

Claude Code

Anthropic's official CLI and agentic coding tool — from first command to advanced workflows.

Last reviewed 28 Sep 2026 ·

Prerequisites: [Git for DE](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundation/s/git-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundation/s/git-for-de/) · [Linux & Bash](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundation/ns/linux-bash/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundation/ns/linux-bash/)

Related: [AI Agents](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/ai-agents/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/ai-agents/) · [LLM APIs](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/llm-apis/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/llm-apis/) · [MCP & Text-to-SQL](#) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mcp-text-to-sql/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

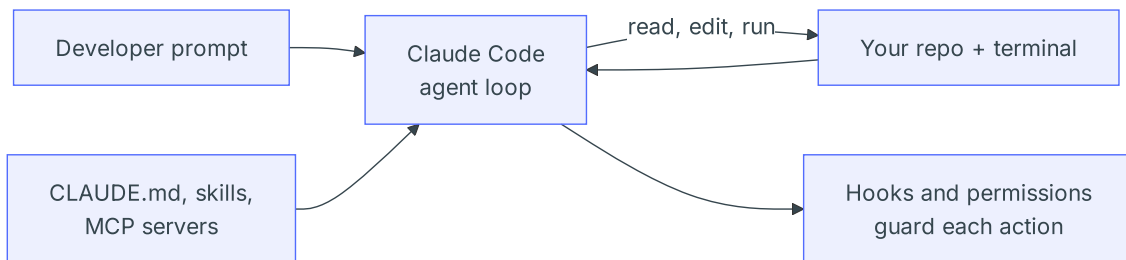
Overview

Challenge: Chat assistants can explain code, but the developer still copies snippets back and forth and performs all editing, running, and testing. The assistant cannot see the repository, run the test suite, or notice that a change breaks a test in another file.

Solution: Claude Code is an AI coding agent that works *inside* the project — in the terminal, IDE, desktop app, or browser. It reads and searches the codebase, edits files, runs commands (tests, linters, build tools, `git`), evaluates the results, and iterates until the task is complete, requesting permission before risky actions.

```
Request: "The orders summary double-counts refunds – fix it📄 and add a test"
  → reads the orders transformation and its upstream sources
  → identifies a join fan-out on the refunds table
  → corrects the join and adds a uniqueness test
  → runs the project's test command    → tests pass
  → summarizes the change and presents the diff for review
```

Configuration: a `CLAUDE.md` file in the repository holds project conventions and is read every session; permission modes and allow-lists control what runs without approval; MCP servers connect it to external systems such as databases and orchestrators; and hooks and skills automate team workflows.



On this page

Basic - [What Is Claude Code](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#what-is-claude-code) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#what-is-claude-code) - [Installation & Setup](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#installation--setup) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#installation--setup) - [First Session](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#first-session) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#first-session) - [Core Commands](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#core-commands) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#core-commands)

Intermediate - [Working with Code](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#working-with-code) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#working-with-code) - [Memory System](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#memory-system) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#memory-system) - [CLAUDE.md Files](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#claudemd-files) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#claudemd-files) - [MCP Servers](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#mcp-servers) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#mcp-servers)

Advanced - [Hooks](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#hooks) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#hooks) - [Custom Slash Commands \(Skills\)](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#custom-slash-commands-skills) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#custom-slash-commands-skills) - [Headless & CI Mode](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#headless--ci-mode) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#headless--ci-mode) - [DE-Specific Workflows](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#de-specific-workflows) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#de-specific-workflows)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#cheat-sheet) - [Interview Questions](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#interview-questions) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#interview-questions) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#further-reading) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/#further-reading)

What Is Claude Code

Claude Code is a terminal-based agentic coding tool. It runs in your shell, reads your codebase, and can edit files, run commands, search, and complete multi-step engineering tasks.

Traditional AI assistant:	You copy-paste code, apply manually, iterate	
Claude Code:	Claude reads your repo, edits files directly,	
	runs tests, fixes errors,	
	commits – you review	

What it can do: - Read and edit files across your entire codebase - Run shell commands (tests, linters, build tools) - Search for symbols, patterns, and files - Understand context from git history, tests, and docs - Remember project-specific rules via CLAUDE.md - Extend with MCP servers (external tools, databases, APIs)

Installation & Setup

```
# Install – native installer (recommended; auto-updates)
curl -fsSL https://claude.ai/install.sh | bash

# Or via npm (requires Node.js 22 or later; the native
installer needs no Node.js)
npm install -g @anthropic-ai/claude-code

# Verify
claude --version

# Authenticate (opens browser for OAuth)
claude

# Or use an API key directly
export ANTHROPIC_API_KEY=sk-ant-...
claude
```

```
# In VS Code – install the extension from the marketplace
# Search: "Claude Code" by Anthropic
```

```
# JetBrains (IntelliJ, PyCharm, etc.)
# Plugins → Marketplace → search "Claude Code"
```

First Session

```
# Start an interactive session in your project directory
cd my-project
claude

# You're now in an interactive session
# Type your task in natural language:

> Explain what this codebase does
> Find all places where we read from S3
> Add error handling to the load_data function in
  pipeline.py
> Run the tests and fix any failures
> Create a new Airflow DAG that runs daily at 2am
```

```
# Quick one-off question (no interactive session)
claude "What does the run_pipeline function do?"

# One-off with output to stdout (useful in scripts)
claude --print "List all Python files that import pandas"
```

Core Commands

```
# In-session slash commands
/help          # show available commands
/clear        # clear conversation context
/compact      # summarize context to save tokens
/usage        # show token usage and cost for
this session (/cost is an alias)
/status       # show current model, context,
settings

# Model and effort
```

```
/model sonnet          # switch model mid-session: an
alias (sonnet, opus, haiku, fable) or a full model ID
/effort                # set how hard the model thinks
(an interactive slider, or /effort high)
/fast                  # toggle fast mode (Opus with
faster output)

# Memory
/memory                # view and edit memory files

# Code review
/code-review           # run a code review on current
branch changes
/code-review ultra     # multi-agent cloud review
(billed)

# Init
/init                  # create a CLAUDE.md for this
project
```

Working with Code

Claude Code reads files before editing them — it always has full context.

```
# Tell it what to do naturally:
> Fix the bug in pipeline/loader.py where it crashes on
empty DataFrames

> Refactor the extract_orders function to use chunked
processing
  so it doesn't OOM on large files

> Add logging to every function in the transforms/
directory

> The fct_orders transformation is too slow – optimize it

> Write tests for the S3Loader class in
tests/test_loader.py
```

```
> The CI is failing on the type check step – fix all mypy errors
```

Permission modes (cycle with Shift+Tab, or pass `--permission-mode`):

Mode	What auto-approves
<code>default</code> (labelled <i>Manual</i> in newer versions)	File reads; prompts on first use of each other tool
<code>acceptEdits</code>	File reads, file edits, and common filesystem commands such as <code>mkdir</code> , <code>touch</code> , <code>mv</code> and <code>cp</code> in the working directory; prompts for other shell commands
<code>plan</code>	Nothing is changed — Claude researches and proposes a plan first
<code>auto</code>	Tool calls, with background safety checks that verify each action matches your request
<code>dontAsk</code>	Nothing new: anything that would prompt is denied, and only pre-approved tools run
<code>bypassPermissions</code> (<code>--dangerously-skip-permissions</code>)	Everything except a few actions no mode approves — only in isolated sandboxes/containers

```
# Auto-accept file edits for this session
claude --permission-mode acceptEdits

# Pre-approve specific tools (and deny others) – also
configurable in .claude/settings.json
claude --allowedTools "Read,Grep,Glob,Edit,Bash(pytest:*)"
--disallowedTools "Bash(rm:*)" 
```

Memory System

Claude Code persists what it learns across sessions as *auto memory* in `~/.claude/projects/<project>/memory/`. It is local to your machine and shared by every worktree of the same repository. Run `/memory` to browse it and to turn it off. The first 200 lines (25 KB) of its `MEMORY.md` index load at the start of each session.

```
# Explicitly ask Claude to remember something
> Remember that we always use snake_case for table and
model names

> Remember that the dev warehouse credentials
are in the team password manager under "Warehouse Dev"

> Remember that CI runs the test suite with fail-fast
enabled

# Claude will save these as memory files and load them in
future sessions
```

Memory types (set in each memory file's frontmatter, with an index in `MEMORY.md`):

- `user` — your role, preferences, expertise
- `feedback` — corrections and confirmed approaches
- `project` — current project context, decisions, deadlines
- `reference` — where to find external information

CLAUDE.md Files

`CLAUDE.md` is a project-level instruction file that Claude always reads. Commit it to your repo.

```
# CLAUDE.md — project-level instructions for Claude Code

## Project overview
This is a data engineering monorepo with orchestration
DAGs, SQL transformations, and Spark jobs.
Stack: cloud warehouse + SQL transformation layer +
orchestrator on Kubernetes + Spark for large jobs.

## Code conventions
- Python: ruff for linting, black for formatting, mypy for
types
- SQL models: snake_case; staging models prefixed stg_,
marts prefixed fct_ or dim_
- SQL: uppercase keywords, 4-space indentation, one column
per line in SELECT
- Airflow: TaskFlow API for new DAGs; legacy DAGs use
PythonOperator
```

```

## Running things
- Tests: `pytest tests/ -v`
- Transformations: `make build-changed` (builds and tests
  changed models only)
- Linting: `ruff check . && mypy .`
- Local stack: `docker compose up -d`

## Important files
- Pipeline configs: config/pipelines.yaml
- Warehouse connection settings: .env.example (never commit
  .env)
- Local connection profiles: ~/.config/ (not in repo)

## What NOT to do
- Never commit credentials — use environment variables
- Never run transformations against production from a
  laptop — deploy through CI
- Never drop tables directly — use versioned migrations
- Don't use SELECT * in production models

```

```

# CLAUDE.md can be at multiple levels:
~/.claude/CLAUDE.md # global (applies
  everywhere)
~/projects/my-data-platform/CLAUDE.md # repo root
~/projects/my-data-platform/transformations/CLAUDE.md #
  subdirectory (loaded when working there)

```

MCP Servers

MCP (Model Context Protocol) extends Claude Code with external tools — databases, APIs, services.

```

# Add an MCP server (default scope: local — just you, this
  project)
claude mcp add my-server -- npx -y @my-org/my-mcp-server

# For all your projects / shared with the team via
  .mcp.json
claude mcp add my-server --scope user -- npx -y @my-

```

```

org/my-mcp-server
claude mcp add my-server --scope project -- npx -y @my-
org/my-mcp-server

# With environment variables (options go BEFORE the server
command)
claude mcp add warehouse \
  -e WAREHOUSE_HOST=warehouse.example.com \
  -e WAREHOUSE_USER=readonly_agent \
  -- npx -y @org/warehouse-mcp-server

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

# List configured MCP servers
claude mcp list

# Remove
claude mcp remove my-server

```

`.mcp.json` (repository root) — project-scoped MCP servers, committed and shared with the team. `${VAR}` values are expanded from each developer's environment; never commit secrets.

```

{
  "mcpServers": {
    "warehouse": {
      "command": "npx",
      "args": ["-y", "@org/warehouse-mcp-server"],
      "env": {
        "WAREHOUSE_HOST": "${WAREHOUSE_HOST}",
        "WAREHOUSE_USER": "${WAREHOUSE_USER}"
      }
    },
    "airflow": {
      "command": "python",
      "args": ["-m", "airflow_mcp_server"],
      "env": {
        "AIRFLOW_URL": "http://localhost:8080",
        "AIRFLOW_TOKEN": "${AIRFLOW_API_TOKEN}"
      }
    }
  }
}

```

```
}  
}  
}
```

With an MCP server, Claude can:

```
> How many orders came in today? → (queries the warehouse directly)  
> Is the daily_orders DAG healthy? → (checks Airflow API)  
> What's in the bronze layer? → (browses S3 via filesystem MCP)
```

Hooks

Hooks run shell commands automatically at specific points in Claude Code's lifecycle — for enforcing standards, running checks, or integrating with external systems. Each hook receives a JSON payload on **stdin** (session ID, tool name, tool input, ...); use `jq` to pull out what you need.

`.claude/settings.json` (project, committed) or `~/.claude/settings.json` (all projects):

```
{  
  "hooks": {  
    "PostToolUse": [  
      {  
        "matcher": "Edit|Write",  
        "hooks": [  
          {  
            "type": "command",  
            "command": "jq -r '.tool_input.file_path' |  
grep '\\.py$' | xargs -r ruff check --fix"  
          }  
        ]  
      }  
    ],  
    "PreToolUse": [  
      {  
        "matcher": "Bash",  
        "hooks": [  
          {
```

```

        "type": "command",
        "command": "jq -r '.tool_input.command' >>
~/.claude/bash-audit.log"
    }
  ],
},
],
"Stop": [
  {
    "hooks": [
      { "type": "command", "command": "osascript -e
'display notification \"Claude Code finished\" with title
\"Claude Code\\\"'\" }
    ]
  }
]
}
}
}
}
}

```

Hook events:

Event	Triggers when
<code>PreToolUse</code>	Before a tool runs — can block it (exit code 2, with the reason on stderr)
<code>PostToolUse</code>	After a tool succeeds — e.g. format or lint the edited file
<code>UserPromptSubmit</code>	When you submit a prompt — can add context or block it
<code>Notification</code>	Claude Code sends a notification (e.g. waiting for permission)
<code>Stop</code> / <code>SubagentStop</code>	The main agent / a subagent finishes responding
<code>PostToolUseFailure</code> / <code>PermissionRequest</code>	A tool call fails / needs a permission decision
<code>SessionStart</code> / <code>SessionEnd</code>	A session starts or ends
<code>PreCompact</code>	Before the conversation is compacted



```
# A PreToolUse script that blocks edits to production
connection profiles
#!/usr/bin/env bash
file=$(jq -r '.tool_input.file_path // empty')
if [[ "$file" == *profiles.yml ]]; then
    echo "Editing profiles.yml is not allowed – change the
template instead." >&2
    exit 2          # exit code 2 = block the tool call and
show the reason to Claude
fi
```

There are many more events (for file changes, worktrees, model switches and so on); the [hooks reference](https://code.claude.com/docs/en/hooks) (<https://code.claude.com/docs/en/hooks>) lists them all. Use `/hooks` in a session to view the configured hooks.

Custom Slash Commands (Skills)

Skills are reusable instructions Claude can load on demand — and you can invoke them as slash commands. Put project skills in `.claude/skills/<name>/SKILL.md` (commit them to share with the team) or personal ones in `~/.claude/skills/`. The older `.claude/commands/<name>.md` format still works.

```
<!-- .claude/skills/sql-review/SKILL.md -->
---
name: sql-review
description: Review a SQL transformation for best
practices. Use when asked to review or check a SQL model.
---
```

Review the SQL model at \$ARGUMENTS (or the most recently edited .sql file):

1. Check for:
 - Hardcoded table names instead of `ref()` / `source()`
 - `SELECT *` usage
 - Missing tests or documentation in the model's YAML
 - Performance issues (cross joins, missing filters on large tables)
 - A materialization that doesn't fit how the model is used

2. Report findings as a prioritized list: HIGH / MEDIUM / LOW
3. Suggest the specific fix for each finding

```
<!-- .claude/skills/pipeline-debug/SKILL.md -->  
---  
name: pipeline-debug  
description: Debug a failing Airflow DAG. Use when a DAG  
run has failed.  
---
```

Debug the DAG named in \$ARGUMENTS:

1. Find the most recent failed run and read the failing task's logs
2. Read the DAG file and the operator implementation
3. Diagnose the root cause
4. Propose a fix with code changes

```
# Use in a session:  
> /sql-review transformations/marts/fct_orders.sql  
> /pipeline-debug daily_orders
```

Subagents are different: specialized assistants with their own system prompt, tools, and context window, defined in `.claude/agents/<name>.md` (or `~/.claude/agents/`). Claude delegates tasks to them (e.g. a read-only `sql-reviewer` agent), which keeps the main conversation's context clean. Ask Claude to create one, or write the file; from version 2.1.198 the `/agents` command no longer opens a creation wizard.

Headless & CI Mode

Run Claude Code non-interactively in scripts and CI pipelines.

```
# Single command, print output, exit – pre-approve only the  
tools it needs  
claude -p "Review the changes in this branch for data  
quality issues" \  
  --allowedTools "Read,Grep,Glob,Bash(git diff:*),Bash(git  
log:*)"
```

```
# Machine-readable output for scripts
claude -p "List SQL models that have no tests" --output-
format json | jq -r '.result'
```

```
# In a CI pipeline (GitHub Actions)
```

```
# .github/workflows/claude-review.yml
name: Claude Code Review

on:
  pull_request:
    paths: ['transformations/**', 'pipelines/**']

jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7
        with:
          fetch-depth: 0    # full history for git diff

      - uses: actions/setup-node@v7
        with:
          node-version: "22"    # Claude Code's npm package
                                # needs Node.js 22 or later

      - name: Install Claude Code
        run: npm install -g @anthropic-ai/claude-code

      - name: Run Claude review
        env:
          ANTHROPIC_API_KEY: ${ secrets.ANTHROPIC_API_KEY }
        run: |
          # Read-only tool allow-list: PR code is untrusted
          # input, so never
          # combine it with --dangerously-skip-permissions
          claude -p \
            "Review the SQL model changes in this PR. Check
            for: missing tests, \
              SELECT *, hardcoded table names, performance
```

```
issues. \
    Output a markdown summary." \
    --allowedTools "Read,Grep,Glob,Bash(git
diff:*)" \
    > review.md

- name: Post review as PR comment
  uses: marocchino/sticky-pull-request-comment@v3
  with:
    path: review.md
```

Anthropic also maintains an official GitHub Action, [anthropics/claude-code-action](https://github.com/anthropics/claude-code-action) (<https://github.com/anthropics/claude-code-action>), which handles PR comments, @claude mentions, and permissions for you. Run `/install-github-app` in a session to set it up.

DE-Specific Workflows

Generate a staging model from a source table

```
> Look at the raw orders table schema in the warehouse (use 
the warehouse MCP tool)
and generate a staging model stg_billing__orders that:
- Renames columns to snake_case
- Casts created_at to a UTC timestamp
- Adds a loaded_at audit column
- Adds not_null and unique tests on the primary key,
following the project's conventions
```

Debug a failing pipeline

```
> The daily_orders Airflow DAG has been failing since 
yesterday.
Check the logs, find the error, trace it to the root
cause,
and propose a fix. Don't make changes yet – just
diagnose.
```

Optimize a slow SQL transformation

> The `fct_revenue_by_region` model takes 45 minutes in production.

Read the model, identify the bottleneck, and rewrite it to run under 5 minutes. Add an explanation of what changed and why.

Write tests for existing pipelines

> Look at all the Airflow DAGs in `dags/` that don't have corresponding tests in `tests/dags/`. Write pytest tests for each one covering:

- DAG imports without errors
- Task dependency structure
- Schedule interval correctness
- Required connections and variables are present

Document a codebase

> Read all the Python files in `pipelines/` and generate a `PIPELINES.md` that documents: what each pipeline does, its schedule, its inputs and outputs, and any known limitations or TODOs.

Migrate code

> Convert all the Airflow DAGs in `dags/` that use `PythonOperator` and task dependencies with `>>` to use the TaskFlow API (`@task` decorator).

Keep the same logic and schedules. Run the tests after each migration.

Common Pitfalls

Pitfall	Symptom	Fix
Vague requests ("clean up the pipeline")	Big, unfocused diffs	Say the goal, the constraints, and how to verify ("...and run the tests for this model")
No <code>CLAUDE.md</code>	Repeating conventions every session; inconsistent style	<code>/init</code> , then keep it short and specific: commands, conventions, gotchas
A bloated <code>CLAUDE.md</code>	Important rules get lost among trivia	Keep it to what's non-obvious; link out to docs for details
<code>--dangerously-skip-permissions</code> on your laptop or in CI on PR code	An agent can run anything, including prompt-injected commands	Allow-lists (<code>--allowedTools</code> , settings permissions); bypass mode only in isolated containers
Giving it production credentials	A mistaken command hits prod data	Dev credentials by default; read-only roles for <u>MCP</u> servers; approvals for writes
One marathon session for many tasks	Context fills up; quality drops	<code>/clear</code> between unrelated tasks; use <code>/compact</code> for long ones; plan mode for large changes
Accepting changes without review	Subtle bugs merged	Ask it to run tests and show the diff; review like a teammate's PR
Secrets pasted into prompts or committed <u>MCP</u> configs	Credentials in logs and Git history	<code>\${ENV_VAR}</code> expansion in <code>.mcp.json</code> ; secrets managers

Cheat Sheet

Task	Command
Start in a project	<code>cd repo && claude</code>
Continue / resume a session	<code>claude -c</code> · <code>claude -r</code> (pick one)

Task	Command
One-shot (scripts, CI)	<code>claude -p "prompt" --output-format json</code>
Create project memory	<code>/init</code> → <code>edit CLAUDE.md</code>
Plan before changing anything	Shift+Tab to plan mode, or <code>claude --permission-mode plan</code>
Pre-approve safe tools	<code>claude --allowedTools "Read,Grep,Glob,Bash(pytest:*)"</code>
Clear / compact context	<code>/clear</code> · <code>/compact</code>
Switch model / effort	<code>/model sonnet</code> · <code>/effort</code>
Add an <u>MCP</u> server	<code>claude mcp add <name> -- <command></code> · <code>--scope project</code> writes <code>.mcp.json</code>
Hooks / permissions	<code>/hooks</code> · <code>/permissions</code> (subagents: ask Claude, or edit <code>.claude/agents/</code>)
Code review	<code>/code-review</code>
Reference a file in a prompt	<code>@models/marts/fct_orders.sql</code>
Run a shell command inline	<code>! pytest tests/unit -q</code>

Where things live

File	Purpose
<code>CLAUDE.md</code> (repo) / <code>~/.claude/CLAUDE.md</code>	Project / personal instructions loaded every session
<code>.claude/settings.json</code>	Team settings: permissions, hooks, env (commit it)
<code>.claude/settings.local.json</code>	Your personal overrides (git-ignored)
<code>.mcp.json</code>	Project <u>MCP</u> servers (commit it; no secrets)
<code>.claude/skills/<name>/SKILL.md</code>	Skills / custom slash commands
<code>.claude/agents/<name>.md</code>	Subagents

Good prompt shape: what to change · where · constraints (style, no new dependencies) · how to verify (tests, a build command, a query) · what "done" looks like

Interview Questions

Q: How is an agentic coding tool different from code completion? A: Completion predicts the next few lines where your cursor is. An agentic tool takes a task, explores the codebase to understand it, makes coordinated edits across files, runs commands to verify (tests, builds, linters), and iterates on failures — closer to delegating a ticket than to autocomplete. That makes it most useful for multi-file changes, debugging, migrations, and writing tests, and it means you review its output like a teammate's pull request.

Q: How would you safely use an AI coding agent on a data platform repository? A: Least privilege and verification: dev or read-only credentials by default, an allow-list of commands (`pytest`, SQL linters, `git diff`), approval for anything that writes to shared systems, and no permission bypass outside sandboxes. Put conventions in `CLAUDE.md`, enforce standards with hooks (formatters, linters, blocking edits to sensitive files), require tests and model builds to pass, and review diffs through normal PRs and CI.

Q: What is MCP and why does it matter for data engineering? A: The Model Context Protocol is an open standard for connecting AI applications to tools and data sources through "MCP servers". Instead of pasting query results into a chat, the agent can query the warehouse, read Airflow run logs, or browse a data catalog directly, through a server you control and scope (for example a read-only database role). One server works across every MCP-compatible client.

Q: How do you give an AI agent the context it needs about a project? A: Layered context: a concise `CLAUDE.md` with conventions, commands, and gotchas; clear code structure and docs it can read; skills for repeatable workflows; MCP connections to live metadata (schemas, lineage, run history); and task prompts that state the goal and how to verify it. Keep the always-loaded context small and let the agent pull in details on demand.

Further Reading

- [Claude Code documentation](https://code.claude.com/docs/en/overview) (https://code.claude.com/docs/en/overview)
 - [Claude Code best practices](https://code.claude.com/docs/en/best-practices) (https://code.claude.com/docs/en/best-practices)
 - [Hooks reference](https://code.claude.com/docs/en/hooks) (https://code.claude.com/docs/en/hooks) • [Skills](https://code.claude.com/docs/en/skills) (https://code.claude.com/docs/en/skills) • [MCP](https://code.claude.com/docs/en/mcp) (https://code.claude.com/docs/en/mcp)
 - [Claude Code GitHub Action](https://github.com/anthropics/claude-code-action) (https://github.com/anthropics/claude-code-action)
 - [Model Context Protocol](https://modelcontextprotocol.io/) (https://modelcontextprotocol.io/)
-

Previous: [MLflow](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/) • **Next:** [Fine-Tuning](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/fine-tuning/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/fine-tuning/) • **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 28, 2026