Claude Code Cheat Sheet

Everything you need in one place — Commands, Shortcuts, Features & Tips

2026 EDITION

Keyboard Shortcuts

ESSENTIAL

`Enter`	Send message / submit
`Esc`	Interrupt / stop generation
`Esc Esc`	Open rewind menu (go back in conversation or code)
`Ctrl+C`	Cancel current operation (hard stop)
`Ctrl+D`	Exit Claude Code
`Shift+Tab`	Cycle modes: Normal → Auto-Accept → Plan

NAVIGATION

`Ctrl+R`	Search command history
`Ctrl+T`	Toggle task list
`Ctrl+O`	Toggle verbose transcript
`Ctrl+G`	Open external editor (write long prompts)
`Ctrl+V`	Paste image (screenshots, diagrams)
`Ctrl+S`	Stash current prompt (save for later)
`Cmd+P / Meta+P`	Open model picker (switch models quick)
`Cmd+T / Meta+T`	Toggle extended thinking

EDITING (BASH-STYLE)

`Ctrl+A / Ctrl+E`	Start / end of line
`Opt+F / Opt+B`	Word forward / back
`Ctrl+W`	Delete previous word
`\ + Enter`	New line (without sending)

BACKGROUND TASKS

`Ctrl+B`	Send running task to background

💡 **Tip:** Run `/terminal-setup` to enable Shift+Enter for multi-line input in iTerm2 & VS Code. Run `/keybindings` to customize all shortcuts.

Slash Commands

SESSION CONTROL

`/clear`	Reset conversation history (fresh start)
`/compact [hint]`	Compress context to save tokens. Optional hint for what to keep.
`/rewind`	Go back in conversation AND/OR code changes
`/export [file]`	Export conversation to file or clipboard
`/cost`	Show session cost & token usage
`/usage`	Show plan usage & rate limits
`/context`	Token consumption visualization

CONFIGURATION

`/config`	Open settings panel
`/model`	Switch between Sonnet / Opus / Haiku
`/permissions`	View & update tool permissions
`/keybindings`	Open keyboard shortcuts config file
`/vim`	Toggle vim mode for input
`/terminal-setup`	Setup Shift+Enter for multi-line input

DEVELOPMENT

`/init`	Create CLAUDE.md for your project — **do this first!**
`/memory`	View & edit CLAUDE.md project memory
`/review`	Code review analysis
`/doctor`	Environment diagnostics & health check
`/agents`	Manage sub-agents
`/mcp`	Manage MCP servers

ADVANCED

`/insights`	Generate HTML usage report NEW
`/pr_comments`	View GitHub PR feedback
`/install-github-app`	Setup automated PR reviews
`/tasks`	Persistent task list management
`/teleport`	Transfer session between web ↔ local

CLI Launch Flags

STARTING SESSIONS

`claude`	Start interactive session
`claude "query"`	Start with an initial prompt
`claude -p "query"`	Print mode — answer & exit (for scripting)
`claude -c`	Continue last conversation
`claude -r "name"`	Resume specific session by name or ID
`claude -w name`	Start in isolated git worktree

MODEL & BEHAVIOR

`--model sonnet`	Use Sonnet (fast, cheap)
`--model opus`	Use Opus (smartest)
`--agent my-agent`	Use a specific sub-agent
`--permission-mode plan`	Start in plan mode
`--max-turns N`	Limit conversation turns
`--max-budget-usd N`	Set max spend limit

CONTEXT & DIRECTORIES

`--add-dir ../path`	Add extra directories to context
`--chrome`	Enable browser integration
`--verbose`	Show detailed logging

PERMISSIONS

`--allowedTools`	Whitelist specific tools
`--disallowedTools`	Block specific tools
`--tools "Bash,Edit"`	Restrict to only these tools

OUTPUT FORMATS (FOR -P MODE)

`--output-format text`	Plain text (default)
`--output-format json`	Structured JSON
`--output-format stream-json`	Real-time streaming JSON

💡 **Tip:** Pipe data in! `git diff | claude -p "review this"` or `cat error.log | claude -p "explain"`

The Big 5 — Claude Code Extension System

1. CLAUDE.MD — PROJECT MEMORY

`What`	A markdown file Claude reads every session. Your project's "brain dump" — coding style, architecture, common commands, conventions.
`Where`	`.claude/CLAUDE.md` (project) or `~/.claude/CLAUDE.md` (global)
`Create`	Run `/init` in your project — Claude generates it for you

2. CUSTOM SLASH COMMANDS

`What`	Your own /commands. Markdown files with prompts that YOU invoke. Like prompt templates.
`Where`	`.claude/commands/` (project) or `~/.claude/commands/` (global)
`Use`	Filename = command name. `review.md` → type `/project:review`

3. SKILLS — AUTO-INVOKED KNOWLEDGE

`What`	Like commands, but Claude decides when to use them automatically. You DON'T invoke them — Claude detects when they're relevant.
`Where`	`.claude/skills/` with a `SKILL.md` inside each skill folder
`Use`	Just work on your project — Claude picks up relevant skills from context

4. SUB-AGENTS — SPECIALIZED HELPERS

`What`	Separate Claude instances with their own context & role. Like team members: reviewer, debugger, architect, etc.
`Where`	`.claude/agents/` (markdown files with YAML metadata)
`Invoke`	`/agents` to manage, or just say "Use the reviewer agent"
`CLI`	`--agent my-agent` or `--agents '{json}'`

5. MCP SERVERS — EXTERNAL TOOL CONNECTIONS

`What`	Connect Claude to external tools: GitHub, Notion, databases, APIs, browsers, etc.
`Setup`	`claude mcp add <name> <command>`
`List`	`claude mcp list`
`Config`	`--mcp-config ./mcp.json` at launch

+ PLUGINS — COMMUNITY EXTENSIONS

`What`	Bundles of commands, skills, hooks & more from the community
`Browse`	`/plugin` to browse, install, enable, disable
`Dir`	`--plugin-dir ./my-plugins` for local plugins

HOW THEY DIFFER: **Custom Commands → YOU invoke them** VS **Skills → CLAUDE invokes them** VS **Sub-Agents → Separate AI instances** VS **MCP → External tool connections**

Permission Modes

`Normal`	Claude asks permission for every tool use (read, write, bash, etc.)
`Auto-Accept`	Claude runs tools WITHOUT asking. Faster but less control. Good for trusted tasks.
`Plan Mode`	Claude ONLY reads & plans. Won't write or run anything. Review first, then switch to Normal to execute.

Shift+Tab → Normal → Auto-Accept → Plan → Normal...

💡 **Best workflow:** Start in Plan Mode to explore & understand the problem. Review Claude's plan. Switch to Normal/Auto-Accept to implement. This is what you already do — it's the right approach!

Hooks — Event Automation

`PreToolUse`	Runs BEFORE Claude uses a tool — validate, block, or modify
`PostToolUse`	Runs AFTER a tool — check results, auto-format, lint
`UserPromptSubmit`	Before your message is processed
`Stop`	When Claude finishes its response
`SessionStart`	When a session begins
`SessionEnd`	When a session ends
`PreCompact`	Before context compression
`Notification`	When Claude sends a notification

💡 **Example:** Auto-run `prettier` after every file edit, or block writes to `.env` files. Configure in your settings JSON.

Input Superpowers

`@ mention`	Type @ to reference files & folders. Claude reads them into context.
`! prefix`	Type ! to run shell commands inline. E.g., `! git status`
`Paste images`	Ctrl+V to paste screenshots, diagrams, error images directly
`Pipe input`	`cat file.py | claude -p "explain"` — feed data directly
`Multi-dir`	`claude --add-dir ../api ../web` — work across multiple projects
`Worktrees`	`claude -w feature` — isolated git branch + Claude session

⚠️ **Pro tip:** Use @ references instead of copy-pasting file contents. It's smarter with context and uses fewer tokens.

Configuration

SETTINGS PRIORITY (HIGHEST → LOWEST)

`Enterprise`	`/etc/claude-code/managed-settings.json`
`Project Local`	`.claude/settings.local.json` (your personal project settings)
`Project Shared`	`.claude/settings.json` (committed to git, shared with team)
`User Global`	`~/.claude/settings.json` (your defaults)

CONFIG CLI

`config list`	`claude config list` — show all settings
`config get`	`claude config get key` — check a value
`config set`	`claude config set key value` — change a value
`config add`	`claude config add key value` — add to array

💡 **Permissions example:** Allow git commands without asking: add `"Bash(git:*)"` to your allowedTools in settings.

File Structure Map

PROJECT LEVEL (.CLAUDE/)

`CLAUDE.md`	Project memory — conventions, architecture, commands
`settings.json`	Shared project settings (committed to git)
`settings.local.json`	Your personal settings (gitignored)
`commands/`	Project slash commands (*.md files)
`skills/`	Project skills (folders with SKILL.md)
`agents/`	Project sub-agents (*.md files)

GLOBAL LEVEL (~/.CLAUDE/)

`CLAUDE.md`	Global memory (applies to ALL projects)
`settings.json`	Global settings
`commands/`	Personal global commands
`skills/`	Personal global skills
`keybindings.json`	Custom keyboard shortcuts

Rewind & Checkpoints

`Esc Esc`	Open rewind menu anywhere
`/rewind`	Same but typed as command

REWIND OPTIONS

`Conversation`	Go back in chat only. Code stays as-is.
`Code`	Restore files only. Conversation stays.
`Full Rewind`	Restore both conversation AND code to a point.

⚠️ **Note:** Bash side-effects (database changes, API calls, deleted files via `rm`) can't be rewound. Checkpoints only track file edits by Claude. Use Git for permanent safety.

Pro Workflow — How to Get the Best Out of Claude Code

STARTING A NEW PROJECT

1. `cd project && claude` → 2. `/init` → 3. Edit CLAUDE.md → 4. Code!

THE PLAN → EXECUTE PATTERN (YOUR CURRENT APPROACH ✅)

Shift+Tab → Plan Mode → Describe what you want → Review Claude's plan → Shift+Tab → Normal/Auto → Execute

SAVING MONEY

`Use /compact`	When context gets big, compress it. Saves tokens dramatically.
`Use /clear`	Between unrelated tasks. Don't carry irrelevant context.
`Use Sonnet`	For routine tasks. Save Opus for complex architecture decisions.
`Use @ refs`	Instead of pasting code — smarter context management.

DEBUGGING LIKE A PRO

`Paste errors`	Copy-paste the full error message. Claude parses stack traces brilliantly.
`Paste screenshots`	Ctrl+V a screenshot of the bug. Claude sees it.
`Pipe logs`	`cat error.log | claude -p "what's wrong?"`
`/doctor`	If something feels broken, run this first.

PARALLEL DEVELOPMENT PRO

`Worktrees`	`claude -w feature-auth` — isolated branch + session
`Multiple dirs`	`--add-dir ../api ../web` — work across repos
`Background`	Ctrl+B sends a task to background so you can start another
`Agent Teams`	Multiple Claude instances collaborating (experimental) NEW

Create Custom Commands

`1. Create file`	`.claude/commands/review.md`
`2. Write prompt`	The markdown content IS the prompt Claude will use
`3. Use it`	Type `/project:review` in Claude Code

OPTIONAL YAML FRONTMATTER

`argument-hint`	Placeholder text for argument input
`description`	Shows in /help listing
`allowed-tools`	Restrict what tools the command can use
`model`	Force a specific model for this command

VARIABLE: $ARGUMENTS

`$ARGUMENTS`	Use $ARGUMENTS in your markdown — it gets replaced with whatever you type after the command

💡 **Example:** `/project:review src/auth.ts` → $ARGUMENTS = "src/auth.ts"

Quick Reference — Most Used Combos

DAILY ESSENTIALS

`Start project`	`cd project && claude`
`Continue where I left off`	`claude -c`
`Quick question, no session`	`claude -p "how do I..."`
`Review my changes`	`git diff | claude -p "review"`
`Explain error`	`cat error.log | claude -p "explain"`
`Check cost`	Type `/cost` anytime
`Undo mistake`	`Esc Esc` → rewind

POWER MOVES

`Parallel sessions`	`claude -w feature-a` + `claude -w feature-b`
`Custom reviewer agent`	Create `.claude/agents/reviewer.md`
`Auto-format on edit`	PostToolUse hook → run prettier
`Web session`	`claude --remote "fix the bug"`
`Transfer to local`	`claude --teleport`
`Budget limit`	`claude -p --max-budget-usd 2 "query"`
`Scripted automation`	`claude -p --output-format json "query" | jq`