
Claude Code Training
Press Space for next page →

Contact Info

Ken Kousen
Kousen IT, Inc.

- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)
- Social Media:
  - @kenkousen (twitter)
  - @kenkousen@foojay.social (mastodon)
  - @kousenit.com (bluesky)
- *Tales from the jar side* (free newsletter)
  - https://kenkousen.substack.com
  - https://youtube.com/@talesfromthejarside

Course Overview

- **Duration**: 5-hours
- **Topics Covered**:
  - Installation and CLI basics
  - Code exploration and understanding
  - Testing and quality assurance
  - Git operations and version control
  - Advanced features (Plan Mode, MCP, SDKs)
- **Hands-on Labs**: Multiple exercises with real codebases
- **Prerequisites**: Command-line experience, development background

Pricing Plans

- **Pro Plan** - $20/month
  - ~10-40 prompts per 5 hours
  - Sonnet 4 only
- **Max Plan 5x** - $100/month
  - ~50-200 prompts per 5 hours
  - Sonnet or Opus 4
- **Max Plan 20x** - $200/month
  - ~200-800 prompts per 5 hours
  - Sonnet or Opus 4
- **Note**: Opus 4 uses 5x more credits than Sonnet 4
- **Limits reset**: Every 5 hours

📖 **Full details**: Using Claude Code with your Pro or Max plan

What is Claude Code?

- Command-line AI tool for development
- Context-aware codebase understanding
- Autonomous and collaborative modes
- Multi-language support
- Integrated git operations
- Test generation and documentation

Installation

- Install via npm: `npm install -g @anthropic-ai/claude-code`
- Or download from GitHub releases
- Set API key: `export ANTHROPIC_API_KEY="your-key"`
- Verify: `claude --version`

Basic Usage

- Start Claude Code: `claude`
- Natural language prompts
- File-specific requests
- Multi-step workflows

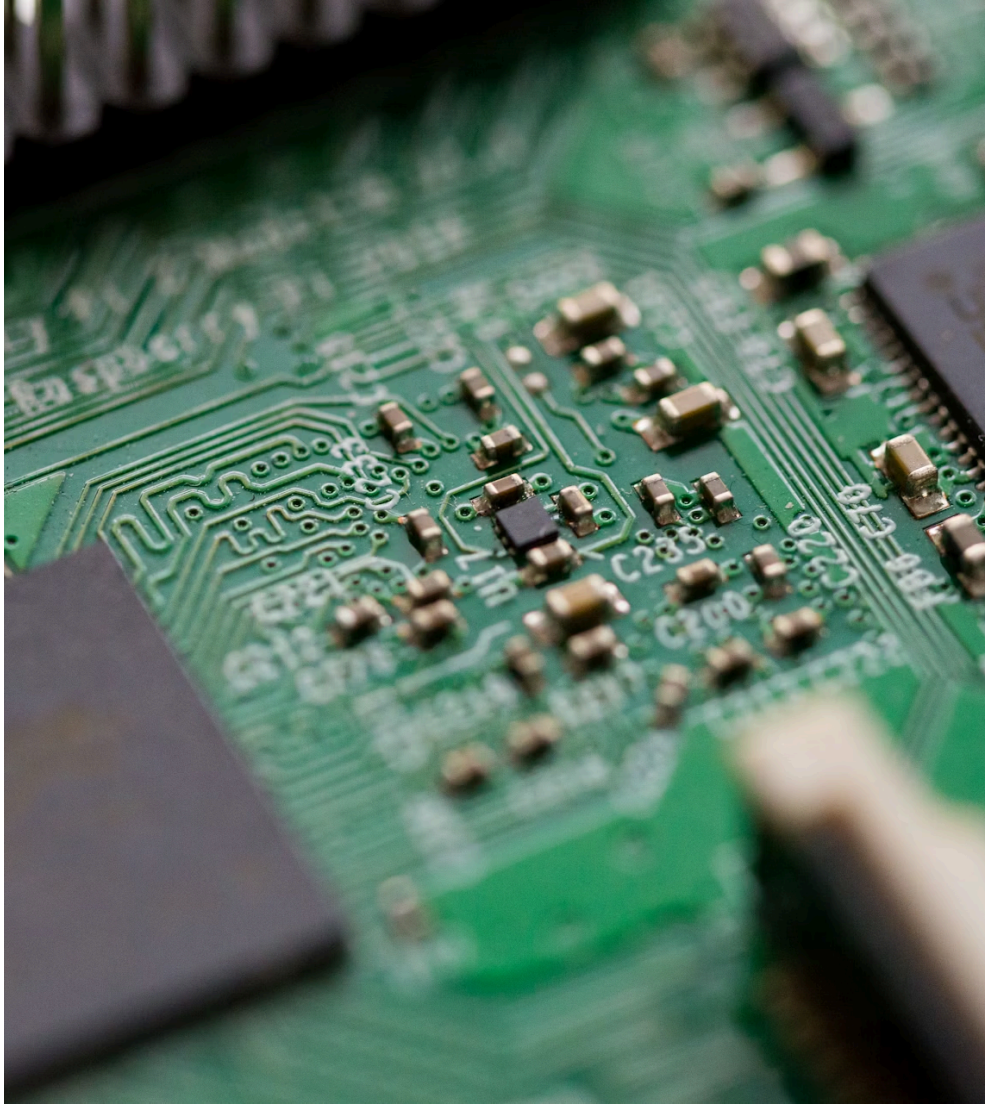
Operation Modes

- **Command Mode** (default) - Interactive conversation
- **Auto-Accept Mode** (Shift+Tab) - Autonomous execution
- **Plan Mode** (Shift+Tab+Tab) - Review plans before execution

Core Productivity Features

Get Productive Immediately

Essential features for daily development work



Code Exploration

- Find files, functions, patterns
- Understand system architecture
- Trace dependencies
- Identify frameworks
- Reference specific files with `@path/to/file.java`

```
"Analyze the UserService class"
"Explain @src/main/java/com/example/UserController.java"
"How does @pom.xml configure Spring Boot?"
```

Test Generation

- Unit test creation
- Edge case identification
- Integration tests
- Mock object setup

```
"Create unit tests for the UserService"
"Add tests for error scenarios"
```

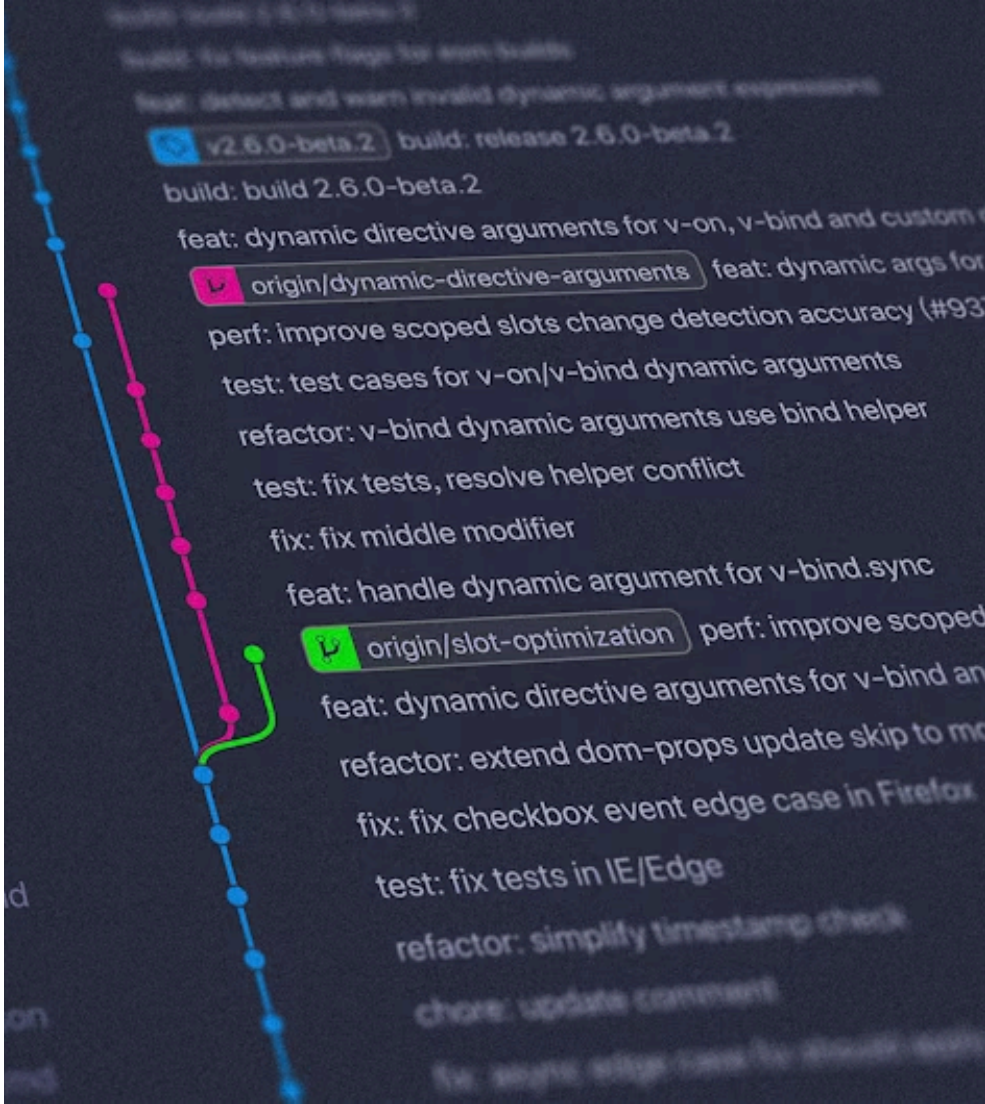
Refactoring Capabilities

- Multi-file operations
- Smart refactoring
- Pattern replacement
- Modern syntax adoption

Git Integration

- Commit message generation
- Branch management
- Merge conflict resolution
- Pull request creation

```
"Commit these changes with an appropriate message"
"Create a pull request for this feature"
```



Documentation Generation

- Inline comments
- `README.md` files
- API documentation
- Architecture docs
- `CLAUDE.md` project configuration (covered later)

Debugging Workflows

- Error message analysis
- Stack trace navigation
- Configuration debugging
- Integration debugging

Essential Workflow Tools

Customize Your Experience

Session management and personalization

CLAUDE.md Files

- **Project memory**: `./CLAUDE.md` (shared with team)
- **User memory**: `~/.claude/CLAUDE.md` (personal preferences)
- Auto-discovered up directory tree
- **Quick add**: Start input with `#` to add memory
- **Commands**: `/memory` to edit, `/init` to bootstrap
- **Import files**: Use `@path/to/import` syntax

Custom Slash Commands

- **Project scope**: `.claude/commands/` (shared with team)
- **User scope**: `~/.claude/commands/` (personal, use `/user:command`)
- **Filename becomes command name** (e.g., `service.md` → `/service`)
- Quick shortcuts for common workflows

Creating Slash Commands

- **Project commands** are shared with the entire team
- **User commands** are personal and require `/user:` prefix
- **Use** `$ARGUMENTS` for dynamic content in commands

```
# Project commands (shared with team)
mkdir -p .claude/commands
echo "Create service for $ARGUMENTS entity" > .claude/commands/service.md

# User commands (personal)
mkdir -p ~/.claude/commands
echo "Fix issue #$ARGUMENTS" > ~/.claude/commands/fix.md

# Usage: /service User  or  /user:fix 123
```

Real example - documentation updater:

```
# .claude/commands/docs.md
Update both the README.md and CLAUDE.md files as appropriate.
If either file does not exist, please create it. Generate the
CLAUDE.md file as though the user invoked the init task.
```

Resuming Conversations

- `--continue` : Automatically resume most recent conversation
- `--resume` : Interactive picker showing conversation history with timestamps and message counts
- **Full history restored**: Complete message context maintained (even hundreds of messages)
- **Original settings preserved**: Model and configuration retained
- **Stored locally**: Complete conversation database maintained on your machine

```
# Continue most recent conversation
claude --continue

# Show conversation picker with details
claude --resume

# Continue with new prompt
claude --continue --print "Continue with my task"
```

Working with Images

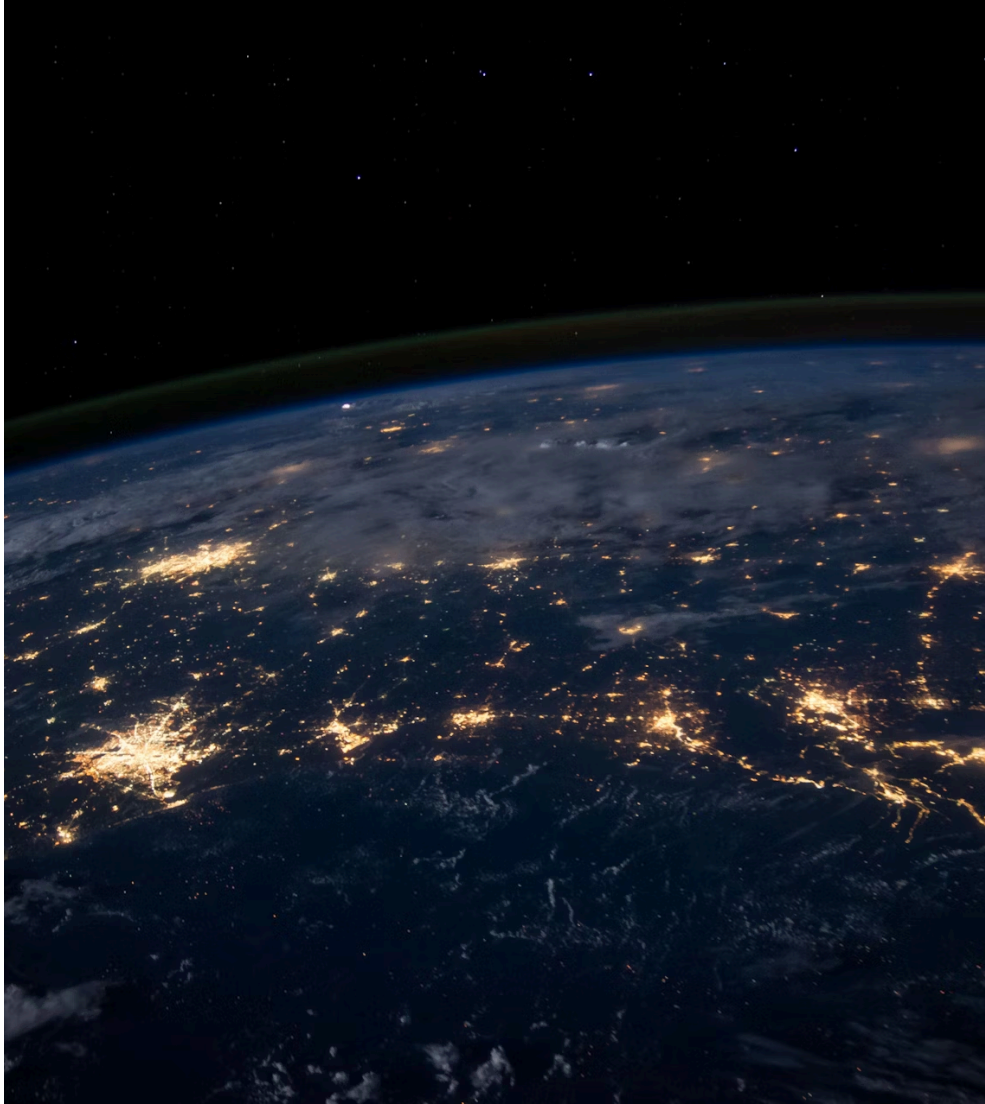
- **Drag and drop** images into Claude Code window
- **Copy/paste** with `Ctrl+V` (not `Cmd+V` even on a Mac!)
- **Provide file path**: "Analyze this image: `/path/to/screenshot.png`"
- Analyze UI designs, error screenshots, diagrams
- Generate code from visual mockups
- Debug visual issues and layouts

```
# Common image workflows
"Analyze this error screenshot and suggest fixes"
"Generate HTML/CSS for this UI mockup"
"Explain what this diagram shows"
"Convert this whiteboard sketch to code"
```

Advanced Features

Power User Capabilities

Complex features for sophisticated workflows



Extended Thinking

- Trigger deeper analysis with "think" in your prompts
- Use "**think more**", "**think harder**", "**ultrathink**" for deeper reasoning
- Shows thinking process as *italic gray text*
- Perfect for complex architectural decisions
- **Verification pattern**: "Before you finish, verify your solution and fix any issues"
- **Note**: Thinking tokens count toward usage but provide higher quality results

```
# Examples of extended thinking prompts
"Think deeply about the best approach for implement
Before you finish, verify your solution and fix any

"Think harder about potential security vulnerabilit
"Think more about the tradeoffs between these two d
```

Plan Mode

- Press `Shift+Tab+Tab` to activate
- Claude presents implementation plan
- Review strategy before execution
- Approve or modify approach
- Perfect for complex changes

Model Context Protocol (MCP)

- Standard protocol for AI-to-system connections
- Tool integration (APIs, databases, services)
- Context enhancement for better AI responses
- Security controls and permissions

MCP Server Examples

- **GitHub MCP** - Repository operations, issues, PRs
- **Sentry MCP** - Error tracking and debugging
- **Linear MCP** - Project management integration
- **Docker MCP** - Container management
- **Database MCP** - Query and schema operations

Setting Up MCP Servers

- Interactive setup: `claude mcp`
- Local servers: Full configuration control
- Remote servers: OAuth authentication, zero maintenance
- Docker MCP Gateway: `docker mcp gateway run`

```
# List existing MCP servers
claude mcp list

# Add local server
claude mcp add my-server -e API_KEY=123 -- /path/to/server

# Add remote server (HTTP)
claude mcp add --transport http remote-server https://example.com/mcp

# Add Docker MCP gateway
claude mcp add docker-mcp docker mcp gateway run
```

Claude Code SDKs

- **Available SDKs**: TypeScript, Python, Command Line
- **Build AI-powered coding assistants** into your workflows
- **Multi-turn conversations** and session management
- **Custom system prompts** and flexible I/O formats
- **MCP integration** for extended capabilities

```
# Command line usage
claude -p "Write a function to calculate Fibonacci numbers"
claude -p "Generate a hello world function" --output-format json
```

```
// TypeScript SDK
import { query } from "@anthropic-ai/claude-code";

for await (const message of query({
  prompt: "Write a haiku about foo.py",
  options: { maxTurns: 3 }
})) {
  // Process messages
}
```

Management & Control

Monitor and Control

Cost, context, and permission management

Cost Monitoring

- Use `/cost` command to check usage
- Shows current usage and limits
- Pro Plan: Displays prompt count vs limit
- Max Plans: Shows monthly usage summary
- Limits reset every 5 hours
- Plan ahead for intensive work sessions

```
# Check your current usage
/cost

# Example output (Pro Plan):
# 📊 Cost information:
#    - Input tokens: 1,245
#    - Output tokens: 3,782
#    - Total cost: $0.076

# Example output (Max Plan):
# With your Claude Max subscription, no need to monitor cost
# — your subscription includes Claude Code usage
```

Context Management

- Use `/compact` command to compress conversation history
- Claude Code automatically compacts when context limit approaches
- Warning message appears before auto-compaction
- Preserves essential information while reducing token usage
- Manual compaction gives you control over timing

```
# Manually compact the conversation
/compact

# Warning message example:
# ⚠️ Context limit approaching. Auto-compacting in next response
# to preserve conversation history and continue working.
```

Configuring Permissions

- **Fine-grained control** over Claude Code's capabilities
- **Use** `/permissions` **UI** to manage tool permissions
- **Allow/Deny rules** for specific tools and actions
- **Enterprise policies** for organization-wide control
- **Permission precedence**: Enterprise → CLI → Project → User

```
# Example permission rules
Bash(npm run test:*)     # Allow npm test commands
Edit(docs/**)           # Allow editing docs directory
Read(src/*)             # Allow reading source files

# Access permissions UI
/permissions
```

Team & Best Practices

Collaborate Effectively

Team workflows and professional practices

Git Worktrees for Parallel Sessions

- Check out multiple branches into separate directories
- Run Claude Code sessions independently on each branch
- Share git history while isolating working files
- Perfect for multi-feature development

```
# Create worktrees for parallel work
git worktree add ../project-feature-a -b feature-a
git worktree add ../project-bugfix bugfix-123

# Run Claude Code in each directory
cd ../project-feature-a && claude
cd ../project-bugfix && claude

# Manage worktrees
git worktree list
git worktree remove ../project-feature-a
```

Best Practices

- **Create a git branch first** - Safe experimentation, easy rollback
- **Update** `README.md` **and** `CLAUDE.md` - Keep documentation current
- **Be specific** - Clear, detailed instructions work better
- **Provide context** - Explain your goals and constraints
- **Iterate gradually** - Make incremental improvements
- **Use examples** - Show desired patterns or styles

Development Process

- Start with clean git state
- Generate tests if none exist
- Commit checkpoints regularly
- Use Claude for git workflows (commits, issues, merges)
- Use git worktrees for parallel sessions on different branches
- Review changes before accepting
- Test generated code thoroughly

Team Collaboration

- Share `CLAUDE.md` configurations
- Document successful patterns
- Review AI-generated code together
- Establish team conventions

Thank You!

Questions?

Kenneth Kousen
Author, Speaker, Java & AI Expert

kousenit.com | @kenkousen