remember to do them.

HOOK TRIGGER POINTS

- → **PreToolUse:** Before any tool call — can block or modify
- → **PostToolUse:** After a tool completes — log, notify, trigger next step
- → **Notification:** When Claude sends you a notification
- → **Stop:** When Claude finishes responding
- → **SubagentStop:** When a background subagent finishes

REAL USE CASES

- → Auto-format code after every file edit
- → Log every tool call to a JSONL file for cost tracking
- → Send an iMessage notification when a background agent finishes
- → Block Claude from editing certain protected files
- → Auto-run tests after file changes

HOW TO SET UP

View current hooks with `/hooks`. Configure in your project's `.claude/settings.json` or global `~/.claude/settings.json` under the "hooks" key.

✓ Bottom line: **Hooks are how you make Claude Code behave like a system, not an assistant. Anything Claude should always do or never do belongs in a hook.**

⟲ Sub-agents & Agents

PARALLEL WORK

Separate Claude instances that run tasks in parallel — your AI team

WHAT SUB-AGENTS ARE

Sub-agents are separate Claude Code sessions that the main session can spawn to run work in parallel. Each has its own context window, can use all the same tools, and reports back when done. Instead of one Claude doing everything sequentially, you get a team of Claudes working simultaneously.

TYPES OF AGENTS

- → **Foreground subagent:** Blocks main session until done
- → **Background subagent (/fork):** Runs while you keep working
- → **Background session (/background):** Full detached session
- → **Named agents (.claude/agents/):** Persistent specialist roles
- → **Batch workflow (/batch):** Auto-spawns N agents in parallel

NAMED AGENTS (LIKE THIS ENGINE)

Store agent definitions in `.claude/agents/agent-name.md`. They appear as agent types the main session can dispatch. This is how the content engine works — CEO agent, Brand Guardian, Content Writer, etc. are all named agents with their own instructions.

KEY LIMITATION

Sub-agents cannot dispatch other sub-agents and cannot reach MCP servers directly. The top-level session handles all MCP tool calls. Design agent pipelines with the main session as orchestrator.

✓ Bottom line: **Sub-agents are how one operator does the work of a team. Research, write, review, and publish all happen in parallel — not sequentially waiting for you.**

◉ Memory System

PERSISTENT CONTEXT

What Claude remembers about you, your preferences, and your projects — across all sessions

THREE MEMORY LAYERS

- → **CLAUDE.md files:** Project instructions (you write these)
- → **Auto-memory:** Claude builds this automatically — user preferences, feedback, project state, references
- → **Session history:** /resume lets you return to any past conversation

AUTO-MEMORY TYPES

- → **User:** Who you are, your role, your expertise level
- → **Feedback:** How you want Claude to behave — dos and don'ts
- → **Project:** Ongoing work, decisions, deadlines
- → **Reference:** Where to find things (Notion DBs, Slack channels, etc.)

✓ Bottom line: **Manage with /memory. Auto-memory means Claude gets better at working with you over time — it learns your preferences without you repeating yourself every session.**

FIRST-SESSION SETUP — DO THIS ONCE IN EVERY NEW PROJECT

1. `/init`	2. `/mcp`	3. `/permissions`	4. `/memory`
Generate CLAUDE.md	Connect your tools	Set approval rules	Refine what Claude knows