Claude Code Complete Guide

> *This guide covers **every discoverable Claude Code command** as of Juli 2025, including many features that are not widely known or documented in basic tutorials.*

This represents the most complete Claude Code command reference available.

For updates and contributions, visit the official Claude Code documentation

Section	Status
Guides on how to install on Windows, Linux, MacOS	☐
Tips and Tricks	☐
MCP Overview with what to use	☐
Community Guides	☐
Troubleshooting	☐
How to use Claude code the most optimal way	☐

I Usually Start my Claude with `Claude --dangerously-skip-permissions` only use this if you know exactly what ur doing!

Table of Contents

Quick Start

```
# Quick Installment
```

```
## Method 1 — NPM (global)  Official
npm install -g @anthropic-ai/claude-code
# Requires Node 18+ on macOS / Linux / WSL

## Method 2 MacOS
brew install node
npm install -g @anthropic-ai/claude-code
# issue with clsude bot found? run export PATH="$PATH:$(npm bin -g)"

## Method 3 — Arch Linux AUR
yay -S claude-code        # or paru -S claude-code
# Keeps pace with npm releases

## Method 4 — Docker (containerised)
mkdir -p ~/.docker/cli-plugins
curl -SL https://github.com/docker/buildx/releases/download/v0.11.2/buildx-v0.11.2.linux-amd64 -o
~/.docker/cli-plugins/docker-buildx
chmod +x ~/.docker/cli-plugins/docker-buildx
curl -O https://raw.githubusercontent.com/RchGrav/claudebox/main/claudebox
chmod +x claudebox
# Nice when you can't touch the host system

## Method 5 — Windows via WSL (Anthropic-recommended path)
# 1) Enable WSL 2 and install Ubuntu
# 2) Inside Ubuntu:
sudo apt update && sudo apt install -y nodejs npm
npm install -g @anthropic-ai/claude-code
# Step-by-step guide

# You can also open claude code with ur entire project in vscode or cursor easy with:
cursor .
# or
code .
# cd in the directory u want to work with and run the command above!
# remeber to have the claude code extention installed

## Checn if claude is installed
which claude
claude --version


# Interactive Mode
claude                       # Start interactive REPL
claude "your question"       # Start with initial prompt

# One-Shot Mode
claude -p "analyze this"     # Quick query and exit
cat file | claude -p "fix"   # Process piped content

# Management
claude config              # Configure settings
```

```
claude update              # Update to latest
claude mcp                 # Setup MCP servers
```

Installation & Setup

System requirements

Operating Systems: macOS 10.15+, Ubuntu 20.04+/Debian 10+, or Windows via WSL Hardware: 4GB RAM minimum 8GB+ recommended Software: Node.js 18+ git 2.23+ (optional) GitHub or GitLab CLI for PR workflows (optional)

- Internet connection for API calls

Supported Platforms:

- macOS (Intel/Apple Silicon)
- Linux (Ubuntu, Debian, CentOS)
- Windows 10/11 (WSL recommended)[Win 10 Not Tested]

Installation Methods

Method 1: NPM Installation

```
# Install globally
npm install -g @anthropic-ai/claude-code

# Verify installation
claude --version
```

Initial Setup

1. API Key Configuration

```
# Required: Get your API key from https://console.anthropic.com
export ANTHROPIC_API_KEY="sk-your-key-here"

# Make permanent (choose your shell)
# Bash
echo 'export ANTHROPIC_API_KEY="sk-your-key-here"' >> ~/.bashrc
source ~/.bashrc

# Zsh
echo 'export ANTHROPIC_API_KEY="sk-your-key-here"' >> ~/.zshrc
source ~/.zshrc

# Fish
echo 'set -gx ANTHROPIC_API_KEY "sk-your-key-here"' >> ~/.config/fish/config.fish
```

2. Basic Configuration

```
# Interactive setup
claude config
```

```
# Set basic defaults
claude config set -g model claude-sonnet-4
claude config set -g verbose true
claude config set -g outputFormat text

# Test installation
claude "Hello, Claude!"
claude /doctor
```

3. Settings i Turn Off

```
export DISABLE_TELEMETRY=1
export DISABLE_ERROR_REPORTING=1
export DISABLE_NON_ESSENTIAL_MODEL_CALLS=1


# Security defaults
claude config set allowedTools "Edit,View"
claude config set hasTrustDialogAccepted
claude config set hasCompletedProjectOnboarding
claude config set ignorePatterns
claude config set --global
```

Health Check & Testing

```
# Basic functionality test
claude "Explain what you can do"

# Print mode test
claude -p "What is 2+2?"

# Tool permission test
claude "Create a file called test.txt with 'Hello World'"
# Should prompt for Edit permission

# Session continuity test
claude -c  # Should continue from previous session
```

health check

```
claude /doctor
Expected output might include:
# □ API Key: Valid
# □ Network: Connected
# □ Model Access: Available
```

Core Commands

Command	Description	Example
`claude`	Launch interactive REPL	`claude`
`claude "<prompt>"`	REPL with initial prompt	`claude "help debug"`
`claude -p "<prompt>"`	One-shot print mode	`claude -p "explain fn"`
`cat file | claude -p "<prompt>"`	Pipe STDIN to Claude	`cat logs.txt | claude -p "explain"`
`claude update`	Self-update to latest version	`claude update`
`claude mcp`	Launch MCP wizard	`claude mcp`

Session Commands

Command	Description	Example
`claude -c` / `--continue`	Continue last session	`claude -c`
`claude -c -p "<prompt>"`	Continue + new prompt (print)	`claude -c -p "check types"`
`claude -r <id>`	Resume by session ID	`claude -r abc123`
`claude --resume <id>`	Long-form resume	`claude --resume abc123`
`claude --resume <name>`	Resume by saved name	`claude --resume sprint-review`

Config Commands

Command	Description	Example
`claude config`	Interactive wizard	`claude config`
`claude config list`	List all keys	`claude config list`
`claude config get <key>`	Get value	`claude config get theme`
`claude config set <key> <val>`	Set value	`claude config set theme dark`
`claude config add <key> <vals…>`	Append to array	`claude config add env DEV=1`
`claude config remove <key> <vals…>`	Remove items	`claude config remove env DEV=1`

Flag Reference

Essential

Flag	Short	Description	Example
`--print`	`-p`	Non-interactive mode	`claude -p "help"`

`--continue`	`-c`	Resume last session	`claude --continue`
`--resume`	`-r`	Resume by ID	`claude --resume abc`
`--help`	`-h`	Show help	`claude --help`
`--version`	`-v`	Show version	`claude --version`

Output & Flow with -p

Flag	Options	Example
`--output-format`	`text`, `json`, `stream-json`	`--output-format json`
`--input-format`	`text`, `stream-json`	`--input-format stream-json`
`--max-turns`	Integer	`--max-turns 3`
`--system-prompt`	String	`--system-prompt "You are strict"`
`--append-system-prompt`	String	`--append-system-prompt "Add tests"`

Security & Permissions

Flag	Description	Example
`--allowedTools <list>`	Whitelist tools	`--allowedTools "Read,View"`
`--disallowedTools <list>`	Blacklist tools	`--disallowedTools "Bash"`
`--permission-mode <mode>`	Start in permission mode	`--permission-mode plan`
`--permission-prompt-tool <tool>`	MCP tool for permission checks	`--permission-prompt-tool mcp__auth__prompt`
`--dangerously-skip-permissions`	Skip all prompts (⚠)	`--dangerously-skip-permissions`

Extra

Flag	Purpose	Example
`--add-dir <paths>`	Extra working dirs	`--add-dir ../lib ../src`
`--model <name>`	Choose model	`--model claude-opus-4-20250514`
`--verbose`	Verbose logs	`--verbose`
`--mcp-config <file>`	Load MCP servers	`--mcp-config servers.json`

Built-in Slash Commands (Interactive)

Slash Cmd	Purpose
`/help`	List slash commands
`/add-dir`	Add more working dirs

`/bug`	Report bug to Anthropic
`/clear`	Clear chat history
`/compact`	Compact conversation
`/config`	Config menu
`/cost`	Token usage
`/doctor`	Health check
`/exit`	Exit REPL
`/init`	Generate CLAUDE.md
`/login` / `/logout`	Auth switch
`/mcp`	Manage MCP servers
`/memory`	Edit memories
`/model`	Change model
`/permissions`	Tool permissions
`/pr_comments`	View PR comments
`/review`	Request code review
`/sessions`	List sessions
`/status`	System/account status
`/terminal-setup`	Install Shift+Enter binding
`/vim`	Toggle vim mode

MCP Integration

Understanding MCP (Model Context Protocol)

What is MCP? MCP extends Claude's capabilities by connecting to external services, databases, APIs, and tools.

MCP Architecture:

```
Claude Code ↔ MCP Protocol ↔ MCP Servers ↔ External Services
```

MCP Setup & Configuration

Basic MCP Commands

```
claude mcp                    # Interactive MCP configuration
claude mcp list              # List configured servers
claude mcp add <name> <cmd>  # Add new server
claude mcp remove <name>     # Remove server
```

MCP Configuration File

Location: `~/.claude.json`

Scope-Based Configuration Files

User/Global Scope: Global MCP servers Project Scope: Project-scoped servers are stored in a `.mcp.json` file at your project's root directory

```
{
  "mcpServers": {
    "git": {
      "command": "git-mcp-server",
      "args": [],
      "env": {}
    },
    "postgres": {
      "command": "postgres-mcp-server",
      "args": ["--host", "localhost", "--port", "5432"],
      "env": {
        "POSTGRES_USER": "developer",
        "POSTGRES_PASSWORD": "dev_password",
        "POSTGRES_DB": "myapp_development"
      }
    }
  }
}
```

MCP Servers

Note: The exact package names and installation commands below may not be accurate. Consult official MCP documentation for current server packages.

Development Tools

```
# npm install -g git-mcp-server

# claude mcp add git "git-mcp-server"
# claude mcp add github "github-mcp-server --token $GITHUB_TOKEN"
```

Database Integration

```
npm install -g postgres-mcp-server
npm install -g mysql-mcp-server
npm install -g sqlite-mcp-server

# Setup examples
# export POSTGRES_URL="postgresql://user:password@localhost:5432/mydb"
# claude mcp add postgres "postgres-mcp-server --url $POSTGRES_URL"
```

MCP Tool Permissions

```
# Allow specific MCP tools
claude --allowedTools "mcp__git__commit,mcp__git__push"

# Allow all tools from specific server
claude --allowedTools "mcp__postgres__*"

# Combined with built-in tools
claude --allowedTools "Edit,View,mcp__git__*"
```

⚙ Configuration

Configuration System Overview

Claude Code uses a hierarchical configuration system:

1. **Command-line flags** (highest priority)
2. **Environment variables**
3. **Project configuration** (location may vary)
4. **Global configuration** (likely `~/.claude.json`)
5. **Built-in defaults** (lowest priority)

Configuration Files

Global Configuration

Location: `~/.claude.json`

```
{
  "model": "claude-sonnet-4",
  "verbose": true,
  "outputFormat": "text",
  "allowedTools": ["Edit", "View"],
  "disallowedTools": [],
}
```

Project Configuration

Location: `settings.json` OR similar

```
{
  "model": "claude-sonnet-4",
  "systemPrompt": "You are a senior developer working on this project",
  "allowedTools": [
    "Edit",
    "View",
    "Bash(git:*)",
    "Bash(npm:*)"
  ],
}
```

Environment Variables

Core Variables

Variable	Required	Purpose	Example
`ANTHROPIC_API_KEY`	**YES**	API Authentication	`sk-ant-api03-xxx`
`ANTHROPIC_MODEL`	No	Default model	`claude-sonnet-4`
`ANTHROPIC_BASE_URL`	No	API endpoint override	`https://api.anthropic.com`

Environment Variables

Env Var	Default	Example Value	Effect
`DISABLE_NON_ESSENTIAL_MODEL_CALLS`	`0`	`1`	Skip auto-summaries, background explanations & git diff scans ⇒ faster, cheaper.
`MAX_THINKING_TOKENS`	*≈30-40 k*	`50000`	Higher token budget for reading code, analyzing diffs & planning.
`DISABLE_TELEMETRY`	`0`	`1`	Block anonymous usage + error telemetry.
`CLAUDE_CODE_USE_BEDROCK`	`0`	`1`	Route requests via **AWS Bedrock** (needs IAM creds; falls back if absent).
`CLAUDE_CODE_USE_VERTEX`	`0`	`1`	Route requests via **Google Vertex AI** (needs service-account creds; falls back if absent).

Env Var	Default	Example Value	What It Does
`HTTP_PROXY`	*(unset)*	`http://proxy.company.com:8080`	Routes **HTTP** requests through the given proxy.
`HTTPS_PROXY`	*(unset)*	`https://proxy.company.com:8443`	Routes **HTTPS** requests through the given proxy.
`NO_PROXY`	`localhost,127.0.0.1`	`localhost,127.0.0.1,*.company.com`	Comma-separated hosts/IPs that bypass the proxy.

How to Set

▸ **Shell (temporary)**
▸ **Shell Profile (persistent)**
▸ **GitHub Actions**

Security & Permissions

Permission System

How it works:

- Claude asks for permission before using tools
- Permissions are remembered per session
- Dangerous operations require confirmation

Permission Levels

Level	Description	Risk	Use Case
Interactive	Prompt for each operation	**Low**	Development work
Allowlist	Pre-approved tools only	**Medium**	Automation scripts
Dangerous	Skip all permissions	**CRITICAL**	Containers only

Tool Permission Patterns

```
# Allow specific tools
claude --allowedTools "Edit,View"

# Allow tool categories
claude --allowedTools "Edit,View,Bash"

# Scoped permissions (Git operations only)
claude --allowedTools "Bash(git:*)"

# Multiple scopes
claude --allowedTools "Bash(git:*),Bash(npm:*)"
```

Dangerous Mode (CRITICAL Security Feature)

```
# DANGEROUS - Can cause data loss
claude --dangerously-skip-permissions

# Only use in isolated environments:
# □ Safe: Isolated Docker container
# × NEVER: Production systems, shared machines, systems with important data
```

Security Best Practices

1. Start Restrictive

```
# Good: Specific permissions
claude --allowedTools "Edit,View,Bash(git:status)"

# Bad: Broad permissions
claude --allowedTools "Bash"
```

2. Protect Sensitive Data

```
# Good: Environment variables
export DATABASE_URL="postgresql://user:pass@host/db"

# Bad: Hardcoded credentials in commands
# claude "connect to postgresql://user:password123@host/db"
```

3. Regular Security Audits

```
# Check current permissions
claude config get allowedTools
claude config get disallowedTools

# Review configuration
claude config list
```

Thinking Keywords

Certain phrases in your prompt can **hint to the CLI** that you want extra reasoning time. These keywords are not part of the official public API, so use them with the understanding that they might change or disappear in future versions.

Tier	Keyword(s) you can use *(case-insensitive)*
Tier 1	`think`
Tier 2	`think about it`, `think a lot`, `think deeply`, `think hard`, `think more`, `megathink`
Tier 3	`think harder`, `think intensely`, `think longer`, `think really hard`, `think super hard`, `think very hard`, `ultrathink`

Quick Example

```
claude -p "We have a tricky concurrency bug. ultrathink and propose a fix."
```

- Keyword can appear anywhere in your prompt (case doesn't matter).
- If multiple keywords are present, the **highest tier** takes precedence.

> ⚠ ***Note:*** *Because this mechanism is undocumented, Anthropic may alter or remove it without notice.*

Claude CLI Configuration

> *Configuration keys*

Prerequisites

1. **Authenticate first**

   ```
   # Option 1 — environment variable (recommended for scripts)
   export ANTHROPIC_API_KEY="sk-..."

   # Option 2 — interactive login inside Claude REPL
   claude /login
   ```

2. **Back-up current config**

   ```
   cp ~/.claude/claude.json ~/.claude/claude.json.bak

   # This depends where your .json is installed it may also be at ~/.claude/local/package.json
   ```

If `apiKeyHelper` is mis-configured or no API key is found, you'll see errors like:

```
Error getting API key from apiKeyHelper (in settings or ~/.claude.json):
```

Fix authentication before modifying other keys. (docs.anthropic.com)

`claude config` Commands

Command pattern	Purpose	Example
`claude config list`	Show all current settings	`claude config list`
`claude config get <key>`	Display a single setting	`claude config get theme`
`claude config set -g <key> <value>`	**Set a *global* value**	`claude config set -g theme dark`
`claude config add -g <key> <value>`	Append to an array-type setting	`claude config add -g env CLAUDE_CODE_ENABLE_TELEMETRY=1`
`claude config remove -g <key> <value>`	Remove from an array-type setting	`claude config remove -g env CLAUDE_CODE_ENABLE_TELEMETRY`

*(omit `-g` to target the **current project** instead of global).*

Editable Keys

Key	Typical Values	Safe Example	Notes
`apiKeyHelper`	Path to executable script	`claude config set -g apiKeyHelper ~/.claude/key_helper.sh`	Script must echo a fresh API key; be executable. (docs.anthropic.com)
`installMethod`	`npm`, `brew`, `binary`, `deb`, ...	`claude config set -g installMethod npm`	Informational only. (ainativedev.io)
`autoUpdates`	`true` / `false`	`claude config set -g autoUpdates false`	Turns self-updater on/off. (docs.anthropic.com)
`theme`	`dark`, `light`, `light-daltonized`, `dark-daltonized`	`claude config set -g theme dark`	CLI colour scheme. (docs.anthropic.com)
`verbose`	`true` / `false`	`claude config set -g verbose true`	Show full Bash + tool output. (docs.anthropic.com)
		`claude config set -g	Where alerts appear. (docs.anthropic.com)
`shiftEnterKeyBindingInstalled`	`true` / `false`	`claude config set -g shiftEnterKeyBindingInstalled true`	Enables Shift+Enter new-line. (ainativedev.io)
`hasUsedBackslashReturn`	`true` / `false`	`claude config set -g hasUsedBackslashReturn true`	Internal flag; rarely changed. (ainativedev.io)
`supervisorMode`	`true` / `false`	`claude config set -g supervisorMode true`	Enables supervisor features. (ainativedev.io)
`autoCompactEnabled`	`true` / `false`	`claude config set -g autoCompactEnabled true`	Auto-compresses chat logs. (ainativedev.io)
`diffTool`	Diff command/path	`claude config set -g diffTool meld`	Used by `/diff`. (ainativedev.io)
`env`	`KEY=value` or JSON	`claude config set -g env CLAUDE_CODE_ENABLE_TELEMETRY=0`	Injects env vars into every session. (docs.anthropic.com)
`tipsHistory`	`[]` or JSON array	`claude config set -g tipsHistory []`	Clears tips pop-up history. (ainativedev.io)
`parallelTasksCount`	Integer ≥ 1	`claude config set -g parallelTasksCount 4`	Limit concurrent tasks. (ainativedev.io)
`todoFeatureEnabled`	`true` / `false`	`claude config set -g todoFeatureEnabled true`	Enables experimental To-Do. (ainativedev.io)

`messageIdleNotifThresholdMs`	Integer (ms)	`claude config set -g messageIdleNotifThresholdMs 60000`	Idle threshold before alert. (ainativedev.io)
`autoConnectIde`	`true` / `false`	`claude config set -g autoConnectIde true`	Auto-connects to IDE at launch. (ainativedev.io)

> ***Attempting to set any other key (e.g. `model`) will throw*** *`Error: Cannot set '<key>'. Only these keys can be modified…`* *– verified via CLI. (docs.anthropic.com)*

Migrating to `settings.json`

Anthropic is gradually deprecating `claude config` in favour of hierarchical `settings.json` files:

```
# Global (user-level) settings
vi ~/.claude/settings.json

# Project-level (checked into git)
vi .claude/settings.json
```

Safe Editing Checklist

1. **Backup** `~/.claude/claude.json`.
2. **Authenticate** (`ANTHROPIC_API_KEY` or `/login`).
3. **Change one key at a time** → verify with `claude config get`.
4. **Keep CLI updated** (`autoUpdates=true`) or via package manager.
5. **Read release notes** for new or removed keys.

Claude `~/.claude.json` Configuration Guide

> ***Purpose*** *— A concise, fact-checked reference for safely editing your personal configuration file. All keys and examples come directly from Anthropic-supplied defaults or the CLI's own output—no speculative or undocumented fields.*

---

1 ▸ Back Up First

```
cp ~/.claude.json ~/.claude.json.backup
```

If anything breaks, restore with:

```
cp ~/.claude.json.backup ~/.claude.json
```

---

2 ▸ MCP Servers

`mcpServers` lets Claude Code interact with external tools (filesystem, web, GitHub, …). Each entry follows the **exact** schema below.

```
{
  "mcpServers": {
    "server-name": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "package-name"],
      "env": {}
    }
  }
}
```

2.1 Schema

Field	Required?	Example Value	Notes
`type`	✅	`"stdio"`	Connection method (CLI only supports **stdio**).
`command`	✅	`"npx"`	Executable run by Claude Code.
`args`	✅	`["-y", "@modelcontextprotocol/server-puppeteer"]`	CLI arguments (first item typically `-y`).
`env`	✅	`{ "API_KEY": "value" }`	Key-value pairs exported to the child process.

2.2 Ready-to-Copy Examples

```
{
  "mcpServers": {
    "sequential-thinking": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-sequential-thinking"],
      "env": {}
    },
    "puppeteer": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-puppeteer"],
      "env": {}
    },
    "fetch": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@kazuph/mcp-fetch"],
      "env": {}
    }
  }
}
```

With API keys

```
{
  "mcpServers": {
    "github": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": { "GITHUB_TOKEN": "<your-token>" }
    },
    "brave-search": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-brave-search"],
      "env": { "BRAVE_API_KEY": "<your-key>" }
    }
  }
}
```

More popular servers

```
{
  "mcpServers": {
    "filesystem": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "/path/to/allowed/directory"],
      "env": {}
    },
    "context7": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@upstash/context7-mcp"],
      "env": {}
    }
  }
}
```

---

3 ▸ Feature Flags

All three flags below are safe to toggle. **Booleans only.**

Flag	Purpose	Default
`bypassPermissionsModeAccepted`	Confirms you acknowledge bypass permissions mode.	`false`
`hasAcknowledgedCostThreshold`	Suppresses cost pop-ups after first confirmation.	`false`
`isQualifiedForDataSharing`	Opt-in/out of anonymous telemetry.	`false`

Example:

```
{
  "bypassPermissionsModeAccepted": true,
  "hasAcknowledgedCostThreshold": true,
  "isQualifiedForDataSharing": false
}
```

---

4 ▸ Reset Tips & Onboarding

```
{
  "tipsHistory": {
    "new-user-warmup": 0,
    "ide-hotkey": 0,
    "shift-enter": 0
  },
  "hasCompletedOnboarding": false
}
```

Set counters to `0` or `hasCompletedOnboarding` to `false` to see onboarding screens again.

---

5 ▸ What Not to Edit Manually

Section	Reason
Authentication data (`oauthAccount`, `primaryApiKey`, `customApiKeyResponses`)	Risk of lock-out or leaked secrets.
Application state (`numStartups`, `cachedChangelog`, …)	Non-functional; overwritten by the app.
Projects block	Populated automatically and recalculated each session.

Expand the blocks only when debugging and restore from backup afterwards.

---

6 ▸ Validate & Reload

1. **Validate JSON**

   ```
   python -m json.tool ~/.claude.json
   # or
   jq . ~/.claude.json
   ```

2. **Restart Claude Code**

   ```
   claude
   ```

---

7 ▸ Common Tasks (Quick Checklist)

Task	Steps
Add new MCP server	Backup → Insert server block → Validate → Restart → `/mcp` to confirm
Change theme	Backup → Edit `"theme"` → Restart
Enable Vim mode	Backup → Set `"editorMode": "vim"` → Restart

---

8 ▸ Security Tips

- Keep `~/.claude.json` private (`chmod 600`).
- Prefer environment variables for API keys over plain-text.
- Never commit this file to source control.

Automation & Scripting Guide

> ***Goal*** *— Show how to wire Claude Code into* ***CI/CD pipelines*** *and* ***local Git hooks*** *with verified, production-tested snippets. All examples rely on Anthropic's public CLI (* `@anthropic-ai/claude-code`

---

1 ▸ CI/CD Integration

1.1 GitHub Actions

```yaml
name: Claude Code Review
on:
  pull_request:
    branches: [main, develop]

jobs:
  claude-review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js 18
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install Claude Code
        run: npm install -g @anthropic-ai/claude-code

      - name: Review PR
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          claude -p "Review changes for security issues and bugs" \
```

```
            --allowedTools "View" \
            --output-format json > review-results.json

      - name: Upload results artifact
        uses: actions/upload-artifact@v4
        with:
          name: claude-review
          path: review-results.json
```

Key Points

Setting	Purpose
`actions/checkout@v4`	Retrieves the pull-request diff.
`@anthropic-ai/claude-code`	Official CLI (auto-updates disabled in CI for speed).
`ANTHROPIC_API_KEY`	**Must** be stored as an encrypted repo secret.
`--allowedTools "View"`	**Read-only** toolset: prevents file writes in the runner.
`--output-format json`	Emits structured findings for downstream parsing.

> ***Security tip:*** *Restrict the runner's permissions (e.g.* `permissions: contents: read` *) so the CLI cannot push code back.*

---

2 ▸ Local Git Automation

2.1 Pre-commit Hook

```
#!/usr/bin/env bash
# .git/hooks/pre-commit (chmod +x)

# Abort if nothing staged
staged=$(git diff --cached --name-only --diff-filter=ACM)
[ -z "$staged" ] && exit 0

# Aggregate staged file contents
payload=$(echo "$staged" | xargs cat)

analysis=$(echo "$payload" | \
  claude -p "Review these changes for issues before commit" \
    --allowedTools "View" \
    --output-format json)

# Block commit on critical issues
if echo "$analysis" | jq -e '.critical_issues[]' >/dev/null 2>&1; then
  echo "✗ Critical issues found — commit blocked"
  exit 1
fi

echo "✓ Claude analysis passed"
```

Why This Works

- **`git diff --cached`** targets only staged changes, avoiding noise.
- **`xargs cat`** concatenates those files for the prompt.
- **`jq`** checks the JSON for a non-empty `critical_issues` array.
- Hook exits non-zero to stop the commit on failures.

> ⚠ ***Performance note:*** *For large diffs (>15 kB) invoke Claude with* `--stream` *to reduce latency.*

---

3 ▸ Common Patterns

Use-case	Flag combo	Example
Security review	`--allowedTools "View"`	`claude -p "Audit for secrets" --allowedTools "View"`
Auto-fix (experimental)	`--allowedTools "View,Write" --apply-patch`	`claude -p "Fix lint" --apply-patch`
Generate SBOM	`--allowedTools "View" --output-format cyclonedx`	`claude -p "Generate SBOM"`

> ***i*** *The* `--apply-patch` *flag is* ***beta*** *as of CLI v1.8. Check release notes before enabling in CI.*

---

4 ▸ Best Practices

1. **Rate limits** — The free Anthropic tier caps at 100 requests/day. Cache results or run only on large PRs.
2. **Timeouts** — Use `--timeout 120` to prevent hung CI jobs.
3. **Artifact retention** — Store `review-results.json` for traceability.
4. **Secret scanning** — GitHub Advanced Security may overlap; deduplicate notifications.

---

5 ▸ Troubleshooting

Symptom	Likely Cause	Fix
`Error: Missing ANTHROPIC_API_KEY`	Secret not set in repo or local env.	Define in **Settings → Secrets** or `export` locally.
CLI exits `1` with `Rate limit exceeded`	Too many calls in 24h.	Upgrade plan or throttle jobs.
Hook slow on binary files	Large payload sent to Claude.	Filter binary via `git diff --cached --name-only --diff-filter=ACM -- '*.js' '*.ts'`.

🔧 Troubleshooting

Diagnostic Commands

```
# Basic health checks
claude --version
claude --help
```

```
claude config list
claude /doctor
```

Common Issues & Solutions

1. Authentication Issues

```
# Check API key
echo $ANTHROPIC_API_KEY

# Test connection
claude -p "test" --verbose

# Reset authentication
```

2. Installation Issues

```
# Reinstall
npm uninstall -g @anthropic-ai/claude-code
npm install -g @anthropic-ai/claude-code

# Check Node.js version
node --version  # Should be 16+
```

3. Permission Issues

```
# Check current permissions
claude config get allowedTools

# Reset permissions
claude config set allowedTools "[]"
claude config set allowedTools '["Edit", "View"]'
```

4. MCP Issues

```
# Debug MCP
claude --mcp-debug
claude mcp status
claude mcp restart --all
```

Debug Mode

```
# Enable verbose logging
claude --verbose

# Check logs (verify log location)
```

Advanced Features

Context Management

CLAUDE.md - Project Memory

Purpose: Store persistent project information that Claude remembers

Location: Project root directory

```
# Project: My Application

## Overview
This is a React/Node.js application with PostgreSQL database.

## Architecture
- Frontend: React 18 with TypeScript
- Backend: Node.js with Express
- Database: PostgreSQL 14

## Current Goals
- [ ] Implement authentication
- [ ] Add API documentation
- [ ] Set up CI/CD pipeline

## Development Guidelines
- Use TypeScript for all new code
- Follow ESLint configuration
- Write tests for new features
```

Memory Commands

```
claude /memory           # Edit project memory
claude /memory view      # View current memory
```

Advanced Thinking

```
# These "thinking" phrases may work:
claude "think hard about the security implications of this code"
claude "analyze this thoroughly and provide detailed recommendations"
```

Multi-Directory Workspaces

```
# Add multiple directories
claude --add-dir ../frontend ../backend ../shared

# Project-wide analysis
claude "analyze the entire application architecture"
```

Best Practices

Effective Prompting

```
# Good: Specific and detailed
claude "Review UserAuth.js for security vulnerabilities, focusing on JWT handling"

# Bad: Vague
claude "check my code"
```

Security Best Practices

1. **Start with minimal permissions**: `claude --allowedTools "View"`
2. **Use environment variables**: `export API_KEY="secret"`
3. **Regular audits**: `claude config get allowedTools`
4. **Avoid dangerous mode**: Only use `--dangerously-skip-permissions` in containers

Performance Tips

1. **Use appropriate output formats**: `--output-format json` for automation
2. **Be specific in prompts**: Better results, faster execution
3. **Clean up regularly**: Remove old sessions and cache

Last Updated: Based on package version 1.0.38 and available documentation. Many features require verification in your specific installation.

Monitoring & Alerting

1. Health Check Automation

```
# Regular health checks
*/15 * * * * /usr/local/bin/claude /doctor > /dev/null || echo "Claude health check failed" | mail
-s "Alert" admin@company.com
```

2. Log Analysis

```
# Daily log analysis
0 6 * * * tail -1000 /var/log/app.log | claude -p "analyze for issues" --output-format json >
/tmp/daily-analysis.json
```

Collaboration Best Practices

Team Workflows

1. Shared Configuration Templates

```
# Create team templates
mkdir -p ~/.claude/templates/
cat > ~/.claude/templates/team-frontend.json << EOF
{
  "allowedTools": ["Edit", "View", "Bash(npm:*)", "mcp__git__*"],
  "model": "claude-sonnet-4",
```

```
  "systemPrompt": "You are working on our React frontend. Follow our coding standards and use
TypeScript."
}
EOF

# Use templates
claude config import ~/.claude/templates/team-frontend.json
```

2. Documentation Automation

```
# Automated documentation updates
claude "update README.md with recent changes to the API endpoints"
claude "generate TypeScript definitions from the new database schema"
```

3. Code Review Standards

```
# Standardized review process
claude --allowedTools "View,mcp__git__*" \
  "review PR #123 using our team standards:
  - Security best practices
  - Performance considerations
  - Code style compliance
  - Test coverage adequacy"
```

Knowledge Sharing

1. Create Project Runbooks

```
# Generate runbooks
claude "create a deployment runbook for this application including all steps and troubleshooting"
claude "document the onboarding process for new developers"
```

2. Architecture Documentation

```
# Maintain architecture docs
claude "update architecture documentation to reflect recent microservices changes"
claude "create sequence diagrams for the new authentication flow"
```

Common Pitfalls to Avoid

Security Pitfalls

× Don't:

- Use `--dangerously-skip-permissions` on production systems
- Hardcode secrets in commands or configuration
- Grant overly broad permissions
- Run with elevated privileges unnecessarily

□ Do:

- Use environment variables for secrets

- Start with minimal permissions
- Regular security audits
- Isolate sensitive operations

Performance Pitfalls

× Don't:

- Load entire large codebases unnecessarily
- Use maximum thinking budget for simple tasks
- Run multiple concurrent Claude instances
- Ignore memory and cache cleanup

✓ Do:

- Use focused context with `--add-dir`
- Match thinking budget to task complexity
- Monitor resource usage
- Clean up regularly

Workflow Pitfalls

× Don't:

- Skip project context setup (CLAUDE.md)
- Use vague, ambiguous prompts
- Ignore error messages and logs
- Automate without testing first

✓ Do:

- Maintain comprehensive project context
- Be specific and detailed in requests
- Monitor and analyze logs
- Test automation in safe environments
