# Claude Code Training Course

## Claude Code Course Overview

This Claude Code course from AGI Training is for engineers who want to move from ad hoc AI assistance to using an agentic coding tool. Find out how to use Claude Code in your terminal: reading your project files, answering questions about a codebase in plain language, and proposing changes you can review. The course helps you use Claude Code confidently, exploring unfamiliar code, delegating multi-file changes, driving the git workflow, writing and running tests, and connecting to the systems where your work already lives.

Discover how Claude Code changes how you work as it modifies files and runs commands. This course is hands-on throughout, building fluency and judgment that make AI-assisted development safe and productive.

No prior experience with Claude Code or other AI coding assistants is needed.

### *Who Should Attend the Claude Code Course*

- Software, firmware, and systems engineers who write or maintain code
- Engineers who have inherited an unfamiliar or legacy codebase and need to understand it quickly
- Data and platform engineers who build analysis scripts, pipelines, and integrations
- Technical program managers and leads evaluating team-level adoption of agentic development tools
- Anyone who wants to move from an autocomplete assistant to an agentic tool.

### *Prerequisites for the Claude Code Course*

- Comfort working in a command-line environment and with Git-based workflows
- Familiarity with at least one programming language
- No prior experience with Claude Code or other AI coding assistants is required

### *What You'll Be Able to Do by the End of the Claude Code Course*

- Install Claude Code and run a productive session independently
- Choose the right tool for a task and give instructions suited to an agentic system
- Explore and explain a codebase you didn't write, using natural language
- Read a multi-file diff critically and distinguish a good change from a plausible-looking bad one
- Delegate features, bug fixes, and refactors at an appropriate scope, and know when to redirect or start over
- Encode team conventions in a CLAUDE.md file and drive the full git workflow through natural language
- Use Claude Code to write and run tests, and judge when autonomous iteration is and isn't appropriate
- Connect Claude Code to external systems via MCP and evaluate those connections for security and value

## Claude Code Training Course Outline

### Foundations: Agentic Tools and Your First Session

Every later module assumes a shared understanding of what makes Claude Code different from an autocomplete assistant and how to work with it responsibly. This module establishes that baseline, covering the spectrum of AI coding tools and what "agentic" actually means for how you give instructions and review output. Participants then install Claude Code and complete a guided first session against a real codebase.

*Find dates and details at [https://www.agitraining.com/ai-classes/claude-training/claude-code-training-course](https://www.agitraining.com/ai-classes/claude-training/claude-code-training-course)*

- Understanding the spectrum of AI coding tools: from autocomplete to conversational to agentic, and where Claude Code sits
- Learning what "agentic" means, and why it changes how you give instructions and review results
- Seeing how an agentic tool's awareness of your whole project differs from file-level autocomplete
- Installing Claude Code and setting up the local environment
- Navigating to a project directory and launching a session
- Using Claude Code to map and explain an unfamiliar codebase in plain language
- Understanding what Claude Code can and cannot see in your environment
- Reviewing proposed changes: reading a diff, understanding scope, accepting or rejecting
- Configuring permission levels: when it asks, when it acts, and when to intervene
- Adopting the verify-don't-accept mindset that anchors the rest of the course

### Precision Prompting and Instruction Design

The single largest source of poor AI output is poor input. Before delegating substantial work, participants build a disciplined approach to communicating intent clearly enough that Claude Code can act on it well. This module is deliberately hands-on: participants rewrite weak instructions into strong ones using before-and-after comparisons, and practice the structure that makes complex, multi-part requests succeed.

- Understanding the anatomy of an effective instruction: role, context, task, constraints, and desired format
- Guiding output with positive examples versus negative examples
- Using structure and delimiters to organize complex inputs so the tool stays focused
- Applying chain-of-thought prompting for diagnostic and analytical tasks
- Iterative prompting: treating the first output as a conversation opener, not a final answer
- Scoping instructions for engineering-specific outputs like test plans, specs, and documentation
- Diagnosing why an instruction didn't work and revising it systematically
- Writing instructions that produce specific, usable output on the first attempt

### Multi-File Work, Refactoring, and Legacy Code

This is the core of delegation. Participants learn to instruct Claude Code to implement features, fix bugs, and execute refactors that span multiple files — and, crucially, to validate what it produces before it touches a repository. The module also covers one of Claude Code's highest-leverage uses: understanding, documenting, and incrementally modernizing legacy systems without breaking what works.

- Writing instructions scoped clearly enough for multi-file changes
- Understanding how Claude Code uses agentic search to build project context
- Delegating a bug fix across dependent files: instruct, review, accept
- Breaking a small feature into sub-steps that can each be delegated
- Reviewing diffs critically: what to look for and what to question
- Handling Claude Code making a wrong assumption mid-task
- Knowing when to break a large task into smaller, sequenced instructions
- Generating documentation for undocumented functions and modules
- Building a dependency map of a legacy system through natural-language queries

- Incremental refactoring: modernizing without a full rewrite
- Writing characterization tests before touching legacy code
- Identifying technical debt systematically and prioritizing it
- Telling the difference between a good diff and a plausible-looking bad one, and knowing when to approve, redirect, or start over

### Workflow: Git, Pull Requests, and CLAUDE.md

Claude Code becomes far more useful when it fits into your existing team practices rather than working around them, and when it understands your team's specific conventions. This module covers the git workflow — staging changes, writing commit messages, creating branches, and opening pull requests through natural language — and CLAUDE.md, the file Claude Code reads at the start of a session to pick up your standards, architecture decisions, and domain vocabulary.

- Using Claude Code to stage changes and generate meaningful commit messages
- Creating and switching branches through natural language
- Opening a pull request and writing a description that accurately reflects AI-assisted changes
- Reviewing AI-generated commits in a team context and communicating provenance
- Generating changelogs and release notes
- Avoiding common pitfalls: over-committing, mixed-scope commits, and lost context between sessions
- Understanding what CLAUDE.md is and how Claude Code uses it at session start
- Deciding what to put in it: coding standards, architecture decisions, library preferences, review checklists
- Deciding what to keep out of it: sensitive credentials and context that changes frequently
- Structuring a CLAUDE.md for a real project and encoding domain-specific vocabulary and constraints
- Team governance: who owns CLAUDE.md, how it's updated and reviewed, and versioning it alongside code

### Testing, Validation, and Autonomous Iteration

Claude Code can write tests, run them, read the failures, and iterate on fixes with limited manual intervention. That autonomous loop is powerful and, used carelessly, risky. This module covers running the test cycle responsibly: defining success criteria clearly, reviewing what the tool proposes before it runs, and deciding where autonomous iteration is appropriate versus where a human must review each step.

- Directing Claude Code to write unit and integration tests from a function description
- Running test suites through Claude Code and interpreting failures
- Understanding the autonomous iteration loop: when to let it run and when to intervene
- Defining acceptance criteria clearly enough for the tool to validate against
- Reviewing test coverage: what Claude Code tends to miss
- Using tests as a specification tool: writing tests first, then delegating the implementation
- Recognizing when not to let it iterate autonomously: risk thresholds in safety-relevant or high-stakes code
- Evaluating AI-generated test suites for coverage gaps and false confidence

## Extending Claude Code: MCP and Data Workflows

Claude Code doesn't have to live in isolation. Through the Model Context Protocol (MCP), it can connect to external systems — ticketing tools, documentation repositories, and internal data sources — and work across them through natural language. This module introduces MCP and applies it to a concrete, high-value use case: data analysis work, where Claude Code can take over the scaffolding of cleaning, transforming, and visualizing data so you can spend your time on interpretation.

- Understanding what MCP is and why it exists: giving Claude Code a longer reach
- Surveying integration categories: project management, documentation, data sources, and custom tooling
- Connecting Claude Code to an internal data source via MCP
- Delegating data cleaning and transformation scripts
- Describing an analysis goal in plain language and reviewing the resulting Python or R code
- Iterating on visualizations conversationally rather than through trial-and-error coding
- Turning ad hoc scripts into repeatable, documented analysis pipelines
- Reviewing statistical logic in AI-generated analysis code: what to verify manually
- Weighing security considerations: what access you grant, to what, and with what controls
- Evaluating whether an integration creates value or just complexity

## Wrap-Up and Adoption Planning

The day closes with a short working session in which participants identify the two or three highest-impact ways to apply Claude Code in their own work, along with the review and governance guardrails those uses require. Participants leave with a shared vocabulary, a disciplined set of working habits, and a concrete starting point for adoption.