Claude Code Quick Reference Guide

AIA Copilot Training | 2026

Your essential command reference for Claude Code - from installation to advanced workflows.

ðŸš€ Getting Started

Installation

```
 # Install Claude Code globally
npm install -g @anthropic-ai/claude-code

# Or use curl (Linux/Mac)
curl -sL https://install.anthropic.com | sh

# Verify installation
claude --version

# Start interactive session
claude
```

First Steps

```
 # Start with a prompt
claude "explain this project"

# Continue most recent conversation
claude -c

# Resume specific session
claude -r SESSION_ID "continue working"
```

```
# Check system health
claude --doctor
```

âš™ï¸� Essential Commands

Session Management

Command	Description
`claude`	Start interactive REPL
`/help`	Show available commands
`/exit` or `/quit`	End session
`/clear`	Clear conversation history
`/compact`	Summarize conversation to save tokens
`/cos`	Show session cost and duration

File & Context Management

Command	Description
`/init`	Create CLAUDE.md for project
`@filename` + Tab	Reference files (autocomplete)
`claude --add-dir ../apps`	Add additional working directories

Permission & Tools

Command	Description
`/permissions`	View/modify tool permissions

Command	Description
`--allowedTools "Read,Write,Bash"`	Allow specific tools without prompting
`--disallowedTools "Bash(rm:*)"`	Block dangerous commands
`--dangerously-skip-permissions`	âš ï¸� Skip all prompts (use carefully!)

ðŸŽ¯ Model Selection

```
 # Use Sonnet (default, balanced)
claude --model sonnet

# Use Opus (more capable, slower)
claude --model opus

# Use Haiku (faster, cheaper)
claude --model haiku
```

When to use each: - **Haiku:** Quick questions, simple tasks, read-only exploration - **Sonnet:** Most development work, balanced speed/capability - **Opus:** Complex architecture, critical decisions, deep reasoning

---

ðŸ’¡ Thinking Modes

Trigger extended reasoning by asking Claude to "think":

Phrase	Thinking Level	Use Case
"think"	Low (~5s)	Simple analysis
"think hard"	Medium (~15s)	Design decisions
"think harder"	High (~30s)	Complex problems

Phrase	Thinking Level	Use Case
"ultrathink"	Maximum (~60s)	Critical architecture

Example:

```
> Think harder about how to structure the database schema for multi-tenancy
```

ðŸ“‹ CLAUDE.md Configuration

Create `.claude/CLAUDE.md` or `CLAUDE.md` in your project root:

```
 # Project: [Your Project Name]

## Commands
- `npm run dev` - Start dev server
- `npm test` - Run test suite
- `npm run build` - Production build

## Tech Stack
- Node.js 20 + TypeScript
- PostgreSQL with Prisma
- Jest for testing

## Code Conventions
- Use async/await, never callbacks
- Prefer named exports
- All functions need JSDoc comments
- Keep functions under 50 lines

## Workflow
- Run tests after code changes
- Use conventional commit messages
- Never commit console.log statements
```

```
## Don't
- Use `any` type
- Modify database directly
- Skip error handling
```

Pro tip: CLAUDE.md is loaded automatically at session start. Update it as your project evolves.

ðŸ¤– Subagents

Built-in Agents

Agent	Model	Purpose	Tools
`@explore`	Haiku	Fast, read-only search	Glob, Grep, Read, limited Bash
`@plan`	Sonnet	Research during planning	Read, Glob, Grep, Bash
Custom	Your choice	Task-specific work	You define

Using Subagents

```
 # Quick exploration
> @explore find all authentication code

# Planning research
> Use the plan agent to outline a refactoring strategy
```

Creating Custom Subagents

Create `.claude/agents/code-reviewer.md`:

```
---
name: code-reviewer
description: Expert code reviewer for security and quality
tools: Read, Grep, Glob, Bash
model: sonnet
---

You are a senior code reviewer ensuring high standards.

When invoked:
1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:
- Code is simple and readable
- No exposed secrets or API keys
- Good test coverage
- Error handling present
```

Invoke:

```
> @code-reviewer review my recent changes
```

ðŸŽ¨ Custom Commands

Create reusable slash commands in `.claude/commands/`:

Example: Test Generator

`.claude/commands/test-gen.md`:

```
---
name: test-gen
description: Generate unit tests for a file
```

```
---

Generate comprehensive unit tests for $ARGUMENTS.

Requirements:
- Use Jest framework
- Cover happy path and edge cases
- Include setup and teardown
- Mock external dependencies
```

Usage:

```
> /test-gen src/utils/validator.js
```

ðŸª� Hooks System

Hooks execute at specific points in Claude's workflow.

Hook Events

Hook	Fires When	Can Block?
`UserPromptSubmit`	User submits prompt	âœ… Yes (exit 2)
`PreToolUse`	Before tool execution	âœ… Yes (exit 2)
`PostToolUse`	After tool completion	â�Œ No (logging only)
`SubagentStop`	Subagent finishes	âœ… Yes (exit 2)
`SessionStart`	Session starts/resumes	â�Œ No
`PreCompact`	Before compaction	â�Œ No

Example: Block Secret Commits

`.claude/hooks/pre-tool/no-secrets.sh`:

```
#!/bin/bash
# Block commits containing potential secrets

if [[ "$TOOL_NAME" == "Bash" ]] && [[ "$TOOL_INPUT" == *"git commit"* ]]; then
  if git diff --cached | grep -iE "(api[_-]?key|secret|password|token)"; then
    echo "ðŸš¨ BLOCKED: Potential secret detected in staged changes!"
    exit 2  # Exit code 2 blocks the tool
  fi
fi
```

Make executable:

```
chmod +x .claude/hooks/pre-tool/no-secrets.sh
```

ðŸ”Œ MCP (Model Context Protocol)

Configuration Locations

File	Scope	Checked In?
`.mcp.json`	Project (shared)	âœ… Yes
`.claude/mcp_servers.json`	Project (local)	â�Œ No
`~/.claude/mcp_servers.json`	Global	â�Œ No

Example MCP Configuration

`.mcp.json`:

```
{
  "mcpServers": {
    "github": {
      "command": "mcp-server-github",
      "args": ["--token", "${GITHUB_TOKEN}"]
    },
    "filesystem": {
      "command": "mcp-server-filesystem",
      "args": ["--root", "${PROJECT_ROOT}"]
    }
  }
}
```

Using MCP Servers

```
 # List available MCP servers
> /mcp

# Debug MCP connections
claude --mcp-debug
```

ðŸ”§ Git Workflows

Commit with AI-Generated Message

```
 > /commit
# Claude analyzes changes and suggests message
# You approve or edit
```

Create Pull Request

```
 > /pr
# Claude creates PR with description
# Opens in browser for final edits
```

Using gh CLI

```
 > Use gh to list open issues assigned to me
> Create an issue for the bug we just found
```

ðŸ†• Skills Marketplace (NEW - Jan 2026)

Add Official Marketplace

```
 claude
/plugin marketplace add anthropics/skills
```

Browse & Install Skills

```
 /plugins
# Select "Browse plugins" from menu
# Install skills like: context7, frontend-design, document-skills
```

Using Installed Skills

```
 # After installing document-skills:
> Use the PDF doc tool to generate a brief on this project

# After installing frontend-design:
> Use frontend-design to create a landing page with header,
  hero section, 3 feature cards, and CTA
```

Community Marketplace (Optional)

```
 /plugin marketplace add alirezarezvani/claude-skills
```

âš ï¸ Safety: Community skills = open source. Inspect before use.

âš¡ Planning Mode (CRITICAL)

Always use Planning Mode for: - Live databases (Notion, Airtable, spreadsheets) - Production systems (APIs, cloud resources) - Bulk operations (>10 files/entries) - Destructive actions (delete, replace, overwrite)

How to Activate

```
> Plan this task first, show me the steps, and wait for approval

> Spend 5 minutes thinking through this. Show me what files you'll
  read, what changes you'll make, and what commands you'll run.
  Wait for me to say 'proceed' before executing.
```

Why it matters: 5 minutes planning saves 2 hours recovery.

ðŸ”„ Common Workflows

Explore â†' Plan â†' Code â†' Commit

```
1. > Give me an overview of this codebase
2. > Plan how to add user preferences feature
3. > Implement the plan, starting with the database schema
4. > /commit
5. > /pr
```

Test-Driven Development

```
1. > Write failing tests for email validation
2. > /commit
3. > Implement email validation to pass the tests
4. > /commit
```

Bug Fix Workflow

```
 1. > Find where the login timeout is configured
2. > Think hard about why timeouts might be occurring
3. > Fix the issue following existing patterns
4. > Write a regression test
5. > /commit
```

ðŸ“Š Output Formats

```
 # JSON output (for scripting)
claude -p "analyze code" --output-format json

# Stream JSON (real-time processing)
claude -p "large task" --output-format stream-json

# Plain text (default)
claude -p "query" --output-format text
```

ðŸš¨ Troubleshooting

Issue	Solution
"command not found: claude"	Add npm global bin to PATH
Authentication fails	Run `claude login` again
Claude doesn't see files	Ensure you're in project root
Slow responses	Use `/compact` to reduce context
Changes not applying	Check if you approved the diff
MCP connector errors	Run with `--mcp-debug`

âŒ¨ï¸� Keyboard Shortcuts

Shortcut	Action
`Tab`	Autocomplete file/folder names
`Ctrl+C`	Cancel current operation
`Ctrl+D`	Exit session
`â†‘` / `â†“`	Navigate command history
`Ctrl+L`	Clear screen

ðŸ’° Cost Management

```
 # Check current session cost
> /cos

# Limit conversation turns (saves tokens)
claude -p --max-turns 3 "focused query"

# Compact conversations frequently
> /compact "keep only the essential context"

# Clear context between unrelated tasks
> /clear
```

ðŸŽ“ Best Practices

Do's âœ…

- **Be specific** - "Add email regex validation to signup form" (not "add validation")
- **Use @ mentions** - Reference files directly for accurate context
- **Approve carefully** - Always review diffs before accepting

- **Iterate** - Start broad, refine with follow-up prompts
- **Use /compact** - Long sessions slow down; compact periodically
- **Commit often** - Smaller commits = easier to review and revert
- **Use Planning Mode** - For live data and bulk operations

Don'ts â�Œ

- **Don't skip reviews** - Always check what Claude is about to change
- **Don't use `--dangerously-skip-permissions`** - Except in isolated/ containerized environments
- **Don't assume context** - Claude doesn't know what you know; provide background
- **Don't ignore errors** - Ask Claude to explain and fix errors immediately
- **Don't commit secrets** - Use hooks to prevent this

ðŸ”� Security Checklist

- [ ] Review all tool permissions (`/permissions`)
- [ ] Use hooks to block dangerous commands
- [ ] Never commit API keys, tokens, or passwords
- [ ] Use `.gitignore` to protect sensitive files
- [ ] Work on git branches (easy rollback)
- [ ] Use Planning Mode for destructive operations
- [ ] Test in non-production environments first
- [ ] Keep Claude Code updated (`claude update`)

ðŸš€ Quick Tips

1. **Three-Layer Memory:**
2. Global: `~/.claude/CLAUDE.md` (your preferences across all projects)
3. Project: `./.claude/CLAUDE.md` or `./CLAUDE.md` (project-specific)

4. Reference: `@filename` mentions (on-demand context)

5. **Compounding Systems:**

6. Build workflows once, execute forever

7. Example: Interview prep template (45 min vs. 3-4 hours manual)

8. **Model Selection:**

9. Start with Sonnet (balanced)
10. Use Haiku for exploration (fast, cheap)

11. Use Opus for critical decisions (best reasoning)

12. **Context Management:**

13. `/clear` between unrelated tasks
14. `/compact` for long sessions

15. `--max-turns` to limit conversation length

16. **Safety First:**

17. Planning Mode prevents mistakes
18. Hooks enforce security rules
19. Git branches enable easy rollback

ðŸ“š Additional Resources

- **Official Docs:** https://code.claude.com/docs
- **Best Practices:** https://anthropic.com/engineering/claude-code-best-practices
- **MCP Specification:** https://modelcontextprotocol.io
- **Community Skills:** https://github.com/alirezarezvani/claude-skills
- **Awesome Claude Code:** https://github.com/hesreallyhim/awesome-claude-code

---

ðŸ†˜ Getting Help

In Claude Code:

```
 > /help                # Show all commands
> /doctor              # Check installation health
> /mcp                 # MCP server status
```

Community: - GitHub Discussions: https://github.com/anthropics/claude-code/discussions - Discord: Ask in #claude-code channel

---

Keep this reference handy - it will save you hours of searching docs!