

CLAUDE CODE QUICK REFERENCE

Getting Started

Install & Launch

```
npm install -g @anthropic-ai/claude-code
claude          # start interactive session
claude --version # check version
claude update   # update to latest
```

Authentication

```
claude login      # browser OAuth
export ANTHROPIC_API_KEY="sk-ant-..."
claude logout     # clear session
```

Slash Commands

Session Commands

<code>/help</code>	Show available commands
<code>/clear</code>	Clear conversation history
<code>/compact</code>	Condense context to save tokens
<code>/cost</code>	Show token usage and cost
<code>/status</code>	Show session info and model
<code>/new</code>	Start a new conversation
<code>/config</code>	Open or view configuration
<code>/doctor</code>	Diagnose configuration issues
<code>/init</code>	Create a CLAUDE.md for the project
<code>/login</code>	Authenticate with Anthropic
<code>/logout</code>	Clear authentication

Workflow Commands

<code>/bug</code>	File a bug report
-------------------	-------------------

<code>/review</code>	Request a code review
<code>/pr</code>	Create or update a pull request
<code>/commit</code>	Commit staged changes with a message

Keyboard Shortcuts

<code>Ctrl+C</code>	Cancel current generation
<code>Ctrl+D</code>	Exit Claude Code (EOF)
<code>Escape</code>	Cancel current input / edit
<code>Tab</code>	Autocomplete file paths and commands
<code>Up/Down</code>	Navigate input history

CLI Flags

Common Flags

<code>-p, --print</code>	Non-interactive (headless) mode
<code>--model</code>	Set model: opus, sonnet, haiku
<code>--output-format</code>	Output as text, json, stream-json
<code>--allowedTools</code>	Restrict tools: Edit, Read, Bash
<code>--max-turns N</code>	Limit conversation turns
<code>--debug</code>	Enable debug logging
<code>-n, --name</code>	Name the session

Headless / Scripting

```
claude -p "explain this error" < log.txt
claude -p "list todos" --output-format json
echo "fix typos" | claude -p
```

Configuration Files

CLAUDE.md Hierarchy

<code>./CLAUDE.md</code>	Project-level instructions (checked into repo)
<code>./.claude/settings.json</code>	Project settings (permissions, hooks)
<code>~/.claude/CLAUDE.md</code>	User-global instructions (all projects)
<code>~/.claude/settings.json</code>	User-global settings
<code>~/.claude/projects/*/CLAUDE.md</code>	Per-project user instructions (not in repo)

Environment Variables

<code>ANTHROPIC_API_KEY</code>	API key for direct authentication
<code>CLAUDE_MODEL</code>	Default model override
<code>CLAUDE_CONFIG_DIR</code>	Custom config directory path

Permissions

settings.json

```
{
  "permissions": {
    "allow": ["Read", "Glob", "Grep"],
    "deny": ["Bash(⌘ *)"]
  }
}
```

Permission Modes

<code>default</code>	Ask before risky tools
<code>--dangerously-skip-permissions</code>	Allow all tools (CI/scripts only)
<code>--allowedTools</code>	CLI flag to restrict tool set

CLAUDE CODE QUICK REFERENCE (continued)

MCP Servers

What Are MCP Servers?

Model Context Protocol servers extend Claude Code with custom tools — databases, APIs, services. Managed via CLI or `.mcp.json`.

CLI Management

```
claude mcp add server-name -- cmd arg1 arg2
claude mcp list
claude mcp remove server-name
```

.mcp.json Config

```
{
  "mcpServers": {
    "my-db": {
      "command": "python",
      "args": ["-m", "mcp_server_db"],
      "env": { "DB_URL": "${DATABASE_URL}" }
    }
  }
}
```

Scope

<code>.mcp.json</code>	Project-level MCP servers
<code>~/.claude/mcp.json</code>	User-global MCP servers
<code>claude mcp add -s user</code>	Add to user scope
<code>claude mcp add -s project</code>	Add to project scope

Hooks

Hook Events

<code>PreToolUse</code>	Runs before a tool is executed
<code>PostToolUse</code>	Runs after a tool completes
<code>Notification</code>	Runs when Claude sends a notification
<code>Stop</code>	Runs when Claude finishes a response

settings.json Example

```
{
  "hooks": {
    "PreToolUse": [{
      "matcher": "Bash",
      "hooks": [{
        "type": "command",
        "command": "check-command.sh $INPUT"
      }]
    }]
  }
}
```

Hook Behavior

<code>exit 0</code>	Allow the tool to proceed
<code>exit 2</code>	Block the tool from running
<code>stdout</code>	JSON feedback to Claude

SDK & Automation

Headless Scripting

```
# Single prompt, get JSON output
claude -p "summarize main.py" \
  --output-format json

# Pipe input
git diff | claude -p "review this diff"
```

CI / GitHub Actions

```
- uses: anthropics/claude-code-action@v1
  with:
    claude_args: >
      --max-turns 5
      --model claude-sonnet-4-6
```

Tips

Use `--max-turns`` to cap cost in automation. Use `--output-format json`` to parse results programmatically. Combine with `--allowedTools`` for safety.

Models

Model Selection

```
claude --model opus      # most capable
claude --model sonnet    # balanced (default)
claude --model haiku     # fastest, cheapest
```

Model Shortcuts

<code>opus</code>	Claude Opus — highest capability
<code>sonnet</code>	Claude Sonnet — default, balanced
<code>haiku</code>	Claude Haiku — fast and economical
<code>--model full-name</code>	e.g. <code>claude-sonnet-4-6</code>

Best Practices

Writing CLAUDE.md

Put project conventions, tech stack, build commands, and test instructions in CLAUDE.md. Keep it concise — Claude reads it every session. Use ``/init`` to scaffold one.

Cost Management

<code>/cost</code>	Check running token usage
<code>/compact</code>	Compress context when it grows large
<code>--max-turns</code>	Cap turns in automation
<code>sonnet / haiku</code>	Use cheaper models for simple tasks

Effective Prompting

<code>Be specific</code>	"Fix the null check in auth.ts:42" > "fix bug"
<code>Give context</code>	Reference files, functions, error messages
<code>Use @file</code>	Reference files directly: <code>@src/main.ts</code>
<code>Use images</code>	Drag & drop screenshots for visual context
<code>Iterate</code>	Follow up to refine; use <code>/clear</code> to reset