Lab: Building Custom Commands

Module: T3 - Claude Code Advanced
Duration: 25 minutes
Difficulty: Intermediate

Overview

In this lab, you will create custom slash commands that extend Claude Code's capabilities. Custom commands are reusable prompt templates that can accept arguments, making repetitive tasks fast and consistent.

Objectives

After completing this lab, you will be able to:

- Create custom commands in `.claude/commands/`
- Use the `$ARGUMENTS` placeholder for dynamic input
- Build commands for documentation, refactoring, and testing
- Share commands with your team via version control

Prerequisites

- Claude Code installed and authenticated (completed in Lab T1)
- Familiarity with Markdown files
- Sample project from Lab T1

Scenario

Your development team frequently needs to: - Generate API documentation for endpoints - Write tests for specific functions - Explain code to new team members

You'll create custom commands to handle each of these tasks consistently.

Exercise 1: Create an API Documentation Command

Duration: 8 minutes

Task 1: Set Up the Commands Directory

1. Navigate to your sample project:

`bash cd labs/sample-project`

1. Create the commands directory:

`bash mkdir -p .claude/commands`

1. Verify the directory exists:

`bash ls -la .claude/`

Task 2: Create the API Documentation Command

1. Create a new file `.claude/commands/api-doc.md`:

`bash code .claude/commands/api-doc.md`

1. Add the following content:

```markdown

---

description: Generate API documentation for an endpoint

---

Analyze the endpoint or route file at $ARGUMENTS and generate comprehensive API documentation.

Include the following sections:

## Endpoint Summary One-line description of what this endpoint does.

## HTTP Details - Method: GET/POST/PUT/DELETE - Path: /api/... - Authentication: Required/Optional/None

## Request Format If POST/PUT, show the expected JSON body with field descriptions.
```

Query Parameters Table of query parameters (name, type, required, description).

Response Format ### Success (200/201) JSON structure with field explanations.

Error Responses Common error codes and when they occur.

Example Requests Provide curl command examples.

Format as clean Markdown suitable for a README or API docs site. ```

1. Save the file.

Task 3: Test Your Command

1. Start Claude Code:

`bash claude`

1. List available commands to verify yours appears:

```

> /help ```

**Expected output:** You should see `/api-doc` listed with your description.

1. Run your command:

```

> /api-doc src/routes/users.js ```

1. Review the generated documentation:
2. Does it include all sections?
3. Are the curl examples correct?
4. Is the format clean?

Validation Checkpoint

- [ ] Command file exists at `.claude/commands/api-doc.md`
- [ ] Command appears in `/help` output
- [ ] Running `/api-doc` generates comprehensive documentation
- [ ] Documentation is properly formatted Markdown

Exercise 2: Create a Test Generator Command

Duration: 8 minutes

Task 1: Create the Test Command

1. Create `.claude/commands/test-this.md`:

```markdown
---

description: Generate Jest tests for a function or file

---

Generate comprehensive Jest tests for $ARGUMENTS.

## Testing Guidelines

1. **Happy Path Tests**
    - Test normal/expected inputs
    - Verify correct output format
2. **Edge Cases**
    - Empty inputs
    - Boundary values
    - Null/undefined handling
3. **Error Cases**
    - Invalid input types
    - Missing required parameters
    - Expected error messages

## Output Format

Generate a complete test file with: - Proper imports - Descriptive `describe` blocks - Clear `it` statements using "should..." format - Meaningful assertions

Follow the existing test patterns in the project if any exist.

Show the complete test file content. ```
```

Task 2: Test the Command

1. In Claude Code, run:

```

> /test-this src/models/user.js ```

1. Review the generated tests:
2. Do they cover the main functionality?
3. Are edge cases included?
4. Do they follow Jest conventions?
5. Optionally save and run the tests:

```

> Save these tests to tests/user.model.test.js and run them ```

Validation Checkpoint

- [ ] Test command generates comprehensive tests
- [ ] Tests include happy path, edge cases, and error cases
- [ ] Generated tests are syntactically correct

Exercise 3: Create an Explanation Command

Duration: 5 minutes

Task 1: Create the Explain Command

1. Create `.claude/commands/explain-like-5.md`:

```markdown

---

description: Explain code for a junior developer

---

Explain $ARGUMENTS as if teaching a junior developer on their first day.
```

Guidelines

- Use simple analogies (restaurants, libraries, post offices)
- Avoid jargon - define any technical terms you must use
- Break complex operations into numbered steps
- Explain WHY, not just WHAT
- Be encouraging - "this is a common pattern" not "obviously..."

Format

Start with a one-sentence summary, then explain in detail. End with "□Key Takeaway:" summarizing the main concept. ```

Task 2: Test the Command

1. Run the command:

```

> /explain-like-5 src/index.js ```

1. Evaluate the explanation:
2. Is it accessible to a beginner?
3. Are analogies helpful and accurate?
4. Does it avoid unnecessary jargon?

---

# Exercise 4: Create a Command with Multiple Arguments (Advanced)

**Duration:** 4 minutes

### Task 1: Create a Refactoring Command

1. Create `.claude/commands/refactor.md`:

```markdown

---

description: Suggest refactoring improvements for code

---

Analyze $ARGUMENTS and suggest refactoring improvements.

## Analysis Criteria

### Code Smells - Long functions (>20 lines) - Deep nesting (>3 levels) - Duplicate code - Magic numbers/strings

### Improvements - Extract method opportunities - Better naming suggestions - Error handling improvements - Performance optimizations

### Modern Patterns - ES6+ syntax improvements - Async/await opportunities - Destructuring possibilities

## Output

For each suggestion: 1. What: The specific issue 2. Why: Impact on maintainability/ performance 3. How: Concrete code example of the fix

Prioritize suggestions by impact (high/medium/low). ```

### Task 2: Test Across Different Files

1. Try on different parts of the codebase:

```

> /refactor src/routes/users.js /refactor src/index.js ```

1. Compare the suggestions - are they contextually appropriate?

Challenge: Team Command Library

Duration: Bonus (outside lab time)

Create a shared commands library for your team:

1. **Code Review Command:** `/review` that checks for common issues
2. **Changelog Command:** `/changelog` that summarizes recent commits
3. **Debug Command:** `/debug` that helps troubleshoot errors
4. **Security Check:** `/security` that looks for vulnerabilities

Share your commands directory with your team via git:
```

```
 git add .claude/commands/
git commit -m "feat: add custom Claude commands for team use"
git push
```

Troubleshooting Guide

Issue	Solution
Command doesn't appear in `/help`	Ensure file is in `.claude/commands/` with `.md` extension
`$ARGUMENTS` not substituted	Check for typos - it's `$ARGUMENTS` (uppercase, no space)
Command runs but output is poor	Refine your prompt in the .md file and restart Claude
Permission denied on file creation	Check directory permissions
Command works differently after restart	Clear cache with `/clear` after modifying commands

---

Summary

In this lab, you learned to:

1. **Create custom commands** in `.claude/commands/`
2. **Use** `$ARGUMENTS` for dynamic input
3. **Build practical commands** for documentation, testing, and explanations
4. **Share commands** with your team via version control

Key Takeaways

- Custom commands are **Markdown files with YAML frontmatter**
- `$ARGUMENTS` is replaced with whatever follows the command

- Commands should have **clear, focused purposes**
- **Version control your commands** to share with your team

Next Steps

- Create commands for your own repetitive tasks
- Explore adding `model:` to frontmatter for specific model selection
- Build a team command library

Additional Resources

- Claude Code Documentation: docs.anthropic.com/claude-code/commands
- Community Commands: github.com/hesreallyhim/awesome-claude-code