Claude Code Slash Commands Reference Guide

This comprehensive guide lists all available slash commands in Claude Code, organized alphabetically. Each command includes a description of what it does and when to use it.

Documentation Status Key

Commands in this guide are marked according to their documentation status:

- **No marker** - Fully documented in the official Slash Commands documentation
- ***(See: [Location])*** - Documented elsewhere in Claude Code documentation, with the location specified
- ***(Undocumented)*** - Functional but not documented anywhere in the official Claude Code documentation

Note: Undocumented commands were discovered through community usage and reported in GitHub Issue #6493. As of September 26, 2025, this guide reflects the current state of Claude Code documentation.

Built-in Slash Commands

/agents *(See: Subagents documentation)*

Description: Opens a comprehensive interface for creating and managing specialized AI subagents in Claude Code. This command allows you to define custom subagents with their own system prompts, tool permissions, and separate context windows for handling specific types of tasks. The interface lists all available tools, including any connected MCP server tools, making it easy to configure exactly which capabilities each subagent should have.

When to Use: Use this command when you want to create specialized assistants for specific workflows like code review, testing, debugging, or documentation. It's particularly valuable for complex projects where you need to delegate tasks to focused AI specialists without polluting your main conversation context. The command is essential for scaling your development workflow by creating reusable, task-specific agents that can work independently while maintaining their own context and tool permissions.

/bashes *(Undocumented)*

Description: Lists and manages background bash shells that are running in your Claude Code session. This command provides visibility into any persistent shell processes that Claude has spawned and allows you to manage them effectively.

When to Use: Use this command when you need to check what background processes Claude is running, especially during complex operations that involve multiple shell sessions. It's

particularly helpful for debugging issues related to lingering processes or when you need to clean up background tasks that may still be consuming resources.

/clear

Description: Resets the conversation history and context within your current session. This clears out all previous messages and file contents from Claude's context window without ending the session itself.

When to Use: Use this command frequently between tasks to keep Claude focused and prevent context pollution. It's especially important when switching from one feature or problem to another within the same project, as it helps avoid confusion from irrelevant prior context.

/compact

Description: Compresses the current conversation by having Claude create a summary of the important context, allowing you to continue without hitting context limits. You can provide specific instructions about what context to preserve, such as "/compact Focus on preserving our authentication implementation and database schema decisions."

When to Use: Use this when you're approaching context limits but need to maintain continuity in your work. It's particularly useful during long debugging sessions or complex feature implementations where you can't afford to lose critical context but need to free up space.

/config

Description: Opens the configuration interface for Claude Code settings. This allows you to modify various preferences and settings for your Claude Code installation, including tool permissions and default behaviors.

When to Use: Use this command when you need to adjust Claude Code's behavior, modify allowed tools, or change other configuration options. It's helpful when setting up Claude Code for the first time or adapting it for specific project requirements.

/context *(Undocumented)*

Description: Visualizes your current context usage, showing how much of Claude's context window is being consumed by the current conversation. This provides a visual representation or statistics about how much space different elements (messages, file contents, tool outputs) are taking up in the context.

When to Use: Use this command to understand how close you are to context limits and what's consuming the most space. It's invaluable for optimizing your prompts and deciding when to

use /clear or /compact. This helps you manage long sessions more effectively by understanding exactly where your tokens are being used.

/cost

Description: Displays detailed token usage statistics and estimated costs for your current session. Shows how many tokens have been consumed and provides cost estimates based on current pricing models.

When to Use: Use this to monitor your token consumption and manage costs effectively. It's particularly important during long sessions or when working on token-intensive tasks to ensure you're optimizing your prompts and staying within budget.

/doctor

Description: Runs a diagnostic health check on your Claude Code installation. This command helps identify and troubleshoot installation issues, configuration problems, or connectivity issues with various services.

When to Use: Use this command when experiencing unexpected behavior, installation problems, or when Claude Code isn't working as expected. It's your first line of defense for troubleshooting technical issues.

/exit

Description: Ends the current interactive Claude Code session completely. This closes the conversation and exits the Claude Code interface, unlike /clear which maintains the session.

When to Use: Use this when you're completely finished with your work and want to close Claude Code. It's the proper way to end a session rather than simply closing your terminal, as it ensures proper cleanup and session saving.

/export *(Undocumented)*

Description: Exports the current conversation to a file or copies it to your clipboard for sharing or archival purposes. This creates a record of your entire session including prompts, responses, and any code or file contents that were discussed.

When to Use: Use this command when you need to save a record of your work session for documentation, sharing with teammates, or future reference. It's particularly valuable after completing complex debugging sessions or architectural discussions that you may want to revisit. The export function helps maintain knowledge continuity across your team and creates a searchable archive of problem-solving approaches.

/help

Description: Displays a comprehensive list of all available slash commands along with brief descriptions of what each command does. This includes both built-in commands and any custom commands you've created or that are available from MCP servers.

When to Use: Use this whenever you need a quick reminder of available commands or when you're looking for a specific functionality but can't remember the exact command. It's especially helpful when starting with Claude Code or after updates that may have added new commands.

/hooks *(See: Hooks guide)*

Description: Opens an interactive menu for configuring hooks - automated shell commands that execute at specific points in Claude Code's lifecycle. You can set up PreToolUse hooks (before Claude executes any tool) and PostToolUse hooks (after successful tool completion).

When to Use: Use this to automate repetitive tasks like running linters after file edits, formatting code before commits, or running tests after changes. It's perfect for enforcing code quality standards and automating your workflow.

/ide *(See: IDE integrations)*

Description: Establishes a connection between Claude Code and your Integrated Development Environment (IDE). This allows Claude to see open files, read linter warnings, and understand your current workspace context.

When to Use: Use this command when you want Claude to have visibility into your IDE state, such as when asking it to "fix the linter issues" or when you want it to understand which files you're currently working on.

/init

Description: Generates a CLAUDE.md file for your project, which serves as a comprehensive project guide. This file contains information about your project's architecture, coding conventions, dependencies, and other context that helps Claude understand your codebase better.

When to Use: Use this at the beginning of a new project or when onboarding Claude to an existing codebase. The CLAUDE.md file acts as Claude's "memory" for your project, improving the quality and relevance of its assistance.

/install-github-app *(See: GitHub Actions)*

Description: Sets up the Claude Code GitHub App for automated pull request reviews. Once

configured, Claude will automatically review your PRs, checking for bugs, security issues, and code quality problems.

When to Use: Use this when you want to automate code reviews in your GitHub workflow. It's particularly valuable when working with AI-generated code that requires consistent quality checks before merging.

/login

Description: Allows you to switch between different Anthropic accounts or re-authenticate your current account. This manages your authentication credentials for Claude Code.

When to Use: Use this when you need to switch between work and personal accounts, when your authentication expires, or when troubleshooting account-related issues.

/mcp

Description: Opens the Model Context Protocol (MCP) configuration interface. This allows you to add, remove, or configure MCP servers that extend Claude's capabilities with external tools and integrations like databases, APIs, or browser automation.

When to Use: Use this when you need to connect Claude to external services or tools beyond its built-in capabilities. It's essential for integrating with services like GitHub, Jira, databases, or browser automation tools.

/memory

Description: Opens an interface to edit project memory files, allowing you to modify the persistent context that Claude maintains about your project across sessions.

When to Use: Use this when you need to update or correct information that Claude should remember about your project, such as architectural decisions, team conventions, or important context that should persist between sessions.

/migrate-installer *(Undocumented)*

Description: Migrates your Claude Code installation from a global npm installation to a local installation. This helps resolve permission issues and ensures better isolation between different projects or Claude Code versions.

When to Use: Use this command when you're experiencing permission problems with a global installation or when you want to maintain different Claude Code versions for different projects. It's particularly useful when transitioning from an older installation method to the newer, more flexible local installation approach that provides better control over updates and dependencies.

/model

Description: Allows you to switch between different Claude models during your session. You can choose between models like Claude Opus (for complex reasoning) or Claude Sonnet (for faster responses).

When to Use: Use this to optimize for either depth or speed depending on your current task. Switch to Opus for complex architectural decisions or difficult debugging, and use Sonnet for routine coding tasks or when you need faster iterations.

/output-style *(See: Output styles)*

Description: Sets a custom output style for how Claude formats and presents its responses. This allows you to configure Claude's response formatting to match your preferences or project requirements, such as adjusting verbosity levels, code formatting preferences, or response structure.

When to Use: Use this command when you want to customize how Claude communicates with you throughout your session. It's particularly useful when you need responses in a specific format for documentation purposes or when you prefer a certain level of detail in explanations. Teams can standardize output styles to ensure consistent communication patterns across all developers using Claude Code.

/output-style:new *(See: Output styles)*

Description: Creates a new custom output style template that can be saved and reused across sessions. This allows you to define and persist specific formatting preferences, creating reusable templates for different types of work or projects.

When to Use: Use this command when you want to create a reusable output style that you can apply in future sessions. It's ideal for establishing different communication styles for different contexts, such as a verbose style for learning new concepts, a concise style for quick iterations, or a formal style for documentation generation. Once created, these styles can be shared with team members to ensure consistent Claude interactions across your organization.

/permissions

Description: Reviews and manages the current permission settings for Claude Code, showing which tools and commands Claude is allowed to execute without asking for permission.

When to Use: Use this to audit and adjust Claude's permissions, especially when working on sensitive projects or when you want to either restrict or expand what Claude can do automatically.

/release-notes *(Undocumented)*

Description: Displays the release notes for Claude Code, showing recent updates, new features, bug fixes, and important changes. This provides a comprehensive changelog of what's been added, modified, or fixed in recent versions of Claude Code.

When to Use: Use this command after updating Claude Code to understand what new capabilities are available or what changes might affect your workflow. It's particularly important to check release notes when you notice different behavior after an update or when you want to explore newly added features. Regular review of release notes helps you stay current with Claude Code's evolving capabilities and ensures you're leveraging the latest improvements in your development workflow.

/resume

Description: Resumes a specific previous session by its ID, restoring all the context and conversation history from that session. You can also use 'claude -c' from the command line to continue your last session.

When to Use: Use this when you need to return to work from a previous session, especially when working on multiple projects or features in parallel. It's invaluable for maintaining context across work sessions.

/security-review *(Undocumented)*

Description: Initiates a comprehensive security review of all pending changes in your current working directory. Claude analyzes your code modifications for potential security vulnerabilities, insecure patterns, exposed credentials, and other security concerns before you commit or deploy the changes.

When to Use: Use this command as a critical checkpoint before committing sensitive code changes, especially when working with authentication systems, data handling, API integrations, or user input processing. It's particularly important to run security reviews when modifying code that handles sensitive data, implements access controls, or interfaces with external services. This command serves as an automated security audit that can catch vulnerabilities early in the development cycle, significantly reducing the risk of security issues making it to production. Teams should consider making this a standard part of their pre-commit workflow for any security-critical codebases.

/status

Description: Displays comprehensive information about your account status, system information, model availability, and current session details. Shows usage limits, current model, and other relevant system information.

When to Use: Use this to check your account status, verify which model you're using, check rate limits, or get general information about your Claude Code installation and configuration.

/upgrade *(Undocumented)*

Description: Opens the interface to upgrade your Claude Code subscription plan. This command provides information about available plans, features included in each tier, and allows you to modify your subscription level to access additional capabilities or higher usage limits.

When to Use: Use this command when you've reached the limits of your current plan and need additional capacity, when you want to access premium features available in higher tiers, or when you need to compare different plan options to find the best fit for your development needs. It's particularly relevant when your usage patterns have evolved and you need more tokens, faster response times, or access to more powerful models. Teams scaling their AI-assisted development should periodically review upgrade options to ensure they have adequate resources for their growing needs.

Custom Slash Commands

Project Commands

Location: `.claude/commands/`
Description: Custom commands specific to your current project that are shared with your team through version control. These appear with "(project)" suffix in the help menu.

When to Use: Create project-specific commands for repetitive workflows like running test suites, deploying applications, or performing code reviews according to your team's standards.

Personal Commands

Location: `~/.claude/commands/`
Description: Personal commands available across all your projects. These appear with "(user)" suffix in the help menu and are not shared with others.

When to Use: Create personal commands for workflows you use across multiple projects, such as personal coding standards, preferred refactoring patterns, or custom analysis tools.

MCP Server Commands

Format: `/mcp__[server]__[command]`
Description: Commands dynamically provided by connected MCP servers. These extend Claude's capabilities with external tools and services.

When to Use: Use these to interact with external services you've connected via MCP, such as GitHub for PR management, Jira for issue tracking, or databases for queries.

Special Command Features

Command Arguments

Commands can accept arguments using the `$ARGUMENTS` placeholder in custom commands, allowing you to pass dynamic values: `/my-command "specific value"`

Positional Arguments

Custom commands support positional arguments using `$1`, `$2`, etc., enabling more complex command structures: `/review-pr 456 high john`

File References (@)

Include file contents in commands using the @ prefix: `/analyze @src/main.js`

Bash Execution (!)

Execute bash commands before slash commands run using the ! prefix, with output included in context: `!git status`

Best Practices

1. **Use /clear frequently** - Reset context between distinct tasks to maintain focus
2. **Monitor with /cost** - Keep track of token usage during long sessions
3. **Create custom commands** - Automate repetitive workflows specific to your project
4. **Configure hooks** - Automate quality checks and formatting
5. **Initialize projects with /init** - Give Claude comprehensive project context
6. **Use /compact strategically** - Preserve critical context when approaching limits
7. **Switch models appropriately** - Use /model to optimize for speed vs. complexity

This reference guide covers all known slash commands available in Claude Code, including:

- Commands documented in the official Slash Commands documentation
- Commands documented in other Claude Code documentation sections
- Commands discovered by the community but not yet documented

As Claude Code continues to evolve rapidly, new commands may be added and currently undocumented commands may receive official documentation. Always check /help for the most current list of available commands in your installation.