

# Lab: Getting Started with Claude Code

---

**Module:** T1 - Claude Enterprise & Claude Code

**Duration:** 28 minutes

**Difficulty:** Beginner

---

## Overview

In this lab, you will learn how to use Claude Code for AI-powered development workflows. You will explore a codebase using natural language, make code changes with AI assistance, and configure project-specific settings.

## Objectives

After completing this lab, you will be able to:

- Install and authenticate Claude Code CLI
- Explore and understand a codebase using natural language queries
- Request and review AI-generated code changes
- Create a CLAUDE.md configuration file for project-specific behavior

## Prerequisites

- Node.js 18 or higher installed
- Claude Enterprise account with active license
- Git installed and configured
- Terminal/command line access
- Visual Studio Code or preferred code editor

## Scenario

You are a developer who has just joined a team working on a Node.js REST API. You need to quickly understand the codebase, add a new feature, and set up project-specific AI configurations to help your team work more efficiently.

---

# Exercise 1: Codebase Exploration

**Duration:** 8 minutes

In this exercise, you will use Claude Code to explore and understand the sample project codebase.

## Task 1: Install Claude Code

1. Open your terminal or command prompt.
2. Install Claude Code globally using npm:

```
```bash
```

```
npm install -g @anthropic-ai/claude-code
```

```
```
```

3. Verify the installation was successful:

```
```bash
```

```
claude --version
```

```
```
```

**Expected output:** You should see a version number (e.g., 1.x.x)

*\*\*Troubleshooting:\*\* If you see "command not found", ensure that npm's global bin directory is in your system PATH.*

## Task 2: Authenticate with Claude Enterprise

4. Run the authentication command:

```
```bash
```

```
claude login
```

```
```
```

5. A browser window will open. Sign in with your Claude Enterprise credentials.
6. After successful authentication, return to your terminal.

**Validation:** You should see a confirmation message: "Successfully authenticated"

## Task 3: Navigate to the Sample Project

7. Clone the sample repository (or navigate to the provided project folder):

```
```bash
```

```
git clone [sample-repo-url]
```

```
cd sample-project
```

```
````
```

*\*\*Note:\*\* If using the provided lab files, navigate to: 'labs/sample-project'*

8. Verify you are in the project root:

```
```bash
```

```
ls
```

```
````
```

**Expected output:** You should see files including package.json, src/, and tests/

## Task 4: Start Claude Code and Explore

9. Launch Claude Code:

```
```bash
```

```
claude
```

```
````
```

10. Ask Claude to provide an overview of the project:

```
````
```

*Give me an overview of this codebase*

```
````
```

**Expected behavior:** Claude will analyze the project structure and provide a summary of the main components, technologies used, and overall architecture.

11. Ask about the API endpoints:

```
````
```

*What are the main API endpoints in this project?*

```
````
```

**Expected behavior:** Claude will list the available endpoints (e.g., /api/users, /api/health) with their HTTP methods.

12. Ask about authentication:

```

*How is authentication handled in this codebase?*

```

**Expected behavior:** Claude will explain the authentication approach used in the project.

### Validation Checkpoint

- [ ] Claude Code is installed and accessible from the command line
  - [ ] You have successfully authenticated with your Claude Enterprise account
  - [ ] Claude can read and describe the project structure
  - [ ] You received meaningful answers about API endpoints and architecture
- 

## Exercise 2: Making Code Changes

**Duration:** 10 minutes

In this exercise, you will use Claude Code to add input validation to an existing API endpoint.

### Task 1: Request Code Changes

13. Ensure Claude Code is running in the sample project directory.
14. Ask Claude to add input validation to the user registration endpoint:

```

*Add input validation to the user registration endpoint in the users route. Validate that email is a valid format and username is at least 3 characters.*

```

15. \*\*Review the proposed changes carefully.\*\* Claude will display a diff showing:

- Which files will be modified
- Exactly what code will be added or changed
- Line numbers for each change

**[SCREENSHOT: Review the diff display in your terminal]**

### Task 2: Approve or Request Modifications

16. If the proposed changes look correct, type `y` or `yes` to approve.
17. If you want modifications, provide feedback:

```

*Can you also add validation for password minimum length of 8 characters?*

```

18. Review the updated diff and approve when satisfied.

### Task 3: Verify the Changes

19. After approving, verify the changes were made:

```

*Show me the updated users route file*

```

20. Run the tests to ensure your changes work correctly:

```bash

npm test

```

**Expected output:** All tests should pass, including any new validation tests.

21. If tests fail, ask Claude for help:

```

*The tests are failing. Can you help fix the issues?*

```

### Validation Checkpoint

- [ ] You successfully requested code changes using natural language
- [ ] You reviewed and approved (or modified) the proposed diff
- [ ] The changes were applied to the codebase
- [ ] Tests pass with the new validation logic

---

## Exercise 3: Create CLAUDE.md Configuration

**Duration:** 10 minutes

In this exercise, you will create a CLAUDE.md configuration file to customize Claude's behavior for this project.

## Task 1: Create the Configuration File

22. In the project root, create a new file called `CLAUDE.md`:

```

*Create a CLAUDE.md file in the project root*

```

Or manually create the file in your code editor.

23. Add the following content to define the project context:

```markdown

# Project Context

## Tech Stack

- Node.js + Express backend
- PostgreSQL database (using pg library)
- Jest for testing

## Coding Conventions

- Use async/await for all asynchronous operations (not callbacks)
- All API endpoints must return JSON responses
- Use meaningful HTTP status codes (200, 201, 400, 401, 404, 500)
- Include error handling on all route handlers
- Use camelCase for variable and function names

## Project Structure

- `src/index.js` - Express server setup
- `src/routes/` - API route handlers
- `src/models/` - Data models
- `tests/` - Jest test files

## DO

- Write descriptive error messages
- Add JSDoc comments to exported functions
- Use environment variables for configuration

## DON'T

- Don't use `var`, use `const` or `let`
- Don't commit console.log statements

- Don't hardcode credentials or API keys

```

24. Save the file.

## Task 2: Test the Configuration

25. Exit the current Claude session:

```

/quit

```

Or press `Ctrl+C`

26. Start a new Claude session:

```bash

claude

```

27. Ask Claude to make a change and observe how it follows your conventions:

```

*Add a new endpoint to get a user by ID. Follow the project conventions.*

```

28. Review the proposed code. \*\*Notice how Claude:\*\*

- Uses `async/await`
- Returns JSON responses
- Includes error handling
- Uses appropriate HTTP status codes
- Follows naming conventions

## Task 3: Customize Further (Optional)

29. Add team-specific instructions to CLAUDE.md. For example:

```markdown

## Team Practices

- All PRs require at least one review

- Use conventional commit messages (feat:, fix:, docs:, etc.)
- Reference Jira tickets in commits when applicable

```

30. Start a new session and test the commit skill:

```

*/commit*

```

**Expected behavior:** Claude should generate a commit message following conventional commit format.

### Validation Checkpoint

- [ ] CLAUDE.md file is created in the project root
  - [ ] The file contains tech stack, conventions, and DO/DON'T rules
  - [ ] Starting a new Claude session respects the CLAUDE.md settings
  - [ ] Generated code follows the conventions you specified
- 

## Troubleshooting Guide

Issue	Solution
"command not found: claude"	Ensure npm's global bin directory is in your PATH. Run `npm config get prefix` to find the path.
Authentication fails	Verify your Claude Enterprise license is active. Contact your admin if needed.
"Permission denied" errors	On Windows, run terminal as Administrator. On Mac/Linux, check file permissions.
Claude doesn't see files	Make sure you're in the project root directory when starting Claude.
Changes weren't applied	Ensure you typed 'y' or 'yes' to approve. Check for error messages in the output.
Tests fail after changes	Ask Claude to review the test failures and suggest fixes.
CLAUDE.md not taking effect	Exit Claude completely (`/quit`) and start a fresh session.

---

## Summary

In this lab, you have learned how to:

31. \*\*Install and authenticate\*\* Claude Code for enterprise development workflows
32. \*\*Explore codebases\*\* using natural language queries to quickly understand project structure
33. \*\*Request and review code changes\*\* with AI assistance, maintaining full control over what gets modified
34. \*\*Configure project settings\*\* with CLAUDE.md to customize Claude's behavior for your team

## Next Steps

- Experiment with other slash commands: `/help`, `/compact`, `/review`
- Share your CLAUDE.md with team members to standardize AI assistance
- Proceed to Lab T2 to learn about MCP connectors and advanced integrations

## Additional Resources

- Claude Code Documentation: [docs.anthropic.com/clause-code](https://docs.anthropic.com/clause-code)
- CLAUDE.md Guide: [docs.anthropic.com/clause-code/clause-md](https://docs.anthropic.com/clause-code/clause-md)
- Claude Enterprise Admin: [admin.claude.ai](https://admin.claude.ai)