
Module 15: Claude Code: Basic to Advanced

Use Claude's coding agent to read, edit and build real projects, create and push GitHub repositories, deploy prototypes, and extend it with MCP, skills, subagents and hooks.



AUTHOR

Varun Kumar

Product Management Master Handbook
v0.1 · 23 October 2026

Built from the 55-lecture course study guide and extended to the 12-module Product Core curriculum.
Hypothetical examples are labelled; no real company results are invented.

Contents

15.1 What Claude Code is	3
15.2 Level 1 · Install and your first session	5
15.3 Creating a repository and pushing to GitHub	7
15.4 Level 2 · Working like a professional	8
15.5 Level 3 · Extending Claude Code	10
15.6 Costs	12
15.7 Labs	13
15.8 PM use cases	15
15.9 Common mistakes	15
15.10 Exercises with model answers	15
15.11 Key takeaways	16
15.12 Check your understanding	17
15.13 Recommended resources	18

MODULE 15

Claude Code: Basic to Advanced

Use Claude's coding agent to read, edit and build real projects, create and push GitHub repositories, deploy prototypes, and extend it with MCP, skills, subagents and hooks.

≈ 10 h study

Commands and menus evolve

Claude Code updates often. If a command here behaves differently, run `/help` inside Claude Code or check the [Claude Code docs](#). Terminal screens below are **illustrative**.

15.1 What Claude Code is

Claude Code is an **agentic coding tool**. You describe what you want in plain language; it reads the files in a project, plans, edits code, runs commands (tests, builds, git), shows you what it changed, and asks permission before risky steps. Module 11 introduced it for vibe coding. This class goes from first install to advanced automation.

Why a PM should learn it

You can build and change prototypes yourself, answer "how does this feature actually work?" by reading the codebase, write data scripts, fix copy and small UI bugs, update documentation, and turn a PRD into a working demo in an afternoon. Engineers still review anything that goes to production.

15.1.1 Where Claude Code runs

TERMINAL (CLI)

- Type claude in any project folder
- Most complete feature set
- macOS, Linux, Windows

IDE EXTENSIONS

- VS Code, Cursor, JetBrains
- See diffs inline
- Good once you edit code daily

DESKTOP APP

- Graphical app on Mac and Windows
- Projects on your laptop
- No terminal needed

WEB: CLAUDE.AI/CODE

- Runs in the cloud on your GitHub repositories
- Start tasks from any browser or phone
- Opens pull requests

GITHUB ACTIONS

- Mention @claude in an issue or pull request
- Claude works and replies in GitHub

AGENT SDK AND HEADLESS

- `claude -p 'task'` in scripts and CI
- Build your own agents on the same engine

The places you can use Claude Code. Start with the desktop app or terminal. Move to the web version to run tasks on GitHub repositories from anywhere.

15.1.2 What you need

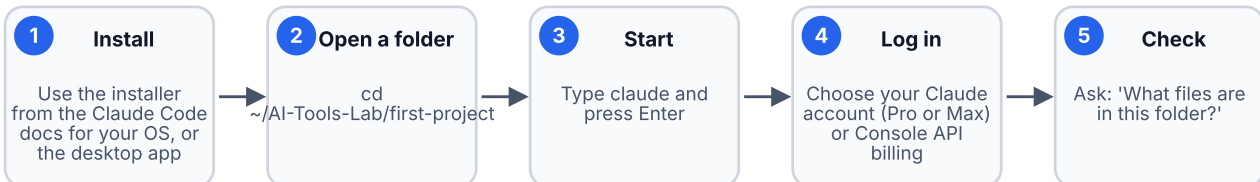
Need	Detail
Claude plan or API	Pro, Max, Team or Enterprise includes Claude Code. Or pay per use with a Claude Console API account.
Git	Version control; installed on macOS, install "Git for Windows" on Windows.
GitHub account	To store and share repositories, and for the web version.
A terminal	Terminal (macOS), PowerShell or Windows Terminal (Windows). Not needed for the desktop app.
Optional	Node.js (some installs and projects), the GitHub CLI <code>gh</code> , a Netlify account for deploying.

15.2 Level 1 · Install and your first session

Level 1 · Beginner

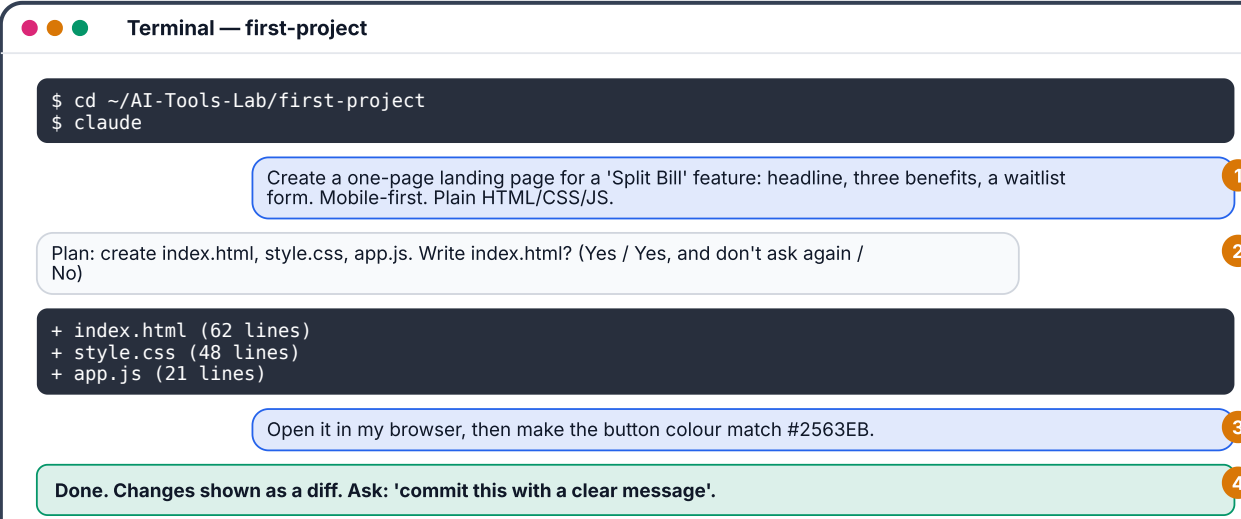
Goal: install Claude Code, make one change to a small project, review it and save it with git.

15.2.1 Install



Install and log in (10 minutes). Typical installer commands are a one-line script for macOS and Linux, a PowerShell line for Windows, or `npm install -g @anthropic-ai/claude-code`. Copy the current command from the docs rather than from a blog.

15.2.2 Your first session



The terminal window, titled "Terminal — first-project", shows a session with Claude Code. It starts with the user navigating to the project directory and running the Claude command. A request is made to create a landing page. Claude asks for confirmation to write the files. The files are created, and the user is asked to open them in a browser and make a color change. Finally, the changes are committed to the repository.

```
$ cd ~/AI-Tools-Lab/first-project
$ claude
```

Create a one-page landing page for a 'Split Bill' feature: headline, three benefits, a waitlist form. Mobile-first. Plain HTML/CSS/JS. 1

Plan: create index.html, style.css, app.js. Write index.html? (Yes / Yes, and don't ask again / No) 2

```
+ index.html (62 lines)
+ style.css (48 lines)
+ app.js (21 lines)
```

Open it in my browser, then make the button colour match #2563EB. 3

Done. Changes shown as a diff. Ask: 'commit this with a clear message'. 4

- 1 Describe the outcome, the constraints (mobile-first, plain HTML) and any data rules (sample data only).
- 2 Claude asks before writing files or running commands. Read the request, then approve once, or approve for the session.
- 3 Iterate in small steps. One change per message is easiest to check.
- 4 Review the diff, then ask Claude to commit. Small commits make mistakes cheap to undo.

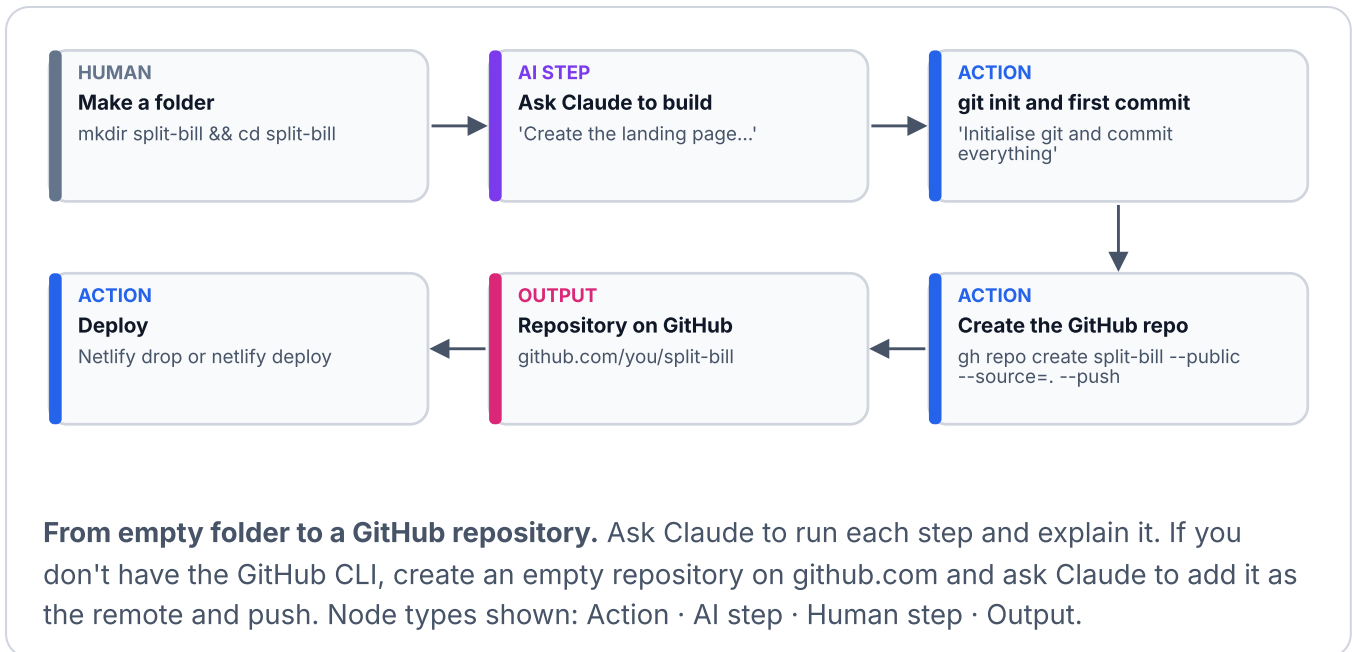
A first Claude Code session in the terminal (illustrative wireframe, not a real product screenshot). Press Esc to stop Claude mid-step. Press Esc twice to rewind to an earlier point in the conversation and undo its changes (checkpoints).

15.2.3 Essential commands

Command or key	What it does
<code>claude</code>	Start a session in the current folder
<code>claude -c</code>	Continue your most recent session here
<code>claude --resume</code>	Pick an older session to resume
<code>/help</code>	List commands available in your version
<code>/init</code>	Create a <code>CLAUDE.md</code> file that describes the project for future sessions
<code>/clear</code>	Start fresh in the same window (clears the conversation)
<code>/compact</code>	Summarise a long conversation to free up context
<code>/model</code>	Switch model (for example a faster or a more capable one)
<code>/permissions</code>	See and change what Claude may do without asking
<code>/mcp</code>	Manage connected MCP servers
Shift+Tab	Cycle modes: normal → auto-accept edits → plan mode
Esc, Esc Esc	Stop the current step; rewind to an earlier checkpoint

15.3 Creating a repository and pushing to GitHub

Every project you build should live in a **git repository**, so you can undo mistakes and share the work.



Prompt that does it all

"Initialise a git repository here, add a sensible .gitignore, commit everything with a clear message, then create a **public** GitHub repository called split-bill with the GitHub CLI and push. Explain each command before you run it."

Two other routes to a repository:

- **Start on GitHub:** create the repository at github.com/new, then ask Claude: "Clone github.com/you/split-bill here and open it."
- **Start in Lovable or Replit:** connect the project to GitHub (Classes 16 and 17), then clone that repository and improve it with Claude Code.

15.4 Level 2 · Working like a professional

Level 2 · Working PM

Goal: make Claude reliable on real projects with project memory, planning before coding, tests, and the right permissions.

15.4.1 CLAUDE.md: the project's instructions

`CLAUDE.md` sits in the project folder. Claude reads it at the start of every session. Run `/init` to create a first version, then edit it.

15.4.4 Permissions and safety

- Claude asks before editing files and running commands. Approve carefully, especially for commands that delete, deploy or spend money.
- Use `/permissions` to pre-approve safe, repeated commands (for example `npm test`) and keep risky ones on "ask".
- Never paste API keys into the chat. Put them in a `.env` file that is listed in `.gitignore`.
- Review the diff before every commit. You are responsible for what you push.

15.5 Level 3 · Extending Claude Code

Level 3 · Advanced

Goal: give Claude new tools (MCP), reusable know-how (skills and commands), helpers (subagents), automatic checks (hooks), and run it without you (headless, GitHub, web).

Knowledge

CLAUDE.md (project rules)

Skills (how to do a task)

Custom slash commands

Tools

MCP servers: Figma, Notion, GitHub, Playwright, your n8n

Helpers

Subagents with their own instructions and tools

Automation

Hooks: run a script before or after a tool

Headless mode and GitHub Actions

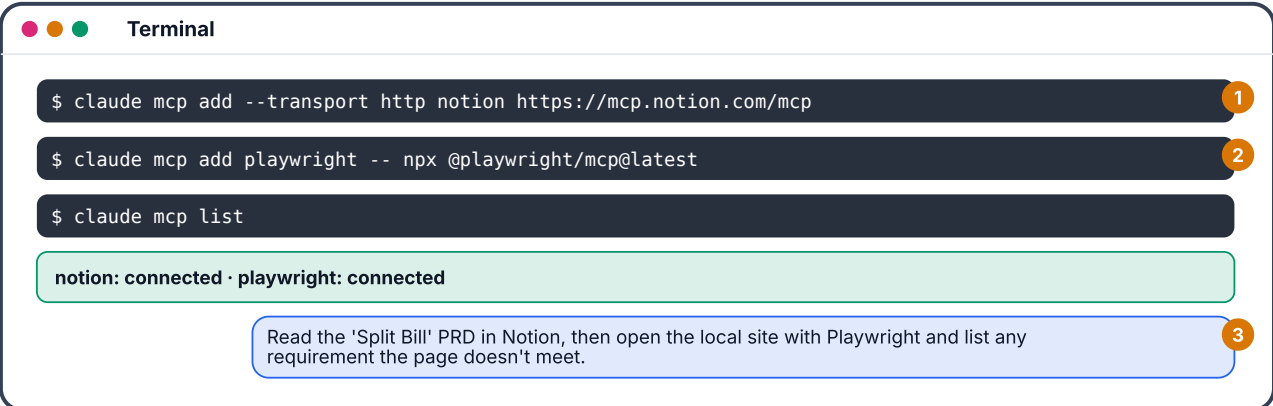
Packaging

Plugins: bundles of skills, agents, commands, hooks and MCP servers

How Claude Code is extended. Add one extension at a time, and only when you repeat the same work at least three times.

15.5.1 MCP servers: new tools for Claude

Add an MCP server once, and Claude can use its tools in every session. Use `claude mcp add`, or `/mcp` inside a session.



The terminal window shows the following commands and output:

```
$ claude mcp add --transport http notion https://mcp.notion.com/mcp
$ claude mcp add playwright -- npx @playwright/mcp@latest
$ claude mcp list
```

Output: `notion: connected · playwright: connected`

Read the 'Split Bill' PRD in Notion, then open the local site with Playwright and list any requirement the page doesn't meet.

- 1 Remote servers: `--transport http` plus the service's MCP URL. The first use opens a browser to sign in. Check each vendor's docs for the current URL.
- 2 Local servers run a command on your machine. Playwright lets Claude open and click through web pages to test them.
- 3 Combine tools in one request: read the spec, test the page, report gaps.

Adding MCP servers (illustrative wireframe, not a real product screenshot). Popular MCP servers for PMs include Figma (build from a design), Notion and Linear or Jira (specs and tickets), GitHub, Playwright (browser testing), and your own n8n workflows (Class 20).

15.5.2 Skills and custom commands

- A **skill** is a folder with a `SKILL.md` file (instructions, examples, optional scripts) inside `.claude/skills/` in your project or your user folder. Claude uses it automatically when a task matches its description. Example: a "release-notes" skill that turns merged pull requests into customer-friendly notes in your house style.
- A **custom slash command** is a Markdown prompt in `.claude/commands/`, run by typing `/its-name`. Example: `/prd-check` asks Claude to compare the code with `docs/prd.md`.

15.5.3 Subagents

A **subagent** is a helper with its own instructions, tools and separate context. Create one with `/agents`. Good PM examples: a **reviewer** that checks a change against the PRD, a **tester** that runs Playwright checks, a **copy editor** that enforces your tone guide. Subagents keep long, noisy work out of your main conversation.

15.5.4 Hooks

Hooks run your own script automatically at set moments, for example **before** a tool runs (block edits to `.env`), or **after** a file is edited (run the formatter). Manage them with `/hooks`. Hooks are rules that always run, unlike instructions, which Claude may occasionally miss.

15.5.5 Plugins

A **plugin** bundles skills, subagents, commands, hooks and MCP servers so a team can install the same setup with one `/plugin` command. Use them to share your team's standard Claude Code setup.

15.5.6 Running without you

Mode	How	PM use
Headless	<code>claude -p "Summarise the changes since last Friday"</code> in a script	Scheduled reports, release notes
GitHub Actions	Run <code>/install-github-app</code> , then write <code>@claude</code> in an issue or pull request	"@claude fix this typo and open a PR"
Claude Code on the web	claude.ai/code → pick a GitHub repository → describe the task	Start a task from your phone; review the pull request later

15.6 Costs

- **Pro and Max plans** include Claude Code; usage is shared with chat, and heavy agentic work uses more.
- **API billing** charges per token. Run `/cost` to see what an API session has spent, and set a spend limit in the Console.
- Cheaper habits: one clear task per session, `/clear` between unrelated tasks, `/compact` in long sessions, plan mode before big changes.

15.7 Labs

Lab 1 • Install and first change (20 minutes)

1. Install Claude Code (desktop app or CLI) and log in.
2. Make a folder `first-project` and start `claude` in it.
3. Ask for a one-page landing page with sample data. Approve the file writes.
4. Ask for one design change. Review the diff.
5. Press **Esc Esc** and rewind the last change, to see that undo works.

Lab 2 • Create and push a GitHub repository (20 minutes)

1. In the same folder: "Initialise git, add a .gitignore, and commit."
2. "Create a public GitHub repository called first-project and push." (Install `gh` and run `gh auth login` first, or create the repository on github.com and ask Claude to push to it.)
3. Open the repository on GitHub and check your files are there.

Lab 3 • From PRD to deployed prototype (60 minutes)

1. Copy `templates/01-prd-template.md` from this handbook into the project as `docs/prd.md` and fill in a small feature.
2. Shift+Tab to **plan mode**: "Read docs/prd.md and plan a clickable prototype that covers every requirement."
3. Approve the plan, leave plan mode, and let Claude build.
4. "Check the page at 375 px width; fix horizontal scroll; label it as a prototype."
5. Deploy: drag the folder onto app.netlify.com/drop, or ask Claude to use the Netlify CLI. Share the link.

Lab 4 • CLAUDE.md and a repeat mistake (15 minutes)

1. Run `/init`. Edit CLAUDE.md using the five-part structure above.
2. Notice one thing Claude got wrong in Lab 3. Add a rule for it.
3. Start a new session and repeat the task. The mistake should not return.

Lab 5 · Test your page with Playwright MCP (20 minutes)

1. `claude mcp add playwright -- npx @playwright/mcp@latest`
2. "Open site/index.html with Playwright, submit the waitlist form empty and with a valid email, and report what happens."
3. Ask Claude to fix any bug it found, then re-run the test.

Lab 6 · A release-notes skill (30 minutes)

1. "Create a skill in .claude/skills/release-notes that turns git commit messages from the last 7 days into customer-friendly release notes, grouped by benefit, max 150 words."
2. Make three small commits.
3. Ask: "Write this week's release notes." Check Claude used the skill.

Lab 7 · Claude Code on the web (15 minutes)

1. Open claude.ai/code and connect GitHub when asked.
2. Choose your `first-project` repository.
3. Task: "Add a FAQ section with three questions and open a pull request."
4. Review the pull request on GitHub and merge it yourself.

15.8 PM use cases

Task	Prompt
Understand a feature	"Explain how refunds work in this codebase, with file names. Don't change anything."
Size a request	"What would it take to add a coupon field at checkout? List files to touch and risks."
Data question	"Write a Python script that reads data/orders.csv and prints weekly retention by cohort."
Copy fix	"Change every 'Checkout' button label to 'Pay now' and show me the diff."
Docs	"Update README.md so a new teammate can run this project in five minutes."
Issue triage	"Read the open GitHub issues and group them into themes with counts."

15.9 Common mistakes

- **Huge first prompts:** build in small steps and commit after each working step.
- **Approving everything without reading:** especially deletes, deploys and package installs.
- **No CLAUDE.md:** Claude rediscovers the project every session and repeats mistakes.
- **Secrets in code:** keys belong in `.env` (git-ignored) or the host's environment variables.
- **Treating a prototype as production:** production code needs engineering review, tests and security checks (Module 11).

15.10 Exercises with model answers

Exercise 1 · Plan or just do it?

Which of these should start in plan mode? (a) Fix a typo in the footer. (b) Add login with email OTP to the prototype. (c) Rename a CSS class.

Model answer

Only **(b)**. It touches several files, adds data storage and has security decisions, so agree the approach first. (a) and (c) are small, contained edits you can review in one diff.

Exercise 2 · Which extension?

Your team keeps asking Claude to check new screens against the Figma design, and someone keeps accidentally committing `.env`. What do you add?

Model answer

Add the **Figma MCP server** (so Claude can read the design) and a short **skill** or **subagent** for "compare the implementation with the Figma frame". For `.env`, add it to `.gitignore` **and** a **PreToolUse hook** that blocks edits or commits touching `.env`. A hook always runs, unlike an instruction.

15.11 Key takeaways

- Claude Code works on **real files and repositories** and asks before risky actions.
- Use **git from the first minute**: commit small, working steps and push to GitHub.
- **CLAUDE.md** and **plan mode** are the two habits that most improve results.
- **MCP, skills, subagents, hooks and plugins** extend Claude Code once you repeat a task.
- Headless mode, **GitHub Actions** and **Claude Code on the web** let it work while you do something else.

15.12 Check your understanding

CHECK YOUR UNDERSTANDING

Which key cycles into plan mode?

- A. Ctrl+C
- B. Shift+Tab
- C. Esc
- D. F5

Why: Shift+Tab cycles normal, auto-accept edits and plan mode.

What does CLAUDE.md do?

- A. Stores your API key
- B. Gives Claude project instructions at the start of every session
- C. Deploys the site
- D. Lists your GitHub followers

Why: Claude reads CLAUDE.md to learn the project's purpose, stack, rules and commands.

You want an automatic check that blocks edits to .env every time. Use...

- A. A line in the prompt
- B. A hook
- C. Plan mode
- D. The /model command

Why: Hooks run your script at fixed points, so the rule is enforced every time.

How do you let Claude read a spec in Notion?

- A. Copy the Notion password into the chat
- B. Add Notion's MCP server with `claude mcp add`
- C. Export Notion as a ZIP
- D. It's not possible

Why: MCP servers give Claude Code tools for other apps; Notion offers a remote MCP server.

15.13 Recommended resources

- [Claude Code docs](#) · [Claude Code best practices](#)
- [Claude Academy: Claude Code in Action](#) (free; formerly Anthropic Academy)
- [DeepLearning.AI: Claude Code: A Highly Agentic Coding Assistant](#) (free, about 2 hours)
- [Coursera: Claude Code: Software Engineering with Generative AI Agents \(Vanderbilt\)](#)
- [GitHub CLI](#) · [Netlify Drop](#) · [Playwright MCP](#)