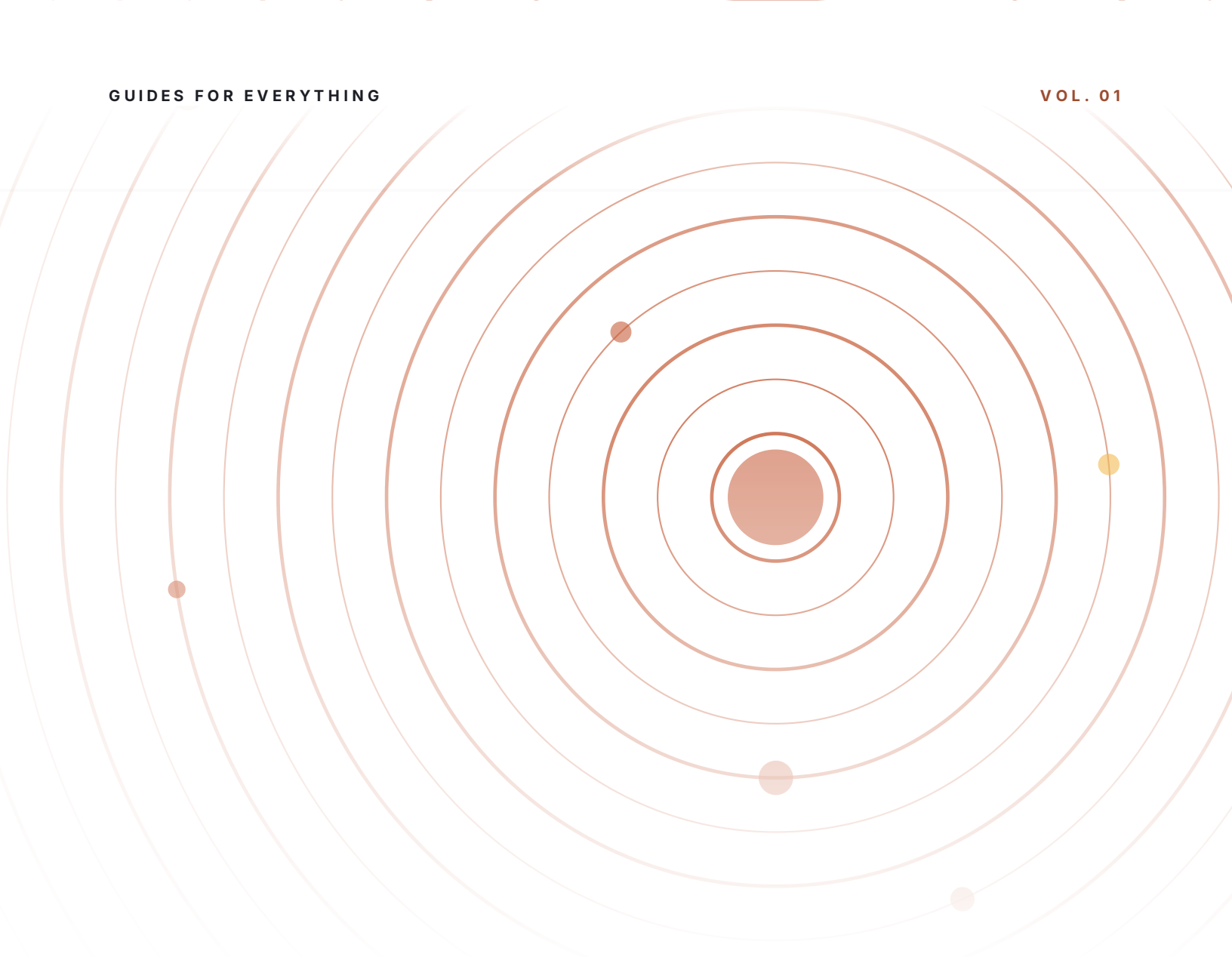




A BEGINNER'S MANUAL

CLAUDE EDITION

Claude Code

Your first AI coding partner, from installation to every slash command, shortcut, and everyday project

Claude Code

Your first AI coding partner, from installation to every slash command, shortcut, and everyday project

CLAUDE EDITION Expanded edition written by Claude (Anthropic's AI) in Claude Code and published in GuidesForEverything PR #4.

Claude Code is an AI assistant from Anthropic that can read your project, write and edit files, run commands, and explain everything it does in plain English. You talk to it the way you'd talk to a helpful colleague, and it does the work.

This manual assumes you have **never used a terminal before**. Each chapter builds on the one before it. You'll install Claude Code on your computer, run your first session, learn every built-in **slash command** (such as `/clear`, `/compact`, and `/rewind`) with examples, and make your own commands. Later chapters cover memory, permissions, plan mode, subagents, hooks, MCP servers, plugins, scripting, GitHub Actions, costs, security, and more than a dozen complete projects you can follow step by step.

HOW TO READ THIS MANUAL

Tip — a shortcut or good habit.

Warning — easy to get wrong.

`$ command` — text you type, exactly as shown.

Note — useful background.

Danger — can lose data, money, or access.

`Settings > Privacy` — a menu path to click.

New words appear in **bold** the first time and are collected in the Glossary at the back.

Contents

01	Welcome to Claude Code	11
1.1	What is an “agent”?	11
1.2	How the loop works under the hood	12
1.3	What can you do with it?	12
1.4	Where Claude Code runs	13
1.5	What you need	13
1.6	How to use this manual	14
02	Terminal Basics in Ten Minutes	15
2.1	Opening a terminal	15
2.2	Folders and paths	16
2.3	The five commands you need	16
2.4	Copying and pasting in a terminal	17
2.5	Stopping and exiting	17
2.6	Environment variables in one minute	18
2.7	Git in two minutes	18
03	Install and Sign In	19
3.1	Before you install	19
3.2	Install with the native installer	20
3.3	Notes for each operating system	21
3.4	Choose a plan	22
3.5	Log in for the first time	23
3.6	Health check and updates	24
3.7	Fixing “command not found”	24
3.8	Common install errors	25
3.9	Uninstalling	26
04	Your First Session	27
4.1	The Claude Code screen	27
4.2	Build a small website	28
4.3	Watch what Claude is doing	29
4.4	Review the changes	29

4.5	Save your work with Git	29
4.6	Ask questions about existing code	29
4.7	Run your own commands with !	30
4.8	Approving, interrupting, and undoing	30
4.9	Ending and resuming	31
4.10	Your first-session practice list	31
05	Talking to Claude Effectively	33
5.1	Five habits of great prompts	33
5.2	Plan mode: look before you leap	34
5.3	Share what you see	34
5.4	Ask a side question	34
5.5	Keep conversations focused	35
5.6	Pattern 1: Explore, plan, code, commit	35
5.7	Pattern 2: Give Claude a way to check its work	36
5.8	Pattern 3: Point at examples and sources	36
5.9	Pattern 4: Let Claude interview you	37
5.10	Pattern 5: Ask for deeper thinking when it matters	37
5.11	A library of reusable prompts	37
5.12	Common mistakes and their fixes	38
06	How Context Works: /clear or /compact?	39
6.1	The context window, explained simply	39
6.2	What fills the context window	40
6.3	See what's using space: /context	40
6.4	Automatic compaction	40
6.5	/clear: a fresh desk	41
6.6	/compact: a tidy desk	41
6.7	The decision table	42
6.8	Other ways to keep context lean	43
6.9	Prompt caching, briefly	43
07	Slash Commands: The Basics and Your Session	44
7.1	How slash commands work	44
7.2	Session and context commands	46
7.3	Commands that shape how Claude works	49
7.4	Background and parallel work	50
08	Slash Commands: Setup, Models, and Cost	52

8.1	Project setup and memory	52
8.2	Settings and permissions	53
8.3	Look and feel	55
8.4	Models and effort	57
8.5	Usage, limits, and plans	58
09	Slash Commands: Tools, Reviews, Account, and Help	60
9.1	Tools, MCP, agents, hooks, and plugins	60
9.2	Bundled helper skills	62
9.3	Reviewing and shipping	63
9.4	Cloud, devices, and sharing	66
9.5	Account	67
9.6	Help and troubleshooting	67
9.7	Just for fun	68
9.8	Removed commands	69
10	Custom Slash Commands and Skills	70
10.1	Commands and skills: one feature, two formats	70
10.2	Where skills live	71
10.3	Your first skill in three minutes	71
10.4	Frontmatter: the settings block	72
10.5	Passing arguments	73
10.6	Worked examples	73
10.7	Stacking skills and adding supporting files	76
10.8	Who can run a skill?	76
10.9	MCP prompts as slash commands	76
10.10	Bundled skills you already have	77
10.11	When a skill misbehaves	77
11	Keyboard Shortcuts and Launch Options	78
11.1	The shortcuts you'll use daily	78
11.2	Model and mode shortcuts	79
11.3	Editing your prompt	79
11.4	Quick prefixes	80
11.5	Prompt history and search	80
11.6	Shortcuts in the transcript viewer	80
11.7	Vim mode	81
11.8	Customize your shortcuts	81
11.9	Starting Claude Code from the terminal	81

12	Plan Mode in Depth	83
12.1	Entering and leaving plan mode	83
12.2	What happens while planning	84
12.3	Reviewing the plan	84
12.4	Plan mode as your default	85
12.5	When to skip planning	85
13	Checkpoints and Rewind	86
13.1	How checkpoints work	86
13.2	Opening the rewind menu	86
13.3	Getting back a cleared conversation	87
13.4	What checkpoints can't undo	87
13.5	Trying an alternative without losing anything	88
14	Permissions and Safety	89
14.1	Permission modes	89
14.2	Allow, ask, and deny rules	91
14.3	Protected paths and actions no mode approves	92
14.4	Where settings live	92
14.5	Safety nets	93
14.6	A sensible starter setup	93
15	Settings Files	94
15.1	The four files	94
15.2	Three ways to change a setting	95
15.3	Precedence: who wins	95
15.4	Settings you'll use most	96
16	Memory: Teaching Claude Your Project	97
16.1	CLAUDE.md files	97
16.2	Create one with /init	98
16.3	What to put in it	98
16.4	Imports	99
16.5	Rules files: <code>.claude/rules/</code>	100
16.6	AGENTS.md	100
16.7	Auto memory	100
16.8	Fixing memory problems	101
17	Sessions and Parallel Work	102

17.1	Managing sessions	102
17.2	Worktrees: one folder per task	103
17.3	Background sessions and agent view	103
17.4	Cloud sessions	103
17.5	Choosing the right way to run in parallel	104
18	Models, Effort, and Speed	105
18.1	The models	105
18.2	Effort levels	106
18.3	Fast mode	106
18.4	The advisor	106
18.5	Extended context	107
19	Subagents: Specialist Helpers	108
19.1	Out-of-the-box extending	108
19.2	Built-in subagents	109
19.3	Create your own	109
19.4	Calling a subagent on purpose	110
19.5	Three useful designs	110
20	Hooks: Automatic Actions	111
20.1	Your first hook: a desktop notification	111
20.2	How a hook is built	112
20.3	Example: format code after every edit	113
20.4	Example: block edits to protected files	113
20.5	Example: remind Claude after compaction	114
20.6	How hooks talk back: exit codes	114
20.7	Debugging hooks	115
21	MCP Servers: Connect Other Apps	116
21.1	Add a remote server	116
21.2	Add a local server	117
21.3	Choose a scope	117
21.4	Manage servers	118
21.5	Worked example: GitHub	118
21.6	Worked example: a database, read-only	118
21.7	Prompts and context cost	118
22	Plugins and Marketplaces	120

22.1	Install a plugin	120
22.2	Add another marketplace	121
22.3	Scopes	121
22.4	Keeping plugins under control	121
23	Scripts, Headless Mode, and the Agent SDK	122
23.1	Print mode basics	122
23.2	Structured output	123
23.3	Permissions in scripts	123
23.4	Faster, repeatable runs with --bare	124
23.5	Continuing a conversation	124
23.6	Adding Claude to a build script	125
23.7	The Agent SDK	125
24	GitHub Actions: Claude on Your Pull Requests	127
24.1	Quick setup	127
24.2	Manual setup	128
24.3	Using @claude	129
24.4	Automation mode: run on a schedule	129
24.5	Costs and safety	130
25	Beyond the Terminal: IDE, Desktop, Web, and Mobile	131
25.1	VS Code and Cursor	131
25.2	JetBrains IDEs	132
25.3	The desktop app	132
25.4	Claude Code on the web and phone	132
25.5	Remote Control	133
25.6	The mobile app	133
25.7	Claude in Chrome	133
26	Costs and Limits	134
26.1	How usage works	134
26.2	When you hit a limit	135
26.3	Make your usage go further	135
26.4	A simple daily routine	136
27	Project Recipes	137
27.1	Recipe 1: Fix a bug from an error message	137
27.2	Recipe 2: Build a small website from scratch	138

27.3	Recipe 3: Write tests for existing code	138
27.4	Recipe 4: Refactor safely	138
27.5	Recipe 5: Review a pull request	139
27.6	Recipe 6: Understand a project you've never seen	139
27.7	Recipe 7: Write or update documentation	139
27.8	Recipe 8: Automate a chore with a script	140
27.9	Recipe 9: Clean up a spreadsheet (CSV)	140
27.10	Recipe 10: Add a feature with an interview and a spec	140
27.11	Recipe 11: Make a team command	140
27.12	Recipe 12: Set up automatic pull request help on GitHub	141
27.13	Recipe 13: Make the same change across many files	141
27.14	Recipe 14: Watch a slow job	141
28	Security and Privacy	142
28.1	Your data	142
28.2	Prompt injection	143
28.3	Protecting secrets	143
28.4	Trusting other people's projects	144
28.5	Extra steps for teams	144
29	Troubleshooting	145
29.1	Your diagnostic toolkit	145
29.2	Installation and login	146
29.3	Everyday use	146
29.4	Extensions	147
29.5	Editors and other surfaces	147
29.6	Getting more help	147
	Slash Command Cheatsheet	148
	Quick Reference	149
30	The Complete Slash-Command Reference	150
30.1	Three kinds of slash command	151
30.2	/clear: start fresh	151
30.3	/compact: shrink without forgetting	152
30.4	/context: see what's using space	152
30.5	/rewind: the undo button	152
30.6	/resume, /branch, /fork, and /rename	153

30.7	/model and /effort	153
30.8	/init and /memory	153
30.9	/permissions, /sandbox, and /fewer-permission-prompts	154
30.10	Review and quality commands	154
30.11	Housekeeping commands	155
30.12	Recipes: commands that work well together	155
30.13	Every documented command, A-Z	155
31	Command-Line Flags and Shortcuts Reference	162
31.1	The ten flags beginners actually use	163
31.2	Terminal subcommands	163
31.3	Every startup flag	165
31.4	Every keyboard shortcut	168
32	Real-World Workflows, Step by Step	172
32.1	Workflow 1: Fix a bug from an error message	172
32.2	Workflow 2: Add a feature with plan mode	173
32.3	Workflow 3: Learn an unfamiliar project	174
32.4	Workflow 4: Tests first	174
32.5	Workflow 5: Ship a pull request	174
32.6	Workflow 6: Long tasks in the background	175
32.7	Workflow 7: Scripts with <code>cClaude -p</code>	175
32.8	Daily routine checklist	175
33	Troubleshooting and FAQ	176
33.1	First-aid routine	176
33.2	Symptom table	177
33.3	Frequently asked questions	177
34	Hooks, Settings, and Environment Reference	179
34.1	Settings files and precedence	179
34.2	Hook events	180
34.3	Worked example: format every file Claude edits	181
34.4	Environment variables	181
	Glossary	199

Welcome to Claude Code

Claude Code is an **AI coding agent**. That means it can do more than answer questions: it can open the files in your project, change them, run programs, check its own work, and keep going until a task is done. You stay in charge the whole time.

IN THIS CHAPTER

- What Claude Code is and what it's good for
- How the “agentic loop” works, and the tools Claude uses
- The different places you can use it
- What you need before you start, and how to use this manual

1.1 What is an “agent”?

You have probably used a **chatbot** such as the Claude app or ChatGPT. You type a question and it types back an answer. If it suggests code, *you* copy it, paste it into a file, and run it.

An **agent** closes that loop. When you ask Claude Code to “add a contact form to my website,” it will:

1. Look through your project to understand how it's built.
2. Decide which files need to change.
3. Edit those files (asking your permission first, unless you've told it not to).
4. Run commands, for example starting your website, to check that the change works.
5. Tell you what it did, in plain language.

This cycle of *look* → *plan* → *act* → *check* is called the **agentic loop**. It's what makes Claude Code feel less like a search engine and more like a teammate.

1.2 How the loop works under the hood

Anthropic describes the loop in three phases that blend together: **gather context**, **take action**, and **verify results**. A question about your code might only need the first phase. A bug fix cycles through all three, again and again, until the bug is gone.

Two things power the loop:

- **A model.** This is the AI “brain” that reads, reasons, and decides what to do next. Claude Code uses Anthropic’s Claude models: Opus, Sonnet, Haiku, and (on some plans) Fable. Chapter 18 explains how to pick one.
- **Tools.** A **tool** is an action Claude can take, such as “read this file” or “run this command.” Without tools, Claude could only reply with text. With tools, it can act.

Claude Code itself is the layer around the model that provides the tools and decides what the model sees. People call this layer the **harness**.

TOOL CATEGORY	WHAT CLAUDE CAN DO WITH IT
File operations	Read files, edit code, create new files, rename and reorganize
Search	Find files by name, search inside files, explore a project
Execution	Run shell commands, start servers, run tests, use Git
Web	Search the web, fetch documentation, look up error messages
Code intelligence	See type errors after edits, jump to definitions (needs a plugin)

When you say “fix the failing tests,” Claude might run the tests, read the error, search for the right file, read it, edit it, and run the tests again. Each step gives it new information for the next one.

NOTE

You’re part of the loop too. You can press `Esc` at any moment to stop Claude and point it in a new direction. It keeps everything it has learned so far.

1.3 What can you do with it?

You don’t need to be a programmer to get value from Claude Code. People use it to:

- **Build things from a description.** “Make me a one-page website for my bakery with our hours and a menu.”
- **Understand unfamiliar code.** “Explain what this project does, as if I’m new to coding.”
- **Fix problems.** “When I run the app I get this error. Find out why and fix it.”

- **Automate chores.** “Rename all these photos by the date they were taken.”
- **Work with Git and GitHub.** “Commit my changes with a clear message and open a pull request.”
- **Write documents and data files**, not just code: reports, spreadsheets, scripts, and more.

 TIP

You can ask Claude Code about itself. Questions like “what can Claude Code do?” or “how do I add an MCP server?” work, because it has its own documentation built in.

1.4 Where Claude Code runs

Claude Code is one product with several “front doors.” They share the same brain and the same settings, so what you learn in this manual works everywhere.

SURFACE	WHAT IT LOOKS LIKE	BEST FOR
Terminal (CLI)	A text window where you type	The full feature set; the focus of this manual
Desktop app	The Claude app’s Code tab	People who prefer buttons and windows over typing commands
VS Code / Cursor extension	A chat panel inside your code editor	Seeing edits side-by-side with your files
JetBrains plugin	Runs inside IntelliJ, PyCharm, etc.	Developers already using those editors
Web	claude.ai/code	Running tasks in the cloud, even from a phone
Slack, GitHub Actions	Mention Claude in a chat or pull request	Teams and automation

CLI stands for *command-line interface*: a program you control by typing text commands instead of clicking.

1.5 What you need

- ☐ A computer running **macOS 13+**, **Windows 10 (1809+)**, or a recent **Linux** (Ubuntu 20.04+, Debian 10+), with at least 4 GB of RAM
- ☐ An internet connection
- ☐ A **paid Claude account**: Pro, Max, Team, or Enterprise, *or* an Anthropic Console account that pays per use
- ☐ A folder to work in (it can be empty, so you don’t need an existing project)

 NOTE

The free Claude plan does **not** include Claude Code. The cheapest way in is the Pro plan. See the pricing table in Chapter 3.

1.6 How to use this manual

The chapters are grouped so you can read straight through or jump to what you need:

PART	CHAPTERS	WHAT YOU'LL LEARN
Getting started	1–5	Terminal basics, installing, your first session, good prompts
Context and commands	6–11	How memory fills up, every slash command, custom commands, shortcuts
Staying in control	12–17	Plan mode, checkpoints, permissions, settings, CLAUDE.md, sessions
Going further	18–25	Models, subagents, hooks, MCP, plugins, scripts, GitHub, other apps
Everyday practice	26–29	Costs, project recipes, security and privacy, troubleshooting
Reference	Back pages	Slash command cheatsheet, quick reference, glossary

Words in **bold** are explained the first time they appear and again in the Glossary at the back. Text in `code style` is something you type exactly. Keys look like `Esc`.

 TIP

Don't try to memorize anything. Type `/` inside Claude Code at any time to see a searchable menu of every command, and press `?` on an empty prompt to see the keyboard shortcuts.

Terminal Basics in Ten Minutes

The terminal looks intimidating, but you only need a handful of ideas to use Claude Code comfortably. Think of it as a very literal text message conversation with your computer.

IN THIS CHAPTER

- How to open a terminal on Mac, Windows, and Linux
- Folders, paths, and the “current directory”
- The commands you’ll actually use
- Copying, pasting, stopping programs, and a two-minute introduction to Git

2.1 Opening a terminal

OPEN YOUR TERMINAL

1 On a Mac.

Press `Cmd` + `Space`, type `Terminal`, and press `Enter`.

2 On Windows.

Press the `Windows` key, type `PowerShell`, and open **Windows PowerShell** (or **Terminal**, which opens PowerShell by default).

3 On Linux.

Press `Ctrl` + `Alt` + `T` on most systems, or search your apps for “Terminal.”

You’ll see a line ending in something like `%`, `$`, or `>`. That’s the **prompt**: the terminal waiting for you to type. When this manual shows a command, type it after the prompt and press `Enter`.

 TIP

On Windows, the prompt starts with `PS C:\` when you're in **PowerShell** and just `C:\` when you're in the older **Command Prompt (CMD)**. Claude Code works in both, but the install commands differ.

What a “shell” is

The program running inside the terminal window is called a **shell**. It reads what you type and runs it. Common shells are **Zsh** (the Mac default), **Bash** (common on Linux), and **PowerShell** or **CMD** on Windows. They mostly agree on the basics in this chapter; where they differ, this manual says so.

2.2 Folders and paths

Everything on your computer lives in **folders** (also called **directories**). A **path** is the address of a folder or file, like `/Users/sam/projects/bakery` on a Mac or `C:\Users\Sam\projects\bakery` on Windows.

The terminal is always “standing in” one folder, called the **current directory** or **working directory**. Claude Code works on the files in whatever folder you start it from. That's how it knows which project you mean.

The `~` symbol is shorthand for your **home folder** (for example `/Users/sam`). A single dot `.` means “this folder,” and two dots `..` mean “the folder above this one.”

2.3 The five commands you need

COMMAND	WHAT IT DOES	EXAMPLE
<code>pwd</code>	Show which folder you're in (“print working directory”)	<code>pwd</code>
<code>ls</code>	List the files here (<code>dir</code> in CMD)	<code>ls</code>
<code>cd</code>	Change into a folder	<code>cd projects/bakery</code>
<code>cd ..</code>	Go up one folder	<code>cd ..</code>
<code>mkdir</code>	Make a new folder	<code>mkdir bakery</code>

Here's how to make a practice folder and step inside it:

TERMINAL

```
cd ~
mkdir my-first-project
cd my-first-project
pwd
```

 TIP

Press `Tab` while typing a folder name and the terminal finishes it for you. Press the `↑` arrow to bring back the last command you typed.

A few more that come in handy

COMMAND	WHAT IT DOES	WINDOWS POWERSHELL
<code>cat notes.txt</code>	Print a file's contents	<code>cat notes.txt</code> works too
<code>open .</code>	Open this folder in Finder (Mac)	<code>explorer .</code>
<code>xdg-open .</code>	Open this folder in your file manager (Linux)	—
<code>clear</code>	Wipe the screen (not your files)	<code>cls</code> or <code>clear</code>
<code>code .</code>	Open this folder in VS Code, if installed	<code>code .</code>

You rarely need these, because Claude Code can run commands for you. Knowing them just helps you follow along.

2.4 Copying and pasting in a terminal

- **Mac:** `Cmd` + `C` and `Cmd` + `V`, as usual.
- **Windows Terminal / PowerShell:** `Ctrl` + `V` pastes; right-click also pastes.
- **Linux:** usually `Ctrl` + `Shift` + `V`.

In a terminal, `Ctrl` + `C` usually means **stop the running program**, not copy.

2.5 Stopping and exiting

- `Ctrl` + `C` stops whatever is running in the terminal.
- `exit` closes the shell (and usually the window).
- Inside Claude Code, `Esc` stops Claude's current task, and `/exit` quits Claude Code itself. You'll meet both again in Chapter 4.

2.6 Environment variables in one minute

An **environment variable** is a named setting that programs can read, like a sticky note attached to your terminal session. For example, `ANTHROPIC_API_KEY` holds an API key. You'll see a few of these later.

SHELL	SET ONE FOR THIS SESSION	READ IT BACK
Mac/Linux (Zsh, Bash)	<code>export NAME=value</code>	<code>echo \$NAME</code>
Windows PowerShell	<code>\$env:NAME = "value"</code>	<code>echo \$env:NAME</code>

A variable you set this way disappears when you close the window. That's usually what you want while experimenting.

2.7 Git in two minutes

Git is a tool that records snapshots of your files, called **commits**, so you can see what changed and go back in time. Claude Code works best in a folder that uses Git, because every change becomes easy to review and undo. A folder tracked by Git is called a **repository** (or **repo**).

You don't need to learn Git commands to start: you can ask Claude to run them. Here's what the common requests mean:

YOU SAY TO CLAUDE	WHAT HAPPENS
"Turn this folder into a Git repo"	Runs <code>git init</code> to start tracking files
"Commit everything with a clear message"	Saves a snapshot with a description
"Show me what changed since the last commit"	Runs <code>git diff</code> and explains it
"Undo my last uncommitted changes to index.html"	Restores the file from the last commit
"Make a new branch called new-menu"	Creates a separate line of work

NOTE

GitHub is a website for storing Git repositories online and collaborating. The *GitHub* manual in this series covers it from scratch.

- ☐ I can open a terminal and see the prompt
- ☐ I can use `cd`, `ls`, and `pwd` to move around
- ☐ I know `Ctrl + C` stops a program in the terminal
- ☐ I have a practice folder called `my-first-project`

CHAPTER 3

Install and Sign In

Installation takes about two minutes. Anthropic’s recommended method is the **native installer**, which also keeps Claude Code up to date automatically.

IN THIS CHAPTER

- Installing on macOS, Linux, and Windows (native or WSL)
- Choosing a plan and logging in
- Checking that everything works, and fixing the most common install errors
- Updating, choosing a release channel, and uninstalling

3.1 Before you install

Claude Code runs on these systems:

REQUIREMENT	SUPPORTED
macOS	13.0 or newer
Windows	10 version 1809 or newer, or Windows Server 2019+
Linux	Ubuntu 20.04+, Debian 10+, Alpine 3.19+
Hardware	4 GB+ RAM, x64 or ARM64 processor
Shell	Bash, Zsh, PowerShell, or CMD

You also need an internet connection, and you must be in one of Anthropic’s supported countries.

3.2 Install with the native installer

INSTALL CLAUDE CODE

1

Open a terminal.

See Chapter 2 if you're not sure how.

2

Paste the command for your system and press Enter.

macOS, Linux, or WSL:

TERMINAL

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell:

POWERSHELL

```
irm https://claude.ai/install.ps1 | iex
```

Windows Command Prompt (CMD):

COMMAND PROMPT

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

3

Wait.

The installer shows no progress bar while it downloads. That's normal.

4

Open a new terminal window

so it picks up the new program.

5

Check the install.

Type `claude --version`. You should see a version number followed by `(Claude Code)`.

⚠️ "COMMAND NOT FOUND"?

If your terminal says `claude` isn't found or isn't recognized, the install folder isn't on your **PATH** yet. (The PATH is the list of folders your terminal searches for programs.) Close every terminal window and open a fresh one first. If it still fails, run the installer again and read its last few lines. It prints the exact fix. The full fix for each system is later in this chapter.

Other ways to install

METHOD	COMMAND	UPDATES
Homebrew (Mac)	<code>brew install --cask claude-code</code>	Manual: <code>brew upgrade claude-code</code>
WinGet (Windows)	<code>winget install Anthropic.ClaudeCode</code>	Manual: <code>winget upgrade Anthropic.ClaudeCode</code>
npm (needs Node.js 22+)	<code>npm install -g @anthropic-ai/claude-code</code>	Manual
apt / dnf / apk (Linux)	See the setup page in the official docs	Through your system updates

⚠ DANGER

Never install with `sudo npm install -g`. Running npm as an administrator causes permission problems and is a security risk.

3.3 Notes for each operating system

macOS

- The native installer puts the program at `~/local/bin/claude`.
- The Homebrew cask `claude-code` follows the **stable** release channel; `claude-code@latest` follows **latest**. Pick one, not both.
- macOS uses Zsh by default, so PATH fixes go in `~/.zshrc`.

Windows

You can run Claude Code natively or inside **WSL** (Windows Subsystem for Linux, a built-in way to run Linux on Windows). Pick based on where your projects live:

OPTION	NEEDS	SANDBOX SUPPORT	USE IT WHEN
Native Windows	Nothing; Git for Windows optional	No	Your projects and tools are Windows-native
WSL 2	WSL 2 enabled	Yes	You use Linux tools or want the command sandbox
WSL 1	WSL 1 enabled	No	WSL 2 isn't available

- You do **not** need to run the installer as Administrator.
- Installing **Git for Windows** (from git-scm.com) is recommended. It gives Claude Code a Bash shell to run commands with. Without it, Claude Code uses PowerShell instead, which also works.

- If you already use **WSL**, install from inside your WSL terminal using the macOS/Linux command, and launch `claude` there too.
- If Claude Code can't find Git Bash, point it there in your settings file (Chapter 15 explains settings files):

~/CLAUDE/SETTINGS.JSON

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Linux

- Most distributions need nothing extra. The program lands in `~/local/bin/claude`.
- On **Alpine** and other musl-based systems, install the helpers first, then tell Claude Code to use the system copy of `ripgrep` (the fast search tool it relies on):

TERMINAL

```
apk add bash curl libgcc libstdc++ ripgrep
```

~/CLAUDE/SETTINGS.JSON

```
{
  "env": { "USE_BUILTIN_RIPGREP": "0" }
}
```

3.4 Choose a plan

Claude Code needs a paid account. Prices below are from claude.com/pricing in October 2026:

PLAN	PRICE	WHO IT'S FOR
Free	\$0	Chat only, no Claude Code
Pro	\$20/month (\$17/month billed yearly)	Learning and light daily use
Max 5x	\$100/month	Heavy daily use: 5× Pro's usage
Max 20x	\$200/month	All-day use: 20× Pro's usage
Team	\$25/seat/month Standard, \$125 Premium (less yearly)	Small teams, 2–150 people
Enterprise	Custom	Large organizations
Console (API)	Pay per use	Developers who prefer metered billing

 TIP

Start with **Pro**. If you regularly hit your usage limit, upgrade to Max later. Inside Claude Code, `/usage` shows how much of your limit you've used.

3.5 Log in for the first time

SIGN IN

- 1 **Go to your project folder.**
For example: `cd ~/my-first-project`.
- 2 **Start Claude Code.**
Type `claude` and press `Enter`.
- 3 **Pick a theme.**
Use the arrow keys to choose light or dark.
- 4 **Choose how to log in.**
Pick your Claude subscription (Pro/Max/Team/Enterprise) or Anthropic Console.
- 5 **Approve in your browser.**
A browser window opens; click **Authorize**. Return to the terminal.
- 6 **Trust the folder.**
Claude Code asks whether you trust the files in this folder. For your own projects, choose **Yes**.

You only log in once; Claude Code remembers your credentials. To switch accounts later, type `/login` inside Claude Code.

 TIP

If the browser doesn't open, press `c` at the login screen to copy the sign-in link, then paste it into any browser. This also helps over SSH (a remote connection to another computer) or inside WSL, where the browser may open on a different machine.

Signing in from the command line

You can manage your login without starting a session:

COMMAND	WHAT IT DOES
<code>claude auth login</code>	Sign in (add <code>--console</code> to use Console billing)
<code>claude auth logout</code>	Sign out
<code>claude auth status</code>	Show whether you're signed in

3.6 Health check and updates

- `claude doctor` checks your installation and settings and suggests fixes. Run it whenever something feels off.
- Native installs **update themselves** in the background. To update right now: `claude update`.
- Prefer slower, battle-tested releases? Type `/config` and set **Auto-update channel** to **stable** (about a week behind, skipping releases with major bugs).

NOTE

A brand-new model can need a newer Claude Code than the stable channel offers. If `/model` says a model isn't supported, switch back to the **latest** channel or run `claude update`.

3.7 Fixing "command not found"

The installer places `claude` in a folder called `.local/bin` inside your home folder. If your shell doesn't search that folder, add it to your PATH once.

FIX YOUR PATH ON MAC OR LINUX

1 Check the program is there.

Run `ls -la ~/.local/bin/claude`. If it says "No such file," the install didn't finish; run the installer again.

2 Add the folder to your PATH.

For Zsh (the Mac default):

TERMINAL

```
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

For Bash on Linux, use `~/.bashrc` instead of `~/.zshrc`. For Bash on a Mac, use `~/.bash_profile`.

3 Test it.

Run `claude --version`.

FIX YOUR PATH ON WINDOWS POWERSHELL**1 Check the program is there.**

Run `Test-Path "$env:USERPROFILE\.local\bin\claude.exe"`. It should print `True`.

2 Add the folder to your user PATH.**POWERSHELL**

```
$p = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH',
"$p;$env:USERPROFILE\.local\bin", 'User')
```

3 Restart the terminal and test it.

Run `claude --version`. If your terminal lives inside an editor such as VS Code, quit and reopen the editor.

3.8 Common install errors

WHAT YOU SEE	WHAT IT MEANS AND WHAT TO DO
<code>command not found: claude</code> / <code>'claude' is not recognized</code>	PATH problem. Open a new window, then follow the PATH fix above
<code>'irm' is not recognized</code>	You're in CMD, not PowerShell. Use the CMD command, or open PowerShell
<code>The token '&&' is not a valid statement separator</code>	You're in PowerShell. Use the PowerShell command
<code>'bash' is not recognized</code> or <code>parameter name 'fsSL'</code>	You pasted the Mac/Linux command into Windows. Use a Windows command
<code>syntax error near unexpected token '<'</code>	The download returned a web page, not the script (often a proxy or network filter). Check your connection and retry
<code>Killed</code> during install on Linux	Not enough memory. Close programs or add swap space, then rerun
<code>EACCES: permission denied</code>	The install folder isn't writable by you. Fix its permissions; don't use <code>sudo</code> with npm
<code>unable to get local issuer certificate</code>	A company network inspects traffic. Ask IT for its certificate setup
<code>App unavailable in region</code>	Claude Code isn't offered in your country
<code>Claude Code does not support 32-bit Windows</code>	Open the normal Windows PowerShell, not the "(x86)" one

Login problems

PROBLEM	FIX
Login keeps failing for no clear reason	Type <code>/logout</code> , close Claude Code, run <code>claude</code> , and sign in again
<code>OAuth error: Invalid code</code>	The code expired or was cut off when copied. Retry quickly; press <code>c</code> to copy the full link
Browser won't open in WSL	Run <code>export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"</code> (or your browser's path), then <code>claude</code>
Pasting the code does nothing	Try your terminal's other paste shortcut (right-click, or <code>Shift</code> + <code>Insert</code>), or use <code>claude auth login</code>
<code>403 Forbidden</code> or "access has not been granted"	Your account or organization role doesn't include Claude Code. Check your plan, or ask your admin

3.9 Uninstalling

If you ever want to remove Claude Code, use the method that matches how you installed it:

INSTALLED WITH	REMOVE WITH
Native installer (Mac/Linux/WSL)	<code>rm -f ~/.local/bin/claude</code> and <code>rm -rf ~/.local/share/claude</code>
Homebrew	<code>brew uninstall --cask claude-code</code>
WinGet	<code>winget uninstall Anthropic.ClaudeCode</code>
npm	<code>npm uninstall -g @anthropic-ai/claude-code</code>

⚠ DANGER

Your settings, history, and memory live in `~/.claude` and `~/.claude.json`. Deleting them erases all your preferences, MCP servers, and saved conversations. Only remove them if you want a completely clean slate.

- ☐ `claude --version` prints a version number
- ☐ I'm signed in with a paid plan or a Console account
- ☐ I trusted my practice folder
- ☐ I know `claude doctor` checks my setup and `claude update` updates it

CHAPTER 4

Your First Session

Let's build something. In this chapter you'll start Claude Code, ask it questions, and have it create and change real files. Follow along at your own computer: it takes about twenty minutes.

IN THIS CHAPTER

- What the Claude Code screen shows you
- A complete walkthrough: build a small website, change it, review it, save it
- Asking questions, pointing at files, and running your own commands
- Approving actions, interrupting, undoing, ending, and resuming

4.1 The Claude Code screen

When you run `claude`, you'll see:

- A **welcome panel** showing the version, the AI model in use, and your current folder.
- A **prompt box** where you type, marked with `>`.
- A **status bar** underneath showing the current *permission mode* (more on this in Chapter 14), and sometimes hints, the effort level, or a link to an open pull request.

Type your request in plain English and press `Enter`. There's no special syntax to learn.

TIP

To write several lines before sending, press `Shift` + `Enter` for a new line (or type `\` then `Enter`, which works in every terminal). `Ctrl` + `J` also adds a new line in any terminal.

The four special characters

Most of what you type is plain English, but four characters at the start of the prompt do something special:

TYPE	WHAT HAPPENS	EXAMPLE
/	Opens the command menu (Chapters 7–10)	/help
@	Autocompletes a file or folder name to point Claude at it	@index.html
!	Runs a shell command yourself; Claude sees the output	! ls
?	On an empty prompt, shows the keyboard shortcuts	?

4.2 Build a small website

YOUR FIRST PROJECT

1 Start in an empty folder.

TERMINAL

```
mkdir ~/bakery-site
cd ~/bakery-site
claude
```

2 Describe what you want.

PROMPT

Create a simple one-page website for a bakery called "Rise & Shine". Include our hours (Tue–Sun, 7am–3pm), a short menu with prices, and a contact section. Keep it to plain HTML and CSS.

3 Watch it plan and work.

Claude explains what it's about to do, then creates files such as `index.html` and `styles.css`.

4 Approve when asked.

If Claude asks permission to create a file, choose **Yes**. (Chapter 14 explains how to approve whole categories at once.)

5 Open the result.

Ask: `open the website in my browser`. Or open `index.html` from your file manager.

6 Change it.

Try: `make the colors warmer and add a photo placeholder at the top`.

That's the whole rhythm: **describe, review, refine**.

4.3 Watch what Claude is doing

While Claude works, it prints short lines such as “Read index.html” or “Edit styles.css.” To see the full detail, press **Ctrl** + **O** to open the **transcript viewer**. It shows every tool Claude used and its results. Press **Esc** or **q** to close it.

For bigger tasks Claude keeps a to-do **task list** for itself. Press **Ctrl** + **T** to show or hide it.

4.4 Review the changes

Before you keep anything, look at what changed. Type:

TEXT

```
/diff
```

You'll see every line Claude added (usually green) and removed (usually red). A view like this is called a **diff**. If something looks wrong, just say so: “Put the menu back the way it was, but keep the new colors.”

4.5 Save your work with Git

If your folder isn't a Git repository yet, ask Claude to set one up and save a first snapshot:

PROMPT

Turn this folder into a Git repository and commit everything with a clear message.

From now on, every commit is a safe point you can return to, even weeks later.

4.6 Ask questions about existing code

Claude Code shines when you point it at a project someone else wrote. Start it inside the project folder and try:

PROMPT

what does this project do?

PROMPT

explain the folder structure like I'm new to coding

PROMPT

where is the main entry point, and what happens when the app starts?

You don't need to tell Claude which files to read. It looks around by itself. If you *want* to point at a file, type `@` and start typing its name; Claude Code autocompletes the path.

4.7 Run your own commands with !

Sometimes you want to run a command yourself and have Claude react to the result. Start the line with `!`:

TEXT

```
! npm test
```

The command runs straight away, without needing Claude's approval, and its output lands in the conversation. Claude then responds to it, for example by explaining the failures. Press `Esc` or `Backspace` on an empty prompt to leave shell mode.

4.8 Approving, interrupting, and undoing

Permission prompts. Before Claude edits a file or runs a command it isn't sure about, it may show what it intends to do and offer choices like:

- **Yes:** allow this one action.
- **Yes, and don't ask again:** allow this kind of action from now on in this project.
- **No:** refuse, and optionally tell Claude what to do instead.

Press `Tab` on **Yes** or **No** to add a short comment with your answer, such as "Yes, but use port 8080."

Interrupting. Press `Esc` at any time to stop Claude mid-task. It keeps what it has done so far, and you can redirect it: "Stop. Use blue instead of green."

Queueing. You don't have to wait. Type a message and press `Enter` while Claude works; it's queued and sent when Claude is ready. Queued messages show in gray until Claude starts on them.

Undoing. Claude Code saves a **checkpoint** before each change. Press `Esc` twice on an empty prompt (or type `/rewind`) to roll the conversation *and/or* your files back to an earlier point. Chapter 13 covers this in depth.

⚠ WARNING

Checkpoints cover file edits Claude makes. They can't undo things outside your computer, like a sent email, a deleted cloud file, or a database change. For real safety, use Git (Chapter 14 and the GitHub manual in this series).

4.9 Ending and resuming

TO...	DO THIS
Leave Claude Code	Type <code>/exit</code> , or press <code>Ctrl</code> + <code>D</code> twice
Start fresh in the same window	<code>/clear</code>
Continue your last conversation	<code>claude -c</code> (from the terminal)
Pick an older conversation	<code>claude -r</code> , or <code>/resume</code> inside Claude Code
Name a session to find it later	<code>/rename bakery-site</code>

When you come back to a terminal after a few minutes away, Claude Code may show a one-line **recap** of what happened so far, so you can pick up where you left off.

4.10 Your first-session practice list

Try each of these in your bakery project. They take a minute each and cover the skills you'll use every day:

PROMPT

Add a "Specials" section under the menu with two weekly specials.

PROMPT

Explain, in three sentences, how `styles.css` changes the page.

PROMPT

Check the page for spelling mistakes and fix them.

TEXT

```
/rename bakery-site
```

TEXT

```
/exit
```

Then run `claude -c` in the same folder and ask `what did we do last time?`.

- ☐ I built a page by describing it in plain English
- ☐ I reviewed the changes with `/diff`
- ☐ I committed my work to Git
- ☐ I used `@` to point at a file and `!` to run a command
- ☐ I interrupted Claude with `Esc` and resumed a session with `claude -c`

Talking to Claude Effectively

Claude Code is capable, but it isn't a mind reader. A little care in how you ask makes a huge difference in what you get back. This chapter collects the habits and patterns that work best, with prompts you can copy.

IN THIS CHAPTER

- The habits that produce great results
- Plan mode: think first, then build
- Sharing screenshots, errors, and files
- Proven prompting patterns, with before-and-after examples
- A library of reusable prompts, and the mistakes to avoid

5.1 Five habits of great prompts

1. **Be specific.** Instead of “fix the bug,” say “users see a blank screen after entering a wrong password, so fix that.”
2. **Give the goal, not just the task.** “I want customers to order ahead online” helps Claude make better choices than “add a form.”
3. **Break big jobs into steps.** Number them, and Claude will work through them in order.
4. **Let Claude explore first.** “Read the code and explain how login works before changing anything.”
5. **Say how you'll judge success.** “When you're done, run the tests and show me they pass.”

PROMPT

1. Add a "Order ahead" page with a form for name, item, and pickup time.
2. Save orders to a file called orders.csv.
3. Add a link to the new page in the site's menu.
4. Open the site and confirm the form works before you finish.

5.2 Plan mode: look before you leap

For anything bigger than a small tweak, ask Claude to **plan first**. In **plan mode** Claude researches and writes a step-by-step plan but won't edit your files until you approve it.

- Press `Shift` + `Tab` until the status bar says `plan mode on`, **or**
- Start your message with `/plan`, for example: `/plan add online ordering`.

When the plan appears you can approve it, ask for changes (“No, keep planning, and don't use a database”), or press `Ctrl` + `G` to edit the plan yourself in a text editor. Chapter 12 covers plan mode in depth.

TIP

Plan mode is the single best habit for beginners. Reading the plan teaches you how the project works, and it catches misunderstandings before any files change.

5.3 Share what you see

- **Error messages:** copy and paste them straight into the prompt. Include the whole thing.
- **Screenshots:** paste an image with `Ctrl` + `V` (`Cmd` + `V` in iTerm2, `Alt` + `V` on Windows). Claude can see it. Each image shows up as a chip like `[Image #1]` you can refer to.
- **Files:** type `@` followed by part of a file name.
- **A command's output:** start a line with `!` to run a shell command yourself, for example `!npm test`. Claude sees the result and responds to it.

5.4 Ask a side question

Want to ask something without derailing the current task? Use `/btw`:

PROMPT

```
/btw what's the difference between HTML and CSS?
```

The answer appears without being added to the main conversation. Side questions can't use tools, so Claude answers only from what it already knows in this session.

5.5 Keep conversations focused

Claude can hold a lot in mind at once, but that space, the **context window**, is not unlimited. Long, wandering sessions get slower and cost more of your usage.

- Use `/clear` when you switch to an unrelated task.
- Use `/compact` to summarize a long conversation and free up space while staying on the same task.
- Use `/context` to see how full the context window is.

Chapter 6 explains exactly how the context window works and when to use each command.

5.6 Pattern 1: Explore, plan, code, commit

Anthropic's recommended workflow for anything non-trivial has four phases:

EXPLORE → PLAN → CODE → COMMIT

1 Explore.

Switch to plan mode and ask Claude to read the relevant parts.

PROMPT

read the files that handle the menu and understand how prices are shown. don't change anything yet.

2 Plan.

Ask for a written plan.

PROMPT

I want to add a "sold out" badge to menu items. what files need to change? write a step-by-step plan.

3 Code.

Approve the plan, then let Claude build and check it.

PROMPT

implement the plan. then open the page and confirm the badge shows on items marked sold out.

4

Commit.

Save the result with a clear message.

PROMPT

commit with a descriptive message

NOTE

If you could describe the change in one sentence, such as “fix the typo in the header,” skip the plan and just ask.

5.7 Pattern 2: Give Claude a way to check its work

Claude stops when the work *looks* done. If it has a test, a build, or a screenshot to compare against, it can check for itself and keep fixing until the check passes. This is the most powerful habit of all.

INSTEAD OF	TRY
“implement a function that validates email addresses”	“write a validateEmail function. <code>user@example.com</code> is valid, <code>invalid</code> is not, <code>user@.com</code> is not. run the tests after implementing”
“make the dashboard look better”	“ <code>[paste screenshot]</code> implement this design. take a screenshot of the result, compare it to the original, and fix the differences”
“the build is failing”	“the build fails with this error: <code>[paste]</code> . fix the root cause, don’t just hide the error, and confirm the build succeeds”

Ask Claude to **show evidence**: the test output, the command it ran, or a screenshot. Reviewing evidence is faster than checking everything yourself.

5.8 Pattern 3: Point at examples and sources

INSTEAD OF	TRY
“add tests for foo.py”	“write a test for foo.py covering the case where the user is logged out. avoid mocks”
“add a calendar widget”	“look at how the existing widgets on the home page are built. follow that pattern for a calendar widget”
“why is this code so weird?”	“look through this file’s Git history and summarize how it came to be this way”
“fix the login bug”	“login fails after the session times out. check the code in src/auth/. write a failing test that shows the bug, then fix it”

5.9 Pattern 4: Let Claude interview you

For a bigger feature you haven't fully thought through, ask Claude to ask *you* the questions first:

PROMPT

I want to build an online ordering page for my bakery. Interview me in detail using the AskUserQuestion tool. Ask about how it should work, what customers see, edge cases, and tradeoffs. Skip obvious questions. When we've covered everything, write a spec to SPEC.md.

When the spec is done, start a fresh session (`/clear`) and ask Claude to build what `SPEC.md` describes. The new session has a clean context focused entirely on building.

5.10 Pattern 5: Ask for deeper thinking when it matters

Claude thinks before it answers. For a hard problem, include the word `ultrathink` anywhere in your prompt to ask for deeper reasoning on that one turn:

PROMPT

ultrathink: why does the order total sometimes come out one cent off?

Other phrases such as “think hard” are treated as ordinary words. To change how much Claude thinks for the whole session, use `/effort` (Chapter 18).

5.11 A library of reusable prompts

SITUATION	PROMPT TO PASTE
New project	"Give me a tour of this project: what it does, the main folders, and how to run it."
Before changing anything	"Read the relevant code and explain your understanding back to me before you edit anything."
A confusing error	"Here's the full error: [paste]. Explain it in plain English, find the cause, and fix it."
Unclear instructions	"Ask me up to five questions before you start, so you understand what I want."
Getting a second opinion	"Review what you just did as a strict senior reviewer. List anything risky or sloppy."
Learning while you work	"Explain each change as you make it, as if I'm learning to code."
Keeping scope small	"Only change files in the menu/ folder. Ask before touching anything else."
Wrapping up	"Summarize what we changed today, what's left, and any risks, in five bullets."

5.12 Common mistakes and their fixes

MISTAKE	WHAT GOES WRONG	FIX
The kitchen-sink session	You mix unrelated tasks; context fills with noise	<code>/clear</code> between unrelated tasks
Correcting over and over	Failed attempts pile up and confuse Claude	After two failed corrections, <code>/clear</code> and write a better first prompt
The giant CLAUDE.md	Important rules get lost in a long file	Keep it short; move procedures into skills (Chapter 10)
Trusting without checking	Plausible code misses edge cases	Always give a test, a build, or a screenshot to check against
Endless exploring	"Investigate this" reads hundreds of files	Narrow the question, or ask Claude to use a subagent (Chapter 19)

- ☐ My prompts say what, where, and how I'll judge success
- ☐ I use plan mode for anything that touches several files
- ☐ I give Claude a way to check its work
- ☐ I `/clear` between unrelated tasks

How Context Works: `/clear` or `/compact`?

Every message you send, every file Claude reads, and every command output it sees goes into Claude's working memory for the session. Understanding that memory, the **context window**, is the key to fast, cheap, accurate sessions.

IN THIS CHAPTER

- What the context window is and what fills it
- How to see what's using space with `/context`
- What automatic compaction does, and what survives it
- Exactly when to use `/clear`, `/compact`, `/rewind`, `/btw`, or a subagent

6.1 The context window, explained simply

Imagine Claude working at a desk. Everything it needs for the current task is spread out on the desk: your instructions, the files it opened, the test results it ran. The desk is the **context window**. It's large, but not infinite.

Two things make the desk matter:

- **Every message re-reads the desk.** Each time you send a prompt, Claude Code sends the whole conversation so far to the model. A cluttered desk costs more usage on *every* turn.
- **A crowded desk is distracting.** When old, unrelated material fills the space, Claude can lose track of what matters now.

Context is measured in **tokens**, chunks of text roughly three-quarters of a word long. Current Claude models on the Anthropic API, such as Opus 5.5, Sonnet 5.5, and Haiku 5.5, run with a window of about **1 million tokens** by default. That's a lot, but long sessions with big files still fill it.

6.2 What fills the context window

Some things load before you type anything; others pile up as you work.

LOADS AT THE START	BUILDS UP AS YOU WORK
Your <code>CLAUDE.md</code> files (Chapter 16)	Your messages and Claude's replies
Auto memory notes	Every file Claude reads
The names of available tools and MCP servers	Command and test output
Short descriptions of available skills	Skills you invoke (their full text)
Claude Code's own instructions	Rules that apply to files Claude opens

NOTE

MCP tools (Chapter 21) are loaded on demand. Until Claude needs a tool, only its name takes up space.

6.3 See what's using space: /context

Type `/context` at any time. Claude Code draws a colored grid showing how full the window is, broken down by category, with suggestions such as “this MCP server is using a lot of space” or “your memory files are large.” Use `/context all` to expand the full per-item breakdown.

TEXT

`/context`

TIP

You can also show context usage permanently at the bottom of the screen with a custom **status line**. Type `/statusline show context usage and the model name` and Claude sets one up for you.

6.4 Automatic compaction

You don't have to manage context by hand. As the window approaches its limit, Claude Code **compacts** the conversation automatically: it first clears out old tool outputs, then replaces the older conversation with a structured summary. Your session keeps going.

The tradeoff is detail. A summary keeps your requests, key decisions, and important code, but specific instructions you gave early in the conversation may be lost. That's why lasting rules belong in `CLAUDE.md`, not in chat.

What survives compaction

KIND OF CONTENT	AFTER COMPACTION
Project-root <code>CLAUDE.md</code> and unscoped rules	Re-read from disk and put back
Auto memory	Re-read from disk
The plan from plan mode	Re-read from disk
Recently read or edited files	Up to five re-read, most recently changed first
Skills you invoked	Put back, trimmed to the first 5,000 tokens each
Nested <code>CLAUDE.md</code> files and path-scoped rules	Reload when Claude next opens a matching file
Instructions you only typed in chat	Summarized, so details may be lost
Background commands and subagents	Keep running

6.5 /clear: a fresh desk

`/clear` starts a brand-new conversation with an empty context. Your project memory (`CLAUDE.md`, auto memory) still loads, so Claude still knows your project's rules; it just forgets the chat.

TEXT

```
/clear
```

- **It costs nothing.** No summary is made, so no usage is spent.
- **Nothing is lost forever.** The old conversation is saved. Get it back with `/resume`, or, in the same Claude Code window, from the top of the `/rewind` menu.
- **Give the old chat a name** to find it easily: `/clear menu-redesign` labels the previous conversation in the `/resume` picker.
- `/reset` and `/new` do the same thing.

6.6 /compact: a tidy desk

`/compact` summarizes the conversation so far and replaces it with the summary. You stay on the same task with most of the useful knowledge, but far fewer tokens.

TEXT

```
/compact
```

The best part is that you can tell it what to keep:

TEXT

/compact focus on the checkout bug and the files we changed

- **It costs some usage**, because Claude has to read the whole conversation to summarize it.
- **Steer it ahead of time.** Add a section to your `CLAUDE.md` that every compaction follows:

CLAUDE.MD

```
# Compact instructions
When compacting, always keep the full list of files we changed
and the exact test commands we used.
```

6.7 The decision table

YOUR SITUATION	USE	WHY
Starting an unrelated task	<code>/clear</code>	Old context is noise and costs tokens every turn
Same task, conversation getting long	<code>/compact focus on ...</code>	Keeps what matters, frees space
Corrected Claude twice and it's still wrong	<code>/clear</code> , then a better prompt	Failed attempts confuse it; a clean start wins
A long detour you no longer need	<code>/rewind</code> → Summarize from here	Compresses only the detour
Long setup chat, but recent work matters	<code>/rewind</code> → Summarize up to here	Compresses the old part, keeps the recent part
Quick question about the work so far	<code>/btw</code>	Answer never enters the conversation
Big research job ("read every file in src/")	Ask for a subagent	Reading happens in the subagent's own context
Want compaction to happen earlier	<code>/autocompact 500k</code>	Compacts sooner, keeping each turn smaller

☆ IMPORTANT

A rule of thumb from Anthropic: if you've corrected Claude more than twice on the same issue, the conversation is cluttered with failed approaches. `/clear` and start again with a more specific prompt that includes what you learned. A clean session with a better prompt almost always wins.

6.8 Other ways to keep context lean

- **Delegate big reads to subagents.** A subagent works in its own context window and hands back only a summary. Just ask: “use a subagent to find every place we calculate tax.”
- **Move procedures out of CLAUDE.md into skills.** A skill’s full text only loads when it’s used (Chapter 10).
- **Turn off MCP servers you aren’t using** with `/mcp`.
- **Be specific.** “Fix the tax rounding in cart.js” reads one file; “improve the code” reads everything.

6.9 Prompt caching, briefly

Claude Code reuses the unchanged start of your conversation from a short-term cache, which makes repeated context much cheaper. On a subscription the cache lasts about an hour. If you step away longer, your next message reprocesses the full context. One more reason to `/clear` before starting something new after a long break.

- ☐ I’ve run `/context` and seen what fills my window
- ☐ I `/clear` between unrelated tasks and `/compact focus on ...` within long ones
- ☐ I keep lasting rules in `CLAUDE.md`, not in chat
- ☐ I ask for a subagent when a task needs lots of reading

Slash Commands: The Basics and Your Session

A **slash command** is a short instruction to Claude Code itself, typed at the start of the prompt with a `/`, such as `/clear` or `/compact`. Commands control the session: they switch models, manage memory, review changes, and much more. This chapter and the next two cover **every** built-in command, grouped by what you use it for.

IN THIS CHAPTER

- How slash commands work: arguments, aliases, the menu, and queueing
- Commands for your session and context: `/clear`, `/compact`, `/context`, `/rewind`, `/resume` and friends
- Commands that shape how Claude works: `/plan`, `/goal`, `/loop`, `/cd`, `/add-dir`
- Commands for background and parallel work

7.1 How slash commands work

Type `/` on an empty prompt and a menu appears. Keep typing to filter it: `/com` narrows the list to `/compact`, `/config`, `/context`, and so on. Press `Enter` to run the highlighted command, or `Tab` to fill it in and keep typing.

RUN YOUR FIRST COMMANDS

- 1 Open the menu.**
Type `/` and look through the list.
- 2 Run a command with no arguments.**
Type `/help` and press `Enter`.

3 Run a command with arguments.

Type `/rename practice-session` and press `Enter`. The name appears on the prompt bar.

4 Try an alias.

Type `/new`. The menu highlights `/clear`, because `/new` is another name for it.

The rules in brief

- **A command only counts at the start of your message.** Whatever follows the name becomes its **arguments** (extra information), as in `/compact focus on the login bug`.
- **Notation in this manual:** `<arg>` means required, `[arg]` means optional. So `/add-dir <path>` needs a folder, while `/clear [name]` works with or without one.
- **Aliases** are alternative names. `/reset` and `/new` both run `/clear`; `/quit` runs `/exit`.
- **Sending a command while Claude is busy** queues it until the current turn ends. A few, such as `/status`, `/tasks`, and `/usage`, run immediately.
- **Typos don't run anything risky.** If you mistype, nothing is highlighted, and pressing `Enter` just reports `Unknown command`.
- **Not every command appears for everyone.** Some depend on your operating system, plan, or settings. A few, such as `/heapdump`, are hidden from the menu on purpose and run only when you type the full name.
- **Mid-sentence completion.** Typing `/` after a space, as in `run the tests, then /com`, offers to complete a command name, but only a command at the very start of a message actually runs.

Three kinds of command

KIND	HOW IT WORKS	EXAMPLES
Built-in command	Fixed behavior coded into Claude Code	<code>/clear</code> , <code>/model</code> , <code>/config</code>
Bundled skill	A ready-made set of instructions Claude follows using its tools	<code>/doctor</code> , <code>/code-review</code> , <code>/loop</code> , <code>/batch</code>
Your own commands	Skills and commands you or your plugins add (Chapter 10)	<code>/fix-issue 123</code>

You use all three the same way. This manual marks bundled skills with *(skill)*. MCP servers can add commands too; Chapter 10 shows how.

💡 TIP

Each command entry below has the same parts: what it does, **When** a beginner would use it, **Try** for an example you can type, and **Watch out** for gotchas.

7.2 Session and context commands

`/clear [name]`

Starts a new conversation with an empty context. Your `CLAUDE.md` files and auto memory still load. Aliases: `/reset`, `/new`.

- **When:** you finish one task and start an unrelated one, or Claude seems confused after several corrections.
- **Try:** `/clear`, or `/clear menu-redesign` to label the old conversation so you can find it in `/resume`.
- **Watch out:** the old chat isn't deleted. Get it back with `/resume`, or from the top of the `/rewind` menu in the same window. Clearing costs no usage.

`/compact [instructions]`

Summarizes the conversation so far and replaces it with the summary, freeing space while you stay on the same task.

- **When:** a long task is still going and `/context` shows the window filling up.
- **Try:** `/compact focus on the checkout bug and the files we changed`
- **Watch out:** compacting uses some usage, because Claude reads everything to summarize it. Details you only typed in chat can be lost; put lasting rules in `CLAUDE.md`.

`/context [all]`

Shows how full the context window is as a colored grid, with suggestions for what to trim.

- **When:** sessions feel slow, or you want to know what's eating space.
- **Try:** `/context`, or `/context all` for the full item-by-item breakdown.
- **Watch out:** if the conversation is already over the limit, the output says by how much and which command frees space.

`/autocompact [auto|<tokens>]`

Sets how full the context window gets before Claude Code compacts automatically.

- **When:** you want smaller, cheaper turns and don't mind compacting more often.
- **Try:** `/autocompact 500k`, or `/autocompact auto` to go back to the default for your model.
- **Watch out:** the value is saved to your user settings and applies to future sessions too. To change it for one session only, start with `claude --autocompact 500k`.

/rewind

Opens the rewind menu to restore code, conversation, or both to an earlier point, or to summarize part of the conversation. Same as pressing **Esc** twice on an empty prompt. Aliases: **/checkpoint**, **/undo**.

- **When:** Claude went down the wrong path, or you want to compress a long detour.
- **Try:** **/rewind**, pick the message before things went wrong, then **Restore code and conversation**.
- **Watch out:** changes made by shell commands (such as **rm** or **mv**) and edits you made outside Claude Code aren't tracked. Chapter 13 has the full story.

/resume [session]

Reopens an earlier conversation, by name or ID, or from a picker. Alias: **/continue**.

- **When:** you closed Claude Code yesterday and want to carry on.
- **Try:** **/resume**, or **/resume bakery-site** to jump straight to a named session.
- **Watch out:** local transcripts are kept for 30 days by default. From the terminal, **claude -c** continues the latest session in this folder and **claude -r** opens the picker.

/rename [name]

Gives the current session a name, shown on the prompt bar and in **/resume**.

- **When:** you'll want to find this conversation again later.
- **Try:** **/rename checkout-bug**. Without a name, Claude Code makes one up from the conversation.
- **Watch out:** names are capped at 200 characters. You can also name a session at launch with **claude -n checkout-bug**.

/branch [name]

Splits the conversation at this point so you can try a different direction. You move into the new branch; the original stays safe in **/resume**.

- **When:** you want to try a risky idea without losing the conversation as it stands.
- **Try:** **/branch try-database-version**
- **Watch out:** this branches the *conversation*, not your Git repository. Your files are shared, so commit or use a Git branch if you need separate code.

/recap

Writes a one-line summary of the session so far.

- **When:** you've lost track, or you're handing the session to someone else.
- **Try:** `/recap`
- **Watch out:** Claude Code also shows a recap on its own when you return after a few minutes away. Turn that off in `/config` under **Session recap**.

/btw [question]

Asks a side question about the current session without adding it to the conversation history.

- **When:** you want to check a detail without derailing the task, even while Claude is still working.
- **Try:** `/btw what was the name of that config file again?`
- **Watch out:** side questions have no tools, so Claude answers only from what's already in the session. In the answer overlay, press `c` to copy it or `f` to hand the question to a subagent that *can* use tools. Run `/btw` alone to reopen your last answer.

/export [filename]

Saves the conversation as plain text.

- **When:** you want a record of a session, or to share it with a colleague.
- **Try:** `/export checkout-bug.txt`, or just `/export` to choose between the clipboard and a file.
- **Watch out:** the file contains everything in the chat, including any secrets you pasted. Check before sharing.

/copy [N]

Copies Claude's last response to your clipboard. `/copy 2` copies the one before that.

- **When:** you want to paste Claude's answer into an email or document.
- **Try:** `/copy`. If the response has code blocks, a picker lets you copy just one.
- **Watch out:** over SSH the clipboard may not reach your computer; press `w` in the picker to write to a file instead. In iTerm2, run `/terminal-setup` once so copying works.

/exit

Quits Claude Code. Alias: `/quit`. Pressing `Ctrl` + `D` twice does the same.

- **When:** you're done for now. Your conversation is saved for `/resume`.
- **Try:** `/exit`
- **Watch out:** in an attached background session, `/exit` *detaches* and the session keeps running. Use `/stop` to actually stop it.

7.3 Commands that shape how Claude works

`/plan [description]`

Switches to plan mode, where Claude researches and proposes a plan but doesn't edit files until you approve.

- **When:** any change that touches several files, or code you don't know well.
- **Try:** `/plan add a gift-card option to checkout`
- **Watch out:** plan mode adds a step. For one-line fixes, just ask. Chapter 12 covers plan mode in depth.

`/goal [condition|clear]`

Sets a finish line. Claude keeps working turn after turn, and after each turn a separate model checks whether the condition is met.

- **When:** a job with a clear, checkable end, like “all tests pass.”
- **Try:** `/goal all tests in tests/cart pass and the linter is clean`. Run `/goal` alone to see the current goal, and `/goal clear` to cancel it.
- **Watch out:** a goal doesn't change your permission mode, so in Manual mode Claude still stops to ask permission. Only one goal can be active per session, and every extra turn uses more of your plan.

`/loop [interval] [prompt] (skill)`

Runs a prompt again and again while the session stays open. Alias: `/proactive`.

- **When:** checking on something slow, like a deploy or a CI run.
- **Try:** `/loop 5m check if the deploy finished and tell me what happened`, or `/loop check whether CI passed` to let Claude choose the interval.
- **Watch out:** loops stop when you close Claude Code, and recurring ones expire after seven days. Each run sends your whole conversation, so it uses usage even while you're away. Press `Esc` to stop a self-paced loop.

`/cd <path>`

Moves the session to a different folder while keeping the conversation.

- **When:** you started Claude Code in the wrong folder.
- **Try:** `/cd ~/projects/bakery-site`. Type part of a path and press `Tab` to complete it.
- **Watch out:** the new folder's project settings start applying. To keep the current folder *and* reach another one, use `/add-dir`.

```
/add-dir <path>
```

Lets Claude read and edit files in an extra folder for this session.

- **When:** your work spans two folders, such as a website and a shared images folder.
- **Try:** `/add-dir ../shared-assets`
- **Watch out:** this grants file access, not configuration: most of the other folder's `.claude/` settings aren't loaded, although its skills are. Network drives such as `\\server\share` usually can't be added.

7.4 Background and parallel work

These commands let Claude work on several things at once. They're handy once you're comfortable; skip them on day one. Chapter 17 explains background sessions and worktrees.

```
/tasks
```

Lists the background work in this session: long-running commands and subagents, including ones that finished. Alias: `/bashes`.

- **When:** you pressed `Ctrl` + `B` to background a server and want to check on it.
- **Try:** `/tasks`
- **Watch out:** this is different from Claude's to-do checklist, which you toggle with `Ctrl` + `T`.

```
/background [prompt]
```

Detaches the whole session so it keeps running as a **background agent** and frees your terminal. Alias: `/bg`.

- **When:** a long task can finish without you watching.
- **Try:** `/background finish the refactor and run the full test suite`
- **Watch out:** check on it later with `claude agents` from your shell. Background sessions still ask for permissions, so a task can pause while waiting for you.

```
/fork [prompt]
```

Copies the conversation into a new background session while you keep working here.

- **When:** you want a second attempt at something to run in parallel.
- **Try:** `/fork try the same fix using a database instead of a file`
- **Watch out:** Claude Code normally tells the copy to create its own Git worktree before changing code, so its edits don't collide with yours. Manage it with `claude agents`.

/subtask <task>

Starts a background subagent that knows the full conversation, works on a side task, and reports back here when it's done.

- **When:** a side job, like writing tests, can happen while you carry on.
- **Try:** `/subtask write unit tests for the cart total function`
- **Watch out:** the result comes back into *this* conversation. For a separate session, use `/fork`; to switch into a copy yourself, use `/branch`.

/stop

Stops the background session you're attached to. Its transcript and any worktree are kept.

- **When:** a background session is going the wrong way.
- **Try:** `/stop`
- **Watch out:** to leave a background session running and just step out, use `/exit` or press `↵` instead.

/batch <instruction> (skill)

Plans a large change, splits it into 5 to 30 independent pieces, and, once you approve, runs one background subagent per piece, each in its own worktree.

- **When:** the same kind of change must happen across many files.
- **Try:** `/batch migrate src/ from JavaScript to TypeScript`
- **Watch out:** it needs a Git repository and can use a lot of usage, because many subagents run at once. Read the plan carefully before approving.

/list-agents

Lists the subagents, teammates, and other Claude Code sessions this session can send messages to. Alias: `/peers`.

- **When:** you're coordinating several sessions and need their names.
- **Try:** `/list-agents`
- **Watch out:** only available where cross-session messaging is turned on; otherwise it reports `Unknown command`.

- ☐ I know a command only runs when it starts my message
- ☐ I can tell `/clear` (fresh start) from `/compact` (summary) from `/rewind` (go back)
- ☐ I can name, branch, export, and resume a session
- ☐ I've tried `/btw` for a side question and `/plan` before a bigger change

Slash Commands: Setup, Models, and Cost

This chapter covers the commands you use to set Claude Code up the way you like it, choose which model does the work, and keep an eye on your usage. Most of them you'll run once or twice and then forget about, which is exactly the point.

IN THIS CHAPTER

- Project setup and memory: `/init`, `/memory`, `/import`
- Settings and permissions: `/config`, `/status`, `/permissions` and more
- Look and feel: `/theme`, `/color`, `/tui`, `/statusline`, `/output-style`, `/voice`
- Models and cost: `/model`, `/effort`, `/fast`, `/advisor`, `/usage` and the `limit` commands

8.1 Project setup and memory

`/init`

Studies your project and writes a starter `CLAUDE.md` file with build commands, test instructions, and conventions it finds.

- **When:** the first time you use Claude Code in a project.
- **Try:** `/init`
- **Watch out:** if a `CLAUDE.md` already exists, it suggests improvements instead of overwriting. Read what it writes and delete anything vague. For a guided version that also offers skills, hooks, and a personal memory file, start Claude Code with the environment variable `CLAUDE_CODE_NEW_INIT=1` set.

/memory

Opens your `CLAUDE.md` files for editing, turns **auto memory** (notes Claude keeps for itself) on or off, and shows what auto memory has saved.

- **When:** Claude keeps forgetting a rule, or you're curious what it remembers about you.
- **Try:** `/memory`, then pick a file to open it in your editor.
- **Watch out:** turning auto memory off here saves the choice to your user settings, so it applies to every project. Chapter 16 covers memory in depth.

/import [codex|gemini|cursor] [--dry-run] [--yes]

Copies your setup from OpenAI Codex, Google Gemini CLI, or Cursor into Claude Code: instruction files, MCP servers, commands, subagents, and skills.

- **When:** you're switching to Claude Code from one of those tools.
- **Try:** `/import cursor --dry-run` to preview what would come across without changing anything.
- **Watch out:** not available on Amazon Bedrock, Google Cloud, or Microsoft Foundry setups. `/init` offers it automatically when it spots Codex or Gemini files.

8.2 Settings and permissions

/config [key=value ...]

Opens the Settings screen for theme, model, editor mode, output style, and other preferences.

Alias: `/settings`.

- **When:** you want to change a preference without editing a file.
- **Try:** `/config`, or set things directly: `/config theme=dark`, `/config model=sonnet`. Run `/config --help` to list the keys it accepts.
- **Watch out:** the `key=value` form can turn *off* a setting that needs your confirmation, such as automatic continue at a usage limit, but can't turn it on. In the VS Code panel and the desktop app, change settings through those apps instead.

/status

Opens the Status tab: your Claude Code version, model, account, and connection health.

- **When:** you want to confirm which account or model you're on, or that a settings file loaded.
- **Try:** `/status`
- **Watch out:** it works even while Claude is busy. It also lists warnings, such as an instruction file that's too long.

/permissions

Opens a dialog to view, add, and remove **allow**, **ask**, and **deny** rules, manage working folders, and review recent auto mode denials. Alias: `/allowed-tools`.

- **When:** Claude keeps asking about the same safe command, or you want to block something for good.
- **Try:** `/permissions`, then add `Bash(npm run test *)` to the allow list.
- **Watch out:** changes made while Claude is working apply from its next action. Chapter 14 explains the rule syntax.

/fewer-permission-prompts (skill)

Scans your past sessions for harmless, read-only commands you approve often, and adds a prioritized allow list to the project's `.claude/settings.json`.

- **When:** permission prompts are slowing you down.
- **Try:** `/fewer-permission-prompts`
- **Watch out:** it writes to the *shared* project settings file, which your team may commit. Review the list before you keep it.

/auto-mode-setup

Drafts a description of your trusted infrastructure (the repositories, storage buckets, and web domains that are yours) for the **auto mode** safety classifier, then saves it to your user settings after you review it.

- **When:** auto mode blocks actions that are actually fine, such as pushing to your company's repositories.
- **Try:** `/auto-mode-setup`
- **Watch out:** requires a Pro, Max, or Team plan. By default the classifier trusts only your working folder and the current repository's remotes.

/update-config [request] (skill)

Describe a settings change in plain English and Claude edits the right `settings.json` file for you.

- **When:** you know what you want but not which file or key it lives in.
- **Try:** `/update-config allow npm test without asking`, or `/update-config show a notification when Claude needs me`
- **Watch out:** for simple preferences like theme or model, `/config` is quicker.

`/sandbox`

Turns the command **sandbox** on or off. The sandbox is a boundary, enforced by your operating system, around the shell commands Claude runs: it limits which files they can write and which websites they can reach.

- **When:** you want fewer permission prompts with an extra safety net.
- **Try:** `/sandbox`, then choose a mode in the panel.
- **Watch out:** it works on macOS, Linux, and WSL 2, not native Windows. On Linux and WSL 2 it needs the `bubblewrap` and `socat` packages; the panel tells you if they're missing. It covers shell commands only, not Claude's file tools or MCP servers.

8.3 Look and feel

`/theme`

Changes the color theme: light, dark, an `auto` option that follows your terminal, colorblind-friendly themes, and themes that use your terminal's own colors.

- **When:** the text is hard to read, or you just prefer a different look.
- **Try:** `/theme`. Inside the picker, `Ctrl` + `T` toggles code coloring.
- **Watch out:** pick **New custom theme...** to build your own.

`/color [color|default]`

Colors this session's prompt bar: `red`, `blue`, `green`, `yellow`, `purple`, `orange`, `pink`, or `cyan`.

- **When:** you have several Claude Code windows open and want to tell them apart.
- **Try:** `/color green`, `/color default` to reset, or just `/color` for a random one.
- **Watch out:** the color is for this session only.

`/tui [default|fullscreen]`

Switches the screen renderer and relaunches with your conversation intact. `fullscreen` is a smoother, flicker-free mode with mouse support.

- **When:** the screen flickers or jumps while Claude works, especially in VS Code's terminal or tmux.
- **Try:** `/tui fullscreen`; run `/tui` alone to see which one you're using.
- **Watch out:** fullscreen mode is a research preview, and some shortcuts behave slightly differently there.

/focus

In fullscreen mode, toggles a tidy view that shows only your last prompt, a one-line summary of Claude's actions, and the final answer.

- **When:** you want less scrolling detail.
- **Try:** `/focus`
- **Watch out:** only available in fullscreen rendering. The choice is remembered across sessions.

/scroll-speed

Adjusts how fast the mouse wheel scrolls in fullscreen mode.

- **When:** scrolling moves too slowly or too fast.
- **Try:** `/scroll-speed`
- **Watch out:** fullscreen only, and not available in JetBrains' built-in terminal.

/statusline

Sets up the **status line**, a custom bar at the bottom of the screen that can show context usage, cost, the Git branch, or anything else.

- **When:** you want to see context usage at a glance.
- **Try:** `/statusline show the model, the git branch, and context used`, or run it alone to copy your shell prompt's style.
- **Watch out:** the status line runs a small script on your machine; Claude writes it for you.

/output-style [style]

Lists **output styles** or switches to one. A style changes how Claude talks to you: **Default**, **Proactive** (gets on with routine decisions), **Concise** (shorter answers), **Explanatory** (adds short "Insight" notes explaining choices), and **Learning** (explains and leaves small pieces for you to write).

- **When:** you're learning to code and want explanations, or you want shorter replies.
- **Try:** `/output-style explanatory`, or `/output-style` to see the list.
- **Watch out:** requires a recent version of Claude Code; run `claude update` if it's missing.

/keybindings

Opens your keyboard shortcut file, `~/.claude/keybindings.json`, creating it if needed.

- **When:** you want to change or add a shortcut.
- **Try:** `/keybindings`
- **Watch out:** changes apply as soon as you save. Chapter 11 shows the file format.

`/terminal-setup`

Sets up your terminal so `Shift` + `Enter` adds a new line, in VS Code, Cursor, Alacritty, Zed, and similar terminals. In Apple Terminal it enables `Option` + `Enter` instead; in iTerm2 it turns on clipboard access so `/copy` works.

- **When:** `Shift` + `Enter` sends your message instead of adding a line.
- **Try:** `/terminal-setup`
- **Watch out:** until then, `\` followed by `Enter` or `Ctrl` + `J` adds a new line in any terminal.

`/voice [hold|tap|off]`

Turns on voice dictation so you can speak your prompts.

- **When:** you'd rather talk than type.
- **Try:** `/voice hold`, then hold `Space` to record; or `/voice tap` to tap once to start and again to stop.
- **Watch out:** requires a Claude.ai account login.

`/ide` and `/chrome`

`/ide` shows and manages the connection to your code editor (VS Code, JetBrains, and others).

`/chrome` configures **Claude in Chrome**, which lets Claude use your Chrome browser to test pages.

- **When:** diffs aren't opening in your editor, or you want Claude to click through your website.
- **Try:** `/ide`, `/chrome`
- **Watch out:** Chrome control needs the Claude in Chrome extension; start with `claude --chrome` to enable it for a session.

8.4 Models and effort

`/model [model]`

Switches the AI model and saves it as your default for new sessions.

- **When:** you want the most capable model for a hard problem, or a faster, lighter one for routine work.
- **Try:** `/model opus`, `/model sonnet`, `/model haiku`, or `/model` to open the picker. In the picker, the left and right arrows adjust effort, and `s` switches for this session only.
- **Watch out:** switching mid-conversation can ask you to confirm, because the new model can't reuse the cached conversation. `/model opusplan` uses Opus while planning and Sonnet while building. Chapter 18 compares the models.

`/effort [level|auto|status]`

Sets how much Claude thinks before acting: `low`, `medium`, `high`, `xhigh`, or `max`.

- **When:** routine edits (lower) or tricky bugs (higher).
- **Try:** `/effort high`, `/effort status` to see the current level, `/effort auto` to go back to the default, or `/effort` alone for a slider.
- **Watch out:** higher effort uses more of your plan. `max` lasts for this session only. On Opus 5.5, Sonnet 5.5, and Haiku 5.5 the default is `medium`.

`/fast [on|off]`

Turns on **fast mode**, which makes Claude Opus respond up to about 2.5 times faster at a higher cost per token.

- **When:** live debugging or rapid back-and-forth where waiting hurts.
- **Try:** `/fast on`, then `/fast off` when you're done.
- **Watch out:** it's a research preview, works only on recent Opus models (not Sonnet or Haiku), and costs more. Turn it off when speed doesn't matter.

`/advisor [model|off]`

Turns on the **advisor**, a second, usually stronger model that Claude consults at key moments, such as before committing to an approach.

- **When:** you run a lighter model day to day but want a stronger second opinion on hard decisions.
- **Try:** `/advisor opus`, or `/advisor off`
- **Watch out:** experimental, and only on the Anthropic API (not Bedrock, Google Cloud, or Foundry). Advisor calls use usage too.

8.5 Usage, limits, and plans

`/usage`

Shows how much of your plan's limits you've used, activity stats, and, on paid plans, a breakdown of what used the most. Aliases: `/cost`, and `/stats` (opens on the Stats tab).

- **When:** you want to know how close you are to a limit.
- **Try:** `/usage`
- **Watch out:** the dollar figure at the top is meant for pay-per-use Console accounts. On a subscription, watch the usage bars instead.

`/rate-limit-options`

When a usage limit stops a request, shows your options: wait and continue automatically when the limit resets, add usage credits, or upgrade.

- **When:** Claude has stopped because you hit a limit.
- **Try:** `/rate-limit-options`
- **Watch out:** requires a claude.ai subscription. By default Claude Code waits and continues by itself once the limit resets, as long as you leave the session open.

`/usage-credits`

Opens your billing settings to add **usage credits** (paid extra usage beyond your plan). On Team and Enterprise without billing access, it sends a request to your admin instead.

- **When:** you hit a limit in the middle of something urgent.
- **Try:** `/usage-credits`
- **Watch out:** credits cost real money; set a spending cap in your billing settings.

`/upgrade`

Opens the upgrade page in your browser to move to a higher plan.

- **When:** you regularly run out of usage on Pro.
- **Try:** `/upgrade`
- **Watch out:** not shown on Enterprise plans.

`/insights`

Creates a web-page report about your recent sessions on this computer: which projects you work in, how you use Claude Code, where things go wrong, and features to try.

- **When:** after a few weeks of use, to learn better habits.
- **Try:** `/insights`
- **Watch out:** generating the report uses some usage, and it isn't available in cloud sessions.

- ☐ I've run `/init` in a project and reviewed the `CLAUDE.md` it wrote
- ☐ I can open `/config` and `/permissions`
- ☐ I've tried `/model` and `/effort` and know what they change
- ☐ I check `/usage` when I'm worried about limits

Slash Commands: Tools, Reviews, Account, and Help

The last group of built-in commands connects Claude Code to other tools, checks your work before you ship it, links your session to the cloud and your other devices, manages your account, and helps when something goes wrong.

IN THIS CHAPTER

- Extensions: `/mcp`, `/agents`, `/hooks`, `/plugin`, `/skills` and related commands
- Bundled helpers: `/run`, `/verify`, `/debug`, `/deep-research` and more
- Reviewing and shipping: `/diff`, `/code-review`, `/simplify`, `/security-review`
- Cloud, devices, and sharing: `/teleport`, `/remote-control`, `/desktop`, `/artifacts`
- Account, help, and troubleshooting commands, plus commands that were removed

9.1 Tools, MCP, agents, hooks, and plugins

```
/mcp [reconnect|enable|disable ...]
```

Manages your **MCP servers**, the connectors that give Claude access to other apps such as GitHub or Notion. Opens a list showing each server's status, and handles signing in.

- **When:** you just added a server and need to sign in, or a server shows “failed.”
- **Try:** `/mcp`, `/mcp reconnect github`, `/mcp reconnect all`, or `/mcp disable notion`
- **Watch out:** disabling a server you aren't using frees context space. Adding servers happens in your shell with `claude mcp add` (Chapter 21).

/agents

Reminds you how to create and manage **subagents**: ask Claude to make one, or edit the files in `.claude/agents/` or `~/.claude/agents/`.

- **When:** you want a specialist helper, such as a code reviewer.
- **Try:** `/agents`, then ask: “Create a project subagent called test-writer that only writes tests.”
- **Watch out:** older versions opened a full menu here; current versions just point you to the easiest path. Chapter 19 covers subagents.

/hooks

Shows the **hooks** you’ve configured: your own commands that run automatically at set moments, such as after every file edit.

- **When:** you added a hook and want to confirm Claude Code sees it.
- **Try:** `/hooks`
- **Watch out:** it’s a viewer. Create hooks by editing a settings file or by asking Claude (Chapter 20).

/plugin [subcommand]

Opens the plugin manager to browse, install, enable, and disable **plugins**: bundles of skills, subagents, hooks, and MCP servers.

- **When:** you want ready-made extras without building them yourself.
- **Try:** `/plugin`, or `/plugin install commit-commands@claude-plugins-official`
- **Watch out:** a plugin runs with your permissions, and every enabled plugin adds a little to your context. Install only what you trust and use (Chapter 22).

/reload-plugins [--force]

Reloads your plugins so recent changes take effect without restarting.

- **When:** you installed or edited a plugin and it isn’t showing up.
- **Try:** `/reload-plugins`
- **Watch out:** if the reload would change which MCP tools are loaded, it warns and skips unless you add `--force`.

/skills, /reload-skills, and /skill-doctor

- `/skills` lists every skill available to you. Type to filter, press `t` to sort by size, and press `Space` to change whether Claude and the `/` menu can see a skill.
- `/reload-skills` re-scans your skill folders so a skill you just created works without restarting.

- `/skill-doctor` shows what each skill costs in context and how often it's used, so you can turn off the ones you don't need.
- **When:** you've written your own skills (Chapter 10) or installed plugins that add some.
- **Try:** `/skills`, `/reload-skills`, `/skill-doctor`
- **Watch out:** `/skill-doctor` needs a recent version of Claude Code.

`/plugin-authoring`

Loads the reference Claude uses to build a **mod**: a plugin that changes Claude Code's own interface, such as a custom panel or toast message.

- **When:** you want Claude to build an interface tweak for you.
- **Try:** "Make me a mod that shows a toast when tests fail." Claude usually loads this by itself.
- **Watch out:** it comes from a built-in plugin you can switch off in `/plugin`.

`/workflows` and `/workflow-authoring`

A **dynamic workflow** is a script Claude writes that runs many subagents at once in the background, for jobs like a codebase-wide bug sweep. `/workflows` opens the progress view to watch, pause, resume, or save workflows. `/workflow-authoring` loads the reference for writing one.

- **When:** a huge job needs more helpers than one conversation can coordinate.
- **Try:** `/workflows`
- **Watch out:** workflows can use a lot of usage. On Pro, turn them on first in `/config`.

9.2 Bundled helper skills

These are ready-made skills that come with Claude Code. Some, like `/run` and `/claude-api`, Claude may also use by itself when they fit.

COMMAND	WHAT IT DOES	TRY
<code>/run</code> (skill)	Launches your app and drives it, to see a change actually working	<code>/run</code>
<code>/verify</code> (skill)	Builds and runs your app to confirm a change does what it should	<code>/verify</code>
<code>/run-skill-generator</code> (skill)	Records how to build and launch your project, so <code>/run</code> and <code>/verify</code> get it right	<code>/run-skill-generator</code>
<code>/debug</code> [description] (skill)	Turns on debug logging and reads the log to diagnose a problem	<code>/debug</code> MCP server keeps disconnecting
<code>/deep-research</code> <question>	Searches the web widely, cross-checks sources, and writes a cited report	<code>/deep-research</code> best way to cache images on a static site
<code>/claude-api</code> [subcommand] (skill)	Loads Claude API reference material for building your own AI apps	<code>/claude-api</code> migrate
<code>/claude-in-chrome</code> [task] (skill)	Carries out a task in your Chrome browser	<code>/claude-in-chrome</code> fill in the contact form and submit it
<code>/dataviz</code> [request] (skill)	Design guidance for charts and dashboards	<code>/dataviz</code> chart monthly sales by product

TIP

`/verify` and `/deep-research` run only when you ask for them, because they can take a while and use a fair amount of usage. Run `/run-skill-generator` once per project if your app needs anything unusual to start, such as a database or a special environment file.

9.3 Reviewing and shipping

`/diff`

Shows every change in your working files, including Claude's edits so far, in an interactive viewer.

- **When:** before you commit, or whenever you want to see exactly what Claude changed.
- **Try:** `/diff`
- **Watch out:** it shows uncommitted changes in your folder, including ones you made yourself.

`/code-review [level] [--fix] [--comment] [target] (skill)`

Reviews your changes for bugs. Alias: `/review`. With no target it reviews your current branch and uncommitted changes. It runs as a background subagent, so you can keep working.

- **When:** before committing or opening a pull request.
- **Try:** `/code-review`, `/code-review high`, `/code-review --fix` to apply the fixes, `/code-review 1234` for a pull request, or `/code-review src/cart` for one folder.
- **Watch out:** the level (`low`, `medium`, `high`, `xhigh`, `max`) trades thoroughness for usage, and is remembered for next time. `--comment` posts findings on the GitHub pull request. `--fix` edits go through your normal permissions.

`/ultrareview [PR or branch]`

Runs a deep, multi-agent review in a cloud sandbox. The preferred spelling is `/code-review ultra`.

- **When:** an important pull request deserves an extra-careful review.
- **Try:** `/code-review ultra 1234`
- **Watch out:** Pro and Max include 3 free runs; after that it needs usage credits.

`/simplify [target] (skill)`

Four review agents look at your recent changes for cleanup: reusing existing helpers, simpler code, efficiency, and the right level of abstraction. Then it applies the fixes.

- **When:** the code works but feels messy.
- **Try:** `/simplify`, or `/simplify src/cart`
- **Watch out:** it doesn't look for bugs. Use `/code-review` for that.

`/security-review`

Checks the changes on your current branch for security problems, such as injection attacks, login flaws, and leaked data.

- **When:** before shipping anything that handles logins, payments, or personal data.
- **Try:** `/security-review`
- **Watch out:** it compares against your `origin` remote's default branch, so the repository needs an `origin` remote.

/install-github-app

Installs the Claude GitHub App on a repository and, optionally, sets up GitHub Actions so people can mention `@claude` in issues and pull requests.

- **When:** you want Claude's help directly on GitHub.
- **Try:** `/install-github-app`
- **Watch out:** github.com only (not GitLab or Bitbucket), needs admin access to the repository and the `gh` command-line tool. Chapter 24 walks through it.

/autofix-pr [prompt]

Starts a cloud session that watches the current branch's pull request and pushes fixes when tests fail or reviewers leave comments.

- **When:** you've opened a PR and want CI failures handled while you do other things.
- **Try:** `/autofix-pr`, or `/autofix-pr only fix lint and type errors`
- **Watch out:** needs the `gh` tool and access to cloud sessions. To watch a different PR, check out its branch first.

/schedule [description]

Creates, updates, lists, or runs **routines**: tasks that run on Anthropic's cloud on a schedule, even when your computer is off. Alias: `/routines`.

- **When:** you want something done every morning, like a summary of new issues.
- **Try:** `/schedule every weekday at 9am, summarize new GitHub issues`
- **Watch out:** routines run in the cloud against your GitHub repositories, not on your local files. For checks while a session is open, use `/loop`.

/install-slack-app

Installs the Claude Slack app through your browser.

- **When:** your team wants to hand coding tasks to Claude from Slack.
- **Try:** `/install-slack-app`
- **Watch out:** a workspace admin may need to approve the app.

9.4 Cloud, devices, and sharing

COMMAND	WHAT IT DOES	WATCH OUT
<code>/teleport</code> (also <code>/tp</code>)	Pulls a cloud session from <code>claude.ai/code</code> into this terminal, with its branch and conversation	Needs a <code>claude.ai</code> subscription
<code>/remote-control</code> (also <code>/rc</code>)	Lets you continue this local session from your phone or a browser	Your computer must stay on with Claude Code running
<code>/remote-env</code>	Chooses the default cloud environment for cloud sessions you start from here	Only matters if you use cloud sessions
<code>/web-setup</code>	Connects your GitHub account for cloud sessions using your local <code>gh</code> login	Requires the <code>gh</code> tool, signed in
<code>/desktop</code> (also <code>/app</code>)	Continues this session in the Claude desktop app	macOS or x64 Windows, with a subscription
<code>/mobile</code> (also <code>/ios</code> , <code>/android</code>)	Shows a QR code to download the Claude mobile app	—
<code>/artifacts</code>	Lists your artifacts (live, private web pages Claude publishes on <code>claude.ai</code>) to attach, open, or copy a link	Only where artifacts are available
<code>/team-onboarding</code>	Writes a setup guide for a teammate from your last 30 days of use	Review it for anything private before sharing

Design, slides, and artifact helpers

COMMAND	WHAT IT DOES
<code>/design [brief]</code> (<i>skill</i>)	Drafts UI mockups, screen flows, or posters on a Claude Design canvas you can edit in your browser
<code>/design-login</code>	Authorizes design-system access for <code>/design-sync</code>
<code>/design-sync [hint]</code> (<i>skill</i>)	Uploads your repository's React design system to Claude Design, so designs use your real components
<code>/slides [brief]</code> (<i>skill</i>)	Makes a slide deck from your brief as an artifact
<code>/artifact-capabilities</code> (<i>skill</i>)	Loads what a published artifact can do, such as live data or file downloads
<code>/artifact-diagramming</code> (<i>skill</i>)	Loads guidance for drawing clear diagrams in artifacts

These depend on artifacts being available to your account, and aren't available on Amazon Bedrock, Google Cloud, or Microsoft Foundry.

9.5 Account

`/login` and `/logout`

`/login` signs in to your Anthropic account or switches accounts; `/logout` signs out.

- **When:** you want to switch between a personal and a work account, or login seems broken.
- **Try:** `/logout`, close Claude Code, then `claude` and sign in again.
- **Watch out:** switching accounts mid-task cancels an automatic wait for a usage limit.

`/privacy-settings`

Shows and updates your privacy settings, such as whether your chats can help improve future models.

- **When:** you want to check or change how your data is used (Chapter 28).
- **Try:** `/privacy-settings`
- **Watch out:** only available on Pro and Max plans; Team and Enterprise data rules are set by your organization's agreement.

`/passes`

Shares a free week of Claude Code with friends, if your account is eligible. It only appears when you can use it.

9.6 Help and troubleshooting

`/help`

Shows help and the commands available to you.

- **Try:** `/help`. You can also just ask Claude: “how do I undo a change?”

`/doctor` (*skill*)

Runs a setup checkup: installation health, settings files, unused extensions, oversized `CLAUDE.md` files, and permission suggestions. It reports first and asks before fixing anything.

Alias: `/checkup`.

- **When:** something feels off, or Claude Code seems slow to start.
- **Try:** `/doctor`, or `/doctor prompt-audit` to check your instruction files for outdated or conflicting rules.
- **Watch out:** `claude doctor` in your shell is a quick, read-only version that runs without starting a session.

`/bug [report]` and `/feedback [report]`

`/bug` reports a bug to Anthropic; `/feedback` sends product feedback. Both let you choose how much of the session to include and ask for consent before sending. Alias for `/bug`: `/share`.

- **When:** you hit a bug or have a suggestion.
- **Try:** `/bug` the diff view doesn't open in VS Code
- **Watch out:** reports that include your conversation are kept for five years, so leave out anything sensitive.

`/release-notes`

Shows what changed in each version of Claude Code, in a version picker.

- **Try:** `/release-notes` after an update, to see what's new.

`/powerup`

Teaches Claude Code features through quick interactive lessons with animated demos.

- **Try:** `/powerup`. It's a friendly way to keep learning after this manual.

`/heapdump`

Saves a memory snapshot for diagnosing high memory use. Hidden from the menu: type it in full.

- **Watch out:** only share the small `-diagnostics.json` file it makes. The large `.heapsnapshot` file contains your whole conversation and credentials.

`/setup-bedrock` and `/setup-vertex`

Wizards that connect Claude Code to Amazon Bedrock or Google Cloud's Agent Platform instead of an Anthropic account. They stay hidden until the matching setting (`CLAUDE_CODE_USE_BEDROCK=1` or `CLAUDE_CODE_USE_VERTEX=1`) is turned on. Most beginners never need them.

9.7 Just for fun

COMMAND	WHAT IT DOES
<code>/radio</code>	Opens Claude FM lo-fi radio in your browser
<code>/stickers</code>	Order Claude Code stickers

9.8 Removed commands

If you read older tutorials, you may see commands that no longer exist:

OLD COMMAND	WHAT TO DO INSTEAD
<code>/pr-comments</code>	Ask Claude directly: "show me the comments on this pull request"
<code>/ultraplan</code>	Use plan mode (<code>/plan</code> or <code>Shift</code> + <code>Tab</code>)
<code>/vim</code>	Turn on Vim editing in <code>/config</code> → Editor mode

- ☐ I know `/mcp`, `/plugin`, and `/skills` manage my extensions
- ☐ I review changes with `/diff` and `/code-review` before committing
- ☐ I know `/doctor` is my first stop when something seems broken
- ☐ I can report a problem with `/bug`

Custom Slash Commands and Skills

The built-in commands are just the start. You can make your own: type `/fix-issue 123` and Claude follows the steps *you* wrote. In Claude Code these are called **skills**, and older-style **custom commands** still work too.

IN THIS CHAPTER

- The difference between `.claude/commands/` files and `.claude/skills/` folders
- Where to save a skill so it works in one project or everywhere
- Frontmatter: the settings at the top of a skill file
- Arguments with `$ARGUMENTS`, `$0`, `$1`, and named arguments
- Seven worked examples, plus MCP prompts as commands

10.1 Commands and skills: one feature, two formats

A **skill** is a folder containing a file called `SKILL.md`. The folder name becomes the command. A **custom command** is a single Markdown file in a `commands` folder. Both create the same kind of `/command`:

FORMAT	FILE	COMMAND YOU TYPE
Custom command (older)	<code>.claude/commands/deploy.md</code>	<code>/deploy</code>
Skill (recommended)	<code>.claude/skills/deploy/SKILL.md</code>	<code>/deploy</code>

Your existing command files keep working. Skills add two extras: a folder for supporting files (scripts, examples, reference notes), and the ability for Claude to use the skill **automatically** when your request matches its description.

NOTE

Skills follow the open **Agent Skills** standard (agentskills.io), so a simple skill can work in other AI tools as well.

10.2 Where skills live

LOCATION	PATH	WORKS IN
Personal	<code>~/.claude/skills/<name>/SKILL.md</code>	All your projects on this computer
Project	<code>.claude/skills/<name>/SKILL.md</code>	This project; commit it to share with your team
Plugin	<code><plugin>/skills/<name>/SKILL.md</code>	Wherever the plugin is on, as <code>/plugin-name:skill-name</code>

Files in subfolders of `.claude/commands/` get a prefix: `.claude/commands/frontend/component.md` becomes `/frontend:component`.

10.3 Your first skill in three minutes

CREATE /SUMMARIZE-CHANGES

1 Make the folder.

TERMINAL

```
mkdir -p ~/.claude/skills/summarize-changes
```

2 Write SKILL.md.

Save this text in that folder:

```
~/.CLAUDE/SKILLS/SUMMARIZE-CHANGES/SKILL.MD

---
description: Summarizes uncommitted changes and flags anything risky.
  Use when the user asks what changed or wants a commit message.
---

## Current changes

!`git diff HEAD`

## Instructions

Summarize the changes above in two or three bullet points, then
list any risks, such as missing error handling or hardcoded
values. If the diff is empty, say there are no changes.
```

3

Test it.

In any Git project, run `/summarize-changes`, or just ask “what did I change?” and Claude picks the skill by itself.

The line `!`git diff HEAD`` is **dynamic context**: Claude Code runs that command first and pastes its output into the skill before Claude reads it. So Claude sees your real, current changes.

TIP

If you create a skill while Claude Code is running and it doesn't appear, run `/reload-skills`.

10.4 Frontmatter: the settings block

Frontmatter is the block between `---` lines at the very top of the file. Every field is optional; `description` is the one to always write.

FIELD	WHAT IT DOES
<code>name</code>	The command name. Defaults to the folder name
<code>description</code>	What the skill does and when to use it. Claude reads this to decide when to load the skill
<code>when_to_use</code>	Extra trigger phrases or example requests
<code>argument-hint</code>	Hint shown in the menu, such as <code>[issue-number]</code>
<code>arguments</code>	Names for positional arguments, such as <code>[issue, branch]</code>
<code>disable-model-invocation</code>	<code>true</code> means only <i>you</i> can run it; Claude won't start it on its own
<code>user-invocable</code>	<code>false</code> hides it from the <code>/</code> menu; only Claude uses it
<code>allowed-tools</code>	Tools Claude may use without asking while the skill runs, such as <code>Bash(git add *)</code>
<code>disallowed-tools</code>	Tools removed while the skill runs
<code>model / effort</code>	Use a particular model or effort level for this skill
<code>context</code>	<code>fork</code> runs the skill in its own subagent
<code>agent</code>	Which subagent type to use with <code>context: fork</code>
<code>paths</code>	Only auto-load when working on matching files, such as <code>src/**/*.ts</code>
<code>hooks</code>	Hooks that switch on when the skill is used

WARNING

Field names must match exactly, hyphens included. Claude Code silently ignores a misspelled field. And the opening `---` must be the very first line of the file.

10.5 Passing arguments

Whatever you type after the command name becomes its **arguments**.

PLACEHOLDER	BECOMES	WITH <code>/MIGRATE SEARCHBAR JS TS</code>
<code>\$ARGUMENTS</code>	Everything you typed	<code>SearchBar JS TS</code>
<code>\$ARGUMENTS[0]</code> or <code>\$0</code>	The first word	<code>SearchBar</code>
<code>\$1</code> , <code>\$2</code>	The second, third word	<code>JS</code> , <code>TS</code>
<code>\$name</code>	A named argument from <code>arguments:</code>	depends on the list

- Wrap multi-word values in quotes: `/greet "Sam Lee" morning` makes `$0` = `Sam Lee`.
- If your skill has no placeholder, Claude Code adds `ARGUMENTS: ...` to the end so Claude still sees what you typed.
- To write a literal dollar amount like `$1.00`, escape it: `\$1.00`.
- Other handy values: `${CLAUDE_SKILL_DIR}` (the skill's own folder), `${CLAUDE_PROJECT_DIR}` (the project root), and `${CLAUDE_SESSION_ID}`.

10.6 Worked examples

1. Fix an issue by number

`.CLAUDE/SKILLS/FIX-ISSUE/SKILL.MD`

```

---
description: Fix a GitHub issue
argument-hint: [issue-number]
disable-model-invocation: true
---

Fix GitHub issue $ARGUMENTS following our coding standards.

1. Read the issue with `gh issue view $ARGUMENTS`
2. Understand the requirements
3. Implement the fix and write tests
4. Run the tests and create a commit

```

Run it with `/fix-issue 123`. Claude receives “Fix GitHub issue 123 following our coding standards...”.

2. Migrate a component, with positional arguments

```
.CLAUDE/SKILLS/MIGRATE-COMPONENT/SKILL.MD
```

```
---
description: Migrate a component from one language to another
argument-hint: [component] [from] [to]
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

Run `/migrate-component SearchBar JavaScript TypeScript`.

3. A safe commit command with pre-approved tools

```
.CLAUDE/SKILLS/COMMIT/SKILL.MD
```

```
---
description: Stage and commit the current changes
disable-model-invocation: true
allowed-tools: Bash(git add *) Bash(git commit *) Bash(git status *)
---

Look at the current changes, stage them, and commit with a short,
clear message in the imperative mood ("Add menu prices").
```

Because of `allowed-tools`, Claude won't ask permission for those Git commands *during this skill*. The approval ends when you send your next message.

4. A weekly update anyone can read

```
~/CLAUDE/SKILLS/WEEKLY-UPDATE/SKILL.MD
```

```
---
description: Writes a friendly weekly progress update from recent Git
  commits. Use when asked for a status update or weekly summary.
---

## Commits this week

!`git log --since="7 days ago" --oneline`

Write a short update with three bullets of what got done and one
bullet of what's next, in plain language a non-programmer understands.
```

5. A pull request summary in a separate subagent

.CLAUDE/SKILLS/PR-SUMMARY/SKILL.MD

```
---
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
---
```

- PR diff: `!`gh pr diff``
 - PR comments: `!`gh pr view --comments``

Summarize this pull request: what it changes, why, and any risks.

`context: fork` runs the skill in a fresh subagent, keeping the big diff out of your main conversation. The subagent can't see your chat, so the skill must contain everything it needs.

6. Reference knowledge Claude applies by itself

.CLAUDE/SKILLS/HOUSE-STYLE/SKILL.MD

```
---
description: Writing style for all customer-facing text on the
  bakery website. Use when writing or editing page copy.
user-invocable: false
---
```

- Warm, short sentences. No jargon.
- Prices always as "\$4.50", never "4.5".
- Call items "bakes", never "products".

With `user-invocable: false` it never appears in the menu; Claude uses it whenever you edit page text.

7. A grouped command in the older format

.CLAUDE/COMMANDS/FRONTEND/COMPONENT.MD

```
---
description: Create a new UI component
argument-hint: [ComponentName]
---
```

Create a new component called \$ARGUMENTS in `src/components/`, following the pattern of the existing components, with a test file.

Run it with `/frontend:component PriceTag`.

10.7 Stacking skills and adding supporting files

You can start a message with several skills: `/write-tests /fix-issue 123` loads both and passes `123` to each (up to six skills).

A skill folder can hold more than `SKILL.md`:

TEXT

```
my-skill/
├── SKILL.md      (required: overview and steps)
├── reference.md  (detailed notes, loaded only when needed)
├── examples.md   (sample outputs)
├── scripts/
│   └── helper.py (a script Claude runs)
```

Mention the extra files in `SKILL.md` (“For full API details see reference.md”) so Claude knows when to open them. Keep `SKILL.md` itself under about 500 lines.

10.8 Who can run a skill?

FRONTMATTER	YOU CAN RUN IT	CLAUDE CAN RUN IT ON ITS OWN
(default)	Yes	Yes
<code>disable-model-invocation: true</code>	Yes	No
<code>user-invocable: false</code>	No	Yes

Use `disable-model-invocation: true` for anything with side effects, such as deploying or sending messages. You don’t want Claude deciding to deploy because the code “looks ready.”

10.9 MCP prompts as slash commands

MCP servers (Chapter 21) can offer ready-made **prompts**, and Claude Code turns them into commands. Type `/` and look for entries marked `(MCP)`, listed as `/servername:promptname`. You can also type the long form:

TEXT

```
/mcp__github__list_prs
/mcp__github__pr_review 456
/mcp__jira__create_issue login-bug high
```

Arguments are separated by spaces, and the prompt's result is added straight into the conversation.

10.10 Bundled skills you already have

Claude Code ships with skills such as `/doctor`, `/code-review`, `/simplify`, `/batch`, `/loop`, `/debug`, `/run`, `/verify`, and `/claude-api` (Chapters 7–9). To turn bundled skills off, set `disableBundledSkills` in your settings.

10.11 When a skill misbehaves

PROBLEM	FIX
Claude doesn't use the skill	Put words you'd naturally say into the <code>description</code> ; or run it directly with <code>/name</code>
<code>/name</code> isn't recognized	Check the folder name for odd characters; run <code>/reload-skills</code> ; check <code>/skills</code>
Fields seem ignored	The YAML may be broken. Start with <code>claude --debug</code> to see the error
Claude uses it too often	Make the description narrower, or add <code>disable-model-invocation: true</code>
Claude stops following it later	Put the key rules near the top; after <code>/compact</code> , run the skill again

- ☐ I've made a personal skill in `~/.claude/skills/`
- ☐ I've used `$ARGUMENTS` or `$0` to pass information in
- ☐ I protect side-effect skills with `disable-model-invocation: true`
- ☐ I know `/skills` and `/reload-skills`

Keyboard Shortcuts and Launch Options

Keyboard shortcuts make Claude Code feel quick. You don't need to learn them all: press `?` on an empty prompt to see the list any time. This chapter also covers the options you can add when you start `claude` from the terminal.

IN THIS CHAPTER

- The everyday shortcuts and what they do
- Editing your prompt, multi-line input, and history search
- Vim mode and customizing shortcuts
- Starting Claude Code with command-line options

11.1 The shortcuts you'll use daily

KEYS	ACTION
<code>Esc</code>	Stop Claude mid-task, or close a dialog
<code>Esc</code> <code>Esc</code>	Clear your draft; on an empty prompt, open rewind
<code>Shift</code> + <code>Tab</code>	Cycle permission modes
<code>Shift</code> + <code>Enter</code>	New line without sending
<code>↑</code> / <code>↓</code>	Scroll through previous prompts
<code>Ctrl</code> + <code>R</code>	Search prompt history
<code>Ctrl</code> + <code>C</code>	Interrupt; on an empty prompt, press twice to exit
<code>Ctrl</code> + <code>D</code>	Exit (press twice)
<code>Ctrl</code> + <code>O</code>	Show the detailed transcript of what Claude did

Ctrl + T	Show or hide Claude's task checklist
Ctrl + G	Edit your prompt (or a plan) in a text editor
Ctrl + V	Paste an image (Cmd + V in iTerm2, Alt + V on Windows)
Ctrl + B	Send a long-running command to the background
Ctrl + S	Stash your draft; press again on an empty prompt to bring it back
Ctrl + L	Redraw a garbled screen
?	Show the shortcut help panel (on an empty prompt)

11.2 Model and mode shortcuts

MAC	WINDOWS / LINUX	ACTION
Option + P	Alt + P	Switch model without clearing your prompt
Option + T	Alt + T	Toggle extended thinking (older models only)
Option + O	Alt + O	Toggle fast mode

NOTE

On a Mac, most **Option** shortcuts only work after you set Option to act as “Meta” in your terminal’s settings. `/terminal-setup` helps with this in Apple Terminal.

11.3 Editing your prompt

KEYS	ACTION
Ctrl + A / Ctrl + E	Jump to the start / end of the line
Ctrl + K	Delete to the end of the line
Ctrl + U	Delete back to the start of the line
Ctrl + W	Delete back to the previous space (a whole path at once)
Ctrl + Y	Paste back what you just deleted
Alt + B / Alt + F	Move back / forward one word
Ctrl + _	Undo your last edit to the prompt

Multi-line prompts

METHOD	WORKS IN
<code>\</code> then <code>Enter</code>	Every terminal
<code>Ctrl</code> + <code>J</code>	Every terminal
<code>Shift</code> + <code>Enter</code>	iTerm2, WezTerm, Ghostty, Kitty, Warp, Apple Terminal, Windows Terminal; others after <code>/terminal-setup</code>
<code>Option</code> + <code>Enter</code>	Mac, after enabling Option as Meta
Paste	Paste a block of text or logs directly

11.4 Quick prefixes

TYPE	MEANING
<code>/</code> at the start	Command or skill
<code>!</code> at the start	Shell mode: run a command yourself
<code>@</code>	File path autocomplete
<code>:</code>	Emoji shortcode, such as <code>:tada:</code>
<code>?</code> on an empty prompt	Shortcut help

11.5 Prompt history and search

Your prompts are remembered per folder, even across sessions, so `↑` can bring back yesterday's request. Press `Ctrl` + `R` and type a few letters to search older prompts; press `Ctrl` + `R` again for older matches, `Tab` to edit the match, or `Enter` to run it.

11.6 Shortcuts in the transcript viewer

After `Ctrl` + `O`: `q` or `Esc` closes it. In fullscreen mode, `{` and `}` jump between your prompts, `v` opens the conversation in your editor, and `?` lists more keys.

11.7 Vim mode

If you know the Vim editor, turn on Vim-style editing in `/config` → **Editor mode**. `Esc` enters NORMAL mode and `i`, `a`, `o` return to typing. To map `jj` to Escape, add this to your settings:

```
~/CLAUDE/SETTINGS.JSON

{
  "editorMode": "vim",
  "vimInsertModeRemaps": { "jj": "<Esc>" }
}
```

11.8 Customize your shortcuts

Run `/keybindings` to open `~/.claude/keybindings.json`. Each block names a **context** (where the keys apply, such as `Chat` or `Global`) and maps keys to actions. Setting a key to `null` unbinds it. Changes apply as soon as you save.

```
~/CLAUDE/KEYBINDINGS.JSON

{
  "$schema": "https://www.schemastore.org/claude-code-keybindings.json",
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+e": "chat:externalEditor",
        "ctrl+s": null
      }
    }
  ]
}
```

11.9 Starting Claude Code from the terminal

COMMAND	WHAT IT DOES
<code>claude</code>	Start an interactive session
<code>claude "fix the build error"</code>	Start with a first request
<code>claude -c</code>	Continue the most recent conversation here
<code>claude -r</code>	Choose an older conversation to resume
<code>claude -r "auth-refactor"</code>	Resume a session by name
<code>claude -n checkout-bug</code>	Start a session with a name

<code>claude -p "summarize README.md"</code>	Answer once and exit (handy in scripts)
<code>claude --model opus</code>	Use a specific model this session
<code>claude --effort high</code>	Use a specific effort level this session
<code>claude --permission-mode plan</code>	Start in a specific permission mode
<code>claude -w new-feature</code>	Work in an isolated copy of the repo (a Git <i>worktree</i>)
<code>claude --add-dir ../shared</code>	Give access to an extra folder
<code>claude --chrome</code>	Enable Chrome browser control
<code>claude --safe-mode</code>	Start with all customizations off, to troubleshoot
<code>claude --debug</code>	Write a detailed debug log
<code>claude update</code>	Update now
<code>claude doctor</code>	Check installation health
<code>claude agents</code>	Open agent view to manage background sessions
<code>claude -v</code>	Print the version

- ☐ I've pressed `?` to see the shortcut panel
- ☐ I can write a multi-line prompt
- ☐ I've searched my history with `Ctrl` + `R`
- ☐ I know how to start Claude Code with a model, a mode, or a name

Plan Mode in Depth

Plan mode is Claude Code’s “measure twice, cut once” setting. Claude reads, explores, and writes a plan, but doesn’t change your files until you approve it. It’s the single most useful habit for beginners.

IN THIS CHAPTER

- How to enter and leave plan mode
- What Claude can and can’t do while planning
- Approving, rejecting, and editing a plan
- Making plan mode your default, and when to skip it

12.1 Entering and leaving plan mode

WAY IN	HOW
Keyboard	Press <code>Shift</code> + <code>Tab</code> until the status bar shows <code>plan mode on</code>
One prompt	Start your message with <code>/plan</code> , such as <code>/plan add a newsletter signup</code>
At launch	<code>claude --permission-mode plan</code>

Press `Shift` + `Tab` again to leave plan mode without approving anything.

12.2 What happens while planning

- Claude **reads files** and **runs commands to explore**, such as listing folders or running tests.
- Claude **does not edit your source files**. Edits stay blocked until you approve a plan.
- For research, Claude often sends a built-in **Plan subagent** to read the code, keeping all that reading out of your main conversation.
- When auto mode is available, a safety classifier reviews exploratory shell commands instead of asking you about each one; otherwise anything beyond simple read-only commands asks for permission.

12.3 Reviewing the plan

When the plan is ready, Claude shows it and asks how to proceed:

CHOICE	WHAT HAPPENS
Yes, and use auto mode	Approve; Claude builds with background safety checks instead of prompts (shown as Yes, auto-accept edits if auto mode isn't available)
Yes, manually approve edits	Approve; you confirm each edit
No, keep planning	Stay in plan mode and tell Claude what to change

Press **Ctrl** + **G** to open the plan in your text editor and change it directly before approving. Approving also gives the session a title based on the plan, unless you named it already. The plan is saved to disk, so it survives compaction.

A COMPLETE PLANNING SESSION

1 Enter plan mode with **Shift** + **Tab**.

2 Ask for understanding first.

PROMPT

read the checkout code and explain how the order total is calculated. don't plan anything yet.

3 Ask for the plan.

PROMPT

I want to add a 10% discount code called WELCOME10. write a step-by-step plan, including how you'll test it.

4

Push back where needed.

“No, keep planning: the discount must not apply to gift cards.”

5

Approve with Yes, manually approve edits the first few times, so you see each change.

12.4 Plan mode as your default

To start every session in a project in plan mode, add this to the project's `.claude/settings.json`:

```
.CLAUDE/SETTINGS.JSON
```

```
{  
  "permissions": { "defaultMode": "plan" }  
}
```

In the VS Code extension, set `claudeCode.initialPermissionMode` to `plan` in VS Code's settings instead.

TIP

Try the `opusplan` model setting (`/model opusplan`): it uses the more capable Opus model while planning and switches to Sonnet to build, a good balance of quality and usage.

12.5 When to skip planning

Planning adds a step. Skip it when you could describe the change in one sentence: fixing a typo, renaming a variable, adding a log line. Plan when you're unsure of the approach, the change touches several files, or you don't know the code well.

- ☐ I can enter and leave plan mode with `Shift` + `Tab`
- ☐ I've edited a plan with `Ctrl` + `G`
- ☐ I know the three approval choices

Checkpoints and Rewind

Claude Code quietly saves the state of your files before every prompt. If something goes wrong, you can rewind: the code, the conversation, or both. This chapter shows how, and where the limits are.

IN THIS CHAPTER

- How checkpoints are taken
- The five rewind options and when to use each
- Summarizing part of a conversation
- What checkpoints can't undo, and why Git still matters

13.1 How checkpoints work

- Every prompt you send that starts a turn creates a **checkpoint** of the files Claude edits with its file tools.
- The last 100 checkpoints in a session keep their file snapshots.
- Checkpoints are saved with the conversation, so `/rewind` still works after you resume a session.
- Snapshots are cleaned up after about 30 days by default.

13.2 Opening the rewind menu

Press `Esc` twice on an empty prompt, or type `/rewind` (aliases `/checkpoint` and `/undo`). You'll see the list of prompts you sent. Pick one, then choose:

OPTION	WHAT IT DOES	USE IT WHEN
Restore code and conversation	Both go back to that point	Claude went down the wrong path entirely
Restore conversation	Rewinds the chat; files stay as they are	You like the code but want to rephrase your next request
Restore code	Reverts files; keeps the chat	The edits broke something but the discussion was useful
Summarize from here	Compresses everything after this point	A long detour you no longer need in detail
Summarize up to here	Compresses everything before this point	Long setup chat; recent work matters most
Never mind	Goes back without changes	You changed your mind

The code options only appear if files changed after that point. After restoring the conversation, your original prompt reappears in the input box, ready to edit and resend.

 TIP

To steer a summary, highlight a **Summarize** option and type instructions where it says **add context (optional)**, such as “keep the list of failing tests.”

13.3 Getting back a cleared conversation

If you ran `/clear` earlier in the same Claude Code window, the top of the rewind menu shows an entry labeled `(previous session)`. Pick it to return to the conversation you cleared.

13.4 What checkpoints can't undo

NOT TRACKED	WHY	WHAT TO DO
Changes made by shell commands (<code>rm</code> , <code>mv</code> , <code>cp</code> , scripts)	Only Claude's file-editing tools are tracked	Commit to Git before big changes
Edits you make in your own editor	Claude Code didn't make them	Use Git
Most edits by background subagents	They run outside your session's checkpoints	Use Git to revert
Symlinked or hard-linked files	Skipped on restore, with a warning	Ask Claude to reverse the edit
Anything outside your computer	Emails, deploys, databases, cloud files	Be careful; use deny rules for risky commands

 IMPORTANT

Checkpoints are a quick “undo” for one session. **Git** is your real safety net: permanent history you can always return to. Ask Claude to “commit everything with a clear message” before any big change.

13.5 Trying an alternative without losing anything

To explore a different approach while keeping the original intact, use `/branch` (Chapter 7) instead of rewinding. You can return to the original with `/resume`.

- ☐ I’ve opened the rewind menu with `Esc` `Esc`
- ☐ I know the difference between restoring code and restoring the conversation
- ☐ I commit to Git before big or risky changes

Permissions and Safety

Claude Code can edit files and run commands on your computer, so it comes with a permission system: rules for what it may do without asking. Understanding it lets you work faster *and* safer.

IN THIS CHAPTER

- The six permission modes, in depth, and how to switch
- Allow, ask, and deny rules, and how to write them
- Protected paths and actions no mode approves
- Safety nets: checkpoints, Git, and the sandbox

14.1 Permission modes

A **permission mode** sets how much Claude can do before checking with you. Press `Shift` + `Tab` to cycle modes during a session. The status bar shows the active one.

MODE	RUNS WITHOUT ASKING	GOOD FOR
Manual (<code>default</code>)	Reading files only	Sensitive work; learning what Claude does
Accept edits (<code>acceptEdits</code>)	Reading, editing files, and simple file commands like <code>mkdir</code> , <code>mv</code> , <code>cp</code>	Iterating on code you'll review afterwards
Plan (<code>plan</code>)	Reading and exploring; no edits until you approve a plan	Thinking before building
Auto (<code>auto</code>)	Almost everything, with a safety classifier checking each action in the background	Long tasks with few interruptions
Don't ask (<code>dontAsk</code>)	Only pre-approved tools; everything else is refused	Scripts and automation
Bypass (<code>bypassPermissions</code>)	Everything, no checks	Disposable containers and virtual machines only

Auto mode is the starting mode for new terminal and VS Code sessions on recent versions of Claude Code. A second AI model (the **classifier**) reviews each risky action, such as running a script from the internet, force-pushing code, deploying, or deleting files that existed before the session, and blocks anything that goes beyond what you asked for.

☆ **IMPORTANT**

You can tell auto mode about limits in plain English: “don’t push anything until I’ve reviewed it.” The classifier treats that as a rule for the rest of the conversation. For a hard guarantee, add a **deny rule** (below).

To start a session in a particular mode:

TERMINAL

```
claude --permission-mode plan
```

The CLI also accepts `manual` as another name for `default`.

Each mode in more detail

Manual stops before most edits, shell commands, and network requests. Choose it when you want to see and approve every step, for example in an unfamiliar project.

Accept edits lets Claude create and edit files inside your working folder and run common file commands (`mkdir`, `touch`, `rm`, `rmdir`, `mv`, `cp`, `sed`) there. Other commands still ask. Review the result afterwards with `/diff`.

Plan is covered in Chapter 12.

Auto needs a supported model (recent Opus, Sonnet, Haiku 5.5, or Fable) and can be turned off by Team and Enterprise admins. By default the classifier only trusts your working folder and your repository’s remotes; `/auto-mode-setup` describes the rest of your trusted setup. Auto mode reduces prompts but doesn’t guarantee safety.

Don’t ask never shows a prompt: anything not pre-approved is simply refused. It’s meant for scripts, and it never appears in the `Shift` + `Tab` cycle.

Bypass only joins the cycle if you start with `--dangerously-skip-permissions` or a similar flag. Deny rules still apply even here.

⚠ **WARNING**

If you set `"defaultMode": "auto"` in a project’s `.claude/settings.json`, it won’t take effect. Put it in your personal `~/.claude/settings.json` instead.

14.2 Allow, ask, and deny rules

Rules let you pre-approve safe actions and forbid dangerous ones. Type `/permissions` to see and edit them, or write them in a settings file.

- **Allow:** run without asking.
- **Ask:** always ask first, even in auto mode.
- **Deny:** never allowed, in any mode.

Deny rules win over ask rules, which win over allow rules.

```
.CLAUDE/SETTINGS.JSON

{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)"
    ],
    "ask": [
      "Bash(git push *)"
    ],
    "deny": [
      "Read(./env)",
      "Bash(rm -rf *)"
    ]
  }
}
```

The `*` matches anything, so `Bash(npm run *)` covers `npm run build`, `npm run test`, and so on. Put the `*` *after* the sub-command: `Bash(git commit *)` is narrow, but `Bash(git *)` allows every Git command.

TIP

Tired of approving the same harmless commands? Type `/fewer-permission-prompts`. Claude scans your recent sessions and proposes a safe allow-list for read-only commands you use often.

Rule syntax cheat sheet

RULE	MATCHES
<code>Bash</code>	Every shell command
<code>Bash(npm test)</code>	Exactly <code>npm test</code>
<code>Bash(git diff *)</code>	Any command starting <code>git diff</code> (the space matters)

<code>Read(/.env)</code>	Reading <code>.env</code> in the current folder
<code>Read(/secrets/**)</code>	Reading anything inside <code>secrets/</code>
<code>Edit(/src/**/*.ts)</code>	Editing TypeScript files under the project's <code>src/</code>
<code>Read(~/Documents/*.pdf)</code>	PDFs in your Documents folder
<code>WebFetch(domain:example.com)</code>	Fetching pages from example.com
<code>mcp__github</code>	Every tool from the <code>github</code> MCP server
<code>mcp__github__*</code>	Same, written as a pattern
<code>Agent(Explore)</code>	The built-in Explore subagent

Path prefixes in `Read` and `Edit` rules: `./` or no prefix means relative to the current folder, `/` means relative to where the settings file applies (the project), `~/` means your home folder, and `//` means an absolute path from the root of the disk.

NOTE

File rules are checked for `Read` and `Edit` only. A `Read` deny rule also blocks images you paste and folders you @-mention from that path.

14.3 Protected paths and actions no mode approves

Some writes are never auto-approved, except in bypass mode: changes to your Git internals (`.git`), your shell setup files (like `.bashrc`), and Claude Code's own `.claude` settings. Allow rules don't override this.

In every mode, including bypass, Claude Code still asks or refuses for:

- tools matched by an **ask** rule;
- the `AskUserQuestion` tool and other tools that need a human answer;
- `rm` and `rmdir` targeting **critical paths**, such as your home folder or the root of the disk.

14.4 Where settings live

Claude Code reads settings from several files. When the same setting appears in more than one place, the higher one in this list wins:

PRIORITY	FILE	APPLIES TO
1 (highest)	Managed settings from your organization	Everyone in your company
2	<code>claude --settings ...</code> on the command line	This session
3	<code>.claude/settings.local.json</code>	You, in this project (not shared)
4	<code>.claude/settings.json</code>	Everyone working in this project
5	<code>~/.claude/settings.json</code>	You, in every project

`~` means your home folder. A file inside `.claude/` lives in the project folder. Chapter 15 covers settings files in depth.

14.5 Safety nets

- **Checkpoints:** `/rewind` (or `Esc Esc`) restores earlier file versions.
- **Git:** commit your work before big changes so you can always get back. Ask Claude: “commit everything with a clear message before you start.”
- **Protected files:** Claude never silently edits things like `.git`, `.bashrc`, or its own `.claude` settings. It always asks, except in bypass mode.
- **Sandbox:** type `/sandbox` (macOS, Linux, WSL2) to run Claude’s shell commands in an isolated box with limited file and network access.

⚠ BYPASS MODE IS FOR THROWAWAY MACHINES

`--dangerously-skip-permissions` turns off every check. A bad instruction hidden in a web page or file (a **prompt injection**) could then make Claude do real damage. Only use it inside a container or virtual machine you’re happy to wipe. The *Maximum Autonomy* guide in this series covers how to set that up.

14.6 A sensible starter setup

~/.CLAUDE/SETTINGS.JSON

```
{
  "permissions": {
    "allow": ["Bash(npm run test *)", "Bash(git status *)"],
    "ask": ["Bash(git push *)"],
    "deny": ["Read(./env)", "Read(./env.*)", "Read(./secrets/**)"]
  }
}
```

- ☐ I’ve added an allow rule for a safe command I run often
- ☐ My `.env` and secret files are protected by deny rules

Settings Files

Almost everything about Claude Code can be configured: permissions, hooks, the default model, environment variables, and more. Settings live in plain JSON files. This chapter shows where they are, how they combine, and how to change them safely.

IN THIS CHAPTER

- The four settings files and who each one affects
- Three ways to change a setting
- How precedence works, and why lists merge
- Common settings you'll actually use

15.1 The four files

SCOPE	FILE	WHO IT AFFECTS	USE IT FOR
User	<code>~/.claude/settings.json</code>	You, in every project	Theme, default model, personal rules
Shared project	<code>.claude/settings.json</code>	Everyone in the project (commit it)	Team permissions, hooks, plugins
Project local	<code>.claude/settings.local.json</code>	You, in this project only	Personal overrides; kept out of Git
Managed	Deployed by your organization	Everyone it's deployed to	Company security policy

On Windows, `~/.claude` means `%USERPROFILE%\ .claude`. Installing Claude Code doesn't create any of these files; Claude Code creates them when you first change something, or you create them yourself.

NOTE

A fifth file, `~/.claude.json`, holds your sign-in, MCP servers, and per-project state such as folder trust. Claude Code manages it; you rarely need to touch it.

15.2 Three ways to change a setting

1. **The menu.** Type `/config` for common personal options (theme, editor mode, verbose output, and more). Or set one directly: `/config verbose=true`.
2. **Ask Claude.** `/update-config` with a plain-English request, or simply “add an allow rule for npm test to my project settings.”
3. **Edit the file.** Open it in any editor:

~/.CLAUDE/SETTINGS.JSON

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "model": "sonnet",
  "autoUpdatesChannel": "stable",
  "permissions": {
    "allow": ["Bash(npm run lint)", "Bash(npm run test *)"],
    "deny": ["Read(./env)", "Read(./env.*)"]
  }
}
```

The `$schema` line gives you autocomplete and error-checking in editors like VS Code. After saving, run `/status` to confirm the file loaded.

WARNING

Settings files are strict JSON. A `//` comment or a trailing comma after the last item is an error, and Claude Code reports a **Settings Error** at the next start. `/doctor` can help fix it.

For one session only, pass settings on the command line: `claude --settings ./my-settings.json`.

15.3 Precedence: who wins

When the same key appears in several places, the highest level wins:

1. Managed settings (your organization)
2. Command line (`--settings`)
3. Project local (`.claude/settings.local.json`)
4. Shared project (`.claude/settings.json`)

5. User (~/.claude/settings.json)

Lists merge instead of replacing. If your user file allows `Bash(npm test)` and the project file allows `Bash(git status *)`, both apply.

15.4 Settings you'll use most

KEY	EXAMPLE	WHAT IT DOES
<code>model</code>	<code>"sonnet"</code>	Default model for new sessions
<code>permissions</code>	<code>{ "allow": [...], "deny": [...] }</code>	Allow, ask, and deny rules (Chapter 14)
<code>permissions.defaultMode</code>	<code>"plan"</code>	Starting permission mode
<code>env</code>	<code>{ "MY_VAR": "1" }</code>	Environment variables for every session
<code>hooks</code>	<code>{ "PostToolUse": [...] }</code>	Automatic actions (Chapter 20)
<code>autoUpdatesChannel</code>	<code>"stable"</code>	Update channel: <code>latest</code> or <code>stable</code>
<code>autoMemoryEnabled</code>	<code>false</code>	Turn auto memory off
<code>cleanupPeriodDays</code>	<code>60</code>	How long local transcripts are kept
<code>editorMode</code>	<code>"vim"</code>	Vim-style prompt editing
<code>autoContinueAtUsageLimit</code>	<code>false</code>	Don't auto-continue after a limit resets
<code>disableBundledSkills</code>	<code>true</code>	Hide the bundled skills
<code>respondToBashCommands</code>	<code>false</code>	Don't reply automatically to <code>!</code> command output

- ☐ I know which settings file to edit for me versus my team
- ☐ I've changed a setting with `/config`
- ☐ I check `/status` after editing a settings file

Memory: Teaching Claude Your Project

Every session starts with a blank slate. Two features carry knowledge from one session to the next: **CLAUDE.md files**, which you write, and **auto memory**, which Claude writes.

IN THIS CHAPTER

- What a CLAUDE.md file is and where to put it
- Generating one with `/init`, and what belongs in it
- Imports, rules files, and AGENTS.md
- How auto memory works, and fixing memory problems

16.1 CLAUDE.md files

A `CLAUDE.md` file is a plain-text note, written in **Markdown** (a simple way to format text with `#` headings and `-` bullets), that Claude reads at the start of every session. Use it for anything you'd otherwise re-explain each time: how to run the project, house style, "always do X" rules.

WHERE	FILE	WHO IT'S FOR
Your whole computer	<code>~/.claude/CLAUDE.md</code>	Your personal preferences, every project
A project (shared)	<code>./CLAUDE.md</code> or <code>./.claude/CLAUDE.md</code>	Everyone on the project, checked into Git
A project (just you)	<code>./CLAUDE.local.md</code>	Your private notes; add it to <code>.gitignore</code>
Your organization	A managed file set up by IT	Company-wide standards

How they load

- Claude Code loads `CLAUDE.md` and `CLAUDE.local.md` from your current folder **and every folder above it**, and combines them. Instructions closer to where you started are read last.
- A `CLAUDE.md` inside a **subfolder** loads only when Claude starts working with files in that subfolder.
- HTML comments (`<!-- like this -->`) are stripped out, so you can leave notes for humans that cost no context.
- The project-root `CLAUDE.md` is re-read after compaction, so its rules survive long sessions.

16.2 Create one with /init

Inside a project, type:

PROMPT

```
/init
```

Claude studies the project and writes a starter `CLAUDE.md` with build commands, test instructions, and conventions it finds. If one already exists, it suggests improvements instead of overwriting.

16.3 What to put in it

CLAUDE.MD

```
# Rise & Shine bakery website

## How to preview
- Open index.html in a browser. No build step.

## Style
- Plain HTML and CSS only. No frameworks.
- Colors: warm cream (#FFF6E9) and cinnamon (#B5562B).
- Write friendly, short copy. No jargon.

## Rules
- Keep every page working on phones.
- Ask before deleting any image.
```

Tips from Anthropic's guidance:

- **Keep it short:** aim for under about 200 lines. Long files get followed less reliably.
- **Be concrete:** “Use 2-space indentation” beats “format code nicely.”
- **Add a line when Claude makes the same mistake twice.**

- **Import other files** with `@`, e.g. `See @README.md for an overview.`

What to leave out

Skip anything Claude can work out by reading the code, such as a list of every folder. Move long, step-by-step procedures into a **skill** (Chapter 10), which only loads when needed. `/doctor` can suggest what to trim.

NOTE

CLAUDE.md is *guidance*, not a lock. To truly block an action, use a deny rule (Chapter 14) or a hook (Chapter 20).

16.4 Imports

Write `@path/to/file` anywhere in a `CLAUDE.md` to pull that file in:

CLAUDE.MD

See @README.md for the project overview and @package.json for scripts.

Git workflow

- @docs/git-instructions.md

- Relative paths are relative to the file doing the importing.
- Imports can nest up to four levels deep.
- To mention a path *without* importing it, wrap it in backticks: ``@README.md``.
- The first time a project imports a file from outside the project, Claude Code asks you to approve it.
- Imports help organization but don't save space: imported files load at the start too.

16.5 Rules files: `.claude/rules/`

For bigger projects, split instructions into topic files in `.claude/rules/`, such as `testing.md` and `api-design.md`. A rule without frontmatter loads every session. Add `paths` to load a rule only when Claude works on matching files:

```
.CLAUDE/RULES/API.MD

---
paths:
  - "src/api/**/*.ts"
---

- Every API endpoint validates its input.
- Use the standard error response format.
```

You can also keep personal rules for every project in `~/.claude/rules/`.

16.6 AGENTS.md

Some projects use an `AGENTS.md` file for other AI coding tools. Claude Code reads it as project instructions when there's no `CLAUDE.md` in the folder or above it. If you have both, Claude reads only `CLAUDE.md`; to share one file, put `@AGENTS.md` inside your `CLAUDE.md`.

16.7 Auto memory

As you work, Claude can note your corrections and preferences (“Sam prefers short commit messages”) in its own memory files, which load automatically next time. Type `/memory` to view or edit them, open your `CLAUDE.md` files, or turn auto memory on or off.

- **Where it lives:** `~/.claude/projects/<project>/memory/`, with an index file called `MEMORY.md` and one small file per memory. All worktrees of a repository share one memory folder. It stays on your computer.
- **What loads:** the first 200 lines (or 25 KB) of `MEMORY.md` each session; topic files are read only when needed.
- **What Claude saves:** your role and preferences, corrections you give, project decisions it can't see in the code, and where to find outside information. It skips what's already in the code or your `CLAUDE.md`.
- **When it saves:** you'll see messages like “Saved 2 memories.” It's all plain Markdown you can read, edit, or delete.
- **Turning it off:** use the toggle in `/memory`, set `"autoMemoryEnabled": false` in a project's settings, or set the environment variable `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`.

16.8 Fixing memory problems

PROBLEM	FIX
Claude ignores my <code>CLAUDE.md</code>	Run <code>/context</code> and check it's listed under memory files; make the instruction more specific; remove conflicting rules
A rule disappeared after <code>/compact</code>	It was only in chat, or in a subfolder file that hasn't reloaded. Move it to the project-root <code>CLAUDE.md</code>
My <code>CLAUDE.md</code> is huge	Trim it, move procedures into skills, use path-scoped rules; run <code>/doctor</code>
I don't know what auto memory saved	<code>/memory</code> → open the auto memory folder
My <code>AGENTS.md</code> isn't read	There's a <code>CLAUDE.md</code> above it; import <code>@AGENTS.md</code> from that file
It must happen <i>every</i> time	Write a hook instead (Chapter 20)

- ☐ My project has a short, concrete `CLAUDE.md`
- ☐ I've added a rule after Claude repeated a mistake
- ☐ I've looked inside my auto memory with `/memory`

Sessions and Parallel Work

A **session** is one conversation with Claude Code, saved so you can come back to it. Once you're comfortable with a single session, you can run several at once, each on its own task.

IN THIS CHAPTER

- Naming, resuming, branching, and exporting sessions
- Git worktrees: separate copies for parallel work
- Background sessions and agent view
- Cloud sessions for work that runs while your laptop is closed

17.1 Managing sessions

TO...	INSIDE CLAUDE CODE	FROM THE TERMINAL
Name a session	<code>/rename checkout-bug</code>	<code>claude -n checkout-bug</code>
Continue the latest one here	—	<code>claude -c</code>
Pick from a list	<code>/resume</code>	<code>claude -r</code>
Resume by name	<code>/resume checkout-bug</code>	<code>claude -r checkout-bug</code>
Try a different direction	<code>/branch</code>	<code>claude -c --fork-session</code>
Save a transcript	<code>/export notes.txt</code>	—
Find sessions linked to a pull request	—	<code>claude --from-pr 123</code>

Local transcripts are kept for 30 days by default; change that with the `cleanupPeriodDays` setting.

17.2 Worktrees: one folder per task

Two sessions editing the same files at once will trip over each other. A Git **worktree** is a second working copy of the same repository, on its own branch, in its own folder. Claude Code makes them for you:

TERMINAL

```
claude -w new-menu
```

This starts Claude in `<repo>/.claude/worktrees/new-menu`, on its own branch. Open another terminal and run `claude -w fix-footer` for a second, independent task. When you're done, ask Claude to merge or open a pull request for each branch.

17.3 Background sessions and agent view

A **background session** keeps working without a terminal attached:

- Inside a session, `/background` detaches it; `/fork` copies it into a new background session.
- From your shell, `claude --bg "fix the flaky test"` starts one directly.
- `claude agents` opens **agent view**: one screen listing every background session, what it's doing, and which ones need your input. Select a row to attach; press `←` to step back out.
- `claude attach <id>`, `claude logs <id>`, and `claude stop <id>` manage sessions from the shell.

NOTE

Background sessions still follow your permission mode, so one may pause waiting for an approval. Agent view shows which.

17.4 Cloud sessions

A **cloud session** runs on Anthropic's computers against a copy of your GitHub repository, so it keeps going when your laptop is closed. Start one at claude.ai/code, from the mobile app, or from the terminal:

TERMINAL

```
claude --cloud "add input validation to the signup form"
```

When it's done, Claude pushes a branch for you to review. Bring a cloud session back to your terminal with `/teleport` or `claude --teleport`. Chapter 25 covers this in more detail.

17.5 Choosing the right way to run in parallel

YOU WANT	USE
A side job that reports back into this chat	<code>/subtask</code> or a subagent
A second attempt you can compare	<code>/fork</code> or <code>/branch</code>
Two tasks editing different code at once	<code>claude -w <name></code> in two terminals
Many similar changes across the codebase	<code>/batch</code>
Work that continues with your laptop closed	A cloud session (<code>claude --cloud</code>)

TIP

Start small: one main session plus one worktree is plenty for most people. Every extra session uses your plan's usage too.

- ☐ I name sessions I'll come back to
- ☐ I use `claude -w` when I want two tasks at once

Models, Effort, and Speed

Claude Code can use several Claude models. Picking the right one, and the right amount of thinking, saves usage and gets better results.

IN THIS CHAPTER

- The model aliases and what each is for
- Effort levels, `ultrathink`, and fast mode
- The `opusplan` hybrid and the advisor
- Extended context

18.1 The models

Use an **alias** (a short nickname) instead of a full version number. Aliases always point to the latest version for your account.

ALIAS	MODEL ON THE ANTHROPIC API	BEST FOR
<code>opus</code>	Opus 5.5	Complex reasoning, architecture, hard bugs
<code>sonnet</code>	Sonnet 5.5	Most everyday coding
<code>haiku</code>	Haiku 5.5	Quick, simple tasks; lowest usage
<code>fable</code>	Fable 5.1	The hardest, longest tasks, where available
<code>best</code>	Fable where available, otherwise Opus	"Give me the strongest"
<code>opusplan</code>	Opus in plan mode, Sonnet otherwise	Good plans at a lower cost
<code>default</code>	Your account's default	Undo any override

Through Amazon Bedrock, Google Cloud, or Microsoft Foundry, the aliases can point to earlier versions.

Switching models

- `/model sonnet` switches and saves it as your default; `/model` alone opens a picker (press `s` to switch for this session only).
- `claude --model opus` sets the model for one session.
- `Option / Alt + P` switches without clearing your prompt.
- To pin an exact version, use its full name, such as `claude-opus-5-5`.

TIP

A sensible pattern: Sonnet for daily work, Opus (or `opusplan`) for planning and tricky problems, Haiku for simple, repetitive jobs. Switching to Opus also affects subagents that inherit your model.

18.2 Effort levels

Effort controls how much Claude thinks before acting. Lower is faster and uses less of your plan; higher is more thorough.

LEVEL	USE IT FOR
<code>low</code>	Quick exchanges you review each time: brainstorming, renames
<code>medium</code>	Day-to-day work with a clear scope. The default on Opus 5.5, Sonnet 5.5, and Haiku 5.5
<code>high</code>	Bug fixes where edge cases matter
<code>xhigh</code>	Deeper reasoning at higher cost
<code>max</code>	Hard problems you want worked through without you; this session only

Set it with `/effort high`, the arrow keys in `/model`, or `claude --effort high`. For a single hard question, put `ultrathink` in your prompt instead of changing the session.

18.3 Fast mode

`/fast on` makes recent Opus models respond up to about 2.5 times faster at a higher price per token. It's the same model with a speed-focused configuration, a research preview. Use it for live debugging, then `/fast off`.

18.4 The advisor

`/advisor opus` lets Claude consult a stronger second model at key moments, such as before committing to an approach. It's experimental and only on the Anthropic API.

18.5 Extended context

Current models such as Opus 5.5, Sonnet 5.5, and Haiku 5.5 run with a 1-million-token context window by default. For some older models, a `[1m]` suffix (such as `sonnet[1m]`) selects the larger window, which may need usage credits depending on your plan.

- ☐ I've switched models with `/model`
- ☐ I've tried a different effort level
- ☐ I know `ultrathink` asks for deep thinking on one turn

Subagents: Specialist Helpers

A **subagent** is a helper Claude can hand work to. Each has its own instructions, its own allowed tools, and its own fresh context, so a big research job doesn't clutter your main conversation.

IN THIS CHAPTER

- The built-in subagents
- Creating your own, by asking or by writing a file
- The settings a subagent file can hold
- Calling a subagent on purpose, and three example designs

19.1 Out-of-the-box extending

Out of the box, Claude Code can work with files, run commands, and search the web. You can extend it in five ways. Each has its own job, and each gets its own chapter:

FEATURE	IN ONE SENTENCE	LIVES IN
Subagents	Specialist helpers with their own instructions and tools	<code>.claude/agents/<name>.md</code>
Hooks	Run your own command automatically when something happens	<code>"hooks"</code> in <code>settings.json</code>
MCP servers	Connect Claude to outside tools: GitHub, Notion, databases, browsers	<code>claude mcp add ...</code>
Plugins	Bundles of the others you can install in one step	<code>/plugin</code>
Skills	Save a reusable set of instructions as a <code>/command</code> (Chapter 10)	<code>.claude/skills/<name>/SKILL.md</code>

19.2 Built-in subagents

SUBAGENT	WHAT IT DOES	TOOLS
Explore	Fast searching and reading of a codebase	Read-only
Plan	Research during plan mode	Read-only
General-purpose	Complex jobs that need both research and changes	All tools

Claude uses these automatically. When you see “Explore(…)” in the transcript, a subagent is reading code on Claude’s behalf; only its summary comes back to your conversation.

19.3 Create your own

The easiest way is to ask:

PROMPT

Create a project subagent called "copy-editor" that checks my website text for spelling, tone, and reading level. It should only be able to read files.

Claude saves it as `.claude/agents/copy-editor.md`. Personal subagents for every project go in `~/.claude/agents/`. Claude Code notices new or changed files within a few seconds.

The file format

```
.CLAUDE/AGENTS/CODE-REVIEWER.MD
```

```
---
name: code-reviewer
description: Expert code reviewer. Use right after writing or
  changing code.
tools: Read, Grep, Glob, Bash
model: sonnet
---
```

```
You are a senior code reviewer. When invoked:
1. Run git diff to see recent changes.
2. Check for clear naming, error handling, exposed secrets,
  input validation, and missing tests.
3. Report findings as Critical, Warnings, and Suggestions, each
  with a specific fix.
```

The frontmatter configures the subagent; the text below becomes its instructions. Only `name` and `description` are required.

FIELD	WHAT IT DOES
<code>tools</code>	Tools it may use, such as <code>Read</code> , <code>Grep</code> . Omit to inherit all
<code>disallowedTools</code>	Tools to remove
<code>model</code>	<code>sonnet</code> , <code>opus</code> , <code>haiku</code> , <code>fable</code> , a full ID, or <code>inherit</code>
<code>permissionMode</code>	Its permission mode, such as <code>plan</code> or <code>acceptEdits</code>
<code>maxTurns</code>	Stop after this many steps
<code>skills</code>	Skills preloaded into it
<code>mcpServers</code>	MCP servers it can use
<code>memory</code>	Give it its own long-term memory: <code>user</code> , <code>project</code> , or <code>local</code>
<code>isolation</code>	<code>worktree</code> gives it its own copy of the repository
<code>effort</code> , <code>color</code> , <code>background</code>	Effort level, display color, always run in background

19.4 Calling a subagent on purpose

HOW	EXAMPLE
Name it in plain English	"Use the code-reviewer subagent to look at my changes"
@-mention it (guarantees it runs)	<code>@"code-reviewer (agent)" look at the auth changes</code>
Run the whole session as it	<code>claude --agent code-reviewer</code>

Subagents usually run in the **background**, so you can keep chatting while they work. Press `Ctrl` + `B` to background one that's running in front, and `/tasks` to check on them.

19.5 Three useful designs

SUBAGENT	TOOLS	INSTRUCTIONS IN BRIEF
test-writer	Read, Edit, Write, Bash	Write tests for new code, run them, report failures
debugger	Read, Edit, Bash, Grep	Reproduce the error, find the root cause, make the smallest fix
docs-checker	Read, Grep, Glob	Compare the README with the code and list anything out of date

TIP

Give a subagent a clear `description` saying *when* to use it ("Use after any change to the checkout code"). That's what Claude reads to decide whether to delegate.

- ☐ I've seen Explore working in the transcript
- ☐ I've created a subagent by asking Claude
- ☐ I know how to @-mention a subagent

Hooks: Automatic Actions

A **hook** runs a command of yours at a specific moment, for example after every file edit or when Claude needs your attention. Unlike CLAUDE.md, hooks *always* run.

IN THIS CHAPTER

- How hooks are configured
- The main hook events
- Four ready-to-use examples, for every operating system
- How hooks talk back to Claude, and how to debug them

20.1 Your first hook: a desktop notification

This example shows a desktop notification on a Mac whenever Claude is waiting for you:

~/CLAUDE/SETTINGS.JSON

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude needs you\" with title \"Claude
Code\"'"
          }
        ]
      }
    ]
  }
}
```

For other systems, replace the `command` line:

SYSTEM	COMMAND
Linux	<code>notify-send 'Claude Code' 'Claude needs you'</code> (package <code>libnotify-bin</code>)
Windows	A PowerShell <code>MessageBox</code> command; ask Claude to write it for you

Type `/hooks` to see what's configured. You can also just describe the hook you want and ask Claude to set it up.

 TIP

On a Mac, if no notification appears, run `osascript -e 'display notification "test"'` once, then allow notifications for **Script Editor** in `System Settings > Notifications`.

20.2 How a hook is built

Each hook has three parts: the **event** (when), an optional **matcher** (which tools or cases), and the **command** (what runs). Several events can live side by side in one `"hooks"` object.

EVENT	FIRES WHEN
<code>SessionStart</code>	A session begins or resumes (also after compaction)
<code>UserPromptSubmit</code>	You send a prompt, before Claude sees it
<code>PreToolUse</code>	Before a tool runs. Can block it
<code>PostToolUse</code>	After a tool succeeds
<code>Notification</code>	Claude needs your input or permission
<code>Stop</code>	Claude finishes responding
<code>SubagentStop</code>	A subagent finishes
<code>PreCompact</code> / <code>PostCompact</code>	Around compaction
<code>SessionEnd</code>	A session ends

There are more for special cases, such as `PermissionRequest`, `FileChanged`, and `ConfigChange`.

20.3 Example: format code after every edit

.CLAUDE/SETTINGS.JSON

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --write"
          }
        ]
      }
    ]
  }
}
```

The hook receives details about the edit as JSON on its input; `jq` (a small JSON tool you may need to install) pulls out the file path, and Prettier formats that file.

20.4 Example: block edits to protected files

Save this script as `.claude/hooks/protect-files.sh` and make it runnable with `chmod +x .claude/hooks/protect-files.sh`:

.CLAUDE/HOOKS/PROTECT-FILES.SH

```
#!/bin/bash
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

for pattern in ".env" "package-lock.json" ".git/"; do
  if [[ "$FILE_PATH" == *"$pattern"* ]]; then
    echo "Blocked: $FILE_PATH is protected" >&2
    exit 2
  fi
done
exit 0
```

Then register it:

.CLAUDE/SETTINGS.JSON

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

20.5 Example: remind Claude after compaction

.CLAUDE/SETTINGS.JSON

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before committing.'"
          }
        ]
      }
    ]
  }
}
```

Text a `SessionStart` hook prints is added to Claude's context.

20.6 How hooks talk back: exit codes

EXIT CODE	MEANING
0	No objection. For <code>SessionStart</code> and <code>UserPromptSubmit</code> , printed text is added to context
2	Block the action. The message you print to error output (stderr) goes to Claude as feedback
Anything else	A non-blocking error; the action goes ahead and a notice appears

Hooks can also print structured JSON for finer control; the official hooks reference covers it.

20.7 Debugging hooks

PROBLEM	FIX
Hook never fires	Check <code>/hooks</code> ; check the event name and matcher spelling
"hook error" notice	Run the command yourself in a terminal to see the error
Script won't run	Make it executable with <code>chmod +x</code> ; use the full path
JSON output ignored	Your shell profile may print text first; keep hook output clean
Need more detail	Start with <code>claude --debug</code> and read the log

WARNING

Hooks run with your full permissions, automatically. Read any hook in a project you didn't write before trusting that folder.

- ☐ I've set up a notification hook
- ☐ I understand exit code 2 blocks an action
- ☐ I check `/hooks` after adding one

MCP Servers: Connect Other Apps

MCP (Model Context Protocol) is an open standard that lets AI tools plug into other software. An **MCP server** is a small connector that gives Claude new abilities, such as reading your Notion pages or creating GitHub issues. (This series has a whole manual on MCP.)

IN THIS CHAPTER

- Adding remote and local MCP servers
- Scopes: just you, this project, or everywhere
- Signing in, checking status, and removing servers
- Two worked examples, and staying safe

21.1 Add a remote server

Most modern services offer a **remote** server: a web address. Add one like this:

TERMINAL

```
claude mcp add --transport http notion https://mcp.notion.com/mcp
```

Then type `/mcp` inside Claude Code, select the server, and sign in through your browser. Some servers use a token in a header instead:

TERMINAL

```
claude mcp add --transport http secure-api https://api.example.com/mcp \  
--header "Authorization: Bearer your-token"
```

21.2 Add a local server

Some servers run as a program on your own computer (a **local** or **studio** server). Everything after the `--` is the command that starts it:

TERMINAL

```
claude mcp add --env API_KEY=your-key myserver -- npx -y @example/mcp-server
```

21.3 Choose a scope

SCOPE	FLAG	LOADS IN	STORED IN
Local (default)	<code>--scope local</code>	This project, just you	<code>~/.claude.json</code>
Project	<code>--scope project</code>	This project, whole team	<code>.mcp.json</code> in the project (commit it)
User	<code>--scope user</code>	All your projects	<code>~/.claude.json</code>

A project `.mcp.json` file looks like this. `${VAR}` reads an environment variable, so secrets stay out of Git:

.MCP.JSON

```
{
  "mcpServers": {
    "shared-server": {
      "type": "http",
      "url": "https://example.com/mcp",
      "headers": { "Authorization": "Bearer ${SHARED_TOKEN}" }
    }
  }
}
```

21.4 Manage servers

COMMAND	WHAT IT DOES
<code>claude mcp list</code>	List servers and whether they're connected
<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>	Sign in without opening <code>/mcp</code>
<code>/mcp</code>	Status, sign-in, enable, disable, reconnect
<code>/mcp reconnect all</code>	Retry every failed server

21.5 Worked example: GitHub

TERMINAL

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer YOUR_GITHUB_PAT"
```

Replace `YOUR_GITHUB_PAT` with a GitHub personal access token. Then try “Show me all open PRs assigned to me” or “Create an issue for the bug we just found.” (If you have the `gh` tool installed, Claude can use it directly too, which uses less context.)

21.6 Worked example: a database, read-only

TERMINAL

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@localhost:5432/shop"
```

Use a database user that can only read. Then ask “What were total sales this month?”

21.7 Prompts and context cost

- MCP servers can add slash commands from their **prompts**, such as `/mcp__github__pr_review` ⁴⁵⁶ (Chapter 10).
- Tools load on demand, so only their names use context until Claude needs one. Still, disable servers you don't use with `/mcp`.

 WARNING

Only install MCP servers you trust. A server can see what Claude sends it and can return text that tries to trick Claude. Prefer official servers published by the company whose product you're connecting.

- ☐ I've added one remote server and signed in with `/mcp`
- ☐ I know the difference between local, project, and user scope
- ☐ I keep tokens out of committed files

Plugins and Marketplaces

A **plugin** packages skills, subagents, hooks, and MCP servers together. Type `/plugin` to browse and install from **marketplaces** (catalogs of plugins). Anthropic's official marketplace is built in.

IN THIS CHAPTER

- Installing a plugin in a few keystrokes
- Adding other marketplaces
- Install scopes, and keeping plugins under control

22.1 Install a plugin

INSTALL FROM THE OFFICIAL MARKETPLACE

- 1 Open the manager.**
Type `/plugin` and go to the **Discover** tab.
- 2 Pick a plugin.**
Read its description and, for official plugins, its **Context cost** estimate.
- 3 Choose a scope** and install. Or type it directly:

```
TEXT
/plugin install commit-commands@claude-plugins-official
```
- 4 Reload if asked.**
Run `/reload-plugins` if the install summary tells you to.

Plugin commands are prefixed with the plugin name, as in `/plugin-name:command`.

22.2 Add another marketplace

TEXT

```
/plugin marketplace add your-org/plugins
```

A marketplace can be a GitHub repository (`owner/repo` , optionally `#tag` to pin a version), any Git URL, a local folder, or a web address of a `marketplace.json` file. From your shell, `claude plugin install ...` and `claude plugin marketplace add ...` do the same without opening a session.

22.3 Scopes

SCOPE	ENABLED FOR
User	You, in every project on this computer
Project	Everyone in the repository (saved in <code>.claude/settings.json</code>)
Local	You, in this repository only

22.4 Keeping plugins under control

- Every enabled plugin adds its skill descriptions to every session, so turn off ones you don't use (`/plugin` → **Installed**, or `claude plugin disable <name>`).
- `/doctor` and `/skill-doctor` point out extensions that cost context but aren't used.
- A plugin runs code as you. Install from sources you trust, and read what a third-party plugin includes first.

- ☐ I've browsed the Discover tab in `/plugin`
- ☐ I know how to disable a plugin I'm not using

Scripts, Headless Mode, and the Agent SDK

Claude Code doesn't have to be a conversation. With `claude -p` it answers once and exits, which means you can use it inside scripts, build tools, and scheduled jobs. The **Agent SDK** goes further, letting you build Claude Code's agent into your own programs.

IN THIS CHAPTER

- `claude -p` ("print mode") and piping data in and out
- Getting JSON, and JSON that matches a shape you define
- Pre-approving tools and choosing a permission mode for scripts
- Continuing conversations from a script
- A first Agent SDK program in Python or TypeScript

23.1 Print mode basics

`claude -p` ("print mode", also called **headless** or non-interactive mode) answers once and exits:

TERMINAL

```
claude -p "What does the auth module do?"
cat error.log | claude -p "explain this error in one paragraph"
cat build-error.txt | claude -p 'explain the root cause' > output.txt
```

It exits with code 0 on success and non-zero on failure, so scripts can react. Piped input is capped at 10 MB; for bigger inputs, save a file and name it in the prompt.

23.2 Structured output

FLAG	OUTPUT
<code>--output-format text</code>	Plain text (the default)
<code>--output-format json</code>	JSON with the <code>result</code> , <code>session_id</code> , <code>cost</code> , and other details
<code>--output-format stream-json</code>	One JSON message per line, as Claude works

TERMINAL

```
claude -p "Summarize this project" --output-format json | jq -r '.result'
```

To get JSON in a shape you define, add `--json-schema`; the answer arrives in `structured_output`:

TERMINAL

```
claude -p "List the main function names in auth.py" \  
  --output-format json \  
  --json-schema '{"type":"object","properties":{"functions":  
    {"type":"array","items":{"type":"string"}},"required":["functions"]}' \  
  | jq '.structured_output'
```

23.3 Permissions in scripts

Nobody is there to click “Yes,” so decide in advance what Claude may do:

TERMINAL

```
claude -p "Run the test suite and fix any failures" \  
  --allowedTools "Bash,Read,Edit"
```

TERMINAL

```
claude -p "Look at my staged changes and create a commit" \  
  --allowedTools "Bash(git diff *),Bash(git status *),Bash(git commit *)"
```

OPTION	EFFECT IN A SCRIPT
<code>--permission-mode acceptEdits</code>	File edits run; other commands are denied unless allowed
<code>--permission-mode dontAsk</code>	Only pre-approved tools run; everything else is refused
<code>--permission-mode auto</code>	The classifier reviews actions
<code>--permission-prompts none</code>	Anything that would prompt is denied, and Claude is told not to retry
<code>--max-turns 5</code>	Stop after five steps
<code>--max-budget-usd 2</code>	Stop when estimated API spend reaches \$2

⚠ WARNING

A `-p` run shows no folder-trust dialog, so a project's hooks and MCP servers run even in a folder you've never trusted. Only run it on code you trust, or add `--bare`.

23.4 Faster, repeatable runs with `--bare`

`--bare` skips loading hooks, skills, plugins, MCP servers, auto memory, and `CLAUDE.md`, so a script behaves the same on every machine. It doesn't use your subscription login: set `ANTHROPIC_API_KEY` first.

TERMINAL

```
claude --bare -p "Summarize README.md" --allowedTools "Read"
```

23.5 Continuing a conversation

TERMINAL

```
claude -p "Review this codebase for performance issues"
claude -p "Now focus on the database queries" --continue

id=$(claude -p "Start a review" --output-format json | jq -r '.session_id')
claude -p "Continue that review" --resume "$id"
```

23.6 Adding Claude to a build script

PACKAGE.JSON

```
{
  "scripts": {
    "lint:claude": "git diff main | claude -p \"report any typos in this diff as filename:line, then the issue\""
  }
}
```

Run it with `npm run lint:claude`. Your own skills work in print mode too: put `/skill-name` in the prompt.

23.7 The Agent SDK

The **Agent SDK** gives you the same tools, agent loop, and context handling that power Claude Code, as a library for Python (3.10+) or TypeScript (Node.js 18+). Use it to build your own agents.

YOUR FIRST SDK AGENT IN PYTHON

1 Install the package.

TERMINAL

```
pip install claude-agent-sdk
```

2 Set an API key from the Claude Console (platform.claude.com):

TERMINAL

```
export ANTHROPIC_API_KEY=your-api-key
```

3 Write `agent.py`.

AGENT.PY

```
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async def main():
    async for message in query(
        prompt="Review utils.py for bugs that would crash. Fix them.",
        options=ClaudeAgentOptions(
            allowed_tools=["Read", "Edit", "Glob"],
            permission_mode="acceptEdits",
        ),
    ):
        print(message)

asyncio.run(main())
```

4 Run it.`python agent.py`

For TypeScript, install `@anthropic-ai/claude-agent-sdk` and call `query({ prompt, options })` in a `for await` loop. The SDK also supports hooks, subagents, MCP, sessions, and plugins.

NOTE

Programs you build with the Agent SDK for other people must use API-key authentication, not a `claude.ai` login, unless Anthropic has approved otherwise.

- ☐ I've piped a file into `claude -p`
- ☐ I've used `--output-format json` with `jq`
- ☐ I pre-approve exactly the tools a script needs

GitHub Actions: Claude on Your Pull Requests

GitHub Actions is GitHub's automation system: small programs, called **workflows**, that run on GitHub's computers when something happens in your repository. The Claude Code GitHub Action lets anyone mention `@claude` in an issue or pull request to get work done.

IN THIS CHAPTER

- Quick setup with `/install-github-app`
- Manual setup, step by step
- Using `@claude`, and running Claude on a schedule
- Keeping costs and credentials under control

24.1 Quick setup

SET UP WITH ONE COMMAND

- 1 Install and sign in to the GitHub CLI.**
Get it from cli.github.com, then run `gh auth login`.
- 2 Open Claude Code in your repository** and run `/install-github-app`.
- 3 Follow the prompts.**
Claude Code installs the Claude GitHub App, saves a secret (an API key as `ANTHROPIC_API_KEY`, or a subscription token as `CLAUDE_CODE_OAUTH_TOKEN`), and opens a pull request with the workflow files.
- 4 Merge that pull request.**
Now `@claude` works in the repository.

You need admin access to the repository, and it must be on github.com.

24.2 Manual setup

1. Install the Claude GitHub App at github.com/apps/claude on your repository.
2. Add a repository secret: `ANTHROPIC_API_KEY` from the Claude Console, or `CLAUDE_CODE_OAUTH_TOKEN`, generated with `claude setup-token` (Pro, Max, Team, and Enterprise).
3. Add a workflow file:

`.GITHUB/WORKFLOWS/CLAUDE.YML`

```
name: Claude Code
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    if: contains(github.event.comment.body, '@claude')
    runs-on: ubuntu-latest
    permissions:
      contents: write
      pull-requests: write
      issues: write
      id-token: write
      actions: read
    steps:
      - uses: actions/checkout@v6
        with:
          fetch-depth: 1
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
```

With a subscription token, replace the last line with `claude_code_oauth_token: ${ secrets.CLAUDE_CODE_OAUTH_TOKEN }`.

24.3 Using @claude

Comment on an issue or pull request:

TEXT

```
@claude implement this feature based on the issue description
@claude fix the TypeError in the user dashboard component
@claude how should I add authentication to this endpoint?
```

Claude replies in the same thread and updates its comment as it works. Only people with write access can trigger it, and bots are ignored unless you allow them.

24.4 Automation mode: run on a schedule

Give the action a `prompt` and it runs without a mention, for example every morning:

.GITHUB/WORKFLOWS/DAILY-REPORT.YML

```
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      issues: read
      id-token: write
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Summarize yesterday's commits and open issues"
          claude_args: |
            --allowedTools "mcp__github__list_commits,mcp__github__list_issues"
```

INPUT	PURPOSE
<code>prompt</code>	Instructions; without it, Claude waits for <code>@claude</code>
<code>claude_args</code>	Any Claude Code command-line options, such as <code>--max-turns 5</code>
<code>trigger_phrase</code>	Change <code>@claude</code> to something else
<code>settings</code>	Claude Code settings as JSON

24.5 Costs and safety

- Each run uses GitHub Actions minutes *and* Claude usage (API tokens, or your subscription with an OAuth token).
- Keep `@claude` requests specific, keep `CLAUDE.md` short, and set `--max-turns` and workflow timeouts.
- **Never** paste keys into workflow files; always use GitHub secrets.
- Review Claude's pull requests before merging, like any teammate's.

PROBLEM	TRY
Claude doesn't respond to <code>@cLaude</code>	Check the app is installed, the workflow is on the default branch, and the commenter has write access
Authentication errors	Check the secret's name matches the workflow input
CI doesn't run on Claude's commits	See the troubleshooting section of the official GitHub Actions docs page

- ☐ The Claude GitHub App is installed on my repository
- ☐ My key or token is stored as a GitHub secret
- ☐ I've tried an `@cLaude` comment on a test issue

Beyond the Terminal: IDE, Desktop, Web, and Mobile

Everything you’ve learned carries over to Claude Code’s other surfaces. They share your settings, skills, and `CLAUDE.md` files on the same computer.

IN THIS CHAPTER

- The VS Code extension and JetBrains plugin
- The desktop app’s Code tab
- Cloud sessions on the web and phone
- Remote Control, and Claude in Chrome

25.1 VS Code and Cursor

In VS Code or Cursor, open the Extensions view (`Ctrl`/`Cmd` + `Shift` + `X`), search for **Claude Code**, and click **Install**. A Claude panel appears in the sidebar. You can review proposed changes as inline diffs and @-mention the lines you’ve selected.

You need VS Code 1.94 or later. The extension includes its own copy of Claude Code for the panel; to run `claude` in VS Code’s terminal too, install the CLI as in Chapter 3.

ACTION	MAC	WINDOWS / LINUX
Switch focus between editor and Claude	<code>Cmd</code> + <code>Esc</code>	<code>Ctrl</code> + <code>Esc</code>
Open a new conversation tab	<code>Cmd</code> + <code>Shift</code> + <code>Esc</code>	<code>Ctrl</code> + <code>Shift</code> + <code>Esc</code>
Insert an @-mention of the selection	<code>Option</code> + <code>K</code>	<code>Alt</code> + <code>K</code>
Reopen a closed Claude tab	<code>Cmd</code> + <code>Shift</code> + <code>T</code>	<code>Ctrl</code> + <code>Shift</code> + <code>T</code>

- Open the panel with the Spark icon in the editor toolbar, or *** Claude Code** in the status bar.

- Claude automatically sees the text you've selected.
- In Manual mode, each edit opens a side-by-side diff with **Accept** and **Reject** buttons.
- Run **Claude Code: Open Walkthrough** from the Command Palette for a guided tour.

25.2 JetBrains IDEs

The plugin works with IntelliJ IDEA, PyCharm, Android Studio, WebStorm, PhpStorm, GoLand, and others. Install the CLI first (Chapter 3), then install **Claude Code** from the JetBrains Marketplace and restart the IDE. Press `Cmd + Esc` (Mac) or `Ctrl + Esc` to open it; diffs open in the IDE's diff viewer, and your current selection is shared with Claude.

25.3 The desktop app

Download the Claude app from claude.com/download, sign in, and open the **Code** tab. You get the same Claude Code with a graphical interface: a folder picker, a visual **diff** (a side-by-side view of what changed), a permission-mode selector next to the send button, and the ability to run several sessions in parallel.

- The app has three tabs: **Chat** (no file access), **Cowork** (a background agent for general tasks), and **Code**.
- Choose **Local** to work on your own files, or **Cloud** to run in the cloud.
- You don't need the CLI installed; the app includes Claude Code. Builds exist for macOS, Windows (x64 and ARM64), and Linux (beta).
- From the terminal, `/desktop` moves your current session into the app.

25.4 Claude Code on the web and phone

At claude.ai/code, or in the **Code** tab of the Claude mobile app, you can start a **cloud session**: Claude works on a copy of your GitHub repository on Anthropic's computers, so it keeps going even when your laptop is closed. When it's done, it pushes a branch you can review. Cloud sessions are available on Pro, Max, Team, and eligible Enterprise seats, and need your GitHub account connected.

From the terminal, `claude --cloud "fix the login bug"` hands a task to the cloud, and `claude --teleport` pulls a cloud session back to your computer.

	CLOUD SESSION	LOCAL SESSION	LOCAL + REMOTE CONTROL
Code runs on	Anthropic's cloud	Your computer	Your computer
Uses your local setup	No, repository only	Yes	Yes
Needs GitHub	Yes	No	No
Keeps going if you disconnect	Yes	No	While your computer stays on

25.5 Remote Control

Remote Control lets you continue a session that's running on your computer from your phone or any browser. Run `/remote-control` (or start with `claude --remote-control`), then open the session at `claude.ai/code` or in the mobile app. Your files and tools stay on your machine, which must stay on. It needs a `claude.ai` subscription; on Team and Enterprise an Owner must switch it on first.

25.6 The mobile app

There's no separate Claude Code app: use the **Code** tab in the Claude app for iOS or Android. Run `/mobile` to show a QR code that opens the right app store. From your phone you can start cloud sessions, drive Remote Control sessions, and get notified when Claude needs you.

25.7 Claude in Chrome

Start with `claude --chrome` (or configure it with `/chrome`) and Claude can use your Chrome browser through the Claude in Chrome extension: open your site, click through it, fill forms, and read the browser console to debug.

- ☐ I've tried Claude Code in at least one editor or the desktop app
- ☐ I know cloud sessions keep working when my laptop is closed
- ☐ I know `/remote-control` lets me continue from my phone

Costs and Limits

Claude Code use is included in paid subscriptions up to a limit, or billed per token on a Console account. A few habits make your usage go much further.

IN THIS CHAPTER

- How usage is measured
- Checking usage with `/usage`
- What happens when you hit a limit
- Fifteen ways to save usage

26.1 How usage works

On a **subscription** (Pro, Max, Team), Claude Code use is included up to your plan's limits, which reset over time. On a **Console** account you pay per **token** (a token is a chunk of text, roughly $\frac{3}{4}$ of a word). Longer conversations and bigger files use more tokens.

Type `/usage` to see your current usage, your limits, and a breakdown of what's using the most.

Why usage climbs in a long session

- **Every turn resends the conversation.** A long, cluttered context costs more on every message.
- **Breaks cost more.** On a subscription, the conversation cache lasts about an hour. Come back later and your next message reprocesses everything.
- **Helpers add up.** Subagents, workflows, `/loop`, and `/goal` all send their own requests.
- **Compacting a huge conversation is itself a big request.** `/clear` costs nothing.

26.2 When you hit a limit

By default, Claude Code waits in the open session and continues the task by itself when your limit resets. The bottom of the screen shows when. Press `Esc` to cancel the wait. You can also:

- run `/rate-limit-options` to choose between waiting, adding usage credits, and upgrading;
- run `/usage-credits` to add paid extra usage;
- run `/upgrade` to move to a bigger plan.

Turn automatic continue off in `/config` (**Continue automatically at usage limit**).

26.3 Make your usage go further

- `/clear` **between unrelated tasks**. Old conversation history is re-read with every message.
- `/compact` **during long tasks** to shrink the history.
- **Use a lighter model for simple jobs:** `/model haiku` or `/model sonnet` for routine work, and save the most powerful model for hard problems.
- **Be specific.** A precise request avoids wasted exploration.
- **Keep CLAUDE.md lean.** It's loaded into every session.

More ways, from Anthropic's guidance:

1. **Use plan mode for big changes** so Claude doesn't build the wrong thing and redo it.
2. **Stop early.** Press `Esc` as soon as Claude heads the wrong way, then `/rewind`.
3. **Lower the effort** for routine work: `/effort low` or `medium`.
4. **Move procedures from CLAUDE.md into skills**, which load only when used.
5. **Disable MCP servers you aren't using** with `/mcp`. Command-line tools like `gh` are cheaper than MCP servers.
6. **Send big reads to a subagent**, ideally on a smaller model, so only a summary comes back.
7. **Give Claude a check to run** (tests, a build) so it gets it right without rounds of fixes.
8. **Use `/btw` for quick questions** that don't need to stay in the conversation.
9. **Avoid leaving `/loop` running** when you don't need it.
10. **Install a code-intelligence plugin** for typed languages, so Claude jumps to definitions instead of reading many files.

TIP

If you hit a limit mid-task, `/rate-limit-options` shows your choices: wait for the reset, add usage credits, or upgrade.

26.4 A simple daily routine

KEEP USAGE LOW EVERY DAY

- 1 **Start each task fresh** with `/clear` and a specific first prompt.
 - 2 **Plan big changes** in plan mode, on `opusplan` or Opus.
 - 3 **Build on Sonnet** at `medium` effort; drop to Haiku for chores.
 - 4 **Compact with a focus** when `/context` shows the window filling.
 - 5 **Check** `/usage` at the end of the day to see what used the most.
- ☐ I check `/usage` once in a while
 - ☐ I `/clear` between tasks and choose lighter models for simple jobs

Project Recipes

These recipes walk through real jobs from start to finish. Each lists what you need, the steps, prompts to paste, and how to know you're done. Adapt the names to your own project.

IN THIS CHAPTER

- Fourteen complete recipes, from fixing a bug to automating pull request reviews
- The commands and habits from earlier chapters, used together

27.1 Recipe 1: Fix a bug from an error message

FIX A BUG

- 1 Commit first** so you can always get back: `! git status`, then “commit everything with a clear message.”
- 2 Paste the full error.**

PROMPT

When I click "Checkout" I get this error: [paste the whole error].
Find the root cause. Write a failing test that shows the bug
before you fix it.
- 3 Let Claude fix it and run the test again.**
Ask to see the passing output.
- 4 Review** with `/diff`, then `/code-review`.
- 5 Commit** with a message that describes the bug.

Done when: the new test passes and the error no longer appears.

27.2 Recipe 2: Build a small website from scratch

BUILD A SITE

1 Make a folder and start:

```
mkdir ~/portfolio && cd ~/portfolio && claude .
```

2 Plan first.

Press `Shift` + `Tab` into plan mode:

PROMPT

Plan a three-page portfolio site (Home, Projects, Contact) in plain HTML and CSS. Mobile-friendly, calm colors, no frameworks.

3 Approve the plan and let Claude build it.

4 Check it in a browser:

“open the site in my browser.”

5 Refine with small requests: “make the headings larger,” “add my three projects with placeholders.”

6 Write a `CLAUDE.md` **with** `/init` **so future sessions know the style.**

27.3 Recipe 3: Write tests for existing code

PROMPT

Find the functions in `src/cart.js` that have no tests. Write tests for them, covering normal cases and edge cases like an empty cart and a negative quantity. Run the tests and show me the results.

If tests fail, ask: “Is the test wrong or the code wrong? Explain before changing anything.” Finish with `/simplify` to tidy the new test code.

27.4 Recipe 4: Refactor safely

REFACTOR

1 Make sure tests exist and pass.

If not, do Recipe 3 first.

2 Plan in plan mode.**PROMPT**

checkout.js is 900 lines. Plan how to split it into smaller files without changing behavior. List each new file and what moves there.

3 Approve, then build in small steps:

“do step 1 only, then run the tests.”

4 Repeat step by step, committing after each passing step.**5 Run `/code-review` at the end.**

27.5 Recipe 5: Review a pull request

TEXT

```
/code-review 1234
```

Or, for a conversation about it: “Check out PR 1234, explain what it changes in plain English, and list anything risky.” Add `--comment` to post the findings on GitHub, or use `/code-review ultra 1234` for a deep cloud review of an important change.

27.6 Recipe 6: Understand a project you’ve never seen

PROMPT

Give me a tour of this project for a beginner: what it does, the main folders, how to run it, and where a change to the login page would go. Use a subagent to explore so our conversation stays short.

Then ask follow-ups, and finish with “write what you learned into CLAUDE.md, keeping it short.”

27.7 Recipe 7: Write or update documentation

PROMPT

Compare README.md with the actual code. List anything that's out of date or missing, then update the README. Keep it friendly and add a "Getting started" section with exact commands.

27.8 Recipe 8: Automate a chore with a script

PROMPT

Write a script that renames every photo in `./photos` to its date taken, like `2026-10-16_001.jpg`. Show me what it would rename first (a dry run), and only rename after I confirm.

TIP

Always ask for a **dry run** first for anything that renames, moves, or deletes files. Shell-command changes aren't covered by checkpoints.

27.9 Recipe 9: Clean up a spreadsheet (CSV)

PROMPT

`customers.csv` has duplicate rows, inconsistent phone formats, and blank emails. Write a cleaned copy to `customers-clean.csv` (never overwrite the original) and give me a summary of what changed.

27.10 Recipe 10: Add a feature with an interview and a spec

SPEC-FIRST FEATURE

- 1 Ask Claude to interview you (Chapter 5, Pattern 4) and write `SPEC.md`.
- 2 Read and edit `SPEC.md` yourself.
- 3 `/clear`, then: "Implement `SPEC.md`. Run the tests after each part."
- 4 Set a finish line:
`/goal every item in SPEC.md works and all tests pass`.
- 5 Review and commit.

27.11 Recipe 11: Make a team command

CREATE /RELEASE-NOTES-DRAFT

- 1 Create the skill folder in the project: `.claude/skills/release-notes-draft/`.

2 Write SKILL.md :`.CLAUDE/SKILLS/RELEASE-NOTES-DRAFT/SKILL.MD`

```

---
description: Draft release notes from commits since the last tag
disable-model-invocation: true
---

!`git log $(git describe --tags --abbrev=0)..HEAD --oneline`

Turn these commits into release notes for customers, grouped
into New, Improved, and Fixed. Plain language, no commit hashes.

```

3 Commit it so teammates get /release-notes-draft too.

27.12 Recipe 12: Set up automatic pull request help on GitHub

Run `/install-github-app` in the repository (Chapter 24), merge the pull request it opens, then comment `@claude review this PR for bugs` on any pull request.

27.13 Recipe 13: Make the same change across many files

TEXT

```
/batch replace the old logger with the new logger in every file under src/
```

Claude plans the work, splits it into independent pieces, and, after you approve, runs them in parallel worktrees. Check the plan carefully; this uses more usage than a single session.

27.14 Recipe 14: Watch a slow job

TEXT

```
/loop 5m check whether the deploy finished and tell me what happened
```

Or, while your CI runs: `/loop check whether CI passed and fix anything that failed`. Press `Esc` to stop a self-paced loop.

- ☐ I commit before starting any recipe
- ☐ I ask for a check (tests, dry run, screenshot) in every recipe
- ☐ I review with `/diff` or `/code-review` before committing

Security and Privacy

Claude Code is powerful because it can act on your computer. This chapter explains how your data is handled and how to keep yourself safe.

IN THIS CHAPTER

- How Anthropic uses and keeps your data
- Prompt injection and the built-in safeguards
- Protecting secrets
- A security checklist

28.1 Your data

ACCOUNT TYPE	TRAINING ON YOUR DATA	HOW LONG IT'S KEPT
Free, Pro, Max (consumer)	Only if you allow it in your privacy settings	5 years if you allow training; 30 days if not
Team, Enterprise, API (commercial)	Not used for training unless your organization opts in	30 days by default

- Change consumer privacy settings at any time at claude.ai/settings/data-privacy-controls, or with `/privacy-settings`.
- Claude Code also keeps transcripts **on your computer** in `~/.claude/projects/` for 30 days by default, so you can resume sessions. They're plain text: treat that folder as private.
- Bug reports sent with `/bug` or `/feedback` that include your conversation are kept for five years.

28.2 Prompt injection

A **prompt injection** is text hidden in a web page, file, issue, or tool result that tries to trick Claude into doing something you didn't ask for. Claude Code's defenses include:

- **Permission prompts** for edits, commands, and network requests in Manual mode.
- **Network commands** such as `curl` and `wget` aren't auto-approved by default.
- **Web pages are summarized** by a separate model call before Claude sees them.
- **Folder trust:** Claude Code asks before working in a folder you haven't trusted, and asks separately before using MCP servers listed in a project's `.mcp.json`.
- **Commands it can't fully understand** require approval.
- **Credentials** are stored in the macOS Keychain, or in a file only your user can read.

⚠ WARNING

No system is immune. Review commands before approving them, don't pipe untrusted content straight into Claude, and use a virtual machine or container for anything risky.

28.3 Protecting secrets

- **Deny reads of secret files:**

.CLAUDE/SETTINGS.JSON

```
{
  "permissions": {
    "deny": ["Read(./env)", "Read(./env.*)", "Read(./secrets/**)"]
  }
}
```

- **Never paste API keys into chat.** If you do, rotate (replace) the key.
- **Keep tokens out of committed files:** use `${VAR}` in `.mcp.json` and GitHub secrets in workflows.
- **Check exports and bug reports** before sharing them.

28.4 Trusting other people's projects

When you open a project someone else wrote, its `.claude/settings.json` hooks, `.mcp.json` servers, and skills can run code. Read them before choosing **Yes** at the trust prompt. Remember that `claude -p` shows no trust prompt at all.

28.5 Extra steps for teams

- Share safe defaults in a committed `.claude/settings.json`: deny rules for secrets, ask rules for `git push`.
 - Ask administrators about **managed settings**, which can enforce rules nobody can override.
 - Review changes to `.claude/`, `.mcp.json`, and workflow files in pull requests as carefully as code.
 - Report suspected security issues in Claude Code through Anthropic's official channels, not in public issues.
-
- ☐ I know my plan's data training and retention rules
 - ☐ My secret files are covered by deny rules
 - ☐ I read hooks, MCP servers, and skills before trusting a stranger's project

Troubleshooting

Most problems have a quick fix. Start with the right diagnostic tool, then look up your symptom in the tables below.

IN THIS CHAPTER

- The four diagnostic tools and when to use each
- A big table of problems and fixes, by category
- How to get more help

29.1 Your diagnostic toolkit

TOOL	WHERE	WHAT IT DOES
<code>claude doctor</code>	Your shell	Quick, read-only check of installation and settings files
<code>/doctor</code>	Inside Claude Code	Full checkup that can fix problems after you confirm
<code>/debug</code> <code>[description]</code>	Inside Claude Code	Turns on logging and analyzes what's going wrong
<code>claude --safe-mode</code>	Your shell	Starts with all customizations off, to see whether one of them is the cause
<code>claude --debug</code>	Your shell	Writes a detailed debug log from the start

29.2 Installation and login

PROBLEM	TRY THIS
<code>claude: command not found</code>	Open a new terminal window. Still failing? Re-run the installer and follow the PATH fix it prints, or run <code>claude doctor</code> .
<code>'irm' is not recognized</code>	You're in CMD, not PowerShell. Use the CMD install command, or open PowerShell.
The token '&&' is not a valid statement separator	You're in PowerShell. Use the PowerShell command instead.
Login window doesn't open	Copy the link Claude Code prints into your browser manually.
Old version keeps running after an update	You may have two installs. Run <code>claude doctor</code> to find duplicates
Prompted to log in again	Your login expired: <code>/login</code>

29.3 Everyday use

PROBLEM	TRY THIS
Claude keeps asking permission for the same thing	Choose Yes, and don't ask again , add an allow rule, or run <code>/fewer-permission-prompts</code> .
Claude ignores an instruction	Put it in CLAUDE.md, make it concrete, and keep the file short. For must-never-happen rules, use a deny rule.
Responses getting slow or confused	<code>/compact</code> or <code>/clear</code> , then restate the goal.
Something is broken and you don't know why	<code>/doctor</code> inside Claude Code. It diagnoses and offers fixes.
Unknown command	Check the spelling; the command may not be available on your plan or platform; run <code>claude update</code>
My own skill doesn't appear	<code>/reload-skills</code> ; check the folder layout <code>skills/<name>/SKILL.md</code>
Rewind says no files were restored	The change was made by a shell command, or the snapshot was cleaned up. Use Git
<code>Shift + Enter</code> sends the message	Run <code>/terminal-setup</code> , or use <code>\ + Enter</code>
<code>Option</code> shortcuts do nothing on a Mac	Set Option as Meta in your terminal's settings
Screen flickers or jumps	Try <code>/tui fullscreen</code> ; press <code>Ctrl + L</code> to redraw
Auto-compaction keeps failing	A single huge file or output is refilling context. <code>/clear</code> and avoid reading it whole

Hit a usage limit

Wait for the automatic continue, or see `/rate-limit-options`

29.4 Extensions

PROBLEM	TRY THIS
MCP server shows “Failed to connect”	<code>claude mcp get <name></code> shows the error; re-authenticate in <code>/mcp</code> .
MCP server needs sign-in again	<code>/mcp</code> , select it, sign in; or <code>claude mcp login <name></code>
Hook doesn't fire	Check <code>/hooks</code> , the event name, and the matcher; make scripts executable
Plugin missing after install	<code>/reload-plugins</code> , then check <code>/plugin</code> → Installed
Settings file ignored	It's probably invalid JSON. Look for comments or trailing commas; check <code>/status</code>
<code>defaultMode: auto</code> has no effect	Move it to <code>~/.claude/settings.json</code>

29.5 Editors and other surfaces

PROBLEM	TRY THIS
VS Code extension doesn't appear	Restart VS Code, or run Developer: Reload Window
JetBrains says it can't launch Claude Code	Install the CLI and make sure <code>claude</code> is on your PATH
Diffs don't open in my editor	<code>/ide</code> to check the connection
Cloud session can't see my private repository	Install the Claude GitHub App on that repository

29.6 Getting more help

NOTE

Stuck? Type `/bug` to report a problem to Anthropic, ask Claude itself (“why did that fail?”), or take the free Claude Code courses at academy.claude.com.

- ☐ I try `/doctor` first when something is wrong
- ☐ I know `claude --safe-mode` rules out my own customizations
- ☐ I can report a bug with `/bug`

Slash Command Cheatsheet

COMMAND	DOES	COMMAND	DOES
<code>/clear</code>	Fresh conversation	<code>/compact [focus]</code>	Summarize to free space
<code>/context</code>	Show context usage	<code>/rewind</code>	Undo code and/or chat
<code>/resume</code>	Reopen a session	<code>/rename <name></code>	Name this session
<code>/branch</code>	Split the conversation	<code>/btw <q></code>	Side question
<code>/export</code>	Save chat as text	<code>/copy [N]</code>	Copy last reply
<code>/plan [task]</code>	Plan mode	<code>/goal <cond></code>	Work until done
<code>/loop 5m <p></code>	Repeat a prompt	<code>/diff</code>	Review changes
<code>/code-review</code>	Find bugs	<code>/simplify</code>	Clean up code
<code>/security-review</code>	Security check	<code>/init</code>	Create CLAUDE.md
<code>/memory</code>	Edit memory	<code>/config</code>	Settings
<code>/status</code>	Version, account	<code>/permissions</code>	Allow/deny rules
<code>/model</code>	Switch model	<code>/effort</code>	Thinking level
<code>/fast</code>	Fast mode	<code>/usage</code>	Limits and usage
<code>/mcp</code>	MCP servers	<code>/plugin</code>	Plugins
<code>/skills</code>	List skills	<code>/hooks</code>	View hooks
<code>/agents</code>	Subagent help	<code>/tasks</code>	Background work
<code>/add-dir <path></code>	Extra folder	<code>/cd <path></code>	Move session
<code>/theme</code>	Colors	<code>/statusline</code>	Status bar
<code>/output-style</code>	Reply style	<code>/terminal-setup</code>	Fix Shift+Enter
<code>/sandbox</code>	Command sandbox	<code>/doctor</code>	Setup checkup
<code>/debug</code>	Diagnose issues	<code>/bug</code>	Report a bug
<code>/login</code>	Sign in	<code>/logout</code>	Sign out
<code>/teleport</code>	Pull cloud session	<code>/remote-control</code>	Use from phone
<code>/install-github-app</code>	@claude on GitHub	<code>/help</code>	Help
<code>/release-notes</code>	What's new	<code>/exit</code>	Quit

Quick Reference

Install (Mac/Linux): `curl -fsSL https://claude.ai/install.sh | bash`

Install (Windows PowerShell): `irm https://claude.ai/install.ps1 | iex`

Start: `cd my-project` then `claude` · **Continue:** `claude -c` · **Resume:** `claude -r`

Stop: `Esc` · **Undo:** `Esc` `Esc` or `/rewind` · **Modes:** `Shift` + `Tab` · **New line:** `Shift` + `Enter`

Plan first: `/plan ...` · **Fresh start:** `/clear` · **Shrink history:** `/compact` · **Project memory:** `/init`

Prefixes: `/` command · `@` file · `!` shell command · `?` shortcuts

Connect an app: `claude mcp add --transport http <name> <url>` then `/mcp` to sign in

Script it: `claude -p "..." --output-format json --allowedTools "Read"`

Your own command: `~/.claude/skills/<name>/SKILL.md` → `/<name> $ARGUMENTS`

Aliases: `/reset` `/new` → `/clear` · `/undo` `/checkpoint` → `/rewind` · `/continue` → `/resume` · `/review` → `/code-review` · `/cost` `/stats` → `/usage` · `/settings` → `/config` · `/quit` → `/exit` · `/bg` → `/background` · `/checkup` → `/doctor`

Health check: `claude doctor` · **Update:** `claude update` · **Usage:** `/usage`

The Complete Slash-Command Reference

Slash commands are the control panel of Claude Code. You type `/` at the start of a message and a searchable menu appears. This chapter lists **every built-in command and bundled skill documented by Anthropic** as of this edition, so you can look any of them up on paper.

IN THIS CHAPTER

- How slash commands, bundled skills, and your own skills differ
- Deep dives on the commands you'll use every day: `/clear`, `/compact`, `/context`, `/rewind`, `/resume`, `/model`, `/init`, `/memory`, `/permissions`
- A full A-Z table of every documented command
- Recipes that chain commands together

WHERE THIS LIST COMES FROM

The A-Z table was generated from Anthropic's official commands page, code.claude.com/docs/en/commands, in October 2026. Claude Code updates often. Some commands appear only on certain plans, platforms, or versions. If a command here doesn't show up for you, run `claude update`, then type `/` and search for it. Your own `/` menu is always the final word.

30.1 Three kinds of slash command

KIND	WHAT IT IS	EXAMPLE
Built-in command	Fixed logic inside Claude Code. It runs instantly and usually doesn't cost tokens.	<code>/clear</code> , <code>/model</code> , <code>/exit</code>
Bundled skill	A ready-made set of instructions that Claude follows using its tools. It uses tokens, like a normal request.	<code>/code-review</code> , <code>/debug</code> , <code>/batch</code>
Your own skill	A <code>SKILL.md</code> file you or your team wrote (Chapter 9).	<code>/deploy</code> , <code>/fix-issue</code> 123

Anything you type after the command name is passed in as **arguments**. For example, in `/compact keep the API decisions`, the text after `/compact` tells the summary what to focus on.

TIP

A command only runs when it's at the **start** of your message. If you write "please `/deploy` this" in the middle of a sentence, nothing runs on its own. Claude just treats it as permission to use that skill if it decides to.

30.2 `/clear`: start fresh

`/clear` wipes the conversation and starts a new one with empty context. Your files are **not** touched. Only Claude's short-term memory of the chat is reset. `CLAUDE.md` loads again automatically.

USE `/CLEAR` BETWEEN TASKS

- 1 Finish or commit the current task.**
Ask Claude to summarize what it did if you want a record.
- 2 Type `/clear`.**
Optionally add a name, such as `/clear login-work`, to label the old conversation so you can find it later.
- 3 Start the next task.**
Claude now has a clean, fast, cheap context.
- 4 Need the old chat back?**
Use `/resume` and pick it from the list.

Use `/clear` when you switch to an unrelated task, when Claude keeps repeating an old mistake, or when the conversation has drifted. `/reset` and `/new` are aliases.

30.3 /compact: shrink without forgetting

Every message, file, and command output takes up room in the **context window**, the AI's working memory. When it fills up, answers get slower, more expensive, and less sharp.

`/compact` replaces the conversation so far with a summary, so you keep the important facts and free most of the space.

PROMPT

`/compact` Keep the database schema decisions and the list of failing tests. Drop the exploration of old files.

SITUATION	USE
Same task, but the chat is long	<code>/compact</code>
New, unrelated task	<code>/clear</code>
Claude went down a wrong path a few steps ago	<code>/rewind</code>
You want a side answer without cluttering the chat	<code>/btw</code>

Claude Code also **auto-compacts** when the window is nearly full. `/autocompact` lets you set how full it gets before that happens. Pass a size such as `500k`, or `auto` to go back to the default.

⚠ SUMMARIES LOSE DETAIL

Compaction is lossy. Exact error messages, file paths, and small decisions can disappear. Before compacting, put anything that must survive into `CLAUDE.md` or a notes file, or list it in your `/compact` instructions.

30.4 /context: see what's using space

`/context` draws a coloured grid of your context window, broken down by system prompt, tools, MCP servers, memory files, skills, and messages. It also suggests what to trim. Check it whenever Claude feels slow or forgetful. If MCP tools take up a big block, disconnect the servers you aren't using with `/mcp`.

30.5 /rewind: the undo button

`/rewind` (aliases `/checkpoint` and `/undo`) rolls the conversation, your code, or both back to an earlier message. Pressing `Esc` twice on an empty prompt opens the same menu.

UNDO A BAD CHANGE

- 1 **Type** `/rewind`.
A list of your earlier messages appears.
- 2 **Pick the last good point.**
Choose the message *before* things went wrong.
- 3 **Choose what to restore.**
Code only, conversation only, or both.
- 4 **Re-ask, more precisely.**
Add the detail that was missing the first time.

⚠ DANGER

Checkpoints track edits Claude made with its file tools. Changes made by shell commands, such as `rm` or a database migration, and edits from background forked skills are **not** undone. Git is your real safety net, so commit often.

30.6 /resume, /branch, /fork, and /rename

COMMAND	USE IT TO...
<code>/resume</code>	Reopen an earlier conversation by name or from a picker
<code>/rename my-feature</code>	Give this session a memorable name
<code>/branch</code>	Split the conversation here so you can try another direction, keeping the original
<code>/fork</code>	Copy the conversation into a new background session and keep working here

30.7 /model and /effort

`/model` switches which Claude model you're using and saves it as the default for new sessions. `/effort` sets how hard the model thinks: `low`, `medium`, `high`, `xhigh`, `max`, or `auto`. Higher effort is slower and uses more of your plan, but it's better for tricky bugs and design work. A good habit is to plan with a strong model at high effort, then do routine edits at a lower setting.

30.8 /init and /memory

`/init` reads your project and writes a starter `CLAUDE.md`. `/memory` opens your `CLAUDE.md` files for editing and lets you turn **auto memory** on or off and browse what it has saved. Run `/init` once per project, then keep the file current with `/memory`.

30.9 /permissions, /sandbox, and /fewer-permission-prompts

`/permissions` shows and edits your allow, ask, and deny rules (Chapter 6). `/sandbox` toggles sandbox mode on supported systems. The bundled `/fewer-permission-prompts` skill scans your history for safe, read-only commands you approve often, then suggests allow rules for them. Review that list before you accept it.

30.10 Review and quality commands

COMMAND	WHAT IT'S FOR
<code>/diff</code>	See every change in your working tree
<code>/code-review</code> (alias <code>/review</code>)	Have Claude review your diff, a branch, or a PR for bugs
<code>/security-review</code>	Scan your branch's changes for security risks
<code>/simplify</code>	Run parallel reviewers that tidy up changed code
<code>/verify</code> and <code>/run</code>	Build and run your app to confirm a change actually works

30.11 Housekeeping commands

`/doctor` runs a setup checkup and offers fixes. `/usage` (aliases `/cost` and `/stats`) shows plan limits and spending. `/status` shows your version, model, and account. `/config` opens settings. `/export` saves the chat as text, `/copy` copies the last answer, and `/release-notes` shows what's new. `/exit` (or `/quit`) leaves.

30.12 Recipes: commands that work well together

Long feature, keep it sharp: `/plan` → approve → work → `/compact` keep decisions → continue → `/code-review` → commit.

Switching tasks: commit → `/clear new-task` → `/context` to confirm it's empty.

Undo a wrong turn: `/rewind` → restore code and conversation → re-prompt with more detail.

Before a pull request: `/diff` → `/code-review` → `/security-review` → `/verify`.

Something's broken in Claude Code itself: `/doctor` → `/debug` describe the problem → `/bug` if it's a real bug.

Running low on plan usage: `/usage` → `/model` to a lighter model → `/effort low` for routine edits.

30.13 Every documented command, A–Z

The **Kind** column shows whether the command is built in, a bundled skill (uses tokens as Claude works), or a workflow. Square brackets `[ ]` mean the argument is optional. Angle brackets `< >` mean it's required.

COMMAND	KIND	WHAT IT DOES
<code>/add-dir <path></code>	Built-in	Add a working directory for file access during the current session.
<code>/advisor [model off]</code>	Built-in	Enable or disable the advisor tool, which consults a second model for guidance at key moments during a task.
<code>/agents</code>	Built-in	Print a reminder to ask Claude to create or manage subagents, or to edit <code>.claude/agents/</code> or <code>~/.claude/agents/</code> directly.
<code>/artifact-capabilities</code>	Skill	Load the reference for the runtime capabilities a published artifact can use, such as calling your connectors or offering a file download, including which ones your account has.
<code>/artifact-diagramming</code>	Skill	Load diagramming guidance for Claude to follow in artifacts: when a diagram helps, what to draw, and how to write inline SVG that stays legible in light and dark themes.

<code>/artifacts</code>	Built-in	List the artifacts you own or that are shared with you, then attach one to the session, open it in your browser, or copy its link.
<code>/auto-mode-setup</code>	Built-in	Draft <code>autoMode.environment</code> entries from your project and recent sessions, then review the draft and save it to your user settings.
<code>/autocompact [auto <tokens>]</code>	Built-in	Set the auto-compact window: how full the context window gets before Claude Code compacts automatically.
<code>/autofix-pr [prompt]</code>	Built-in	Spawn a cloud session that watches the current branch's PR and pushes fixes when CI fails or reviewers leave comments.
<code>/background [prompt]</code>	Built-in	Detach the current session to run as a background agent and free this terminal. Alias: <code>/bg</code> .
<code>/batch <instruction></code>	Skill	Orchestrate large-scale changes across a codebase in parallel.
<code>/branch [name]</code>	Built-in	Create a branch of the current conversation at this point, so you can try a different direction without losing the conversation as it stands.
<code>/btw [question]</code>	Built-in	Ask a side question about the current session without adding to the conversation.
<code>/bug [report]</code>	Built-in	Report a bug or share your conversation. Alias: <code>/share</code> .
<code>/cd <path></code>	Built-in	Move this session to a new working directory, keeping the conversation.
<code>/chrome</code>	Built-in	Configure Claude in Chrome settings.
<code>/claude-api [migrate upgrade managed-agents-onboard prompt-audit cost-optimize build-eval hillclimb preserved-thinking-migration]</code>	Skill	Load Claude API and Managed Agents reference material for your project's language.
<code>/claude-in-chrome [task]</code>	Skill	Have Claude carry out a task in your browser, such as testing a page, filling a form, or reading console logs, through Claude in Chrome.
<code>/clear [name]</code>	Built-in	Start a new conversation with empty context. Alias: <code>/reset</code> , <code>/new</code> .
<code>/code-review [low medium high xhigh max ultra] [--fix] [--comment] [--max-findings n all default] [pr# branch path]</code>	Skill	Review the current diff, or a PR number, branch, or path you pass, for correctness bugs. Alias: <code>/review</code> .
<code>/color [color default]</code>	Built-in	Set the prompt bar color for the current session.
<code>/compact [instructions]</code>	Built-in	Free up context by summarizing the conversation so far.
<code>/config [key=value ...]</code>	Built-in	Open the Settings interface to adjust theme, model, output style, and other preferences. Alias: <code>/settings</code> .
<code>/context [all]</code>	Built-in	Visualize current context usage as a colored grid.

<code>/copy [N]</code>	Built-in	Copy the last assistant response to clipboard.
<code>/cost</code>	Built-in	Alias for <code>/usage</code> .
<code>/dataviz [request]</code>	Skill	Design guidance for charts, graphs, and dashboards.
<code>/debug [description]</code>	Skill	Enable debug logging for the current session and troubleshoot issues by reading the session debug log.
<code>/deep-research <question></code>	Workflow	Fan out web searches on a question, fetch and cross-check sources, and synthesize a cited report.
<code>/design [brief]</code>	Skill	Draft UI mockups, screen flows, landing pages, or posters as artboards on one canvas, published as a Claude Design artifact, for example <code>/design a settings screen for a mobile banking app</code> .
<code>/design-login</code>	Built-in	Authorize design-system access for <code>/design-sync</code> with your claude.ai account.
<code>/design-sync [hint]</code>	Skill	Convert your repo's React design system and upload it to Claude Design, so designs it produces use your real components.
<code>/desktop</code>	Built-in	Continue the current session in the Claude Code Desktop app. Alias: <code>/app</code> .
<code>/diff</code>	Built-in	Review the changes in your working tree, including the edits Claude has made so far.
<code>/doctor [prompt-audit [path]]</code>	Skill	Run a setup checkup that diagnoses installation, settings, extension, and <code>CLAUDE.md</code> problems and proposes fixes that Claude applies after you confirm. Alias: <code>/checkup</code> .
<code>/effort [level auto status ultracode [on off]]</code>	Built-in	Set the effort level: <code>low</code> to <code>xhigh</code> , <code>max</code> , or <code>auto</code> ; <code>status</code> prints it. <code>ultracode</code> or <code>ultracode on</code> turns ultracode on for the session at the current level, and <code>ultracode off</code> turns it off; the <code>ultracode</code> key...
<code>/exit</code>	Built-in	Exit the CLI. In an attached background session, this detaches and the session keeps running. Alias: <code>/quit</code> .
<code>/export [filename]</code>	Built-in	Export the current conversation as plain text.
<code>/fast [on off]</code>	Built-in	Toggle fast mode on or off.
<code>/feedback [report]</code>	Built-in	Send product feedback about Claude Code.
<code>/fewer-permission-prompts</code>	Skill	Scan your transcripts for common read-only Bash and MCP tool calls, then add a prioritized allowlist to project <code>.claude/settings.json</code> to reduce permission prompts.
<code>/focus</code>	Built-in	Toggle the focus view, which shows only your last prompt, a one-line tool-call summary with edit diffstats, and the final response.
<code>/fork [prompt]</code>	Built-in	Copy the current conversation into a new background session and keep working here.
<code>/goal [condition clear]</code>	Built-in	Set a goal: Claude keeps working across turns until the condition is met or the goal clears for another reason.
<code>/heapdump</code>	Built-in	Write a JavaScript heap snapshot and a memory breakdown to <code>~/Desktop</code> , or your home directory on Linux without a Desktop folder, for diagnosing high memory usage.
<code>/help</code>	Built-in	Show help and available commands.
<code>/hooks</code>	Built-in	View hook configurations.
<code>/ide</code>	Built-in	Manage IDE integrations and show status.

<code>/import [codex gemini cursor] [--dry-run] [--yes]</code>	Built-in	Bring configuration from OpenAI Codex, Google Gemini CLI, or Cursor on your machine into Claude Code, including instruction files, MCP servers, commands, subagents, and skills.
<code>/init</code>	Built-in	Initialize project with a <code>CLAUDE.md</code> guide.
<code>/insights</code>	Built-in	Generate an HTML report analyzing your recent sessions on this machine: which projects you work in, how you use Claude Code, where things go wrong, and features to try.
<code>/install-github-app</code>	Built-in	Install the Claude GitHub App for a repository, with an optional step to set up GitHub Actions workflows and secrets.
<code>/install-slack-app</code>	Built-in	Install the Claude Slack app.
<code>/keybindings</code>	Built-in	Open your keyboard shortcuts file.
<code>/list-agents</code>	Built-in	List the subagents, agent team teammates, and other Claude Code sessions Claude can message, with the name to use for each.
<code>/login</code>	Built-in	Sign in to your Anthropic account.
<code>/logout</code>	Built-in	Sign out from your Anthropic account.
<code>/loop [interval] [prompt]</code>	Skill	Run a prompt repeatedly while the session stays open. Alias: <code>/proactive</code> .
<code>/mcp [reconnect (<server> all)] [enable disable [<server> all]]</code>	Built-in	Manage MCP server connections and OAuth authentication.
<code>/memory</code>	Built-in	Edit <code>CLAUDE.md</code> files, enable or disable auto memory, and view auto memory entries.
<code>/mobile</code>	Built-in	Show QR code to download the Claude mobile app. Alias: <code>/ios</code> , <code>/android</code> .
<code>/model [model]</code>	Built-in	Switch the AI model and save it as your default for new sessions.
<code>/output-style [style]</code>	Built-in	List output styles or switch to one, for example <code>/output-style concise</code> .
<code>/passes</code>	Built-in	Share a free week of Claude Code with friends.
<code>/permissions</code>	Built-in	Manage allow, ask, and deny rules for tool permissions. Alias: <code>/allowed-tools</code> .
<code>/plan [description]</code>	Built-in	Enter plan mode directly from the prompt.
<code>/plugin [subcommand]</code>	Built-in	Manage Claude Code plugins.
<code>/plugin-authoring</code>	Built-in	Load the reference Claude works from to write a mod.
<code>/powerup</code>	Built-in	Discover Claude Code features through quick interactive lessons with animated demos.

<code>/pr-comments [PR]</code>	Built-in	Removed in v2.1.91. Ask Claude directly to view pull request comments instead.
<code>/privacy-settings</code>	Built-in	View and update your privacy settings.
<code>/radio</code>	Built-in	Open Claude FM lo-fi radio in your browser.
<code>/rate-limit-options</code>	Built-in	Show ways to keep working when a claude.ai usage limit blocks a request: wait and continue automatically when the limit resets, add usage credits, or upgrade your plan.
<code>/recap</code>	Built-in	Generate a one-line summary of the current session on demand.
<code>/release-notes</code>	Built-in	View the changelog in an interactive version picker.
<code>/reload-plugins [--force]</code>	Built-in	Reload all active plugins to apply pending changes without restarting.
<code>/reload-skills</code>	Built-in	Re-scan skill and command directories so skills added or changed on disk during the session become available without restarting.
<code>/remote-control</code>	Built-in	Make this session available for Remote Control from claude.ai. Alias: <code>/rc</code> .
<code>/remote-env</code>	Built-in	Choose the default cloud environment for cloud sessions you start from the CLI.
<code>/rename [name]</code>	Built-in	Rename the current session and show the name on the prompt bar.
<code>/resume [session]</code>	Built-in	Resume a conversation by ID or name, or open the session picker. Alias: <code>/continue</code> .
<code>/review [low medium high xhigh max ultra] [--fix] [--comment] [--max-findings n all default] [pr# branch path]</code>	Built-in	Alias of <code>/code-review</code> : reviews the current diff, or a PR number, branch, or path you pass, such as <code>/review 1234</code> , and takes the same effort levels and flags.
<code>/rewind</code>	Built-in	Rewind the conversation and/or code to a previous point, or summarize from a selected message. Alias: <code>/checkpoint</code> , <code>/undo</code> .
<code>/run</code>	Skill	Launch and drive your project's app to see a change working, not only passing tests.
<code>/run-skill-generator</code>	Skill	Teach <code>/run</code> and <code>/verify</code> how to build, launch, and drive your project's app from a clean environment by writing a per-project skill.
<code>/sandbox</code>	Built-in	Toggle sandbox mode.
<code>/schedule [description]</code>	Built-in	Create, update, list, or run routines, which execute in the cloud. Alias: <code>/routines</code> .
<code>/scroll-speed</code>	Built-in	Adjust mouse wheel scroll speed interactively.
<code>/security-review</code>	Built-in	Analyze the changes on your current branch for security vulnerabilities.

<code>/setup-bedrock</code>	Built-in	Configure Amazon Bedrock authentication, region, and model pins through an interactive wizard.
<code>/setup-vertex</code>	Built-in	Configure Google Cloud's Agent Platform authentication, project, region, and model pins through an interactive wizard.
<code>/simplify [target]</code>	Skill	Review the changed code for cleanup opportunities and apply the fixes.
<code>/skill-doctor</code>	Built-in	Show what each of your skills costs in context and how often it gets used, so you can find skills to turn off.
<code>/skills</code>	Built-in	List available skills.
<code>/slides [brief]</code>	Skill	Make a new presentation as a Claude Slides artifact filled from your brief, for example <code>/slides a quarterly review of the platform team</code> .
<code>/stats</code>	Built-in	Alias for <code>/usage</code> .
<code>/status</code>	Built-in	Open the Settings interface on the Status tab, showing version, model, account, and connectivity.
<code>/statusline</code>	Built-in	Configure Claude Code's status line.
<code>/stickers</code>	Built-in	Order Claude Code stickers.
<code>/stop</code>	Built-in	Stop the background session you're attached to, or the one you send it to as a peek reply; the transcript and any worktree are kept.
<code>/subtask <task></code>	Built-in	Spawn a forked subagent: a background subagent that inherits the full conversation and works on the task while you keep working.
<code>/tasks</code>	Built-in	View and manage background work in the current session, including subagents that have finished.
<code>/team-onboarding</code>	Built-in	Generate a team onboarding guide from your Claude Code usage history.
<code>/teleport</code>	Built-in	Pull a cloud session into this terminal.
<code>/terminal-setup</code>	Built-in	Install a Shift+Enter keybinding for newlines in VS Code, Cursor, Devin Desktop, Alacritty, or Zed.
<code>/theme</code>	Built-in	Change the color theme.
<code>/tui [default fullscreen]</code>	Built-in	Set the terminal UI renderer and relaunch into it with your conversation intact. <code>fullscreen</code> enables the flicker-free alt-screen renderer.
<code>/ultraplan <prompt></code>	Built-in	Removed.
<code>/ultrareview [PR or branch]</code>	Built-in	Run a deep, multi-agent code review in a cloud sandbox with ultrareview.
<code>/update-config [request]</code>	Skill	Describe a settings change, such as allowing a command, setting an environment variable, or adding a hook, and Claude edits the matching <code>settings.json</code> file.
<code>/upgrade</code>	Built-in	Open the upgrade page in your browser to switch to a higher plan tier.
<code>/usage</code>	Built-in	Show session cost, plan usage limits, and activity stats.

<code>/usage-credits</code>	Built-in	Configure usage credits, or request them from your admin, when you hit a limit.
<code>/verify</code>	Skill	Confirm a code change does what it should by building your project's app, running it, and observing the result, rather than relying on tests or type checks.
<code>/vim</code>	Built-in	Removed in v2.1.92. To toggle between Vim and Normal editing modes, use <code>/config</code> → Editor mode.
<code>/voice</code> <code>[hold tap off]</code>	Built-in	Toggle voice dictation, or enable it in a specific mode.
<code>/web-setup</code>	Built-in	Connect your GitHub account for cloud sessions using your local <code>gh</code> CLI credentials.
<code>/workflow-authoring</code>	Skill	Load the reference for writing dynamic workflow scripts: the script API, resume behavior, quality patterns, and worked examples.
<code>/workflows</code>	Built-in	Open the workflow progress view to watch, pause, resume, or save running and completed workflows.

i NOTE

A few rows say “Removed.” They’re kept here because older tutorials still mention them. For example, `/vim` was replaced by **Editor mode** in `/config`.

Command-Line Flags and Shortcuts Reference

This chapter is the companion to the slash-command reference. It covers what you can type **in your terminal before** Claude Code starts, and every keyboard shortcut you can press **inside** it. The tables were generated from Anthropic's CLI reference (code.claude.com/docs/en/cli-reference) and interactive-mode page (code.claude.com/docs/en/interactive-mode) in October 2026.

IN THIS CHAPTER

- Every `claude ...` subcommand
- Every startup flag, with the ones beginners need marked below
- Every keyboard shortcut and line-editing key

31.1 The ten flags beginners actually use

FLAG	WHY YOU'D USE IT
<code>-c</code> / <code>--continue</code>	Pick up exactly where you left off in this folder
<code>-r</code> / <code>--resume</code>	Choose an older conversation
<code>-p</code> / <code>--print</code>	Ask one question, print the answer, exit, which is great for scripts
<code>--model</code>	Pick the model for this session only
<code>--permission-mode plan</code>	Start in plan mode, so nothing gets edited until you approve
<code>--add-dir ../other</code>	Let Claude also work in a second folder
<code>--effort high</code>	Ask for deeper thinking this session
<code>--debug</code>	Write a detailed log when something misbehaves
<code>--chrome</code>	Turn on browser control for web testing
<code>--dangerously-skip-permissions</code>	Skip all prompts. Only use it in a throwaway container or VM (see Vol. 8)

⚠ DANGER

Flags with “dangerously” in their name remove safety checks. On your everyday computer, use plan mode, accept-edits mode, or allow rules instead.

31.2 Terminal subcommands

COMMAND	WHAT IT DOES
<code>claude</code>	Start interactive session.
<code>claude "query"</code>	Start interactive session with initial prompt.
<code>claude -p "query"</code>	Query via SDK, then exit.
<code>claude -c</code>	Continue most recent conversation in current directory.
<code>claude -c -p "query"</code>	Continue via SDK.
<code>claude -r "<session>" "query"</code>	Resume session by ID or name.
<code>claude update</code>	Update to latest version.
<code>claude gateway</code>	Start the self-hosted Claude apps gateway server, for administrators deploying SSO and policy in front of Claude Code on Amazon Bedrock, Google Cloud's Agent Platform, or Microsoft Foundry.
<code>claude install [version]</code>	Install or reinstall the native binary.
<code>claude auth login</code>	Sign in to your Anthropic account.

<code>claude auth logout</code>	Log out from your Anthropic account.
<code>claude auth status</code>	Show authentication status as JSON. Use <code>--text</code> for human-readable output.
<code>claude agents</code>	Open agent view to monitor and dispatch parallel background sessions.
<code>claude attach <id name></code>	Attach to a background session in this terminal.
<code>claude auto-mode defaults</code>	Print the built-in auto mode classifier rules as JSON. Use <code>claude auto-mode config</code> to see your effective config with settings applied. <code>--label <prefix></code> prints only the rules whose label starts...
<code>claude auto-mode reset</code>	Restore the default auto mode configuration by removing the <code>autoMode</code> section from your user settings file.
<code>claude daemon logs</code>	Follow the background-session supervisor's log file, <code>~/.claude/daemon.log</code> , printing new lines as they arrive until you press <code>Ctrl+C</code> .
<code>claude daemon run</code>	Run the background-session supervisor in the foreground of this terminal, printing its log.
<code>claude daemon status</code>	Print the background-session supervisor's state, version, socket directory, and worker count for diagnostics.
<code>claude daemon stop --any</code>	Stop the background-session supervisor and the sessions it hosts.
<code>claude doctor</code>	Print read-only installation and settings diagnostics from the terminal without starting a session, including install health, settings-file validation errors, and Remote Control eligibility.
<code>claude import [source]</code>	Start an interactive session that runs <code>/import</code> to bring configuration from other coding agents into Claude Code.
<code>claude logs <id name></code>	Print recent output from a background session.
<code>claude mcp</code>	Configure Model Context Protocol (MCP) servers.
<code>claude mcp login <name></code>	Run a configured MCP server's OAuth flow without opening the interactive <code>/mcp</code> panel.
<code>claude mcp logout <name></code>	Clear stored OAuth credentials for an MCP server.
<code>claude plugin</code>	Manage Claude Code plugins.
<code>claude purge [path]</code>	Delete all local Claude Code state for a project: transcripts, task lists, debug logs, file-edit history, prompt history lines, and the project's entry in <code>~/.claude.json</code> .
<code>claude remote- control</code>	Start a Remote Control server to control Claude Code from Claude.ai or the Claude app.
<code>claude respawn <id></code>	Restart a background session, running or stopped, with its conversation intact.
<code>claude rm <id></code>	Remove a background session from the list.
<code>claude self- hosted-runner</code>	Start a runner process that registers this machine or container with a self-hosted environment and hosts Claude Code cloud sessions on your infrastructure.
<code>claude setup- token</code>	Generate a long-lived OAuth token for CI and scripts.
<code>claude stop <id></code>	Stop a background session.
<code>claude ultrareview [target]</code>	Run ultrareview non-interactively.

31.3 Every startup flag

FLAG	WHAT IT DOES
<code>--add-dir</code>	Add additional working directories for Claude to read and edit files.
<code>--advisor <model></code>	Enable the server-side advisor tool for this session with a model alias, <code>fable</code> , <code>opus</code> , or <code>sonnet</code> , or a full model ID. Takes precedence over the <code>advisorModel</code> ...
<code>--agent</code>	Specify an agent for the current session (overrides the <code>agent</code> setting).
<code>--agents</code>	Define custom subagents dynamically via JSON. Accepts the fields listed for CLI-defined subagents.
<code>--allow-dangerously-skip-permissions</code>	Add <code>bypassPermissions</code> to the <code>Shift+Tab</code> mode cycle without starting in it.
<code>--allowedTools</code> , <code>--allowed-tools</code>	Tools that execute without prompting for permission, apart from reads from network paths.
<code>--append-subagent-system-prompt</code>	Append custom text to the end of every subagent's system prompt, nested subagents included, apart from a forked subagent, which reuses the conversation's own prompt.
<code>--append-subagent-system-prompt-file</code>	Load text from a file and append it to subagent system prompts.
<code>--append-system-prompt</code>	Append custom text to the end of the default system prompt.
<code>--append-system-prompt-file</code>	Load additional system prompt text from a file and append to the default prompt.
<code>--autocompact <auto tokens></code>	Set the auto-compact window for this session without changing your saved settings.
<code>--ax-screen-reader</code>	Render screen-reader friendly output: flat text without decorative borders or animations.
<code>--bare</code>	Minimal mode: skip auto-discovery of hooks, skills, custom commands, subagents, installed plugins, MCP servers, auto memory, and CLAUDE.md so scripted calls start faster.
<code>--betas</code>	Beta headers to include in API requests (API key users only).
<code>--bg</code> , <code>--background</code>	Start the session as a background agent and return immediately.
<code>--channels</code>	(Research preview) MCP servers whose channel notifications Claude should listen for in this session.
<code>--chrome</code>	Enable Chrome browser integration for web automation and testing.
<code>--cloud</code>	With a task description, create a new cloud session.
<code>--continue</code> , <code>-c</code>	Load the most recent conversation in the current directory, including a background session that has finished; opening finished background sessions requires Claude...
<code>--dangerously-load-development-channels</code>	Enable channels that are not on the approved allowlist, for local development.
<code>--dangerously-skip-permissions</code>	Skip permission prompts.
<code>--debug</code>	Enable debug mode with optional category filtering, such as <code>--debug='mcp, startup'</code> or <code>--debug='!lp'</code> .
<code>--debug-file <path></code>	Write debug logs to a specific file path.

<code>--desktop</code>	Open the Claude Desktop app on the current directory and exit without starting a session in the terminal.
<code>--disable-slash-commands</code>	Disable all skills and commands for this session.
<code>--disallowedTools</code> , <code>--disallowed-tools</code>	Deny rules.
<code>--effort</code>	Set the effort level for the current session.
<code>--enable-auto-mode</code>	Removed in v2.1.111. Auto mode is now in the <code>Shift+Tab</code> cycle by default; use <code>--permission-mode auto</code> to start in it.
<code>--environment <environment-id></code>	Create a new cloud session that runs on the self-hosted environment with the given ID. Environment IDs start with <code>ccpool_</code> .
<code>--exclude-dynamic-system-prompt-sections</code>	Move per-user context, such as the auto memory location, out of the system prompt and into the first user message.
<code>--exec</code>	Run a shell command as a PTY-backed background job instead of starting a Claude session.
<code>--fallback-model</code>	Enable automatic fallback to the specified model(s) when the primary model is overloaded or not available, for example a retired model.
<code>--fork-session</code>	When resuming, create a new session ID instead of reusing the original (use with <code>--resume</code> or <code>--continue</code>).
<code>--forward-subagent-text</code>	Emit subagent text and thinking blocks in the output stream as <code>assistant</code> and <code>user</code> messages with <code>parent_tool_use_id</code> set, so you can reconstruct each subagent's...
<code>--from-pr</code>	Open the session picker filtered to sessions linked to a specific pull request.
<code>--ide</code>	Automatically connect to IDE on startup if exactly one valid IDE is available.
<code>--init</code>	Run Setup hooks with the <code>init</code> matcher before the session (print mode only).
<code>--init-only</code>	Run Setup and <code>SessionStart</code> hooks, then exit without starting a conversation.
<code>--include-hook-events</code>	Include hook lifecycle events in the output stream. <code>SessionStart</code> and <code>Setup</code> hook events are always included and don't need this flag.
<code>--include-partial-messages</code>	Include partial streaming events in output.
<code>--input-format</code>	Specify input format for print mode (options: <code>text</code> , <code>stream-json</code>).
<code>--json-schema</code>	Get validated JSON output matching a JSON Schema after the agent completes its workflow (print mode only).
<code>--maintenance</code>	Run Setup hooks with the <code>maintenance</code> matcher before the session (print mode only).
<code>--max-budget-usd</code>	Stop the run once estimated spend on API calls reaches this amount (print mode only).
<code>--max-turns</code>	Limit the number of agentic turns (print mode only).
<code>--mcp-config</code>	Load MCP servers from JSON files or strings (space-separated).
<code>--model</code>	Sets the model for the current session with a model alias such as <code>sonnet</code> , <code>opus</code> , <code>haiku</code> , or <code>fable</code> , or a model's full name.
<code>--name</code> , <code>-n</code>	Set a display name for the session, shown in <code>/resume</code> and the terminal title.
<code>--no-chrome</code>	Disable Chrome browser integration for this session.
<code>--no-session-persistence</code>	Disable session persistence so sessions are not saved to disk and cannot be resumed.
<code>--output-format</code>	Specify output format for print mode (options: <code>text</code> , <code>json</code> , <code>stream-json</code>).
<code>--permission-mode</code>	Begin in a specified permission mode.

<code>--permission-prompt-tool</code>	Specify an MCP tool to handle permission prompts in non-interactive mode.
<code>--permission-prompts</code>	Set who answers permission prompts in print mode.
<code>--plugin-dir</code>	Load a plugin from a directory or <code>.zip</code> archive, or several from a folder of plugins, for this session only.
<code>--plugin-url</code>	Fetch a plugin <code>.zip</code> archive from a URL for this session only.
<code>--print</code> , <code>-p</code>	Print response without interactive mode (see Agent SDK documentation for programmatic usage details).
<code>--prompt-suggestions</code>	Emit a <code>prompt_suggestion</code> message with a predicted next user prompt after each turn that generates one; very short conversations can produce none.
<code>--ref <branch></code>	With <code>--environment</code> , base the new session's checkout on a named ref instead of local <code>HEAD</code> .
<code>--remote</code>	Deprecated alias for <code>--cloud</code> , including the existing-session form.
<code>--remote-control</code> , <code>--rc</code>	Start an interactive session with Remote Control enabled so you can also control it from claude.ai or the Claude app.
<code>--remote-control-session-name-prefix <prefix></code>	Prefix for auto-generated Remote Control session names when no explicit name is set.
<code>--replay-user-messages</code>	Re-emit user messages from stdin back on stdout for acknowledgment.
<code>--restricted</code>	Start in restricted mode.
<code>--resume</code> , <code>-r</code>	Resume a specific session by ID or name, or show an interactive picker to choose a session.
<code>--safe-mode</code>	Start with all customizations disabled to troubleshoot a broken configuration: CLAUDE.md, skills, plugins, hooks, MCP servers, custom commands and agents, output...
<code>--session-id</code>	Use a specific session ID for the conversation (must be a valid UUID).
<code>--setting-sources</code>	Comma-separated list of setting sources to load (<code>user</code> , <code>project</code> , <code>local</code>).
<code>--settings</code>	Path to a settings JSON file or an inline JSON string.
<code>--strict-mcp-config</code>	Only use MCP servers from <code>--mcp-config</code> , ignoring all other MCP configurations.
<code>--system-prompt</code>	Replace the entire system prompt with custom text.
<code>--system-prompt-file</code>	Load system prompt from a file, replacing the default prompt.
<code>--system-prompt-snapshot</code>	Pass <code>off</code> to rebuild the system prompt on every request instead of reusing the prompt recorded on the conversation's first request, for example while you iterate on...
<code>--teleport</code>	Resume a cloud session in your local terminal.
<code>--teammate-mode</code>	Set how agent team teammates display: <code>in-process</code> (default), <code>auto</code> , <code>tmux</code> , or <code>iterm2</code> .
<code>--tmux</code>	Create a tmux session for the worktree.
<code>--tools</code>	Restrict which built-in tools Claude can use.
<code>--verbose</code>	Enable verbose logging, shows full turn-by-turn output.
<code>--version</code> , <code>-v</code>	Output the version number.
<code>--worktree</code> , <code>-w</code>	Start Claude in an isolated git worktree at <code><repo>/claude/worktrees/<name></code> .
<code>--system-prompt</code>	Replaces the entire default prompt.
<code>--system-prompt-file</code>	Replaces with file contents.
<code>--append-system-prompt</code>	Appends to the default prompt.

<code>--append-system-prompt-file</code>	Appends file contents to the default prompt.
<code>--system-prompt-snapshot</code>	With <code>off</code> , rebuilds the prompt on every request.

31.4 Every keyboard shortcut

KEYS	WHAT IT DOES
<code>Ctrl+C</code>	Interrupt, or clear input.
<code>Ctrl+X Ctrl+K</code>	Stop all running background subagents in this session, and turn off artifact auto-replies for the rest of it.
<code>Ctrl+D</code>	Exit Claude Code session.
<code>Ctrl+G</code> or <code>Ctrl+X Ctrl+E</code>	Open in default text editor.
<code>Ctrl+L</code>	Redraw the screen.
<code>Ctrl+O</code>	Toggle transcript viewer.
<code>Ctrl+R</code>	Reverse search command history.
<code>Ctrl+V</code> or <code>Cmd+V</code> (iTerm2) or <code>Alt+V</code> (Windows and WSL)	Paste image from clipboard.
<code>Ctrl+B</code>	Background running tasks.
<code>Ctrl+T</code>	Toggle Claude's task checklist.
<code>Ctrl+S</code>	Stash or restore prompt.
<code>Ctrl+Z</code>	Suspend Claude Code.
<code>Left/Right arrows</code>	Cycle through dialog tabs.
<code>Tab</code>	Accept an autocomplete suggestion, or add a comment to a permission answer.
<code>Up/Down arrows</code> or <code>Ctrl+P</code> / <code>Ctrl+N</code>	Move cursor or navigate command history.
<code>Esc</code>	Interrupt Claude, or close a dialog.
<code>Esc + Esc</code>	Clear input draft, or rewind.
<code>Ctrl+Enter</code> or <code>Ctrl+X Ctrl+S</code>	Send queued messages now.
<code>Shift+Tab</code> , or <code>Alt+M</code> on Windows when the Node or Bun runtime doesn't enable VT input mode	Cycle permission modes.
<code>Option+P</code> (macOS) or <code>Alt+P</code> (Windows/Linux)	Switch model.
<code>Option+T</code> (macOS) or <code>Alt+T</code> (Windows/Linux)	Toggle extended thinking.
<code>Option+O</code> (macOS) or <code>Alt+O</code> (Windows/Linux)	Toggle fast mode.
<code>Ctrl+A</code>	Move cursor to start of current line.
<code>Ctrl+E</code>	Move cursor to end of current line.
<code>Ctrl+K</code>	Delete to end of line.
<code>Ctrl+U</code>	Delete from cursor to line start.
<code>Ctrl+W</code>	Delete back to previous whitespace.
<code>Ctrl+Y</code>	Paste deleted text.
<code>Alt+Y</code> (after <code>Ctrl+Y</code>)	Cycle paste history.

Alt+B	Move cursor back one word.
Alt+F	Move cursor forward one word.
Alt+D	Delete to end of word.
Ctrl+_ or Ctrl+Shift+-	Undo last input edit.
Ctrl+T	Toggle syntax highlighting for code blocks.
/ at start	Command or skill.
! at start	Shell mode.
@	File path mention.
:	Emoji shortcode.
? on empty input	Toggle the shortcut help panel.
?	Toggle the keyboard shortcut help panel.
{ / }	Jump to the previous or next user prompt, like vim paragraph motion.
Ctrl+E	Toggle show all content.
[	Write the full conversation to your terminal's native scrollbar so Cmd+F , tmux copy mode, and other native tools...
v	Write the conversation to a temporary file and open it in \$VISUAL or \$EDITOR .
q , Ctrl+C , Esc	Exit transcript view.
Esc or Ctrl+[	Enter NORMAL mode.
i	Insert before cursor.
I	Insert at beginning of line.
a	Insert after cursor.
A	Insert at end of line.
o	Open line below.
O	Open line above.
v	Start character-wise visual selection.
V	Start line-wise visual selection.
h / j / k / l	Move left/down/up/right.
Space	Move right.
w	Next word.
e	End of word.
b	Previous word.
0	Beginning of line.
\$	End of line.
^	First non-blank character.
gg	Beginning of input.

G	Beginning of last line.
f{char}	Jump to next occurrence of character on the current line.
F{char}	Jump to previous occurrence of character on the current line.
t{char}	Jump to just before next occurrence of character on the current line.
T{char}	Jump to just after previous occurrence of character on the current line.
;	Repeat last f/F/t/T motion.
,	Repeat last f/F/t/T motion in reverse.
%	Jump to the matching () , [] , or { } bracket, starting from the next bracket on the current line when the cursor...
/	Open reverse history search, same as Ctrl+R .
x	Delete character.
r{char}	Replace character under cursor with {char} .
dd	Delete line.
D	Delete to end of line.
dw / de / db	Delete word/to end/back.
df{char} / dt{char}	Delete to and including, or up to, the next occurrence of a character on the current line.
dj / dk	Delete the current line and the line below or above.
dgg / dG	Delete from the current line to the first or last line.
d0 / c0 / y0	Delete, change, or yank from the cursor back to the beginning of the line.
d% / c% / y%	Delete, change, or yank from the cursor through the bracket that % jumps to.
cc	Change line.
C	Change to end of line.
cw / ce / cb	Change word/to end/back.
s	Substitute character: delete the character under the cursor and enter INSERT mode.
S	Substitute line: clear the line and enter INSERT mode.
yy / Y	Yank (copy) line.
yw / ye / yb	Yank word/to end/back.
p	Paste after cursor.
P	Paste before cursor.
>>	Indent line.
<<	Dedent line.
J	Join lines.
u	Undo.
.	Repeat last change.
iw / aw	Inner/around word.
iW / aW	Inner/around WORD (whitespace-delimited).

<code>i" / a"</code>	Inner/around double quotes.
<code>i' / a'</code>	Inner/around single quotes.
<code>i(/ a(</code>	Inner/around parentheses.
<code>i[/ a[</code>	Inner/around brackets.
<code>i{ / a{</code>	Inner/around braces.
<code>d / x</code>	Delete selection.
<code>y</code>	Yank selection.
<code>c / s</code>	Change selection.
<code>p</code>	Replace selection with register contents.
<code>r{char}</code>	Replace every selected character with <code>{char}</code> .
<code>~ / u / U</code>	Toggle, lowercase, or uppercase selection.
<code>> / <</code>	Indent or dedent selected lines.
<code>J</code>	Join selected lines.
<code>o</code>	Swap cursor and anchor.
<code>iw / aw / i" / ...</code>	Select a text object.
<code>v / V</code>	Toggle between character-wise and line-wise, or exit.
<code>Space</code> , <code>Enter</code> , <code>Escape</code>	Dismiss the answer and return to the prompt.
<code>Up</code> / <code>Down</code>	Scroll the answer.
<code>Shift+Left</code> / <code>Shift+Right</code>	Step between this answer and your earlier <code>/btw</code> answers. <code>Shift+Left</code> moves to older answers and <code>Shift+Right</code> ...
<code>c</code>	Copy the answer to your clipboard as raw Markdown.
<code>f</code>	Start a forked subagent that inherits the parent conversation plus this question and answer, so it can continue with...
<code>x</code>	Clear the list of earlier <code>/btw</code> exchanges shown above the current answer.

Real-World Workflows, Step by Step

Knowing the commands is half the job. This chapter strings them into complete, repeatable workflows you can follow with the book open next to your keyboard.

IN THIS CHAPTER

- Fix a bug from an error message
- Add a feature safely with plan mode
- Understand a project you've never seen
- Write and run tests
- Ship a pull request
- Run long tasks in the background
- Automate with `claude -p`

32.1 Workflow 1: Fix a bug from an error message

1

Reproduce it.

Run the program and copy the full error, including the lines above it.

2

Start Claude in the project folder.

Type `claude`.

3 Paste the error with context.**PROMPT**

When I run `npm start` I get the error below. Find the cause, explain it in plain English, then propose the smallest fix. Don't change anything else.
[paste error]

4 Read the explanation before approving.

If it doesn't make sense to you, ask "why?" until it does.

5 Approve the edit, then ask Claude to rerun the command.

A fix isn't done until the error is gone.

6 Commit.

Ask: "Commit this with a clear message."

32.2 Workflow 2: Add a feature with plan mode

1 Enter plan mode.

Press `Shift` + `Tab` until the footer says *plan*, or type `/plan add a dark-mode toggle`.

2 Let Claude research.

It reads files but can't edit them.

3 Review the plan.

Push back on anything too big. Press `Ctrl` + `G` to edit the plan in your text editor.

4 Approve.

Claude switches out of plan mode and starts working.

5 Check in at milestones.

Ask for a short status after each step.

6 Review with `/diff`, then `/code-review`.**7 Compact or clear.**

If the session is long and you're continuing, use `/compact`. If you're done, commit and `/clear`.

32.3 Workflow 3: Learn an unfamiliar project

PROMPT

Give me a beginner-friendly tour of this project: what it does, how to run it, the main folders, and the five files I should read first. Don't edit anything.

Follow up with “trace what happens when a user clicks *Sign up*.” This **trace** question is the fastest way to learn real code. When you're done, run `/init` so the next session starts with what you learned.

32.4 Workflow 4: Tests first

- 1 **Ask for tests that describe the behaviour you want**, and tell Claude they should fail for now.
- 2 **Run them and confirm they fail.**
This proves the tests actually check something.
- 3 **Ask Claude to make the tests pass without changing the tests.**
- 4 **Run the full test suite** so you know nothing else broke.

TIP

Saying “don't modify the tests” stops a common shortcut, where an AI “fixes” a failing test by weakening it.

32.5 Workflow 5: Ship a pull request

- 1 **Branch first.**
“Create a branch called `feature/contact-form`.”
- 2 **Do the work** (Workflow 2).
- 3 **Review.**
`/diff` → `/code-review` → `/security-review`.
- 4 **Commit and push.**
“Commit in logical chunks and push the branch.”
- 5 **Open the PR.**
“Open a pull request with a summary and testing notes.” This uses the `gh` tool (Vol. 2).

6

Address feedback.

Paste reviewer comments back to Claude, or try `/autofix-pr` if your plan supports cloud sessions.

32.6 Workflow 6: Long tasks in the background

Press `Ctrl + B` to send a long-running command, like a dev server or test run, to the background. Use `/tasks` to check on it. To detach the whole session and get your terminal back, use `/background`. Then watch it from your shell with `claude agents`. `/subtask` hands a side job to a helper that sees the full conversation.

32.7 Workflow 7: Scripts with `claude -p`

`-p` (print mode) runs one request with no interactive screen. That makes it ideal for scripts and scheduled jobs.

TERMINAL

```
claude -p "Summarize what changed in git today in five bullets" > today.md
git diff | claude -p "Write a commit message for this diff"
```

⚠ WARNING

In print mode nobody is around to approve prompts. Give it only the permissions it needs, with `--allowedTools` or allow rules, and never broad shell access on a machine you care about.

32.8 Daily routine checklist

- ☐ `git status` is clean before you start
- ☐ Start with `claude -c` to continue, or `claude` for a fresh task
- ☐ One task per conversation
- ☐ Use plan mode for anything touching more than two files
- ☐ `/context` when things feel slow, then `/compact` or `/clear`
- ☐ `/diff` before every commit
- ☐ Commit small and often
- ☐ End by asking for a summary of what changed and what's next

Troubleshooting and FAQ

When something goes wrong, work through this chapter from top to bottom. Most problems are one of a dozen common causes.

IN THIS CHAPTER

- A first-aid routine for any problem
- Install and sign-in fixes
- When Claude gets slow, forgetful, or stuck
- Permission and tool problems
- Frequently asked questions

33.1 First-aid routine

- 1 Read the whole message.**
The fix is often in the last line.
- 2 Run `claude doctor` in your terminal, or `/doctor` inside a session.**
- 3 Update.**
Run `claude update`, then restart.
- 4 Check sign-in.**
Run `claude auth status`.
- 5 Turn on logging.**
Restart with `claude --debug`, or run `/debug describe the problem`.
- 6 Search the official docs at code.claude.com/docs.**
- 7 Report it.**
If it's a real bug, use `/bug`. It shows you what will be sent and asks you to confirm first.

33.2 Symptom table

SYMPTOM	LIKELY CAUSE	FIX
<code>command not found: claude</code>	The install folder isn't on your PATH	Open a new terminal. Run <code>claude doctor</code> . Reinstall with the native installer
Browser sign-in never finishes	Pop-up blocked or wrong browser profile	Copy the sign-in URL into the right browser. Try <code>claude auth login</code>
"Usage limit reached"	Your plan's limit for this period is used up	<code>/usage</code> to see when it resets. <code>/rate-limit-options</code> lists choices. Switch to a lighter <code>/model</code>
Answers get vague or repetitive	The context window is full of old material	<code>/context</code> , then <code>/compact</code> or <code>/clear</code>
Claude keeps redoing a mistake	The bad attempt is still in context	<code>/rewind</code> to before the mistake. Re-prompt with the missing detail
Too many permission prompts	No allow rules yet	<code>/permissions</code> to add rules, or <code>/fewer-permission-prompts</code>
Claude edits files you didn't want touched	The request was too vague	Use plan mode. Add "only change X" to the prompt. Add deny rules
MCP server shows "failed"	Bad config or expired sign-in	<code>/mcp</code> to reconnect or sign in. Check the command and path
A skill doesn't appear in <code>/</code>	Wrong folder, bad YAML, or a name clash	Check <code>.claude/skills/<name>/SKILL.md</code> . Run <code>/reload-skills</code>
Hooks don't fire	Wrong event name or matcher	Run <code>/hooks</code> to inspect them, then <code>--debug</code>
Garbled screen	The terminal display got out of sync	<code>Ctrl</code> + <code>L</code> to redraw
Pasting an image does nothing	Your terminal uses a different key	Try <code>Ctrl</code> + <code>V</code> , <code>Cmd</code> + <code>V</code> (iTerm2), or <code>Alt</code> + <code>V</code> (Windows)
<code>Shift</code> + <code>Enter</code> sends instead of adding a new line	Your terminal doesn't support it	Run <code>/terminal-setup</code> , or use <code>\</code> + <code>Enter</code> or <code>Ctrl</code> + <code>J</code>

33.3 Frequently asked questions

Does `/clear` delete my files? No. It only clears the conversation. Your files and git history stay as they are.

Is `/compact` the same as `/clear`? No. `/compact` keeps a summary and stays on the same task. `/clear` starts from nothing.

Can Claude see my whole computer? By default it works in the folder you started it in, plus any folders you add with `/add-dir`. It asks before it does anything outside its allowed permissions.

Does it remember me between sessions? Only through `CLAUDE.md`, auto memory, and the conversations you resume. A fresh session doesn't automatically know what you discussed yesterday.

What does it cost? It depends on your plan and changes over time. See the Costs chapter, and run `/usage` for your own numbers. This manual doesn't quote prices, because they change.

Is my code used for training? That depends on your account type and settings. Check Anthropic's current privacy settings for your plan. Don't take a third-party guide's word for it.

Can I use it on my phone? You can use Claude Code on the web and in the Claude app, and `/remote-control` lets you drive a session running on your computer. See the "Beyond the Terminal" chapter.

Hooks, Settings, and Environment Reference

This is the deep reference for people who want to automate Claude Code. You don't need it on day one. The tables were generated from Anthropic's hooks reference (code.claude.com/docs/en/hooks) and environment-variable reference (code.claude.com/docs/en/env-vars) in October 2026.

IN THIS CHAPTER

- Every hook event and when it fires
- A worked hook example
- Where settings files live and which one wins
- Every documented environment variable

34.1 Settings files and precedence

FILE	SCOPE	COMMIT TO GIT?
Managed settings (set by IT)	Whole organisation, can't be overridden	Not applicable
<code>.claude/settings.local.json</code>	You, in this project	No, keep it private
<code>.claude/settings.json</code>	Everyone on this project	Yes
<code>~/.claude/settings.json</code>	You, in every project	Not applicable

When two files disagree, the more specific one wins: managed, then command-line flags, then local project, then shared project, then user. **Deny** rules beat allow rules wherever they come from.

34.2 Hook events

A **hook** is a command that runs automatically when an event fires. Use `/hooks` to see the hooks you have, or `/update-config` to ask Claude to add one.

EVENT	FIRES...
<code>SessionStart</code>	When a session begins or resumes.
<code>Setup</code>	When you start Claude Code with <code>--init-only</code> , or with <code>--init</code> or <code>--maintenance</code> in <code>-p</code> mode.
<code>UserPromptSubmit</code>	When a prompt is submitted, before Claude processes it.
<code>UserPromptExpansion</code>	When a user-typed command expands into a prompt, before it reaches Claude.
<code>PreToolUse</code>	Before a tool call executes.
<code>PermissionRequest</code>	When a tool call needs a permission decision.
<code>PermissionDenied</code>	When auto mode denies a tool call, including denials without a classifier verdict.
<code>PostToolUse</code>	After a tool call succeeds.
<code>PostToolUseFailure</code>	After a tool call fails.
<code>PostToolBatch</code>	After a full batch of parallel tool calls resolves, before the next model call.
<code>Notification</code>	When Claude Code sends a notification.
<code>MessageDisplay</code>	While assistant message text is displayed.
<code>SubagentStart</code>	When a subagent is spawned.
<code>SubagentStop</code>	When a subagent finishes.
<code>TaskCreated</code>	When a task is being created via <code>TaskCreate</code> .
<code>TaskCompleted</code>	When a task is being marked as completed.
<code>Stop</code>	When Claude finishes responding.
<code>StopFailure</code>	When the turn ends due to an API error.
<code>TeammateIdle</code>	When an agent team teammate is about to go idle.
<code>InstructionsLoaded</code>	When a <code>CLAUDE.md</code> or <code>.claude/rules/*.md</code> file is loaded into context.
<code>ConfigChange</code>	When a configuration file changes during a session.
<code>CwdChanged</code>	When the working directory changes, for example when Claude executes a <code>cd</code> command.
<code>DirectoryAdded</code>	When a working directory is added mid-session via <code>/add-dir</code> or the SDK <code>register_repo_root</code> control request.
<code>FileChanged</code>	When a watched file changes on disk.
<code>WorktreeCreate</code>	When a worktree is being created via <code>--worktree</code> , <code>isolation: "worktree"</code> , or for a background session.
<code>WorktreeRemove</code>	When a worktree that a <code>WorktreeCreate</code> hook created is being removed.
<code>PreCompact</code>	Before context compaction.
<code>PostCompact</code>	After context compaction completes.
<code>PreModelSwitch</code>	Before Claude Code applies a model switch that you or a client requested.

PostModelSwitch	After the session's model changes, including changes Claude Code makes on its own, such as restoring the model when you resume a session.
Elicitation	When an MCP server requests user input during a tool call.
ElicitationResult	After a user responds to an MCP elicitation, before the response is sent back to the server.
SessionEnd	When a session terminates.

34.3 Worked example: format every file Claude edits

```
.CLAUDE/SETTINGS.JSON

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "jq -r '.tool_input.file_path' | xargs npx prettier --write" }
        ]
      }
    ]
  }
}
```

⚠ WARNING

Hooks run with **your** user permissions and no prompt. Only add hooks you've read and understand, and be careful with hooks that come inside repositories you cloned.

The hook receives details of the tool call as JSON on its input. `jq` pulls out the edited file's path, so Prettier formats only that file. You need `jq` and Prettier installed.

34.4 Environment variables

Set these in your shell or in the `env` block of `settings.json`. Most people never need more than a handful. Search this table when a guide or error message mentions one.

VARIABLE	WHAT IT DOES
ANTHROPIC_API_KEY	API key sent as <code>X-API-Key</code> header.
ANTHROPIC_AUTH_TOKEN	Custom value for the <code>Authorization</code> header (the value you set here will be prefixed with <code>Bearer </code>).
ANTHROPIC_AWS_API_KEY	Workspace API key for Claude Platform on AWS, generated in the AWS Console.
ANTHROPIC_AWS_BASE_URL	Override the Claude Platform on AWS endpoint URL. Use for custom regions or when routing through an LLM...

<code>ANTHROPIC_AWS_WORKSPACE_ID</code>	Required for Claude Platform on AWS. Sent on every request as the <code>anthropic-workspace-id</code> header.
<code>ANTHROPIC_BASE_URL</code>	Override the API endpoint to route requests through a proxy or gateway.
<code>ANTHROPIC_BEDROCK_BASE_URL</code>	Override the Amazon Bedrock endpoint URL. Use for custom Amazon Bedrock endpoints or when routing through...
<code>ANTHROPIC_BEDROCK_MANTLE_BASE_URL</code>	Override the Amazon Bedrock Mantle endpoint URL. See Mantle endpoint.
<code>ANTHROPIC_BEDROCK_REGION_PREFIX</code>	Cross-region inference profile prefix (<code>us</code> , <code>eu</code> , <code>apac</code> , <code>jp</code> , <code>au</code> , or <code>global</code>) Claude Code tries...
<code>ANTHROPIC_BEDROCK_SERVICE_TIER</code>	Amazon Bedrock service tier (<code>default</code> , <code>flex</code> , or <code>priority</code>).
<code>ANTHROPIC_BETAS</code>	Comma-separated list of additional <code>anthropic-beta</code> header values to include in API requests.
<code>ANTHROPIC_CUSTOM_HEADERS</code>	Custom headers to add to requests (<code>Name: Value</code> format, newline-separated for multiple headers).
<code>ANTHROPIC_CUSTOM_MODEL_OPTION</code>	Model ID to add as a custom entry in the <code>/model</code> picker.
<code>ANTHROPIC_CUSTOM_MODEL_OPTION_DESCRIPTION</code>	Display description for the custom model entry in the <code>/model</code> picker.
<code>ANTHROPIC_CUSTOM_MODEL_OPTION_NAME</code>	Display name for the custom model entry in the <code>/model</code> picker.
<code>ANTHROPIC_CUSTOM_MODEL_OPTION_SUPPORTED_CAPABILITIES</code>	Comma-separated list of capabilities the custom model supports, for example <code>effort,thinking</code> .
<code>ANTHROPIC_DEFAULT_FABLE_MODEL</code>	Model ID that the <code>fable</code> alias resolves to, and the ID Claude Code recognizes as a Fable model for...
<code>ANTHROPIC_DEFAULT_FABLE_MODEL_DESCRIPTION</code>	Display description for the pinned Fable model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_FABLE_MODEL_NAME</code>	Display name for the pinned Fable model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_FABLE_MODEL_SUPPORTED_CAPABILITIES</code>	Comma-separated list of capabilities the pinned Fable model supports, for example <code>effort,thinking</code> .
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Model ID that the <code>haiku</code> alias resolves to, also used for background functionality.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL_DESCRIPTION</code>	Display description for the pinned Haiku model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL_NAME</code>	Display name for the pinned Haiku model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL_SUPPORTED_CAPABILITIES</code>	Comma-separated list of capabilities the pinned Haiku model supports, for example <code>effort,thinking</code> .
<code>ANTHROPIC_DEFAULT_MODEL</code>	Model that new sessions start on by default.
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	Model ID that the <code>opus</code> alias resolves to, and that <code>opusplan</code> uses while Plan Mode is active.
<code>ANTHROPIC_DEFAULT_OPUS_MODEL_DESCRIPTION</code>	Display description for the pinned Opus model in the <code>/model</code> picker.

<code>ANTHROPIC_DEFAULT_OPUS_MODEL_NAME</code>	Display name for the pinned Opus model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_OPUS_MODEL_SUPPORTED_CAPABILITIES</code>	Comma-separated list of capabilities the pinned Opus model supports, for example <code>effort,thinking</code> .
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Model ID that the <code>sonnet</code> alias resolves to, and that <code>opusplan</code> uses when Plan Mode is not active.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL_DESCRIPTION</code>	Display description for the pinned Sonnet model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL_NAME</code>	Display name for the pinned Sonnet model in the <code>/model</code> picker.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL_SUPPORTED_CAPABILITIES</code>	Comma-separated list of capabilities the pinned Sonnet model supports, for example <code>effort,thinking</code> .
<code>ANTHROPIC_FEDERATION_RULE_ID</code>	Federation rule ID for Workload Identity Federation.
<code>ANTHROPIC_FOUNDRY_API_KEY</code>	API key for Microsoft Foundry authentication (see Microsoft Foundry).
<code>ANTHROPIC_FOUNDRY_AUTH_TOKEN</code>	Bearer token for Microsoft Foundry authentication, such as a Microsoft Entra access token.
<code>ANTHROPIC_FOUNDRY_BASE_URL</code>	Full base URL for the Microsoft Foundry resource (for example,...
<code>ANTHROPIC_FOUNDRY_RESOURCE</code>	Microsoft Foundry resource name (for example, <code>my-resource</code>).
<code>ANTHROPIC_MODEL</code>	Name of the model setting to use (see Model Configuration).
<code>ANTHROPIC_ORGANIZATION_ID</code>	Organization ID for Workload Identity Federation.
<code>ANTHROPIC_PROFILE</code>	Name of the Anthropic profile to authenticate with, such as one created by <code>ant auth login</code> or by signing...
<code>ANTHROPIC_SMALL_FAST_MODEL</code>	[DEPRECATED] Name of Haiku-class model for background tasks.
<code>ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION</code>	Override AWS region for the Haiku-class model when using Amazon Bedrock or Amazon Bedrock Mantle.
<code>ANTHROPIC_VERTEX_BASE_URL</code>	Override Google Cloud's Agent Platform endpoint URL. Use for custom Google Cloud's Agent Platform...
<code>ANTHROPIC_VERTEX_PROJECT_ID</code>	GCP project ID that Google Cloud's Agent Platform requests are addressed to.
<code>ANTHROPIC_WORKSPACE_ID</code>	Workspace ID for workload identity federation.
<code>API_FORCE_IDLE_TIMEOUT</code>	Override the 5-minute body idle timeout that aborts a streaming model response when no bytes arrive.
<code>API_TIMEOUT_MS</code>	Timeout for API requests in milliseconds (default: 600000, or 10 minutes; maximum: 2147483647).
<code>AWS_BEARER_TOKEN_BEDROCK</code>	Amazon Bedrock API key for authentication (see Amazon Bedrock API keys).
<code>BASH_DEFAULT_TIMEOUT_MS</code>	Default timeout for a foreground Bash or PowerShell tool command, in milliseconds (default: 120000, or 2...
<code>BASH_MAX_OUTPUT_LENGTH</code>	Maximum number of characters of bash output that Claude Code reads back into a command's result (default:...

<code>BASH_MAX_TIMEOUT_MS</code>	Maximum timeout the model can set for a foreground Bash or PowerShell tool command, in milliseconds...
<code>BETA_TRACING_ENDPOINT</code>	OTLP/HTTP endpoint for detailed beta tracing: with <code>ENABLE_BETA_TRACING_DETAILED=1</code> , logs and traces go...
<code>CCR_FORCE_BUNDLE</code>	Set to <code>1</code> to force <code>claude --cloud</code> to bundle and upload your local repository instead of cloning from...
<code>CLAUDECODE</code>	Set to <code>1</code> in subprocesses Claude Code spawns (Bash and PowerShell tools, tmux sessions, hook commands,...)
<code>CLAUDE_AFK_COUNTDOWN_MS</code>	How many milliseconds before auto-continue the on-screen countdown appears on an unanswered...
<code>CLAUDE_AFK_TIMEOUT_MS</code>	How many milliseconds of idle time before an unanswered <code>AskUserQuestion</code> dialog auto-continues without you.
<code>CLAUDE_AGENT_SDK_DISABLE_BUILTIN_AGENTS</code>	Set to <code>1</code> to disable all built-in subagent types such as Explore and Plan.
<code>CLAUDE_AGENT_SDK_MCP_NO_PREFIX</code>	Set to <code>1</code> to skip the <code>mcp__<server>__</code> prefix on tool names from SDK-created MCP servers.
<code>CLAUDE_ASYNC_AGENT_STALL_TIMEOUT_MS</code>	Stall timeout in milliseconds for subagents.
<code>CLAUDE_AUTOCOMPACT_PCT_OVERRIDE</code>	Set the percentage (1-100) of the auto-compact window at which auto-compaction triggers.
<code>CLAUDE_AUTO_BACKGROUND_TASKS</code>	Set to <code>1</code> to force-enable automatic backgrounding of long-running agent tasks.
<code>CLAUDE_AX_PREPARK_MS</code>	In screen reader mode, how many milliseconds Claude Code waits before it writes a new or changed line.
<code>CLAUDE_AX_SCREEN_READER</code>	Set to <code>1</code> to render screen-reader friendly output: flat text without decorative borders or animations.
<code>CLAUDE_AX_STARTUP_QUIET_MS</code>	In screen reader mode, how many milliseconds Claude Code holds the first interface render after the...
<code>CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR</code>	Return to the original working directory after each Bash or PowerShell command in the main session.
<code>CLAUDE_BYTE_STREAM_IDLE_TIMEOUT_MS</code>	Timeout in milliseconds for the byte-level streaming idle watchdog; when set, it takes precedence over...
<code>CLAUDE_CLIENT_PRESENCE_FILE</code>	Path to a file that an external tool, such as a screen-lock listener, creates when you unlock your screen...
<code>CLAUDE_CODE_ACCESSIBILITY</code>	Set to <code>1</code> to keep the native terminal cursor visible and disable the inverted-text cursor indicator.
<code>CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD</code>	Set to <code>1</code> to load memory files from directories specified with <code>--add-dir</code> .
<code>CLAUDE_CODE_ALT_SCREEN_FULL_REPAINT</code>	Set to <code>1</code> to repaint the entire screen on every frame in fullscreen rendering instead of sending...
<code>CLAUDE_CODE_ALWAYS_ENABLE_EFFORT</code>	Set to <code>1</code> to send the effort parameter with every request, even when Claude Code does not recognize the...
<code>CLAUDE_CODE_API_KEY_HELPER_TTL_MS</code>	Interval in milliseconds at which credentials should be refreshed (when using <code>apiKeyHelper</code>).
<code>CLAUDE_CODE_ARTIFACT_AUTO_OPEN</code>	Set to <code>0</code> to stop Claude Code from opening the browser automatically when a new artifact is published.

<code>CLAUDE_CODE_ARTIFACT_COMMENTS</code>	Set to <code>0</code> to stop Claude reading and replying to comments on an artifact.
<code>CLAUDE_CODE_ARTIFACT_COMMENTS_AUTOREACT</code>	Set to <code>0</code> to stop Claude replying on its own to comments sent to it.
<code>CLAUDE_CODE_ARTIFACT_PUBLISH_EDITS</code>	Set to <code>0</code> to stop Claude from applying small changes to an artifact's file as part of publishing it.
<code>CLAUDE_CODE_ATTRIBUTION_HEADER</code>	Set to <code>0</code> to omit the attribution block, which carries the client version and a prompt fingerprint, from...
<code>CLAUDE_CODE_AUTO_BACKGROUND_WORKER_CHECKIN_SECONDS</code>	Removed in v2.1.283. Use <code>CLAUDE_CODE_WORKER_CHECKIN_SCHEDULE</code> instead.
<code>CLAUDE_CODE_AUTO_COMPACT_WINDOW</code>	Set the auto-compact window in tokens, from <code>100000</code> to <code>1000000</code> .
<code>CLAUDE_CODE_AUTO_CONNECT_IDE</code>	Override automatic IDE connection.
<code>CLAUDE_CODE_AUTO_MODE_SERVER</code>	Controls whether Claude Code asks the server to review auto mode actions.
<code>CLAUDE_CODE_AWS_CHAIN_RESOLVE_TIMEOUT_MS</code>	Time in milliseconds Claude Code waits for the AWS default credential provider chain to produce...
<code>CLAUDE_CODE_BASH_EDIT_DIFF</code>	Set to <code>0</code> to turn off the diff of the files that changed while a Bash command ran, or <code>1</code> to record it in...
<code>CLAUDE_CODE_BG_TASKS_REPORT_RUNNING</code>	Set to <code>0</code> to make a non-interactive session report an idle status to its host at every turn end, even...
<code>CLAUDE_CODE_BRIDGE_SESSION_ID</code>	Set automatically in Bash tool and hook command subprocesses while the session has an active Remote...
<code>CLAUDE_CODE_BS_AS_CTRL_BACKSPACE</code>	Set to <code>0</code> to make Claude Code read the <code>0x08</code> byte, also written <code>^H</code> , as plain Backspace, or <code>1</code> to read...
<code>CLAUDE_CODE_CERT_STORE</code>	Comma-separated list of CA certificate sources for TLS connections. <code>bundled</code> is the Mozilla CA set...
<code>CLAUDE_CODE_CHILD_SESSION</code>	Set to <code>1</code> in subprocesses Claude Code spawns via the Bash, PowerShell, and Monitor tools, hook commands,...
<code>CLAUDE_CODE_CLIENT_CERT</code>	Path to client certificate file for mTLS authentication.
<code>CLAUDE_CODE_CLIENT_KEY</code>	Path to client private key file for mTLS authentication.
<code>CLAUDE_CODE_CLIENT_KEY_PASSPHRASE</code>	Passphrase for encrypted <code>CLAUDE_CODE_CLIENT_KEY</code> (optional).
<code>CLAUDE_CODE_CONNECT_TIMEOUT_MS</code>	Removed in v2.1.186 and now a no-op.
<code>CLAUDE_CODE_DEBUG_LOGS_DIR</code>	Override the debug log file path.
<code>CLAUDE_CODE_DEBUG_LOG_LEVEL</code>	Minimum log level written to the debug log file.
<code>CLAUDE_CODE_DISABLE_1M_CONTEXT</code>	Set to <code>1</code> to turn off 1M context window support.
<code>CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING</code>	Set to <code>1</code> to disable adaptive reasoning on Opus 4.6 and Sonnet 4.6 and fall back to the fixed thinking...
<code>CLAUDE_CODE_DISABLE_ADMIN_ENV_UNION</code>	Set to <code>1</code> to stop Claude Code from merging managed settings <code>env</code> blocks per key across admin sources, so...
<code>CLAUDE_CODE_DISABLE_ADVISOR_TOOL</code>	Set to <code>1</code> to disable the advisor tool.

<code>CLAUDE_CODE_DISABLE_AGENT_VIEW</code>	Set to <code>1</code> to turn off background agents and agent view: <code>claude agents</code> , <code>--bg</code> , <code>/background</code> , and the...
<code>CLAUDE_CODE_DISABLE_ALTERNATE_SCREEN</code>	Set to <code>1</code> to disable fullscreen rendering and use the classic main-screen renderer.
<code>CLAUDE_CODE_DISABLE_ARTIFACT</code>	Set to <code>1</code> to turn off the Artifact tool, which publishes session output as a private web page on claude.ai.
<code>CLAUDE_CODE_DISABLE_ATTACHMENTS</code>	Set to <code>1</code> to disable attachment processing.
<code>CLAUDE_CODE_DISABLE_AUTH_REFRESH_LOCK</code>	Set to <code>1</code> to make a Claude Code process run its <code>gcpAuthRefresh</code> or <code>awsAuthRefresh</code> command itself...
<code>CLAUDE_CODE_DISABLE_AUTO_MEMORY</code>	Set to <code>1</code> to disable auto memory.
<code>CLAUDE_CODE_DISABLE_BACKGROUND_TASKS</code>	Set to <code>1</code> to disable all background task functionality, including the <code>run_in_background</code> parameter on...
<code>CLAUDE_CODE_DISABLE_BEDROCK_CONTENT_TYPE_DEFAULT</code>	Set to <code>1</code> to stop Claude Code from treating an Amazon Bedrock streaming response with a missing or empty...
<code>CLAUDE_CODE_DISABLE_BEDROCK_CONTENT_TYPE_GUARD</code>	Set to <code>1</code> to skip the check that an Amazon Bedrock streaming response carries the...
<code>CLAUDE_CODE_DISABLE_BG_EXIT_HANDOFF</code>	Set to <code>1</code> to stop a background session's running background shell commands, dynamic workflows, and, as of...
<code>CLAUDE_CODE_DISABLE_BG_SHELL_PRESSURE_REAP</code>	Set to <code>1</code> to stop Claude Code from terminating background shell commands under memory pressure.
<code>CLAUDE_CODE_DISABLE_BUNDLED_SKILLS</code>	Set to <code>1</code> to disable the skills and workflows included with Claude Code: bundled skills and workflows are...
<code>CLAUDE_CODE_DISABLE_CFC_PROMPT</code>	Set to <code>1</code> to keep the Claude in Chrome browser tools available while omitting the Chrome section of the...
<code>CLAUDE_CODE_DISABLE_CLAUDE_MDS</code>	Set to <code>1</code> to prevent loading any CLAUDE.md memory files into context, including user, project, and auto...
<code>CLAUDE_CODE_DISABLE_CRON</code>	Set to <code>1</code> to disable scheduled tasks.
<code>CLAUDE_CODE_DISABLE_DANGEROUS_RM_TIMEOUT</code>	Set to <code>1</code> to turn off the time limit on critical-path removal prompts.
<code>CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS</code>	Set to <code>1</code> to strip pre-release <code>anthropic-beta</code> request headers, the body fields that pair with them, and...
<code>CLAUDE_CODE_DISABLE_EXPLORE_PLAN_AGENTS</code>	Set to <code>1</code> to disable the built-in Explore and Plan subagents.
<code>CLAUDE_CODE_DISABLE_FAST_MODE</code>	Set to <code>1</code> to disable fast mode.
<code>CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY</code>	Set to <code>1</code> to disable the "How is Claude doing?" session quality surveys.
<code>CLAUDE_CODE_DISABLE_FILE_CHECKPOINTING</code>	Set to <code>1</code> to disable file checkpointing.
<code>CLAUDE_CODE_DISABLE_FILE_MENTIONS</code>	Set to <code>1</code> to stop Claude Code from expanding <code>@</code> mentions of files and directories.
<code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code>	Set to <code>1</code> to remove built-in commit and PR workflow instructions and the git status snapshot from...
<code>CLAUDE_CODE_DISABLE_INLINE_SHELL_RM_PROMPT</code>	Set to <code>1</code> to stop Claude Code from reading scripts passed to a shell with <code>-c</code> , such as <code>'bash -c 'rm -rf...</code>
<code>CLAUDE_CODE_DISABLE_LEGACY_MODEL_REMAP</code>	Set to <code>1</code> to prevent automatic remapping of Opus 4 and Opus 4.1 to the current Opus version on the...

<code>CLAUDE_CODE_DISABLE_MODEL_ACCESS_FALLBACK</code>	Set to <code>1</code> to stop Claude Code on Amazon Bedrock and Google Cloud's Agent Platform from switching to an...
<code>CLAUDE_CODE_DISABLE_MOUSE</code>	Set to <code>1</code> to disable mouse tracking in fullscreen rendering.
<code>CLAUDE_CODE_DISABLE_MOUSE_CLICKS</code>	Set to <code>1</code> to disable click, drag, and hover handling in fullscreen rendering while keeping mouse-wheel...
<code>CLAUDE_CODE_DISABLE_MTLS_RELOAD_ON_STALE_CONNECTION</code>	Set to <code>1</code> to stop Claude Code from re-reading the mTLS client certificate and key when an API request...
<code>CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC</code>	Set to any non-empty value, such as <code>1</code> , to disable nonessential network traffic: auto-updates, telemetry,...
<code>CLAUDE_CODE_DISABLE_NONSTREAMING_FALLBACK</code>	Set to <code>1</code> to disable the non-streaming fallback when a streaming request fails mid-stream.
<code>CLAUDE_CODE_DISABLE_NOTIFICATION_PRESENCE_CHECK</code>	Set to <code>1</code> to send the <code>PushNotification</code> tool's desktop notification even while you are typing in or...
<code>CLAUDE_CODE_DISABLE_OFFICIAL_MARKETPLACE_AUTOINSTALL</code>	Set to <code>1</code> to disable automatic registration of the official plugin marketplace.
<code>CLAUDE_CODE_DISABLE_PERMISSION_PROMPT_NOTIFY_HOOKS</code>	Set to <code>1</code> to stop Claude Code from running your <code>Notification</code> hooks for unanswered permission requests...
<code>CLAUDE_CODE_DISABLE_POLICY_SKILLS</code>	Set to <code>1</code> to skip loading skills from the system-wide managed skills directory.
<code>CLAUDE_CODE_DISABLE_POWERSHELL_CMD_RM_DENY</code>	Set to <code>1</code> to turn off the PowerShell tool check that denies the <code>cmd</code> built-ins <code>rd</code> , <code>rmdir</code> , <code>del</code> , and...
<code>CLAUDE_CODE_DISABLE_REFUSAL_FALLBACK</code>	Set to <code>1</code> to turn off the automatic model switch when a safety classifier flags a request, the behavior...
<code>CLAUDE_CODE_DISABLE_STRUCTURED_OUTPUTS</code>	Set to <code>1</code> to stop Claude Code from sending the structured-output <code>output_config.format</code> field and the...
<code>CLAUDE_CODE_DISABLE_SUBSTITUTION_RM_PROMPT</code>	Set to <code>1</code> to turn off the critical-path check for a recursive <code>rm</code> whose target is entirely the output of...
<code>CLAUDE_CODE_DISABLE_TERMINAL_TITLE</code>	Set to <code>1</code> to disable automatic terminal title updates based on conversation context.
<code>CLAUDE_CODE_DISABLE_THINKING</code>	Set to <code>1</code> to omit the <code>thinking</code> parameter from API requests entirely.
<code>CLAUDE_CODE_DISABLE_UNKNOWN_MODEL_WINDOW_ENFORCEMENT</code>	Set to <code>1</code> to skip proactive auto-compaction when Claude Code doesn't recognize the model ID, such as an...
<code>CLAUDE_CODE_DISABLE_VIRTUAL_SCROLL</code>	Set to <code>1</code> to disable virtual scrolling in fullscreen rendering and render every message in the transcript.
<code>CLAUDE_CODE_DISABLE_WEB_FETCH</code>	Set to <code>1</code> to turn off the WebFetch tool.
<code>CLAUDE_CODE_DISABLE_WINDOWS_SHELL_LAUNCHER</code>	Set to <code>1</code> to start PowerShell tool commands on Windows directly instead of through the <code>cmd.exe</code> launcher.
<code>CLAUDE_CODE_DISABLE_WORKFLOWS</code>	Set to <code>1</code> to disable workflows.
<code>CLAUDE_CODE_EFFORT_LEVEL</code>	Set the effort level for supported models.
<code>CLAUDE_CODE_EMIT_SESSION_STATE_EVENTS</code>	Set to <code>1</code> to add <code>session_state_changed</code> messages, which carry the session's state, to the message stream.
<code>CLAUDE_CODE_ENABLE_AUTO_MODE</code>	Accepted for compatibility with older releases and has no effect.

<code>CLAUDE_CODE_ENABLE_AWAY_SUMMARY</code>	Override session recap availability.
<code>CLAUDE_CODE_ENABLE_BACKGROUND_PLUGIN_REFRESH</code>	Set to <code>1</code> to refresh plugin state at turn boundaries in non-interactive mode after a background install...
<code>CLAUDE_CODE_ENABLE_CFC</code>	Set to <code>1</code> to start a CLI session with Chrome integration on, or <code>0</code> to start with it off.
<code>CLAUDE_CODE_ENABLE_FEEDBACK_SURVEY_FOR_OTEL</code>	Set to <code>1</code> to route the “How is Claude doing?” session quality survey to your own OpenTelemetry collector...
<code>CLAUDE_CODE_ENABLE_FINE_GRAINED_TOOL_STREAMING</code>	Controls whether tool call inputs stream from the API as Claude generates them.
<code>CLAUDE_CODE_ENABLE_GATEWAY_MODEL_DISCOVERY</code>	Set to <code>1</code> to populate the <code>/model</code> picker from your gateway's <code>/v1/models</code> endpoint when...
<code>CLAUDE_CODE_ENABLE_OPUS_4_7_FAST_MODE</code>	Removed in v2.1.142, when the fast mode default moved from Opus 4.6 to Opus 4.7.
<code>CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION</code>	Set to <code>false</code> to turn off prompt suggestions, the grayed-out predictions that appear in your prompt input.
<code>CLAUDE_CODE_ENABLE_TASKS</code>	Selects which task-tracking tools Claude Code provides in sessions that have them.
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	Set to <code>1</code> to enable OpenTelemetry data collection for metrics and logging.
<code>CLAUDE_CODE_ENABLE_TODO_TOOLS</code>	Set to <code>1</code> to get the task-tracking tools on every model.
<code>CLAUDE_CODE_EXIT_AFTER_STOP_DELAY</code>	Time in milliseconds to wait after the query loop becomes idle before automatically exiting.
<code>CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS</code>	Set to <code>1</code> to enable agent teams.
<code>CLAUDE_CODE_EXTRA_BODY</code>	JSON object to merge into the top level of every API request body.
<code>CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS</code>	Override the default token limit for file reads, which is 25,000 tokens.
<code>CLAUDE_CODE_FORCE_SESSION_PERSISTENCE</code>	Set to <code>1</code> to force transcript persistence, prompt history, and <code>claude agents</code> registration even when...
<code>CLAUDE_CODE_FORCE_STRIKETHROUGH</code>	Set to <code>1</code> to force strikethrough rendering for <code>~~text~~</code> in Claude's responses when your terminal...
<code>CLAUDE_CODE_FORCE_SYNC_OUTPUT</code>	Set to <code>1</code> to force-enable DEC private mode 2026 synchronized output when your terminal supports it but is...
<code>CLAUDE_CODE_FORCE_TERMINAL_IMAGES</code>	Set to <code>1</code> to draw mod <code>Image</code> elements as pictures when your terminal draws kitty graphics protocol...
<code>CLAUDE_CODE_FORK_SUBAGENT</code>	Controls fork mode, which lets Claude spawn forked subagents itself and is on by default in interactive...
<code>CLAUDE_CODE_FORWARD_SUBAGENT_TEXT</code>	Set to <code>1</code> to emit subagent text and thinking blocks in <code>claude -p -output-format stream-json</code> output...
<code>CLAUDE_CODE_GATEWAY_HINT_HEADERS</code>	Set to <code>1</code> to send the gateway hint headers, such as <code>x-claude-code-request-class</code> and...
<code>CLAUDE_CODE_GATEWAY_MODEL_DISCOVERY_TIMEOUT_MS</code>	Timeout in milliseconds for the gateway model discovery request that...
<code>CLAUDE_CODE_GIT_BASH_PATH</code>	Windows only: path to the Git Bash executable (<code>bash.exe</code>).

<code>CLAUDE_CODE_GLOB_HIDDEN</code>	Set to <code>false</code> to exclude dotfiles from results when Claude invokes the Glob tool.
<code>CLAUDE_CODE_GLOB_NO_IGNORE</code>	Set to <code>false</code> to make the Glob tool respect <code>.gitignore</code> patterns.
<code>CLAUDE_CODE_GLOB_TIMEOUT_SECONDS</code>	Timeout in seconds for Glob tool file discovery.
<code>CLAUDE_CODE_GOAL_CHECKIN_MINUTES</code>	How many minutes background work can keep an active goal waiting before Claude Code asks Claude to check...
<code>CLAUDE_CODE_GZIP_REQUEST_BODIES</code>	Set to <code>0</code> to turn off gzip compression of Claude API, telemetry, and artifact publish request bodies sent...
<code>CLAUDE_CODE_HIDE_CWD</code>	Set to <code>1</code> to hide the working directory in the startup logo.
<code>CLAUDE_CODE_IDE_HOST_OVERRIDE</code>	Override the host address used to connect to the IDE extension.
<code>CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL</code>	Set to <code>1</code> to skip auto-installation of IDE extensions.
<code>CLAUDE_CODE_IDE_SKIP_VALID_CHECK</code>	Set to <code>1</code> to skip validation of IDE lockfile entries during connection.
<code>CLAUDE_CODE_IDLE_COMPACT_MIN_TOKENS</code>	Context size, in tokens, at which idle compaction applies to a conversation (default: <code>200000</code>).
<code>CLAUDE_CODE_MAX_CONCURRENT_SUBAGENTS</code>	How many subagents can be running in one session before the Agent tool refuses to spawn another (default: 20).
<code>CLAUDE_CODE_MAX_CONTEXT_TOKENS</code>	Override the context window size Claude Code assumes for the active model.
<code>CLAUDE_CODE_MAX_MCP_DESCRIPTION_LENGTH</code>	Maximum length in characters of MCP tool descriptions and MCP server instructions that Claude Code sends...
<code>CLAUDE_CODE_MAX_OUTPUT_TOKENS</code>	Set the maximum number of output tokens for most requests.
<code>CLAUDE_CODE_MAX_RETRIES</code>	Override the number of times to retry failed API requests (default: 10).
<code>CLAUDE_CODE_MAX_SUBAGENTS_PER_SESSION</code>	Removed in v2.1.224 and now a no-op.
<code>CLAUDE_CODE_MAX_SUBAGENT_SPAWN_DEPTH</code>	Number of subagent layers allowed below the main conversation (default: 3).
<code>CLAUDE_CODE_MAX_TOOL_USE_CONCURRENCY</code>	Maximum number of read-only tools and subagents that can execute in parallel (default: 10).
<code>CLAUDE_CODE_MAX_TURNS</code>	Cap the number of agentic turns when no explicit limit is passed.
<code>CLAUDE_CODE_MAX_WEB_SEARCHES_PER_SESSION</code>	Cap on WebSearch calls (default: 200).
<code>CLAUDE_CODE_MCP_ALLOWLIST_ENV</code>	Set to <code>1</code> to spawn stdio MCP servers with only a safe baseline environment plus the server's configured...
<code>CLAUDE_CODE_MCP_AUTO_BACKGROUND_MS</code>	Elapsed time in milliseconds before a still-running MCP tool call moves to a background task (default:...
<code>CLAUDE_CODE_MCP_STARTUP_WAIT_MS</code>	How long in milliseconds the first turn of a non-interactive session waits for MCP servers that are still...
<code>CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT</code>	Idle timeout in milliseconds for MCP tool calls.
<code>CLAUDE_CODE_MESSAGING_SOCKET</code>	Set by Claude Code, not by you: in sessions that bind an inbox socket, Claude Code exports that socket's...
<code>CLAUDE_CODE_MESSAGING_TOKEN</code>	Set by Claude Code, not by you: in sessions that bind an inbox socket, Claude Code exports this...
<code>CLAUDE_CODE_NATIVE_CURSOR</code>	Set to <code>1</code> to show the terminal's own cursor at the input caret instead of a drawn block.

<code>CLAUDE_CODE_NEW_INIT</code>	Set to <code>1</code> to make <code>/init</code> run an interactive setup flow.
<code>CLAUDE_CODE_NONBLOCKING_STDOUT</code>	Set to <code>1</code> to write terminal output through a second non-blocking file descriptor, so a terminal that...
<code>CLAUDE_CODE_NONSTREAMING_TIMEOUT_RETRIES</code>	Limit how many times Claude Code re-sends a non-streaming request that times out.
<code>CLAUDE_CODE_NO_FLICKER</code>	Set to <code>1</code> to enable fullscreen rendering, a research preview that reduces flicker and keeps memory flat...
<code>CLAUDE_CODE_OAUTH_REFRESH_TOKEN</code>	OAuth refresh token for Claude.ai authentication.
<code>CLAUDE_CODE_OAUTH_SCOPES</code>	Space-separated OAuth scopes the refresh token was issued with, such as <code>"user:profile user:inference..."</code>
<code>CLAUDE_CODE_OAUTH_TOKEN</code>	OAuth access token for claude.ai authentication.
<code>CLAUDE_CODE_OPUS_4_6_FAST_MODE_OVERRIDE</code>	Removed in v2.1.160 and now a no-op.
<code>CLAUDE_CODE_OTEL_CONTENT_MAX_LENGTH</code>	Maximum length of content-bearing OpenTelemetry attributes (model responses, tool content, system prompts,...)
<code>CLAUDE_CODE_OTEL_DIAG_STDERR</code>	Set to <code>1</code> to write OpenTelemetry exporter diagnostic errors to stderr.
<code>CLAUDE_CODE_OTEL_FLUSH_TIMEOUT_MS</code>	Timeout in milliseconds for flushing pending OpenTelemetry spans (default: 5000).
<code>CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS</code>	Interval for refreshing dynamic OpenTelemetry headers in milliseconds (default: 1740000 / 29 minutes).
<code>CLAUDE_CODE_OTEL_SHUTDOWN_TIMEOUT_MS</code>	Timeout in milliseconds for the OpenTelemetry exporter to finish on shutdown (default: 2000).
<code>CLAUDE_CODE_OVERLOADED_RETRY_BASE_DELAY_MS</code>	Starting delay in milliseconds, in place of the default 500, of the exponential backoff between automatic...
<code>CLAUDE_CODE_OVERLOADED_RETRY_MAX_DELAY_MS</code>	Ceiling in milliseconds, in place of the default 32000, on the exponential backoff between automatic...
<code>CLAUDE_CODE_PACKAGE_MANAGER_AUTO_UPDATE</code>	Set to <code>1</code> to let Claude Code run your package manager's upgrade command in the background when a new...
<code>CLAUDE_CODE_PERFORCE_MODE</code>	Set to <code>1</code> to enable Perforce-aware write protection.
<code>CLAUDE_CODE_PLUGIN_CACHE_DIR</code>	Override the plugins root directory.
<code>CLAUDE_CODE_PLUGIN_DIRS</code>	Plugin directories to load for the session, each loaded the way a <code>-plugin-dir</code> flag loads it.
<code>CLAUDE_CODE_PLUGIN_DIR_WATCH</code>	Controls whether Claude Code reloads a mod when the mod's files change.
<code>CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS</code>	Timeout in milliseconds for cloning or refreshing a plugin marketplace (default: 120000).
<code>CLAUDE_CODE_PLUGIN_KEEP_MARKETPLACE_ON_FAILURE</code>	Set to <code>1</code> to skip the re-clone attempt and keep using the existing marketplace checkout when a...
<code>CLAUDE_CODE_PLUGIN_PREFER_HTTPS</code>	Set to <code>1</code> to clone GitHub <code>owner/repo</code> shorthand sources over HTTPS instead of SSH. Applies to plugin...
<code>CLAUDE_CODE_PLUGIN_SEED_DIR</code>	Path to one or more read-only plugin seed directories, separated by <code>:</code> on Unix or <code>;</code> on Windows.
<code>CLAUDE_CODE_POWERSHELL_RESPECT_EXECUTION_POLICY</code>	Set to <code>1</code> to stop Claude Code from passing <code>-ExecutionPolicy Bypass</code> when spawning PowerShell for tool...

<code>CLAUDE_CODE_PRINT_BG_WAIT_CEILING_MS</code>	Ceiling in milliseconds on idle waiting for background work, such as subagents and workflows, after the...
<code>CLAUDE_CODE_PROCESS_WRAPPER</code>	Launch the processes Claude Code starts from its own binary, such as the background service that hosts...
<code>CLAUDE_CODE_PROJECT_DIR_NAME</code>	Set together with <code>CLAUDE_CONFIG_DIR</code> to choose the <code>projects/</code> directory name Claude Code stores that...
<code>CLAUDE_CODE_PROMPT_CACHE_TTL</code>	Set <code>5m</code> or <code>1h</code> , the only values Claude Code accepts, to choose the prompt cache TTL for the main...
<code>CLAUDE_CODE_PROPAGATE_TRACEPARENT</code>	Set to <code>1</code> to propagate W3C trace context when <code>ANTHROPIC_BASE_URL</code> points at a custom proxy.
<code>CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST</code>	Set by host platforms that embed Claude Code and manage model provider routing on its behalf.
<code>CLAUDE_CODE_PROXY_RESOLVES_HOSTS</code>	Set to <code>1</code> to allow the proxy to perform DNS resolution instead of the caller.
<code>CLAUDE_CODE_REMOTE</code>	Set automatically to <code>true</code> when Claude Code is running as a cloud session.
<code>CLAUDE_CODE_REMOTE_SESSION_ID</code>	Set automatically in cloud sessions to the current session's ID. Read this to construct a link back to the...
<code>CLAUDE_CODE_RESTRICTED</code>	Set to <code>1</code> to start the session in restricted mode, the same as passing <code>--restricted</code> .
<code>CLAUDE_CODE_RESUME_INTERRUPTED_TURN</code>	Set to <code>1</code> to automatically resume if the previous session ended mid-turn.
<code>CLAUDE_CODE_RESUME_INTERRUPTED_TURN_MAX_AGE_MS</code>	Maximum age in milliseconds of the last transcript message for a session that ended mid-turn to continue...
<code>CLAUDE_CODE_RESUME_PROMPT</code>	Override the continuation message Claude Code sends to Claude when <code>CLAUDE_CODE_RESUME_INTERRUPTED_TURN</code> ...
<code>CLAUDE_CODE_RETRY_WATCHDOG</code>	Set to <code>1</code> for unattended sessions such as eval harnesses, CI jobs, or remote workers.
<code>CLAUDE_CODE_RETRY_WATCHDOG_MAX_WAIT_MS</code>	Maximum time in milliseconds that each API request spends waiting out <code>429</code> and <code>529</code> errors when...
<code>CLAUDE_CODE_SAFE_MODE</code>	Set to <code>1</code> to start in safe mode: CLAUDE.md, skills, plugins, hooks, MCP servers, custom commands and...
<code>CLAUDE_CODE_SCRIPT_CAPS</code>	JSON object limiting how many times specific scripts may be invoked per session when...
<code>CLAUDE_CODE_SCROLL_SPEED</code>	Set the mouse wheel scroll multiplier in fullscreen rendering.
<code>CLAUDE_CODE_SEND_FEEDBACK</code>	Set to <code>0</code> to turn off Claude-drafted feedback for a session.
<code>CLAUDE_CODE_SESSIONEND_HOOKS_TIMEOUT_MS</code>	Override the time budget in milliseconds for SessionEnd hooks.
<code>CLAUDE_CODE_SESSION_ID</code>	Set automatically to the current session ID in Bash and PowerShell tool subprocesses, hook command...
<code>CLAUDE_CODE_SHELL</code>	Set the shell Claude Code uses to run Bash tool commands.
<code>CLAUDE_CODE_SHELL_PREFIX</code>	Command prefix that wraps shell commands Claude Code spawns: Bash tool calls, hook commands, status line...
<code>CLAUDE_CODE_SIMPLE</code>	Set to <code>1</code> to run with a minimal system prompt and only the Bash, file read, and file edit tools.

<code>CLAUDE_CODE_SIMPLE_SYSTEM_PROMPT</code>	Choose between Claude Code's full system prompt and a shorter one with abbreviated tool descriptions.
<code>CLAUDE_CODE_SKIP_ANTHROPIC_AWS_AUTH</code>	Skip client-side authentication for Claude Platform on AWS, for gateways that sign requests themselves.
<code>CLAUDE_CODE_SKIP_AWS_CRED_CACHE</code>	Set to <code>1</code> to turn off the in-process cache of credentials resolved from the AWS default credential...
<code>CLAUDE_CODE_SKIP_BEDROCK_AUTH</code>	Skip AWS authentication for Amazon Bedrock (for example, when using an LLM gateway).
<code>CLAUDE_CODE_SKIP_FAST_MODE_NETWORK_ERRORS</code>	Set to <code>1</code> to treat a failed fast mode availability check as available, for networks that block the...
<code>CLAUDE_CODE_SKIP_FAST_MODE_ORG_CHECK</code>	Set to <code>1</code> to skip the client-side fast mode availability check, for proxies that intercept the check's...
<code>CLAUDE_CODE_SKIP_FOUNDRY_AUTH</code>	Skip Azure authentication for Microsoft Foundry, for a proxy or gateway that injects its own...
<code>CLAUDE_CODE_SKIP_MANTLE_AUTH</code>	Skip AWS authentication for Amazon Bedrock Mantle (for example, when using an LLM gateway).
<code>CLAUDE_CODE_SKIP_MODEL_ACCESS_MEMORY</code>	The startup model checks on Amazon Bedrock and Google Cloud's Agent Platform remember on this machine...
<code>CLAUDE_CODE_SKIP_PROMPT_HISTORY</code>	Set to <code>1</code> to skip writing prompt history and session transcripts to disk.
<code>CLAUDE_CODE_SKIP_VERTEX_AUTH</code>	Skip Google authentication for Google Cloud's Agent Platform (for example, when using an LLM gateway).
<code>CLAUDE_CODE_STARTUP_FAILURE_RESULTS</code>	Set to <code>1</code> to have a session started with <code>--output-format stream-json</code> write a result message naming why...
<code>CLAUDE_CODE_STOP_HOOK_BLOCK_CAP</code>	Maximum number of consecutive times a Stop or SubagentStop hook may block the turn from ending before...
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	The default model for subagents, agent team teammates, and workflow agents that aren't assigned a model...
<code>CLAUDE_CODE_SUBAGENT_MODEL_FORCE</code>	Set to <code>1</code> to force one model onto subagents, teammates, and workflow agents.
<code>CLAUDE_CODE_SUBAGENT_PROMPT_CACHE_TTL</code>	Set <code>5m</code> or <code>1h</code> , the only values Claude Code accepts, to choose the prompt cache TTL for requests outside...
<code>CLAUDE_CODE_SUBPROCESS_ENV_SCRUB</code>	Set to <code>1</code> to strip credentials from the environments of the subprocesses Claude Code starts, such as Bash...
<code>CLAUDE_CODE_SYNC_PLUGIN_INSTALL</code>	Set to <code>1</code> in non-interactive mode (the <code>-p</code> flag) to wait for plugin installation to complete before the...
<code>CLAUDE_CODE_SYNC_PLUGIN_INSTALL_TIMEOUT_MS</code>	Timeout in milliseconds for synchronous plugin installation.
<code>CLAUDE_CODE_SYNC_SKILLS</code>	Set to <code>1</code> in non-interactive mode with the <code>-p</code> flag to make Claude Code download the skills enabled for...
<code>CLAUDE_CODE_SYNC_SKILLS_INSTALL_TIMEOUT_MS</code>	Timeout in milliseconds for the skills resync that runs mid-session when an app built on the Agent SDK...
<code>CLAUDE_CODE_SYNC_SKILLS_WAIT_TIMEOUT_MS</code>	Timeout in milliseconds for the first query to wait for the initial skill list when...
<code>CLAUDE_CODE_SYNTAX_HIGHLIGHT</code>	Set to <code>false</code> to disable syntax highlighting in diff output.
<code>CLAUDE_CODE_TASK_LIST_ID</code>	Share a task list across sessions.

<code>CLAUDE_CODE_TEAM_TEARDOWN_PARK_TIMEOUT_MS</code>	Override, in milliseconds, how long a non-interactive session waits at exit for its agent team to finish...
<code>CLAUDE_CODE_TMPDIR</code>	Override the temp directory used for internal temp files.
<code>CLAUDE_CODE_TMUX_TRUECOLOR</code>	Set to any non-empty value, such as <code>1</code> , to allow 24-bit truecolor output inside tmux. **Setting it to <code>0</code> ...
<code>CLAUDE_CODE_TOOL_MEMORY_CGROUP_EXCLUDE</code>	On Linux and WSL, set to a comma-separated list of the kinds of processes Claude Code excludes from the...
<code>CLAUDE_CODE_TOOL_MEMORY_LIMIT</code>	On Linux and WSL, set to a size such as <code>4G</code> to cap the memory that Bash and PowerShell tool commands can...
<code>CLAUDE_CODE_TRANSCRIPT_LOCAL_GC</code>	Set to <code>1</code> to limit how large the transcript file of a long <code>-p</code> or Agent SDK session grows.
<code>CLAUDE_CODE_USER_DIALOG_TIMEOUT_MS</code>	Deadline in milliseconds before Claude Code cancels a dialog it forwards to a remote client such as a...
<code>CLAUDE_CODE_USE_ANTHROPIC_AWS</code>	Use Claude Platform on AWS.
<code>CLAUDE_CODE_USE_BEDROCK</code>	Use Amazon Bedrock.
<code>CLAUDE_CODE_USE_FOUNDRY</code>	Use Microsoft Foundry.
<code>CLAUDE_CODE_USE_MANTLE</code>	Use the Amazon Bedrock Mantle endpoint.
<code>CLAUDE_CODE_USE_NATIVE_FILE_SEARCH</code>	Set to <code>1</code> to discover custom commands, subagents, and output styles using Node.js file APIs instead of...
<code>CLAUDE_CODE_USE_POWERSHELL_TOOL</code>	Controls the PowerShell tool.
<code>CLAUDE_CODE_USE_VERTEX</code>	Use Google Cloud's Agent Platform.
<code>CLAUDE_CODE_WEBFETCH_CACHE_TTL_MS</code>	Set to the number of milliseconds WebFetch keeps each fetched URL's response cached.
<code>CLAUDE_CODE_WEBFETCH_DEADLINE_MS</code>	Upper bound in milliseconds on how long WebFetch waits for a page to download, including any redirects it...
<code>CLAUDE_CODE_WEB_SEARCH_REFILLS_PER_HOUR</code>	Rate at which a session's WebSearch limit refills, in calls per hour.
<code>CLAUDE_CODE_WORKER_CHECKIN_SCHEDULE</code>	When <code>CLAUDE_AUTO_BACKGROUND_TASKS</code> is set to <code>1</code> , how long Claude Code waits before each reminder to...
<code>CLAUDE_CODE_WORKFLOW_MAX_CONCURRENT_AGENTS</code>	How many agents a single workflow run executes at once, from <code>1</code> to <code>256</code> .
<code>CLAUDE_CODE_WORKFLOW_PREFIX_STAGGER_MS</code>	Upper bound in milliseconds on how long a workflow agent waits for a same-prefix sibling's first response...
<code>CLAUDE_CODE_YOU_SHOULD_KNOW_DEBUG</code>	Set to <code>1</code> to show, above the prompt, a line for each of the last three checks the You should know mod...
<code>CLAUDE_CONFIG_DIR</code>	Override the configuration directory (default: <code>~/ .claude</code>).
<code>CLAUDE_DISABLE_ADOPT</code>	Set to <code>1</code> to stop in-flight background work instead of carrying it over when you background a session by...
<code>CLAUDE Effort</code>	Set automatically in Bash tool subprocesses and hook commands to the effort level in effect when the...
<code>CLAUDE_ENABLE_BYTE_WATCHDOG</code>	Set to <code>1</code> to force-enable the byte-level streaming idle watchdog, or set to <code>0</code> to force-disable it. <code>0</code> ...
<code>CLAUDE_ENABLE_BYTE_WATCHDOG_BEDROCK</code>	Set to <code>1</code> to enable the byte-level streaming idle watchdog on Amazon Bedrock <code>vnd.amazon.eventstream</code> ...

<code>CLAUDE_ENABLE_STREAM_WATCHDOG</code>	Set to <code>0</code> to force-disable the event-level streaming idle watchdog, or set to <code>1</code> to force-enable it.
<code>CLAUDE_ENV_FILE</code>	Path to a shell script whose contents Claude Code runs before each Bash command in the same shell process,...
<code>CLAUDE_JOB_DIR</code>	Set by Claude Code in each background session to that session's <code>~/.claude/jobs/<id></code> directory.
<code>CLAUDE_PID</code>	Claude Code sets this to its own process ID in the subprocesses it spawns: Bash and PowerShell tool...
<code>CLAUDE_REMOTE_CONTROL_SESSION_NAME_PREFIX</code>	Prefix for auto-generated Remote Control session names when no explicit name is provided.
<code>CLAUDE_STREAM_FIRST_BYTE_TIMEOUT_MS</code>	Deadline in milliseconds for the first response byte of a streaming request, on the connections where the...
<code>CLAUDE_STREAM_IDLE_TIMEOUT_MS</code>	Timeout in milliseconds before the event- and byte-level streaming idle watchdogs close a stalled connection.
<code>CLAUDE_SUBAGENT_BG_SHELL_MAX_MS</code>	Removed in v2.1.260 and now a no-op.
<code>DEBUG</code>	Set to <code>1</code> to enable debug mode, equivalent to launching with <code>--debug</code> .
<code>DISABLE_AUTOUPDATER</code>	Set to <code>1</code> to disable automatic background updates.
<code>DISABLE_AUTO_COMPACT</code>	Set to <code>1</code> to disable automatic compaction when approaching the context limit, along with idle compaction.
<code>DISABLE_COMPACT</code>	Set to <code>1</code> to disable all compaction: both automatic compaction and the manual <code>/compact</code> command.
<code>DISABLE_COST_WARNINGS</code>	Set to <code>1</code> to disable cost warning messages.
<code>DISABLE_DOCTOR_COMMAND</code>	Set to <code>1</code> to hide the <code>/doctor</code> setup checkup skill and its <code>/checkup</code> alias.
<code>DISABLE_ERROR_REPORTING</code>	Set to any non-empty value, such as <code>1</code> , to opt out of error reporting. **Setting it to <code>0</code> or <code>false</code> ...
<code>DISABLE_EXTRA_USAGE_COMMAND</code>	Set to <code>1</code> to hide the <code>/usage-credits</code> command that lets users purchase additional usage beyond rate limits.
<code>DISABLE_FEEDBACK_COMMAND</code>	Set to <code>1</code> to disable the <code>/feedback</code> command and Claude-drafted feedback.
<code>DISABLE_GROWTHBOOK</code>	Set to <code>1</code> or <code>true</code> to disable GrowthBook feature-flag fetching and use code defaults for every flag.
<code>DISABLE_INSTALLATION_CHECKS</code>	Set to <code>1</code> to disable installation warnings.
<code>DISABLE_INSTALL_GITHUB_APP_COMMAND</code>	Set to <code>1</code> to hide the <code>/install-github-app</code> command.
<code>DISABLE_INTERLEAVED_THINKING</code>	Set to <code>1</code> to prevent sending the interleaved-thinking beta header.
<code>DISABLE_LOGIN_COMMAND</code>	Set to <code>1</code> to hide the <code>/login</code> command.
<code>DISABLE_LOGOUT_COMMAND</code>	Set to <code>1</code> to hide the <code>/logout</code> command.
<code>DISABLE_PROMPT_CACHING</code>	Set to <code>1</code> to disable prompt caching for all models (takes precedence over per-model settings).
<code>DISABLE_PROMPT_CACHING_FABLE</code>	Set to <code>1</code> to disable prompt caching for Fable models.
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Set to <code>1</code> to disable prompt caching for the default Haiku model, wherever it runs.

<code>DISABLE_PROMPT_CACHING_OPUS</code>	Set to <code>1</code> to disable prompt caching for the default Opus model.
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Set to <code>1</code> to disable prompt caching for the default Sonnet model.
<code>DISABLE_TELEMETRY</code>	Set to any non-empty value, such as <code>1</code> , to opt out of telemetry. **Setting it to <code>0</code> or <code>false</code> still...
<code>DISABLE_UPDATES</code>	Set to <code>1</code> to block all updates including manual <code>claude update</code> and <code>claude install</code> .
<code>DISABLE_UPGRADE_COMMAND</code>	Set to <code>1</code> to hide the <code>/upgrade</code> command.
<code>DO_NOT_TRACK</code>	Set to <code>1</code> to opt out of telemetry, with the same effect as <code>DISABLE_TELEMETRY</code> , including on feature-flag...
<code>ENABLE_BETA_TRACING_DETAILED</code>	Set to <code>1</code> , and set <code>BETA_TRACING_ENDPOINT</code> to your OTLP/HTTP collector endpoint, to turn on detailed beta...
<code>ENABLE_CLAUDEAI_MCP_SERVERS</code>	Set to <code>false</code> to stop Claude Code from fetching claude.ai MCP servers.
<code>ENABLE_PROMPT_CACHING_1H</code>	Set to <code>1</code> to request a 1-hour prompt cache TTL instead of the default 5 minutes.
<code>ENABLE_PROMPT_CACHING_1H_BEDROCK</code>	Deprecated.
<code>ENABLE_TOOL_SEARCH</code>	Controls MCP tool search.
<code>FALLBACK_FOR_ALL_PRIMARY_MODELS</code>	Set to any non-empty value, such as <code>1</code> , to make Claude Code stop retrying on repeated overload errors for...
<code>FORCE_AUTOUPDATE_PLUGINS</code>	Set to <code>1</code> to force plugin auto-updates even when the main auto-updater is disabled via <code>DISABLE_AUTOUPDATER</code> .
<code>FORCE_HYPERLINK</code>	Set to <code>1</code> to enable clickable OSC 8 hyperlinks when your terminal supports them but isn't auto-detected,...
<code>FORCE_PROMPT_CACHING_5M</code>	Set to <code>1</code> to force the 5-minute prompt cache TTL even when 1-hour TTL would otherwise apply.
<code>HTTP_PROXY</code>	Specify HTTP proxy server for network connections.
<code>HTTPS_PROXY</code>	Specify HTTPS proxy server for network connections.
<code>IS_DEMO</code>	Set to any non-empty value, such as <code>1</code> , to enable demo mode: hides your email and organization name from...
<code>MAX_MCP_OUTPUT_TOKENS</code>	Maximum number of tokens allowed in MCP tool responses (default: 25000).
<code>MAX_STRUCTURED_OUTPUT_RETRIES</code>	Number of attempts Claude Code allows when the model's response fails validation against the...
<code>MAX_THINKING_TOKENS</code>	Fixed token budget for extended thinking.
<code>MCP_CLIENT_SECRET</code>	OAuth client secret for MCP servers that require pre-configured credentials.
<code>MCP_CONNECTION_NONBLOCKING</code>	Controls whether startup waits for MCP servers to connect before the first query.
<code>MCP_CONNECT_TIMEOUT_MS</code>	How long blocking MCP startup waits, in milliseconds, for the connection batch before snapshotting the...
<code>MCP_DISCOVERY_CACHE</code>	Turns the MCP discovery cache on or off.
<code>MCP_DISCOVERY_CACHE_MAX_STALE_S</code>	Maximum age, in seconds, of a discovery-cache entry (default: 14400, or 4 hours).
<code>MCP_DISCOVERY_CACHE_STRIKES</code>	At a start where a discovery-cache entry is older than <code>MCP_DISCOVERY_CACHE_TTL_S</code> , Claude Code refreshes...
<code>MCP_DISCOVERY_CACHE_TTL_S</code>	Seconds for which Claude Code uses a discovery-cache entry without refreshing it (default: 900).

<code>MCP_OAUTH_CALLBACK_PORT</code>	Fixed port for the OAuth redirect callback, as an alternative to <code>--callback-port</code> when adding an MCP...
<code>MCP_PROTOCOL_NEGOTIATION</code>	On the v2 MCP client runtime only, whether Claude Code probes servers for MCP protocol revision...
<code>MCP_REMOTE_SERVER_CONNECTION_BATCH_SIZE</code>	Maximum number of remote MCP servers (HTTP/SSE) to connect in parallel during startup (default: 20).
<code>MCP_SDK_GENERATION</code>	Pin which MCP client runtime this process connects to MCP servers with: <code>v1</code> , built on MCP TypeScript SDK...
<code>MCP_SERVER_CONNECTION_BATCH_SIZE</code>	Maximum number of local MCP servers (stdio) to connect in parallel during startup (default: 3).
<code>MCP_TIMEOUT</code>	Timeout in milliseconds for MCP server startup (default: 30000, or 30 seconds).
<code>MCP_TOOL_TIMEOUT</code>	Timeout in milliseconds for MCP tool execution (default: 100000000, about 28 hours).
<code>NO_PROXY</code>	List of domains and IPs to which requests will be directly issued, bypassing proxy.
<code>OTEL_ATTRIBUTE_VALUE_LENGTH_LIMIT</code>	Standard OpenTelemetry SDK limit on attribute value length.
<code>OTEL_LOG_ASSISTANT_RESPONSES</code>	Set to <code>1</code> to include the model's response text on <code>assistant_response</code> OpenTelemetry log events.
<code>OTEL_LOG_MANAGED_SETTINGS</code>	Set to <code>1</code> to add the redacted managed settings, and a SHA-256 digest of the settings before redaction, to...
<code>OTEL_LOG_RAW_API_BODIES</code>	Emit Anthropic Messages API request and response JSON as <code>api_request_body</code> / <code>api_response_body</code> log events.
<code>OTEL_LOG_TOOL_CONTENT</code>	Set to <code>1</code> to include tool content in the <code>tool.output</code> OpenTelemetry span event.
<code>OTEL_LOG_TOOL_DETAILS</code>	Set to <code>1</code> to include tool input arguments; MCP server names; user-authored workflow names; raw error...
<code>OTEL_LOG_USER_PROMPTS</code>	Set to <code>1</code> to include user prompt text in OpenTelemetry traces and logs.
<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code>	Set to <code>false</code> to exclude account UUID from metrics attributes (default: included).
<code>OTEL_METRICS_INCLUDE_ENTRYPPOINT</code>	Set to <code>true</code> to include the session entrypoint in metrics attributes (default: excluded).
<code>OTEL_METRICS_INCLUDE_REPOSITORY</code>	Set to <code>true</code> to tag OpenTelemetry metrics and events with <code>vcs.*</code> attributes identifying the session's...
<code>OTEL_METRICS_INCLUDE_RESOURCE_ATTRIBUTES</code>	As of v2.1.161, Claude Code attaches <code>OTEL_RESOURCE_ATTRIBUTES</code> keys to metric datapoint labels.
<code>OTEL_METRICS_INCLUDE_SESSION_ID</code>	Set to <code>false</code> to exclude session ID from metrics attributes (default: included).
<code>OTEL_METRICS_INCLUDE_VERSION</code>	Set to <code>true</code> to include Claude Code version in metrics attributes (default: excluded).
<code>SLASH_COMMAND_TOOL_CHAR_BUDGET</code>	Override the character budget for skill metadata shown to the Skill tool.
<code>TASK_MAX_OUTPUT_LENGTH</code>	Removed in v2.1.277 and now a no-op, together with the <code>TaskOutput</code> tool it sized.
<code>USE_BUILTIN_RIPGREP</code>	Set to <code>0</code> to use system-installed <code>rg</code> instead of <code>rg</code> included with Claude Code.

<code>VERTEX_REGION_CLAUDE_3_5_HAIKU</code>	Override region for Claude 3.5 Haiku when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_3_5_SONNET</code>	Override region for Claude 3.5 Sonnet when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_3_7_SONNET</code>	Override region for Claude 3.7 Sonnet when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_0_OPUS</code>	Override region for Claude Opus 4 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_0_SONNET</code>	Override region for Claude Sonnet 4 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_1_OPUS</code>	Override region for Claude Opus 4.1 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_5_OPUS</code>	Override region for Claude Opus 4.5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_5_SONNET</code>	Override region for Claude Sonnet 4.5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_6_OPUS</code>	Override region for Claude Opus 4.6 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_6_SONNET</code>	Override region for Claude Sonnet 4.6 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_7_OPUS</code>	Override region for Claude Opus 4.7 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_4_8_OPUS</code>	Override region for Claude Opus 4.8 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_5_5_OPUS</code>	Override region for Claude Opus 5.5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_5_5_SONNET</code>	Override region for Claude Sonnet 5.5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_5_OPUS</code>	Override region for Claude Opus 5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_5_SONNET</code>	Override region for Claude Sonnet 5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_FABLE_5</code>	Override region for Claude Fable 5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_FABLE_5_1</code>	Override region for Claude Fable 5.1 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_HAIKU_4_5</code>	Override region for Claude Haiku 4.5 when using Google Cloud's Agent Platform.
<code>VERTEX_REGION_CLAUDE_HAIKU_5_5</code>	Override region for Claude Haiku 5.5 when using Google Cloud's Agent Platform.
<code>ANTHROPIC_API_KEY</code> , <code>AWS_SECRET_ACCESS_KEY</code>	Removes it.
<code>NPM_TOKEN</code> , <code>DB_PASSWORD</code>	Removes it, because the name looks like a credential.
<code>DATABASE_URL</code> that contains a password`	Removes it, because the value looks like a credential.
<code>PIP_INDEX_URL</code> or <code>NPM_CONFIG_REGISTRY</code> that contains a password`	Keeps the URL and cuts the username and password from it.

<code>GITHUB_TOKEN</code> , <code>GH_TOKEN</code> , <code>GH_ENTERPRISE_TOKEN</code> , <code>GITHUB_ENTERPRISE_TOKEN</code>	Leaves it in place, so that <code>gh</code> and scripts that call the GitHub API keep working.
<code>HTTP_PROXY</code> , <code>HTTPS_PROXY</code>	Leaves it in place, including a username and password in the URL. The sandbox can set these variables...
<code>GIT_CONFIG_COUNT</code> , <code>GIT_CONFIG_KEY_<n></code> , <code>GIT_CONFIG_VALUE_<n></code>	Leaves it in place, whatever it holds.

Glossary

Agent	An AI that can take actions, such as reading and editing files or running commands, in a loop until a task is finished, rather than only replying with text.
Agent SDK	A Python and TypeScript library that lets you build Claude Code's agent loop and tools into your own programs.
Agent view	The screen opened with <code>claude agents</code> that lists and manages background sessions.
Agentic loop	The cycle an agent repeats: gather context, plan, act, check the result.
Alias	An alternative name for a command or model, such as <code>/new</code> for <code>/clear</code> , or <code>sonnet</code> for the latest Sonnet model.
Argument	Extra information typed after a command's name, such as the focus text in <code>/compact focus on the bug</code> .
Artifact	A live web page Claude publishes on claude.ai from your session, private unless you share it.
Auto memory	Notes Claude writes for itself about your preferences and project, loaded at the start of each session.
Auto mode	A permission mode where a separate safety classifier reviews Claude's actions instead of asking you about each one.
Background session	A Claude Code session that keeps running without a terminal attached.
Bundled skill	A ready-made skill that ships with Claude Code, such as <code>/doctor</code> or <code>/loop</code> .
Bypass permissions	A permission mode (<code>--dangerously-skip-permissions</code>) that skips all checks. Meant only for isolated, disposable environments.
Checkpoint	An automatic snapshot taken before Claude edits files, which you can return to with <code>/rewind</code> .

Classifier	A second AI model that judges whether an action in auto mode matches what you asked for.
CLAUDE.md	A Markdown file of instructions Claude reads at the start of every session.
CLI	Command-line interface: a program you control by typing commands in a terminal.
Cloud session	A Claude Code session that runs on Anthropic's computers against a copy of your GitHub repository.
Commit	A saved snapshot of your files in Git, with a description.
Compaction	Replacing the older part of a conversation with a summary to free context space, automatically or with <code>/compact</code> .
Context window	The amount of conversation and file content Claude can consider at once.
Current directory	The folder your terminal is "in." Claude Code works on the files in this folder. Also called the working directory.
Custom command	A Markdown file in <code>.claude/commands/</code> that creates your own slash command; the older form of a skill.
Diff	A view showing exactly which lines of a file were added, removed, or changed.
Dynamic context	A <code>!`command`</code> line in a skill that runs before Claude reads the skill, inserting the command's output.
Dynamic workflow	A script Claude writes that runs many subagents at once in the background.
Effort	How much Claude thinks before acting, from <code>low</code> to <code>max</code> .
Environment variable	A named setting that programs can read, such as <code>ANTHROPIC_API_KEY</code> .
Fast mode	A faster, more expensive configuration of Claude Opus, toggled with <code>/fast</code> .
Frontmatter	The settings block between <code>---</code> lines at the top of a skill or subagent file.
Git	A tool that records the history of your files so you can review changes and go back in time.
GitHub Actions	GitHub's automation system, which runs workflows when things happen in a repository.
Harness	The software around the AI model, here Claude Code itself, that provides tools and manages what the model sees.
Headless mode	Running Claude Code non-interactively with <code>claude -p</code> , for scripts and automation. Also called print mode.

Hook	A command you configure to run automatically at a set moment, such as after every file edit.
Markdown	A simple text format that uses symbols like <code>#</code> and <code>-</code> for headings and lists.
MCP	Model Context Protocol, an open standard for connecting AI tools to other apps and data.
MCP prompt	A ready-made prompt an MCP server offers, which appears as a slash command.
MCP server	A connector that gives an AI new tools, such as access to Notion, GitHub, or a database.
Model	The specific AI “brain” doing the work, e.g. Opus (most capable), Sonnet (balanced), Haiku (fastest).
Output style	A setting that changes how Claude responds, such as Concise or Learning.
PATH	The list of folders your terminal searches when you type a program’s name.
Permission mode	The setting that decides which actions Claude can take without asking you.
Plan mode	A permission mode in which Claude researches and proposes a plan but makes no edits until you approve.
Plugin	An installable bundle of skills, subagents, hooks, and MCP servers.
Plugin marketplace	A catalog of plugins you can add and install from.
Prompt	The request you type to Claude. Also, the terminal’s “ready for input” marker.
Prompt caching	Reusing the unchanged start of a conversation so repeated context costs less.
Prompt injection	Hidden instructions in a web page, file, or tool result that try to trick an AI into doing something you didn’t ask for.
Remote Control	A feature that lets you continue a session running on your computer from your phone or a browser.
Repository	A folder whose history Git tracks. Often shortened to “repo.”
Routine	A task that runs on Anthropic’s cloud on a schedule, created with <code>/schedule</code> .
Sandbox	An isolated environment that limits which files and network addresses a command can reach.
Session	One saved conversation with Claude Code, which you can name and resume.

Shell	The program inside a terminal that reads and runs your commands, such as Zsh, Bash, or PowerShell.
Skill	A saved set of instructions, in a <code>SKILL.md</code> file, that Claude can use automatically or that you can run as a <code>/command</code> .
Slash command	An instruction to Claude Code itself, typed at the start of the prompt with a <code>/</code> , such as <code>/clear</code> .
Status line	A customizable bar at the bottom of the Claude Code screen.
Subagent	A specialist helper with its own instructions and tools that Claude can delegate work to.
Terminal	A text window for typing commands to your computer. Also called a shell, console, or command prompt.
Token	A small chunk of text, roughly three-quarters of a word, used to measure AI usage.
Tool	An action Claude can take, such as reading a file or running a command.
Usage credits	Paid extra usage you can add when you hit your plan's limit.
Workflow (GitHub)	An automation file in <code>.github/workflows/</code> that GitHub Actions runs.
Worktree	A separate working copy of a Git repository, so parallel tasks don't step on each other.
WSL	Windows Subsystem for Linux, a built-in way to run Linux on Windows.