

Claude Code Cheatsheet

By Dejan Panovski • Updated on Oct 20, 2026

Quick reference for Claude Code in the terminal: session flags, print mode for scripts, slash commands, keyboard shortcuts, permission rules, MCP servers, hooks, and config file locations.

Claude Code works as an interactive agent in your terminal, but much of what it can do sits behind flags, slash commands, and keyboard shortcuts that are easy to forget between sessions. This cheatsheet collects the ones you reach for daily, from resuming a conversation and scripting with print mode to permission rules, MCP servers, and the files where settings and project instructions live.

Start and Resume Sessions

Launch options for the interactive session. Flags and commands on this page match Claude Code 2.1.302.

<code>claude</code>	Start an interactive session in the current directory
<code>claude "explain this project"</code>	Start a session with a first prompt
<code>claude -c</code>	Continue the most recent conversation in this directory
<code>claude -r</code>	Open the session picker
<code>claude -r SESSION_ID</code>	Resume a specific conversation
<code>claude -c --fork-session</code>	Continue under a new session ID, keeping the original intact
<code>claude -n "auth refactor"</code>	Name the session for the prompt bar and the <code>/resume</code> picker
<code>claude --from-pr 123</code>	Resume the session linked to a pull request
<code>claude -w feature-x</code>	Start in a new git worktree named <code>feature-x</code>
<code>claude --add-dir ../shared</code>	Allow file access to another directory
<code>claude --model opus</code>	Pick the model by alias, such as <code>opus</code> or <code>sonnet</code> , or by full name
<code>claude --effort high</code>	Set the effort level: <code>low</code> , <code>medium</code> , <code>high</code> , <code>xhigh</code> , or <code>max</code>

Print Mode for Scripts

`-p` runs a single prompt, prints the answer, and exits, which makes Claude Code usable in pipes and CI jobs.

<code>claude -p "summarize README.md"</code>	Print the response and exit
<code>cat error.log claude -p "explain this error"</code>	Send piped input along with the prompt
<code>claude -c -p "now add tests"</code>	Continue the last conversation without opening the interface
<code>claude -p "..." --output-format json</code>	Return the result as a single JSON object
<code>claude -p "..." --output-format stream-json --verbose</code>	Stream events as they arrive
<code>claude -p "..." --json-schema '{...}'</code>	Validate structured output against a JSON Schema
<code>claude -p "..." --max-budget-usd 2</code>	Stop when estimated API spend reaches \$2; final spend may exceed this
<code>claude -p "..." --allowedTools "Bash(npm test *)" Read</code>	Pre-approve the tools the job needs
<code>claude -p "..." --append-system-prompt "..."</code>	Add instructions to the default system prompt
<code>claude -p "..." --no-session-persistence</code>	Do not save the session to disk

Print mode skips the workspace trust dialog, so run it only in directories you trust.

Session and Context Commands

Slash commands that manage the conversation itself. Type `/` to see everything available, including skills and plugin commands.

<code>/clear</code>	Start a new conversation with empty context
<code>/compact [instructions]</code>	Summarize the conversation to free context, with optional focus
<code>/context</code>	Show context usage as a colored grid
<code>/usage</code>	Show session cost, plan limits, and activity (<code>/cost</code> is an alias)
<code>/model [model]</code>	Switch the model and save it as the default
<code>/effort [level]</code>	Change the effort level for this session
<code>/fast [on off]</code>	Toggle fast mode
<code>/resume [session]</code>	Resume a conversation by ID or name, or open the picker
<code>/rename [name]</code>	Rename the current session
<code>/branch [name]</code>	Branch the conversation to try a different direction
<code>/rewind</code>	Roll the conversation, the code, or both back to an earlier point
<code>/btw [question]</code>	Ask a side question that does not enter the conversation
<code>/copy [N]</code>	Copy the last response, or the Nth latest, to the clipboard
<code>/export [filename]</code>	Export the conversation as plain text
<code>/exit</code>	Leave Claude Code (<code>/quit</code> also works)

Project and Setup Commands

Commands that change how Claude Code behaves in this project and on this machine.

<code>/init</code>	Generate a starting <code>CLAUDE.md</code> from the codebase
<code>/memory</code>	Edit <code>CLAUDE.md</code> files and turn auto memory on or off
<code>/config</code>	Open the settings interface
<code>/permissions</code>	Manage allow, ask, and deny rules
<code>/add-dir <path></code>	Add a working directory during the session
<code>/mcp</code>	Manage MCP server connections and OAuth sign-in
<code>/hooks</code>	View hook configurations
<code>/skills</code>	List available skills
<code>/plugin</code>	Install, enable, and disable plugins
<code>/status</code>	Show version, model, account, and connectivity
<code>/doctor</code>	Diagnose installation, settings, and <code>CLAUDE.md</code> problems
<code>/terminal-setup</code>	Install a Shift+Enter newline binding for supported terminals
<code>/keybindings</code>	Open the keyboard shortcuts file
<code>/statusline</code>	Configure the status line

Review and Background Work

Commands for checking changes and running longer jobs while you keep typing.

<code>/plan [description]</code>	Enter plan mode, optionally starting on a task
<code>/diff</code>	Review working tree changes, including the edits Claude made
<code>/code-review</code>	Review the current diff, a PR number, a branch, or a path (<code>/review</code> is an alias)
<code>/security-review</code>	Check the branch changes for security vulnerabilities
<code>/simplify</code>	Review changed code for cleanup and apply the fixes
<code>/tasks</code>	View and manage background shells and subagents
<code>/loop [interval] [prompt]</code>	Run a prompt repeatedly while the session stays open
<code>/install-github-app</code>	Set up the Claude GitHub App and Actions workflow for a repository
<code>claude --bg "run the tests and fix failures"</code>	Start a session in the background and print its ID
<code>claude agents</code>	List background sessions
<code>claude attach ID</code>	Open a background session in this terminal
<code>claude logs ID</code>	Print a background session's recent output
<code>claude stop ID</code>	Stop a background session and keep its conversation

Keyboard Shortcuts

The shortcuts that control the session. Press `?` on an empty prompt to see the full list.

<code>Esc</code>	Interrupt Claude and keep the work done so far
<code>Esc Esc</code>	Clear the draft, or open the rewind menu on an empty prompt
<code>Ctrl+C</code>	Interrupt, or clear the input; a second press exits
<code>Ctrl+D</code>	Exit Claude Code (press twice)
<code>Shift+Tab</code>	Cycle permission modes
<code>Ctrl+R</code>	Search prompt history
<code>Ctrl+O</code>	Toggle the transcript viewer with full tool output
<code>Ctrl+B</code>	Send a running Bash command or agent to the background (press twice in tmux)
<code>Ctrl+T</code>	Show or hide Claude’s task checklist
<code>Ctrl+G</code>	Edit the prompt in your default text editor
<code>Ctrl+S</code>	Stash the prompt, then press again to restore it
<code>Ctrl+V</code>	Paste an image from the clipboard
<code>Ctrl+L</code>	Redraw the screen
<code>Alt+P</code>	Switch the model without clearing the prompt (<code>Opt ion+P</code> on macOS)
<code>Alt+T</code>	Toggle extended thinking (<code>Opt ion+T</code> on macOS)
<code>Ctrl+Z</code>	Suspend Claude Code; run <code>fg</code> to resume

Input and Multiline Prompts

Prefixes change what the prompt does, and newlines need a key that does not submit.

/ at the start	Run a command or skill
! at the start	Shell mode: run the command and add its output to the session
@	Autocomplete a file path to mention
\ then Enter	Insert a newline in any terminal
Ctrl+J	Insert a newline without terminal configuration
Shift+Enter	Insert a newline in iTerm2, WezTerm, Ghostty, Kitty, Warp, Apple Terminal, and Windows Terminal
Option+Enter	Insert a newline on macOS with Option set as Meta
Ctrl+A / Ctrl+E	Move to the start or end of the line
Ctrl+W	Delete back to the previous whitespace
Ctrl+U	Delete from the cursor to the line start
Ctrl+K	Delete to the end of the line
Ctrl+Y	Paste the text you last deleted

On macOS, the `Alt` and `Option` shortcuts work only after you configure Option as Meta in your terminal.

Permission Modes

How much Claude Code may do before it asks. Cycle with `Shift+Tab` or set one at launch.

default	Ask on first use of each tool (shown as Manual)
acceptEdits	Accept file edits and common filesystem commands in the working directory
plan	Read and explore without editing source files
auto	Skip routine prompts; a classifier reviews shell commands and network requests first
dontAsk	Deny anything that would prompt, unless a rule allows it
bypassPermissions	Skip permission prompts
claude --permission-mode plan	Start the session in a given mode
claude --dangerously-skip-permissions	Start with every permission check bypassed

Use `bypassPermissions` and `--dangerously-skip-permissions` only inside an isolated container or VM, since Claude can then change any file and run any command without asking.

Permission Rules

Rules use the format `Tool` or `Tool(specifier)` . Add them with `/permissions` , in a settings file, or with launch flags.

<code>Bash</code>	Match every Bash command
<code>Bash(npm run build)</code>	Match one exact command
<code>Bash(git *)</code>	Match every command that starts with <code>git</code>
<code>Read(./env)</code>	Match reads of <code>.env</code> in the current directory
<code>WebFetch(domain:example.com)</code>	Match fetches from one domain
<code>claude --allowedTools "Bash(git *)" Edit</code>	Allow tools for this session
<code>claude --disallowedTools "Bash(rm *)"</code>	Deny tools for this session

Claude Code checks deny rules first, then ask, then allow, and the first match wins, so an allow rule cannot carve an exception out of a deny rule. When you approve a Bash command permanently from its prompt, Claude Code saves the rule to `.claude/settings.local.json` .

MCP Servers

Connect external tools and data sources through the Model Context Protocol.

<code>claude mcp add --transport http NAME URL</code>	Add a remote HTTP server
<code>claude mcp add --transport http NAME URL --header "Authorization: Bearer ..."</code>	Add a remote server with a request header
<code>claude mcp add NAME -e API_KEY=... -- npx my-mcp-server</code>	Add a local stdio server with an environment variable
<code>claude mcp add --scope user NAME -- COMMAND</code>	Make the server available in every project
<code>claude mcp add --scope project NAME -- COMMAND</code>	Share the server with the team through <code>.mcp.json</code>
<code>claude mcp add-json NAME '{...}'</code>	Add a server from a JSON definition
<code>claude mcp list</code>	List configured servers and their health
<code>claude mcp get NAME</code>	Show details for one server
<code>claude mcp remove NAME</code>	Remove a server
<code>claude mcp login NAME</code>	Authenticate with an HTTP or SSE server
<code>claude mcp reset-project-choices</code>	Reset approvals for <code>.mcp.json</code> servers in this project
<code>claude --mcp-config servers.json</code>	Load servers from a file for one session

The default scope is `local` , which keeps the server private to you in the current project. Do not commit API keys inside `.mcp.json` ; pass them through environment variables instead.

Settings and Instruction Files

Where Claude Code reads configuration and project instructions.

~/.claude/settings.json	User settings for every project
.claude/settings.json	Project settings shared through version control
.claude/settings.local.json	Personal settings for one project, kept out of git
claude --settings FILE	Extra settings for one session
~/.claude/CLAUDE.md	Personal instructions for every project
./CLAUDE.md or ~/.claude/CLAUDE.md	Project instructions for the whole team
./CLAUDE.local.md	Personal project instructions; add it to .gitignore
.claude/rules/*.md	Instruction files that can be scoped to paths
@path/to/file	Import another file from inside CLAUDE.md
.claude/skills/NAME/SKILL.md	Project skill, run as /NAME
~/.claude/skills/NAME/SKILL.md	Personal skill for every project
.claude/agents/	Project subagent definitions
.mcp.json	Project-scoped MCP servers

When a key is set in several places, managed settings win, then `--settings` , then `.claude/settings.local.json` , then `.claude/settings.json` , then the user file.

Hooks

Hooks run your own shell commands at fixed points in a session. Define them under the `hooks` key in a settings file.

SessionStart	A session begins or resumes
UserPromptSubmit	A prompt is submitted, before Claude processes it
PreToolUse	Before a tool call runs; the hook can block it
PermissionRequest	A tool call needs a permission decision
PostToolUse	After a tool call succeeds
PostToolUseFailure	After a tool call fails
Notification	Claude Code sends a notification
SubagentStop	A subagent finishes
Stop	Claude finishes responding
PreCompact	Before the context is compacted
SessionEnd	The session ends

Hooks run with your user account’s permissions. Read any hook that arrives with a project or plugin before you trust it.

Install, Update, and Sign In

Maintenance commands that run outside a session.

<code>claude --version</code>	Print the installed version
<code>claude update</code>	Check for an update and install it
<code>claude install stable</code>	Install the native build from the <code>stable</code> or <code>latest</code> channel, or a specific version
<code>claude doctor</code>	Check the health of the installation
<code>claude auth login</code>	Sign in to your Anthropic account
<code>claude auth status</code>	Show the current authentication status
<code>claude auth logout</code>	Sign out
<code>claude setup-token</code>	Create a long-lived token for automation (requires a Claude subscription)
<code>claude plugin install NAME@MARKETPLACE</code>	Install a plugin from a marketplace
<code>claude plugin list</code>	List installed plugins
<code>claude --safe-mode</code>	Start with <code>CLAUDE.md</code> , skills, plugins, hooks, and MCP servers disabled
<code>claude --debug="api, hooks"</code>	Start with debug logging for selected categories

Treat the token from `claude setup-token` as a password and store it in your CI secret manager, not in the repository.

Troubleshooting

Common problems and the first thing to check.

claude: command not found after installing	Add ~/.local/bin to your PATH and reload the shell
Shift+Enter submits the prompt	Run /terminal-setup, or use \ then Enter or Ctrl+J
Alt or Option shortcuts do nothing on macOS	Set Option as Meta in the terminal preferences
Context fills up in a long session	Run /context, then /compact or /clear
A project MCP server shows Pending approval	Approve it when prompted, or reset choices with claude mcp reset-project-choices
Claude keeps asking for the same command	Add an allow rule with /permissions
Startup breaks after a config change	Start with claude --safe-mode, then run /doctor

Related Guides

Setup and the tools Claude Code works alongside.

How to Install Claude Code on Ubuntu and Linux	Native installer, package repositories, npm, and sign-in
git worktree: Work on Multiple Branches at Once	The worktrees that claude -w creates
GitHub CLI: Manage Repositories, Issues, and Pull Requests	Turn Claude's changes into pull requests
tmux Command in Linux: Sessions, Windows, and Panes	Keep long sessions running on a remote server
How to Install and Use Ollama on Ubuntu: Run LLMs Locally	Run models locally instead
Git cheatsheet	Quick reference for the commits Claude Code prepares