GitHub Copilot Slash Commands

Slash commands in GitHub Copilot Chat provide quick access to specialized AI capabilities. These commands help you perform specific development tasks efficiently within the conversational interface.

How to Use Slash Commands

Type a forward slash (`/`) in the chat interface, followed by the command name. Most commands work better when combined with context variables (`#file` , `#selection` , etc.) or specific instructions.

Core Slash Commands

`/explain`

Provides detailed explanations of code, concepts, or implementations.

Usage:

```
/explain this algorithm's time complexity
/explain #selection why this pattern is used
/explain the difference between async/await and Promises
```

Best with: - `#selection` for specific code analysis - `#file` for architectural explanations - Complex algorithms or design patterns

`/fix`

Identifies and resolves bugs, errors, or code issues.

Usage:

```
/fix #selection this function has a memory leak
/fix the TypeScript errors in this file
/fix #terminalLastCommand compilation error
```

Best with: - `#selection` for targeted fixes - `#terminalLastCommand` for error context - Specific error descriptions

`/optimize`

Suggests performance improvements and code optimizations.

Usage:

```
/optimize this database query for better performance
/optimize #selection for reduced memory usage
/optimize bundle size for this React component
```

Best with: - `#selection` for targeted optimization - Performance metrics or requirements - Specific optimization goals (speed, memory, size)

`/tests`

Generates comprehensive test cases and testing strategies.

Usage:

```
/tests create unit tests for this service class
/tests #selection generate edge case tests
/tests integration tests for this API endpoint
```

Best with: - `#selection` for specific functions - `#file` for class or module testing - Testing framework preferences (Jest, Mocha, pytest, etc.)

`/doc`

Creates documentation, comments, and technical specifications.

Usage:

```
/doc generate API documentation for this endpoint
/doc #selection add JSDoc comments
/doc create README for this project structure
```

Best with: - `#file` for comprehensive documentation - `#selection` for inline comments - Documentation format specifications (JSDoc, Sphinx, etc.)

Advanced Slash Commands

`/review`

Conducts thorough code reviews with suggestions for improvement.

Usage:

```
/review #selection for security vulnerabilities
/review this pull request for best practices
/review #codebase architecture and design patterns
```

Best with: - `#selection` or `#file` for focused reviews - `#codebase` for architectural analysis - Specific review criteria (security, performance, maintainability)

`/refactor`

Suggests and implements code refactoring improvements.

Usage:

```
/refactor #selection extract into smaller functions
/refactor this class to use composition over inheritance
/refactor for better testability and separation of concerns
```

Best with: - `#selection` for targeted refactoring - Specific refactoring goals - Design pattern preferences

`/scaffold`

Creates boilerplate code and project structures.

Usage:

```
/scaffold a React component with TypeScript and tests
/scaffold REST API endpoints for user management
/scaffold CI/CD pipeline for Node.js deployment
```

Best with: - Project requirements and specifications - Framework and technology preferences - Architectural patterns

Specialized Commands

`/new`

Creates new files, components, or project structures.

Usage:

```
/new React component with hooks and TypeScript
/new Express.js route with validation middleware
/new Dockerfile for Node.js application
```

`/newNotebook`

Creates Jupyter notebooks with specific configurations.

Usage:

```
/newNotebook for data analysis with pandas
/newNotebook machine learning experiment setup
/newNotebook API testing and documentation
```

`/terminal`

Provides command-line assistance and shell scripting help.

Usage:

```
/terminal set up automated deployment script
/terminal Git workflow for feature branching
/terminal Docker commands for development environment
```

`/commit`

Generates meaningful commit messages based on changes.

Usage:

```
/commit generate message for these changes
/commit follow conventional commits format
/commit with detailed description of refactoring
```

Context-Aware Command Usage

Combining Commands with Context

```
# Effective combinations:
/explain #selection + specific questions
/fix #terminalLastCommand + error description
/tests #file + testing framework preference
/doc #codebase + documentation standard
/review #selection + review criteria
```

Multi-Step Workflows

```
1. /explain #selection understand the current implementation
2. /refactor extract common functionality
3. /tests generate tests for refactored code
4. /doc update documentation for changes
```

Language-Specific Command Usage

JavaScript/TypeScript

```
/tests generate Jest tests with mocks and spies
/optimize #selection reduce bundle size impact
/refactor use modern ES6+ features and patterns
```

Python

```
/tests create pytest with fixtures and parametrization
/doc generate Sphinx documentation with type hints
/optimize profile and improve algorithmic complexity
```

Java

```
/tests generate JUnit 5 tests with proper setup
/refactor apply SOLID principles and design patterns
/review check for proper exception handling
```

Go

```
/tests create table-driven tests with proper error handling
/optimize concurrent processing with goroutines
/doc generate godoc-compatible documentation
```

Command Modifiers and Options

Specifying Frameworks and Tools

```
/tests using Jest and React Testing Library
/scaffold with Spring Boot and JPA annotations
/optimize for webpack bundle splitting
/doc in OpenAPI 3.0 specification format
```

Setting Constraints and Requirements

```
/refactor maintaining backward compatibility
/fix without changing public API
/optimize for mobile performance
/tests with 100% code coverage goal
```

Best Practices

1. Be Specific

```
❌ /fix this code
✅ /fix #selection memory leak in event listeners
```

2. Provide Context

```
❌ /tests for this
✅ /tests #selection for React component with props validation
```

3. Specify Standards

```
❌ /doc add documentation
✅ /doc #file generate JSDoc with TypeScript types
```

4. Chain Related Commands

```
/explain #selection current implementation approach
/refactor for better separation of concerns
/tests create comprehensive test suite
```

Troubleshooting Commands

Issue	Command	Example
Build errors	`/fix`	`/fix #terminalLastCommand compilation errors`
Performance issues	`/optimize`	`/optimize #selection database query performance`
Code smells	`/review`	`/review #file for maintainability issues`
Missing tests	`/tests`	`/tests #codebase identify untested functions`
Poor documentation	`/doc`	`/doc #file comprehensive API documentation`

Enterprise and Team Commands

Custom Slash Commands

Organizations can create custom slash commands for internal workflows:

```
/deploy generate deployment scripts for our infrastructure
/security review code against company security policies
/standards apply company coding standards and conventions
```

Future Command Enhancements

Expected additions in late 2025: - `/analyze` for deep code analysis and metrics - `/migrate` for framework and language migrations
- `/performance` for detailed performance profiling - `/accessibility` for WCAG compliance checking - `/security` for comprehensive security analysis

Command Reference Quick Card

Command	Primary Use	Best Context
`/explain`	Code understanding	`#selection`
`/fix`	Bug resolution	`#terminalLastCommand`
`/optimize`	Performance improvement	`#selection` + metrics
`/tests`	Test generation	`#file` or `#selection`
`/doc`	Documentation	`#file` or `#codebase`
`/review`	Code analysis	`#selection` or `#file`
`/refactor`	Code improvement	`#selection` + goals
`/scaffold`	Boilerplate creation	Requirements + framework

Remember: Slash commands are most effective when combined with specific context and clear requirements!