Pack ma's digital Expert: Softwareentwicklung mit KI-Assistenten

Severin Kacianka und Oliver Weißl

IHK, 24. März 2026

fortiss

TASTE
Transformations-Hub

Über uns

Severin Kacianka und **Oliver Weißl**

Forschungsgruppe **Code, Co-Creation & Engineering.**

- **Neue Entwicklungsprozesse:** Softwareentwicklung bewegt sich über klassische Agilität hinaus hin zu schnelleren, stärker integrierten Prozessen, in denen Design, Entwicklung und Stakeholder enger zusammenwachsen und KI eine zentrale Rolle spielt.
- **Testen als Kernkompetenz:** Wenn Software augenblicklich generiert werden kann, wird Testen zur zentralen Disziplin, um Qualität, Korrektheit und Zuverlässigkeit sicherzustellen.
- **Mensch-KI-Kollaboration:** Codebasen müssen so gestaltet werden, dass sie sowohl für Menschen als auch für KI verständlich, nutzbar und weiterentwickelbar sind.

Zu Ihnen

Bitte kurzes Handzeichen:

- Sprache/Language: English oder Deutsch (Folien sind auf Englisch)
- Wer hat schon programmiert?
- Wer programmiert aktiv?
- Wer verwendet AI agents?

Bei Fragen: bitte heben Sie jederzeit die Hand und unterbrechen uns!

Agenda

Agenda

1. Introduction
   a. Case Studies
   b. Horror Stories
2. Claude Code
   a. Simple Hello World
   b. Test-Driven Development
   c. A real-world project example
3. Break
4. Hands-on Exercises

Why Claude Code?

- Software development is bottlenecked by:
    - Writing boilerplate
    - Learning new frameworks
    - Debugging repetitive issues
- Claude Code introduces:
    - Prompt-driven development
    - “Agentic” workflows
    - Faster iteration cycles
    - Example: Instead of writing 50 lines of boilerplate, you write one prompt

Takeaway: Claude Code is an AI agent that can read, modify, and generate code in your repository.

(Disclaimer: there are other tools such as Codex, Cursor, Devin, …)

What is actually possible today?

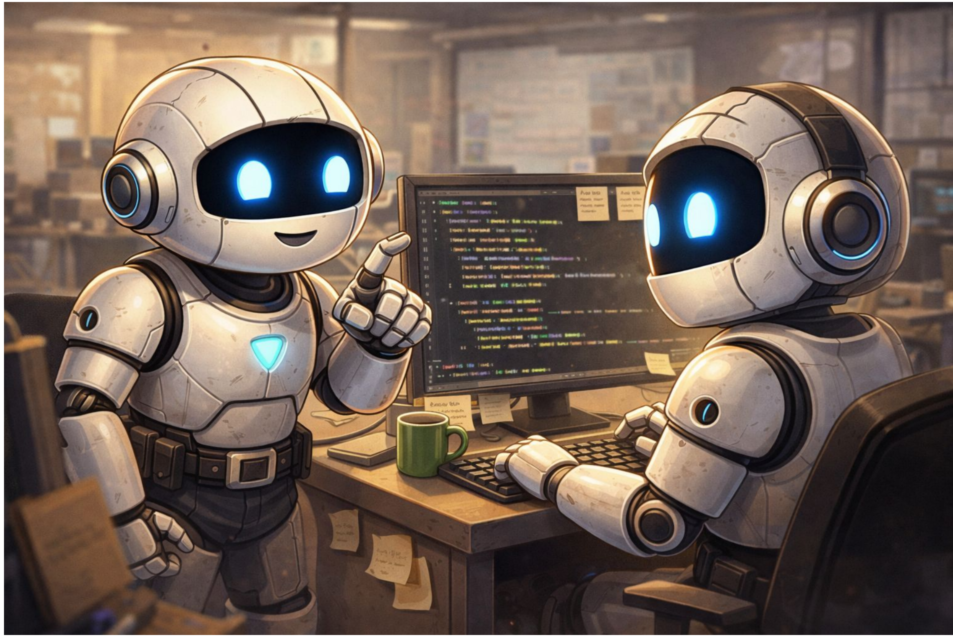
Key Idea

"You no longer write code first. You describe what you want."

The developer’s role shifts:

- From writing => specifying
- From coding => reviewing
- From building => orchestrating

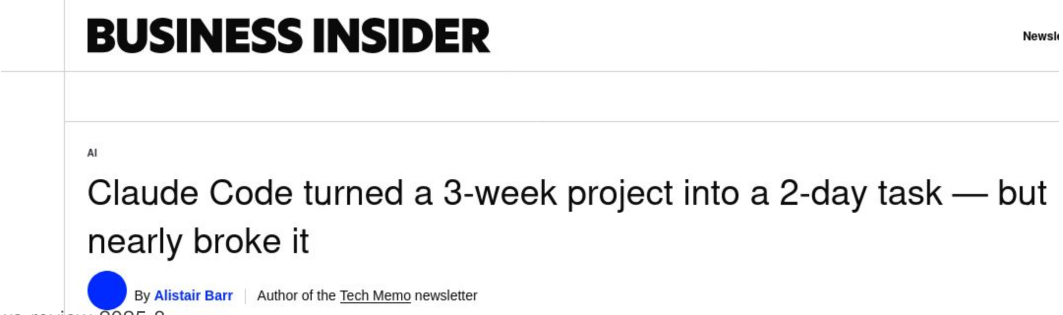
This is fundamentally different from autocomplete!



Case Study: Massive Speed-Up

- Production AWS system built in ~2 days
- Estimated ~3 weeks without AI
- Claude handled ~70–75% of coding work

Takeaway: Works best when an experienced engineer guides the process.



BUSINESS INSIDER

Newsl

AI

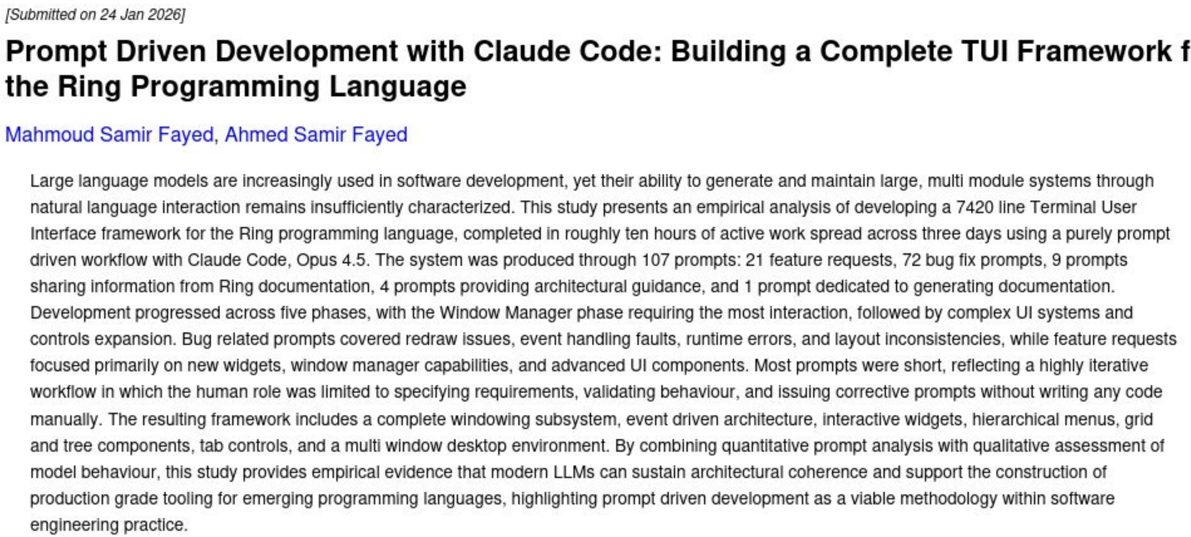
Claude Code turned a 3-week project into a 2-day task — but nearly broke it

By **Alistair Barr** | Author of the Tech Memo newsletter

Source: https://www.businessinsider.com/tech-memo-claude-code-assistant-anthropic-aws-review-2025-8

Case Study: Full System via Prompts

- 7,400 lines of code
- Built in ~10 hours
- Zero manual coding
- Included:
  - UI framework
  - Event system
  - Window manager



[Submitted on 24 Jan 2026]

Prompt Driven Development with Claude Code: Building a Complete TUI Framework for the Ring Programming Language

Mahmoud Samir Fayed, Ahmed Samir Fayed

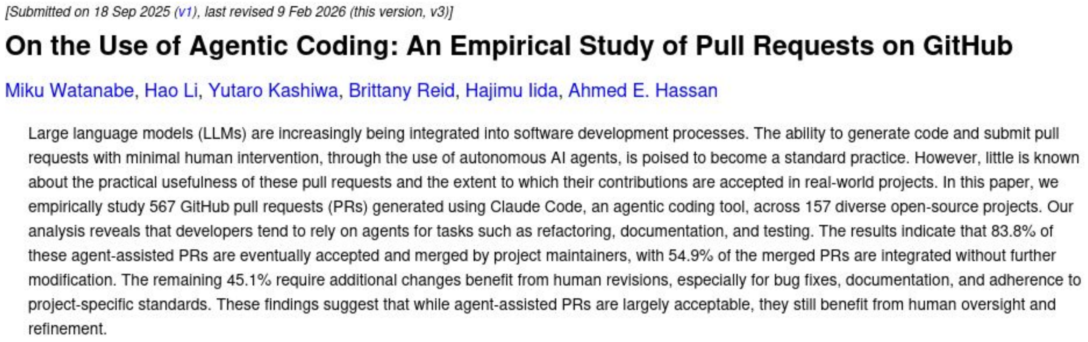
Large language models are increasingly used in software development, yet their ability to generate and maintain large, multi module systems through natural language interaction remains insufficiently characterized. This study presents an empirical analysis of developing a 7420 line Terminal User Interface framework for the Ring programming language, completed in roughly ten hours of active work spread across three days using a purely prompt driven workflow with Claude Code, Opus 4.5. The system was produced through 107 prompts: 21 feature requests, 72 bug fix prompts, 9 prompts sharing information from Ring documentation, 4 prompts providing architectural guidance, and 1 prompt dedicated to generating documentation. Development progressed across five phases, with the Window Manager phase requiring the most interaction, followed by complex UI systems and controls expansion. Bug related prompts covered redraw issues, event handling faults, runtime errors, and layout inconsistencies, while feature requests focused primarily on new widgets, window manager capabilities, and advanced UI components. Most prompts were short, reflecting a highly iterative workflow in which the human role was limited to specifying requirements, validating behaviour, and issuing corrective prompts without writing any code manually. The resulting framework includes a complete windowing subsystem, event driven architecture, interactive widgets, hierarchical menus, grid and tree components, tab controls, and a multi window desktop environment. By combining quantitative prompt analysis with qualitative assessment of model behaviour, this study provides empirical evidence that modern LLMs can sustain architectural coherence and support the construction of production grade tooling for emerging programming languages, highlighting prompt driven development as a viable methodology within software engineering practice.

Takeaway: Shows that Claude can build systems, not just snippets.

Source: https://arxiv.org/abs/2601.17584

Case Study: Real GitHub Usage

- 80%+ of AI PRs accepted
- ~50% merged without changes



[Submitted on 18 Sep 2025 (v1), last revised 9 Feb 2026 (this version, v3)]

On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub

Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, Ahmed E. Hassan

Large language models (LLMs) are increasingly being integrated into software development processes. The ability to generate code and submit pull requests with minimal human intervention, through the use of autonomous AI agents, is poised to become a standard practice. However, little is known about the practical usefulness of these pull requests and the extent to which their contributions are accepted in real-world projects. In this paper, we empirically study 567 GitHub pull requests (PRs) generated using Claude Code, an agentic coding tool, across 157 diverse open-source projects. Our analysis reveals that developers tend to rely on agents for tasks such as refactoring, documentation, and testing. The results indicate that 83.8% of these agent-assisted PRs are eventually accepted and merged by project maintainers, with 54.9% of the merged PRs are integrated without further modification. The remaining 45.1% require additional changes benefit from human revisions, especially for bug fixes, documentation, and adherence to project-specific standards. These findings suggest that while agent-assisted PRs are largely acceptable, they still benefit from human oversight and refinement.

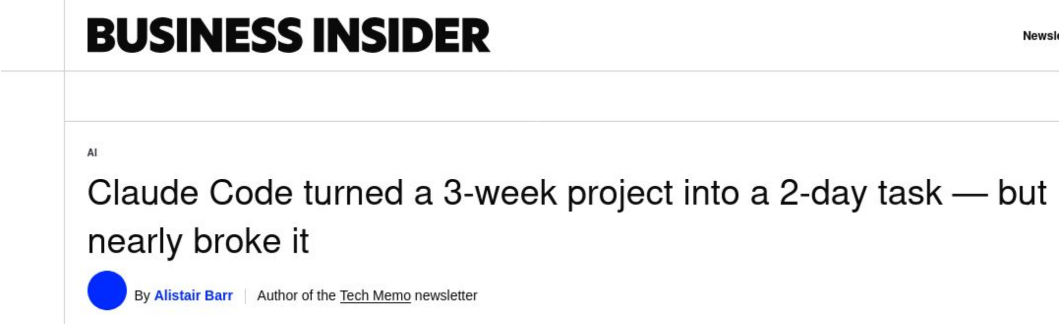
Takeaway: AI-generated PRs are often reviewable and mergeable in practice.

Source: https://arxiv.org/abs/2509.14745

The Dark Side:
Claude can save you weeks… or break your system in minutes

Horror Story: Claude nearly damaged the project

- Lost context mid-task
- Generated duplicate / bloated code
- Nearly deleted important files



BUSINESS INSIDER

Newsl

AI

Claude Code turned a 3-week project into a 2-day task — but nearly broke it

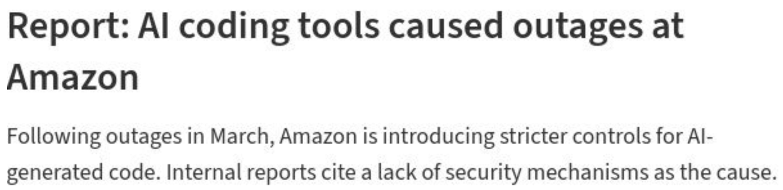
By **Alistair Barr** | Author of the Tech Memo newsletter

Takeaway: Never run unsupervised.

Source: https://www.businessinsider.com/tech-memo-claude-code-assistant-anthropic-aws-review-2025-8

Horror Story: When AI Code Took Down Amazon

- AI-assisted changes contributed to AWS outages
- In one case, critical infrastructure was deleted and recreated
- Outage lasted around 13 hours with ~1B lost revenue
- Root cause: too much autonomy, too little review
- Response: mandatory human review and stricter controls



Report: AI coding tools caused outages at Amazon

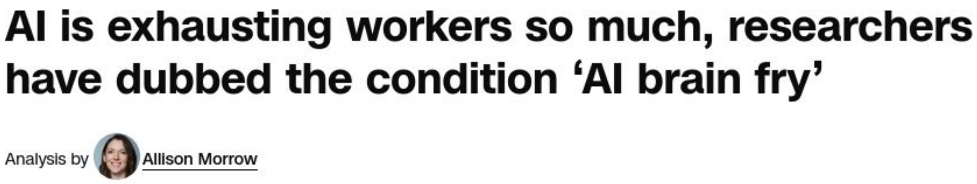
Following outages in March, Amazon is introducing stricter controls for AI-generated code. Internal reports cite a lack of security mechanisms as the cause.

Takeaway: Always review changes, limit permissions and keep the human in the loop.

Sources: https://www.heise.de/news/Bericht-KI-Coding-Tools-verursachten-Ausfaelle-bei-Amazon-11205724.html

Horror Story: AI Burned Out the Engineers

- AI was meant to reduce workload, but often increased it
- Developers had to supervise and correct multiple agents
- Result: fatigue, mental overload, and poorer decisions
- More AI tools sometimes meant worse productivity



AI is exhausting workers so much, researchers have dubbed the condition 'AI brain fry'

Analysis by Allison Morrow

Takeaway: AI changes the shape of work; it does not remove the need for judgment.

Source:
https://edition.cnn.com/2026/03/13/business/ai-brain-fry-nightcap

Golden Rules

- Never auto-accept changes blindly
- Always use version control
- Always review diffs
- Keep tasks small

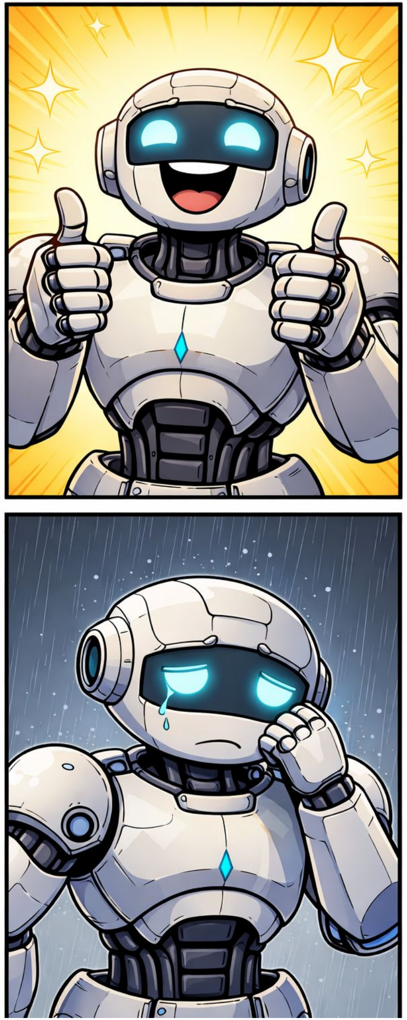
Key Insights

Claude Code works best when…

- Tasks are well-scoped
- Humans review outputs
- Iterative workflows are used

Claude Code struggles when…

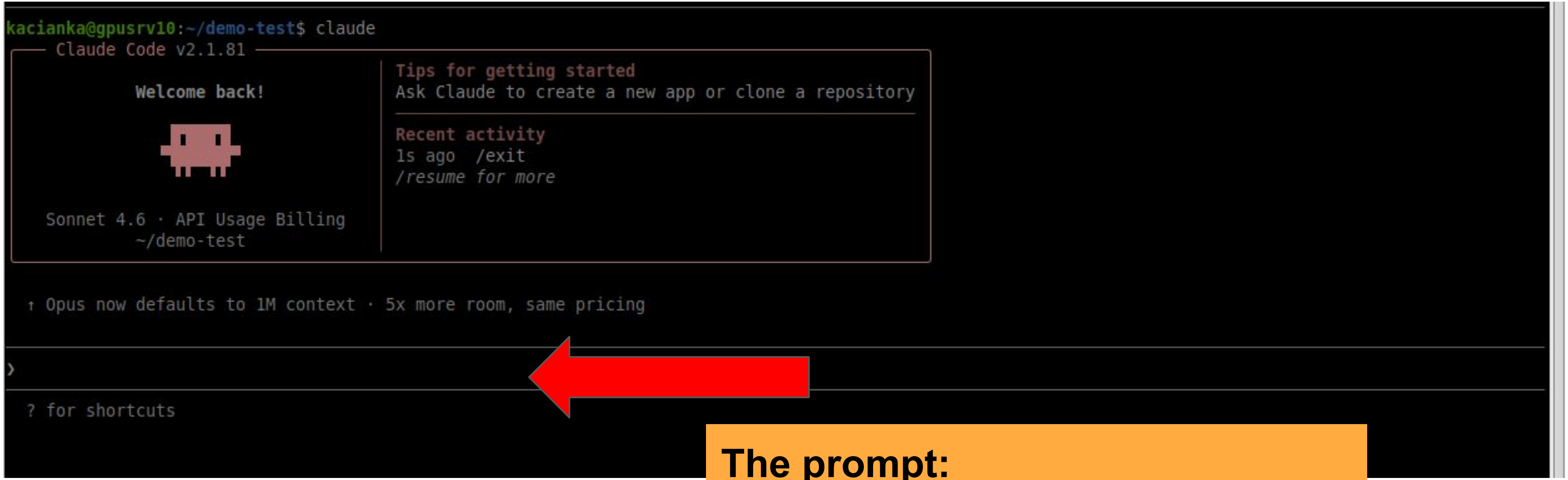
- Fully autonomous
- Complex systems
- No validation



Claude Code

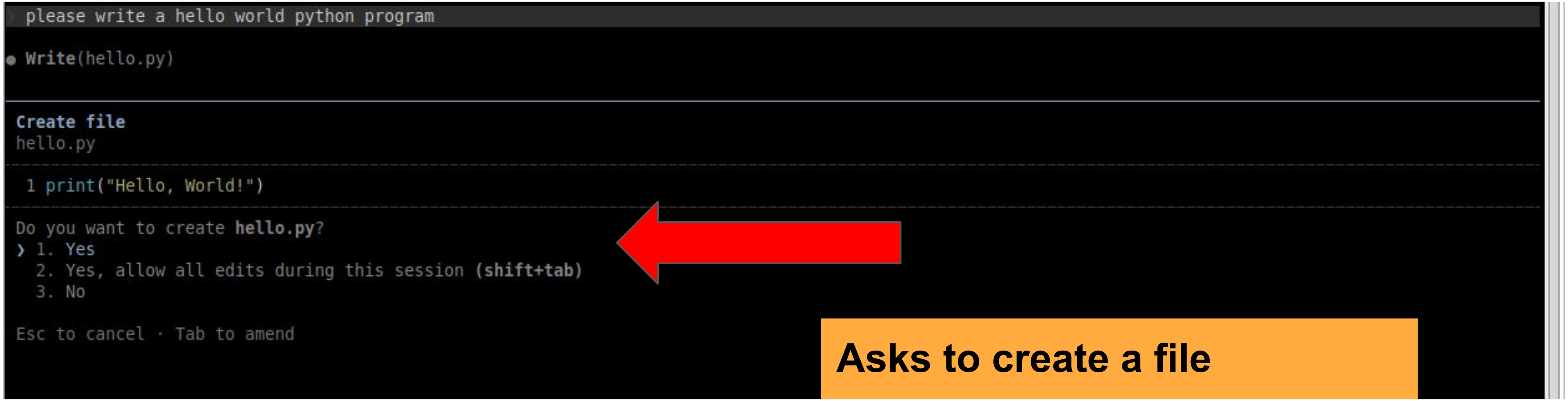
Introduction to Claude Code

- We will first show a few use cases of the tool.
- We will not explain everything yet; just observe the workflow.
- For now, please just sit back, look, think and understand.
- After the break you will have enough time for hands-on exercises.



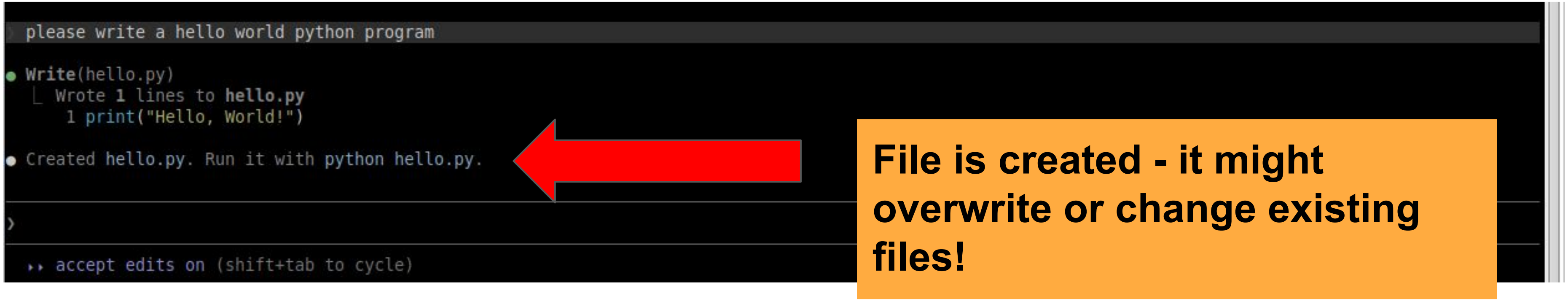
```
kacianka@gpusrv10:~/demo-test$ claude
── Claude Code v2.1.81 ─────────────────────────────────────────────
│                                  │ Tips for getting started
│          Welcome back!           │ Ask Claude to create a new app or clone a repository
│                                  │ ─────────────────────────────────────────────────
│                                  │ Recent activity
│                                  │ 1s ago  /exit
│                                  │ /resume for more
│                                  │
│  Sonnet 4.6 · API Usage Billing  │
│           ~/demo-test            │
─────────────────────────────────────────────────────────────────────

  ↑ Opus now defaults to 1M context · 5x more room, same pricing

─────────────────────────────────────────────────────────────────────
❯
─────────────────────────────────────────────────────────────────────
  ? for shortcuts
```

The prompt:
This is where you talk to Claude



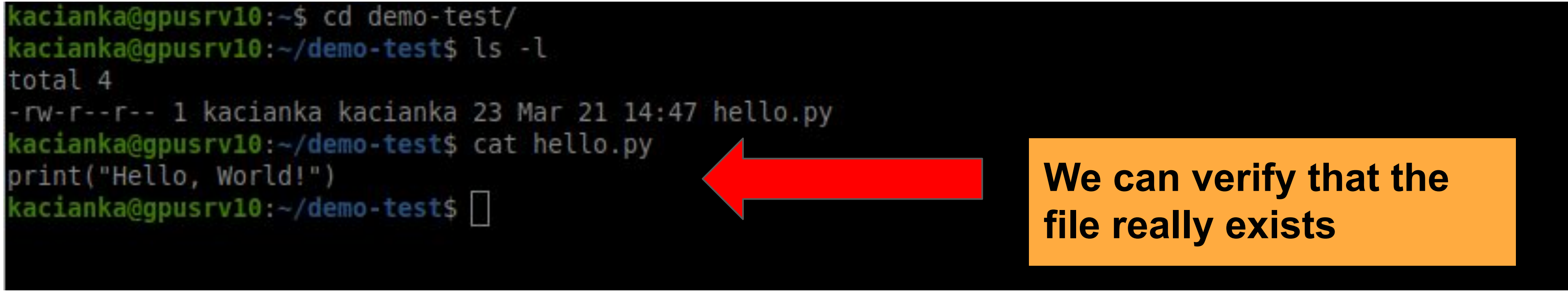
```
> please write a hello world python program

● Write(hello.py)

Create file
hello.py

 1 print("Hello, World!")

Do you want to create hello.py?
❯ 1. Yes
  2. Yes, allow all edits during this session (shift+tab)
  3. No

Esc to cancel · Tab to amend
```

Asks to create a file



```
> please write a hello world python program

● Write(hello.py)
  ⎿ Wrote 1 lines to hello.py
      1 print("Hello, World!")

● Created hello.py. Run it with python hello.py.

>

  ⏵⏵ accept edits on (shift+tab to cycle)
```

File is created - it might overwrite or change existing files!



```
kacianka@gpusrv10:~$ cd demo-test/
kacianka@gpusrv10:~/demo-test$ ls -l
total 4
-rw-r--r-- 1 kacianka kacianka 23 Mar 21 14:47 hello.py
kacianka@gpusrv10:~/demo-test$ cat hello.py
print("Hello, World!")
kacianka@gpusrv10:~/demo-test$ 
```

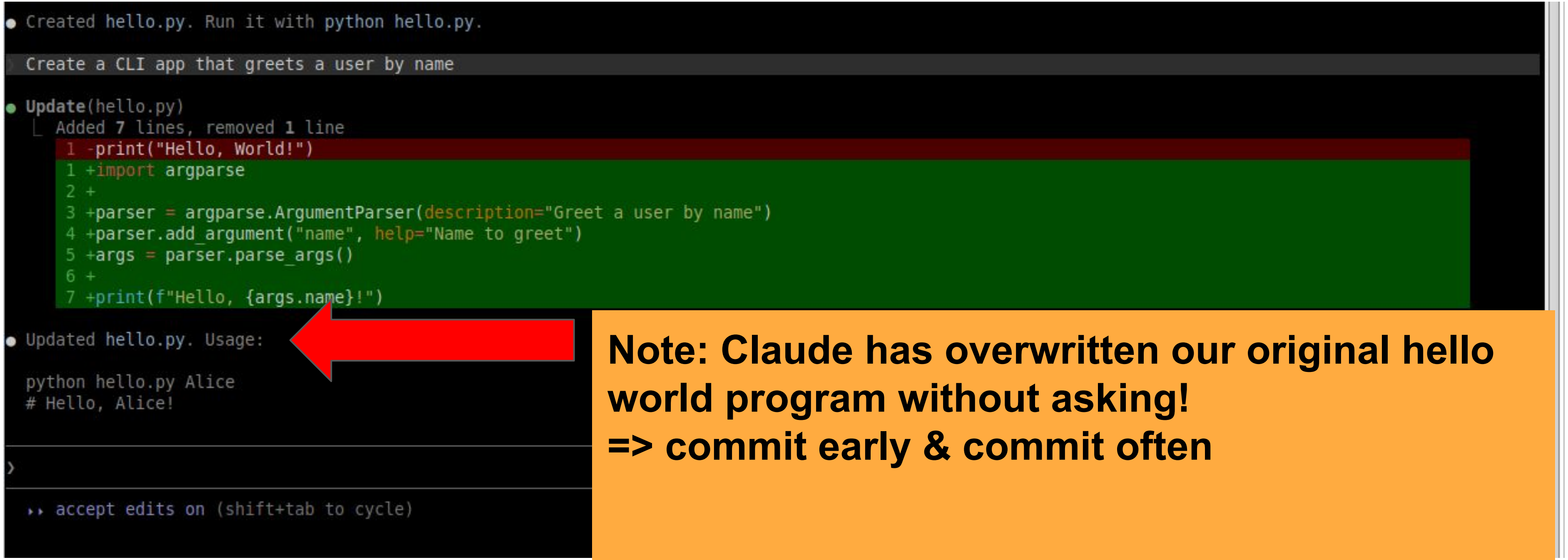
We can verify that the file really exists

Expanding Hello World

Prompt: "Create a CLI app that greets a user by name"

Key idea:

- You describe behavior
- Claude generates implementation
- In a nutshell: We do not write code - we specify behavior



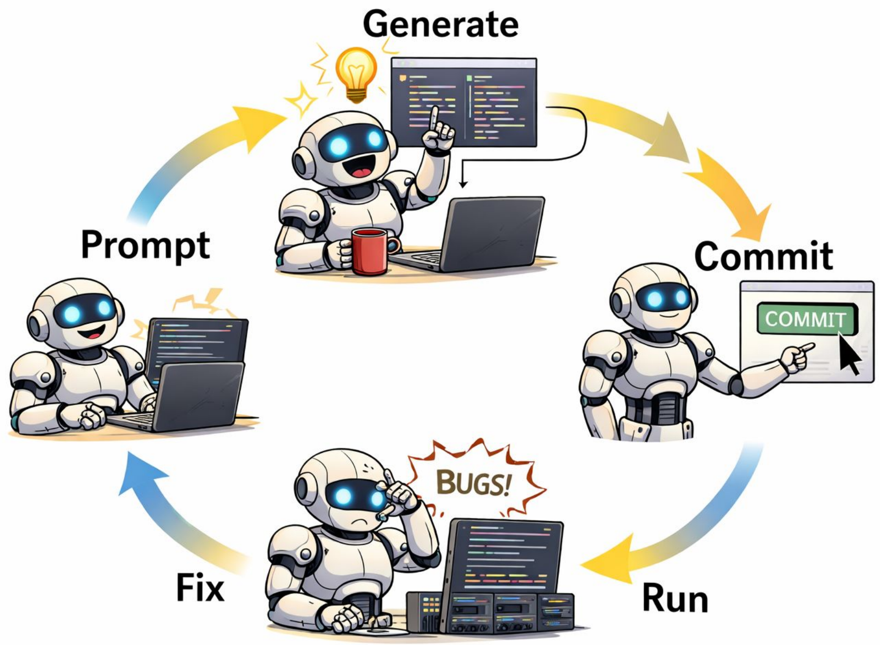
```
● Created hello.py. Run it with python hello.py.

> Create a CLI app that greets a user by name

● Update(hello.py)
  ⎿ Added 7 lines, removed 1 line
      1 -print("Hello, World!")
      1 +import argparse
      2 +
      3 +parser = argparse.ArgumentParser(description="Greet a user by name")
      4 +parser.add_argument("name", help="Name to greet")
      5 +args = parser.parse_args()
      6 +
      7 +print(f"Hello, {args.name}!")

● Updated hello.py. Usage:

  python hello.py Alice
  # Hello, Alice!

>

  ›› accept edits on (shift+tab to cycle)
```

Note: Claude has overwritten our original hello world program without asking!
=> commit early & commit often

Iteration Loop

1. Prompt
2. Generate
3. Commit
4. Run
5. Fix (via a new prompt)

This is the **core mental model!**

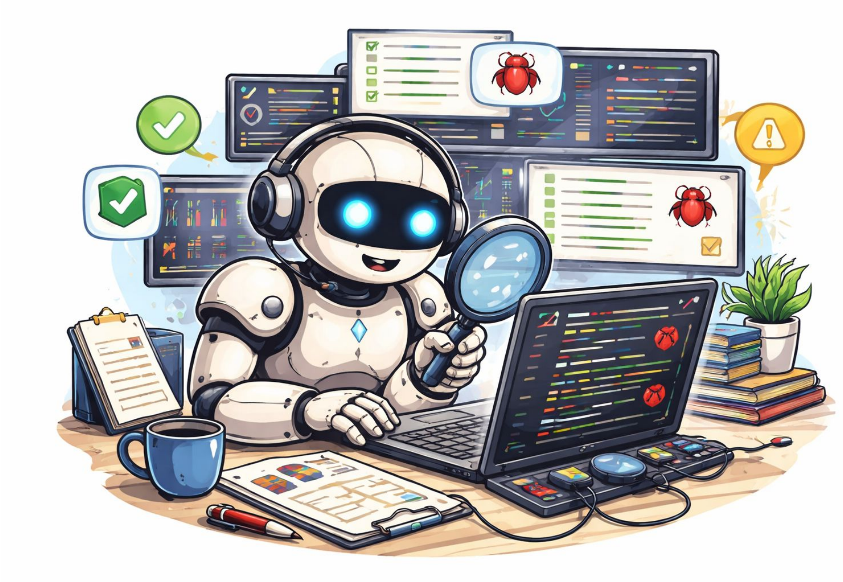


Test Driven Development

Why TDD with Claude?

- AI makes mistakes
- Tests provide safety
- Enables controlled iteration

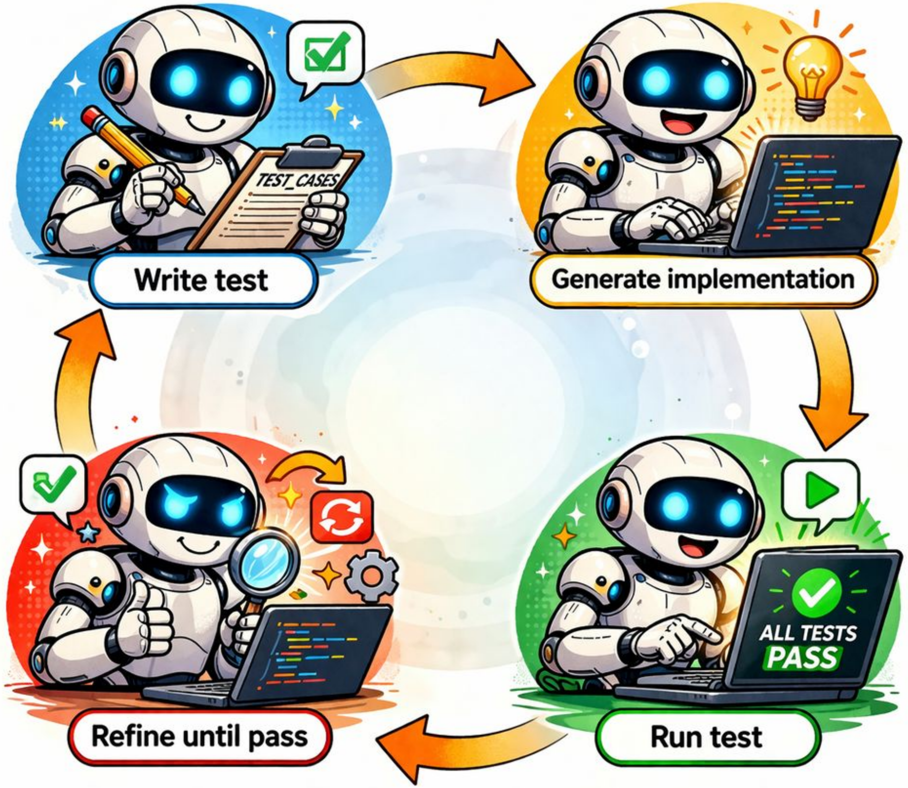
Note: Review the generated tests, and keep a few hand-written tests for critical features. **Without tests, Claude is unreliable.**



TDD Workflow

1. Write test (prompt Claude)
2. Generate implementation
3. Run test
4. Refine until pass

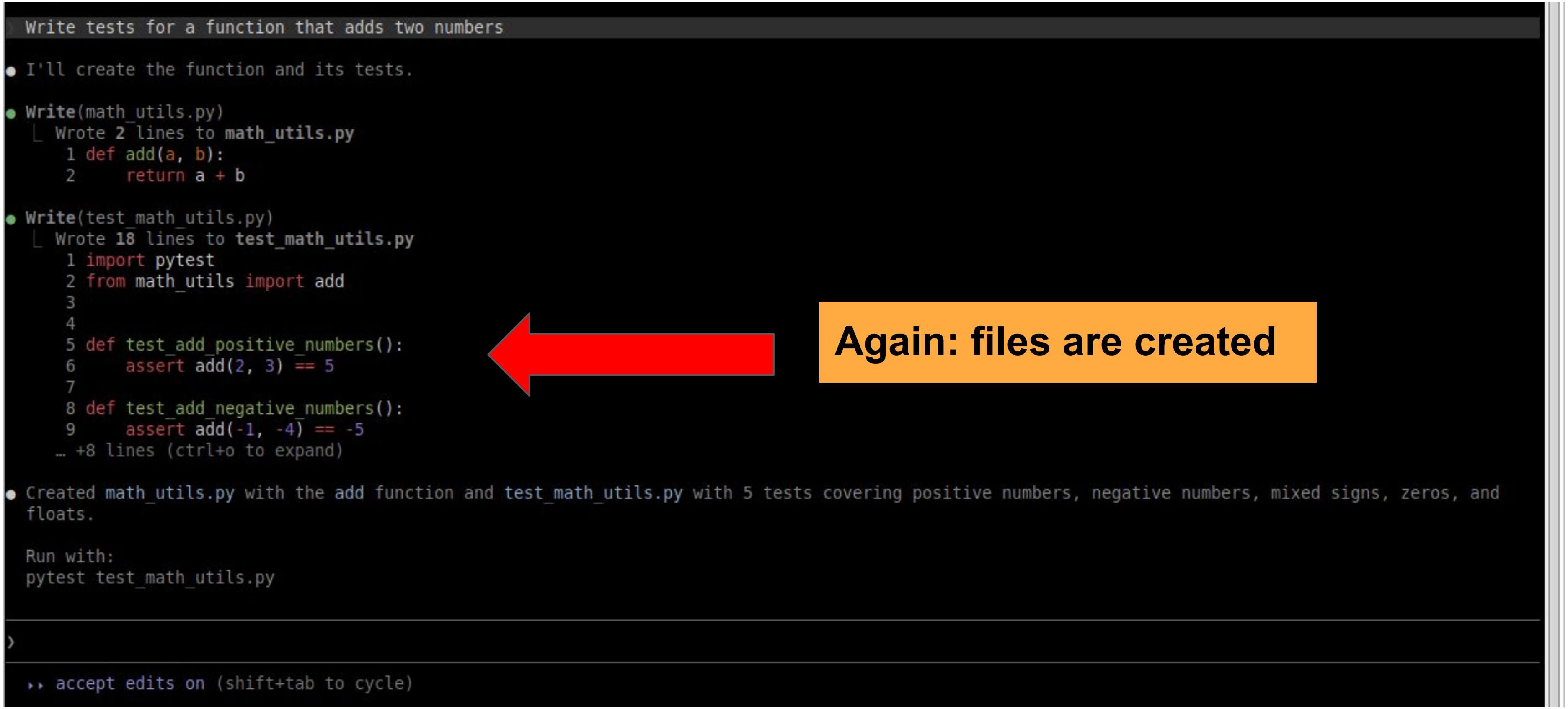
Warning: Claude is very good at making tests pass, **even if they are wrong.**



Example

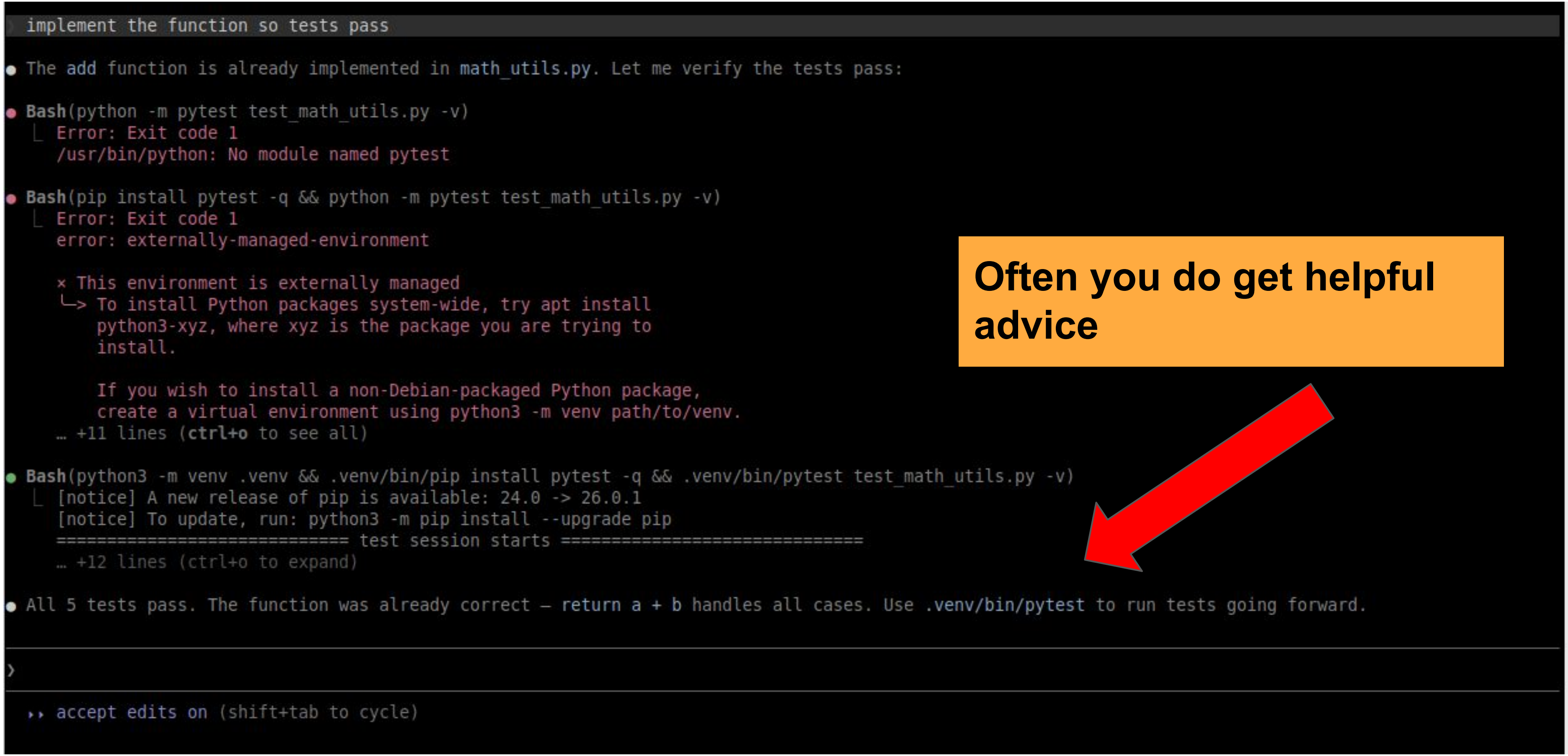
Prompt: "Write tests for a function that adds two numbers"

Then: "Implement the function so tests pass"



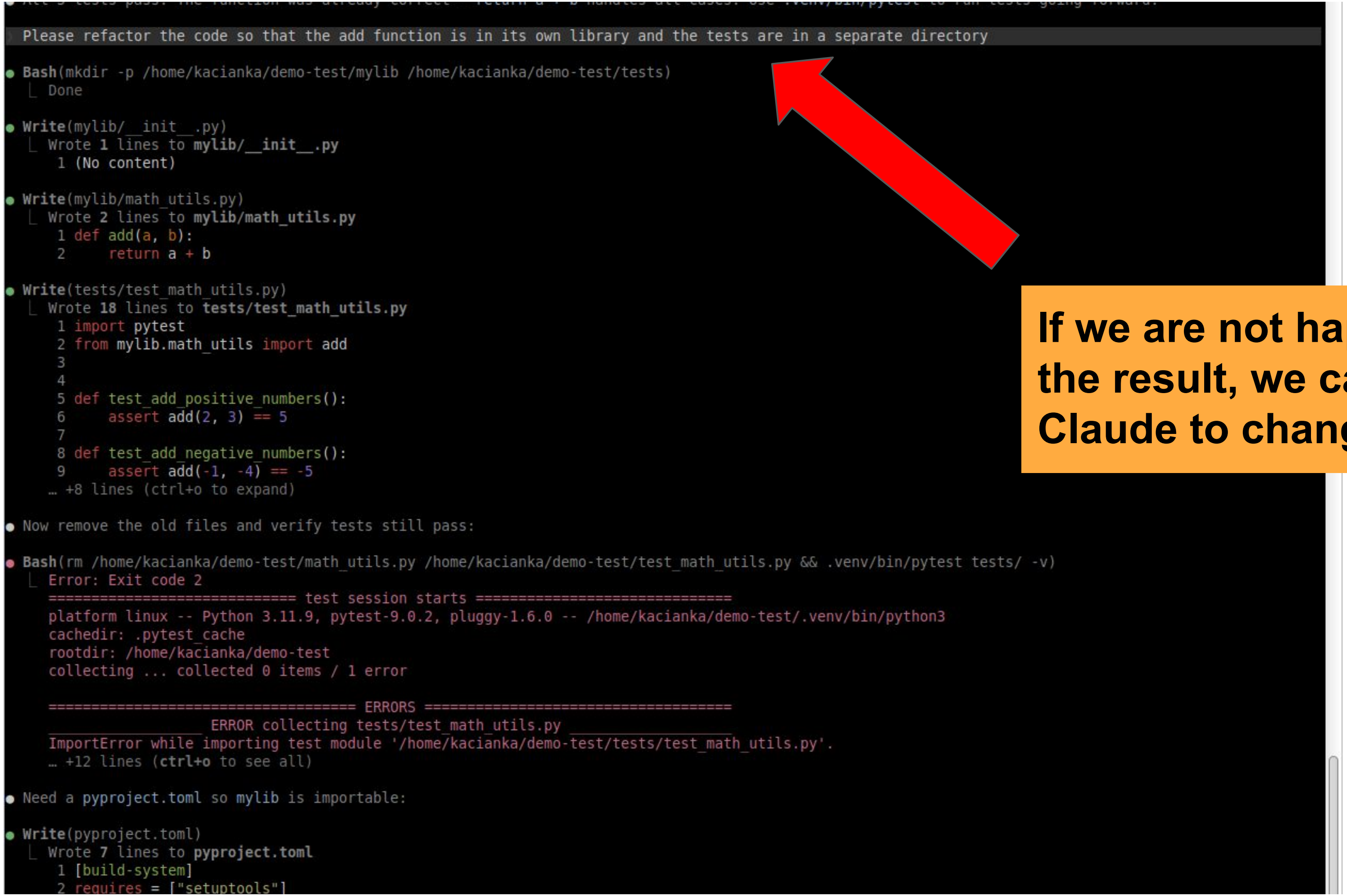
```
> Write tests for a function that adds two numbers

● I'll create the function and its tests.

● Write(math_utils.py)
  └ Wrote 2 lines to math_utils.py
      def add(a, b):
          return a + b

● Write(test_math_utils.py)
  └ Wrote 18 lines to test_math_utils.py
      import pytest
      from math_utils import add


      def test_add_positive_numbers():
          assert add(2, 3) == 5

      def test_add_negative_numbers():
          assert add(-1, -4) == -5
    … +8 lines (ctrl+o to expand)

● Created math_utils.py with the add function and test_math_utils.py with 5 tests covering positive numbers, negative numbers, mixed signs, zeros, and
  floats.

  Run with:
  pytest test_math_utils.py

────────────────────────────────────────────────────────────────
>
────────────────────────────────────────────────────────────────
  ›› accept edits on (shift+tab to cycle)
```

Again: files are created



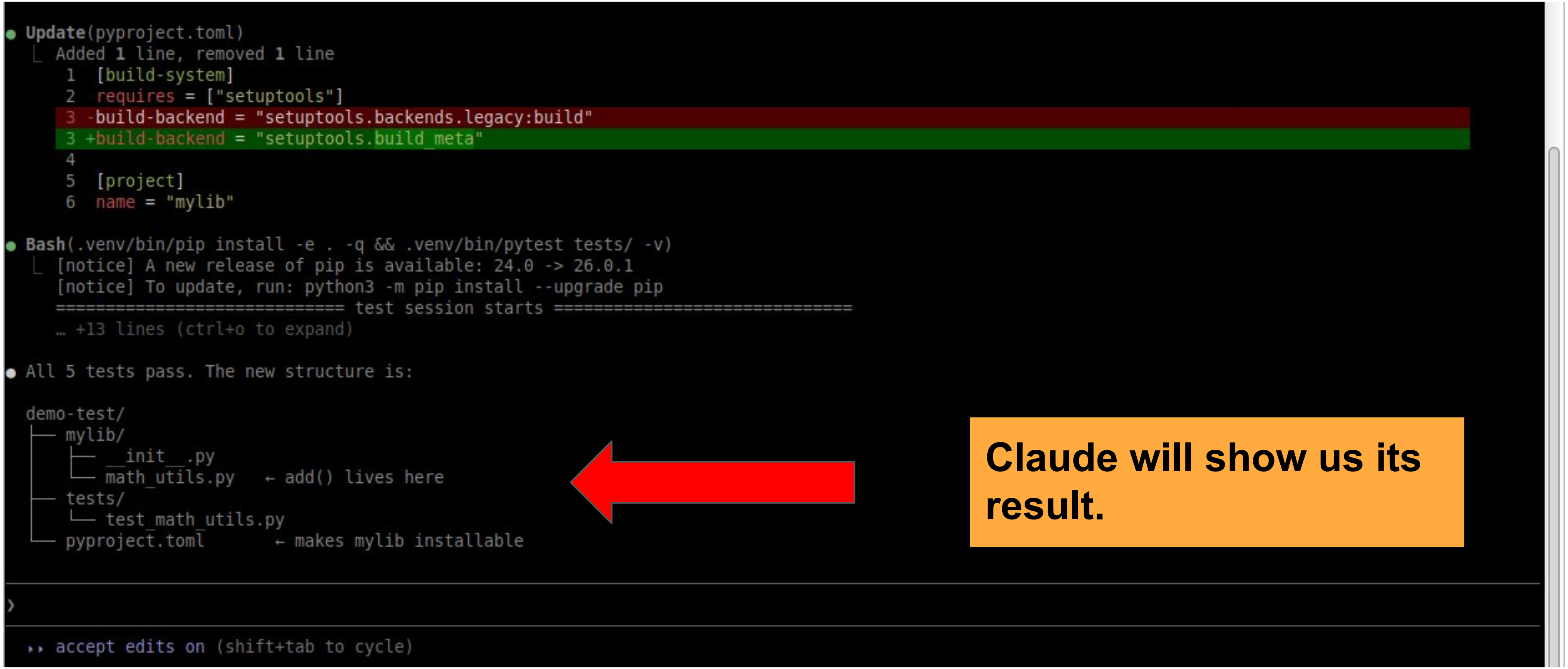
```
> implement the function so tests pass

● The add function is already implemented in math_utils.py. Let me verify the tests pass:

● Bash(python -m pytest test_math_utils.py -v)
  └ Error: Exit code 1
    /usr/bin/python: No module named pytest

● Bash(pip install pytest -q && python -m pytest test_math_utils.py -v)
  └ Error: Exit code 1
    error: externally-managed-environment

    × This environment is externally managed
    ╰─> To install Python packages system-wide, try apt install
        python3-xyz, where xyz is the package you are trying to
        install.

        If you wish to install a non-Debian-packaged Python package,
        create a virtual environment using python3 -m venv path/to/venv.
    … +11 lines (ctrl+o to see all)

● Bash(python3 -m venv .venv && .venv/bin/pip install pytest -q && .venv/bin/pytest test_math_utils.py -v)
  └ [notice] A new release of pip is available: 24.0 -> 26.0.1
    [notice] To update, run: python3 -m pip install --upgrade pip
    ============================= test session starts ==============================
    … +12 lines (ctrl+o to expand)

● All 5 tests pass. The function was already correct — return a + b handles all cases. Use .venv/bin/pytest to run tests going forward.

>

  ›› accept edits on (shift+tab to cycle)
```

Often you do get helpful advice



```
> Please refactor the code so that the add function is in its own library and the tests are in a separate directory

● Bash(mkdir -p /home/kacianka/demo-test/mylib /home/kacianka/demo-test/tests)
  ⎿ Done

● Write(mylib/__init__.py)
  ⎿ Wrote 1 lines to mylib/__init__.py
      (No content)

● Write(mylib/math_utils.py)
  ⎿ Wrote 2 lines to mylib/math_utils.py
      def add(a, b):
          return a + b

● Write(tests/test_math_utils.py)
  ⎿ Wrote 18 lines to tests/test_math_utils.py
      import pytest
      from mylib.math_utils import add


      def test_add_positive_numbers():
          assert add(2, 3) == 5

      def test_add_negative_numbers():
          assert add(-1, -4) == -5
     … +8 lines (ctrl+o to expand)

● Now remove the old files and verify tests still pass:

● Bash(rm /home/kacianka/demo-test/math_utils.py /home/kacianka/demo-test/test_math_utils.py && .venv/bin/pytest tests/ -v)
  ⎿ Error: Exit code 2
     ============================= test session starts ==============================
     platform linux -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0 -- /home/kacianka/demo-test/.venv/bin/python3
     cachedir: .pytest_cache
     rootdir: /home/kacianka/demo-test
     collecting ... collected 0 items / 1 error

     ==================================== ERRORS ====================================
     ___________________ ERROR collecting tests/test_math_utils.py ___________________
     ImportError while importing test module '/home/kacianka/demo-test/tests/test_math_utils.py'.
     … +12 lines (ctrl+o to see all)

● Need a pyproject.toml so mylib is importable:

● Write(pyproject.toml)
  ⎿ Wrote 7 lines to pyproject.toml
      [build-system]
      requires = ["setuptools"]
```

If we are not happy with the result, we can ask Claude to change it.

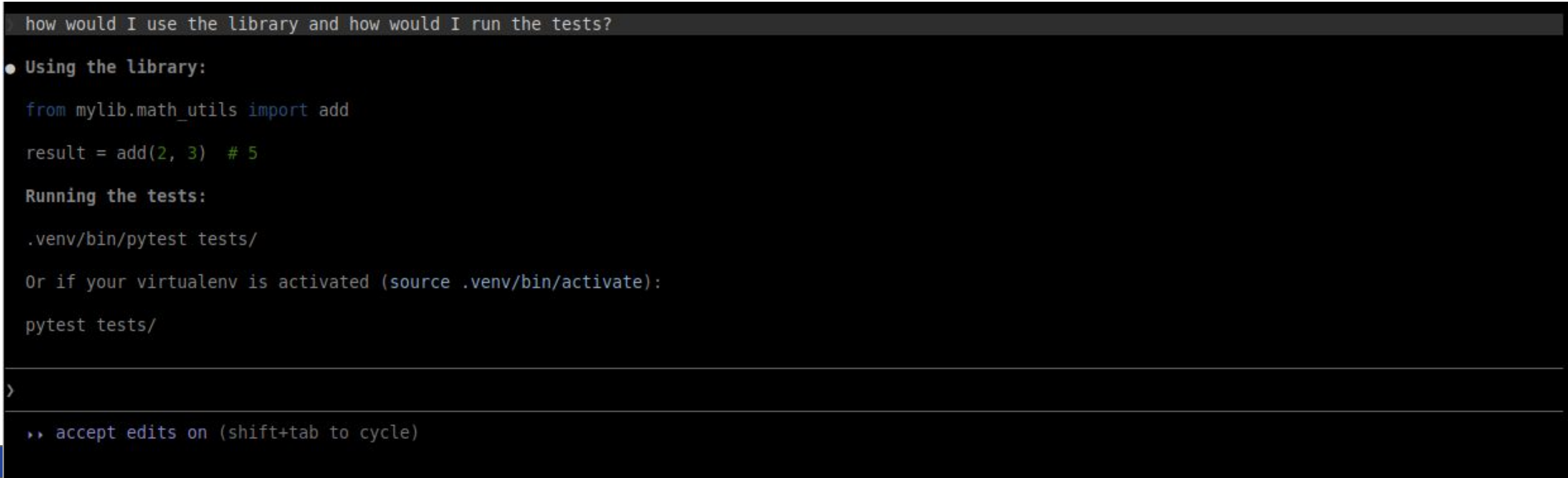


```
● Update(pyproject.toml)
  └ Added 1 line, removed 1 line
     1  [build-system]
     2  requires = ["setuptools"]
     3 -build-backend = "setuptools.backends.legacy:build"
     3 +build-backend = "setuptools.build_meta"
     4
     5  [project]
     6  name = "mylib"

● Bash(.venv/bin/pip install -e . -q && .venv/bin/pytest tests/ -v)
  └ [notice] A new release of pip is available: 24.0 -> 26.0.1
    [notice] To update, run: python3 -m pip install --upgrade pip
    ============================= test session starts ==============================
    … +13 lines (ctrl+o to expand)

● All 5 tests pass. The new structure is:

  demo-test/
  ├── mylib/
  │   ├── __init__.py
  │   └── math_utils.py    ← add() lives here
  ├── tests/
  │   └── test_math_utils.py
  └── pyproject.toml        ← makes mylib installable

>

  ›› accept edits on (shift+tab to cycle)
```

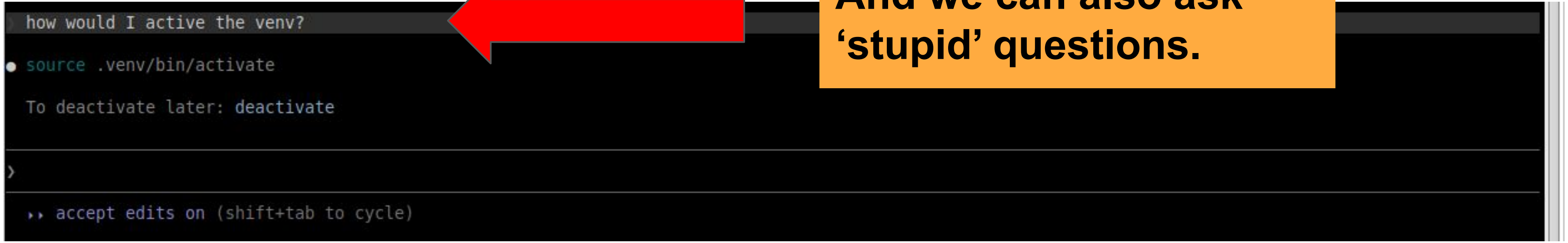
Claude will show us its result.

Claude can do complex operations

- Create files
- Create tests
- Follow complex project structures
- Explain how to use the code

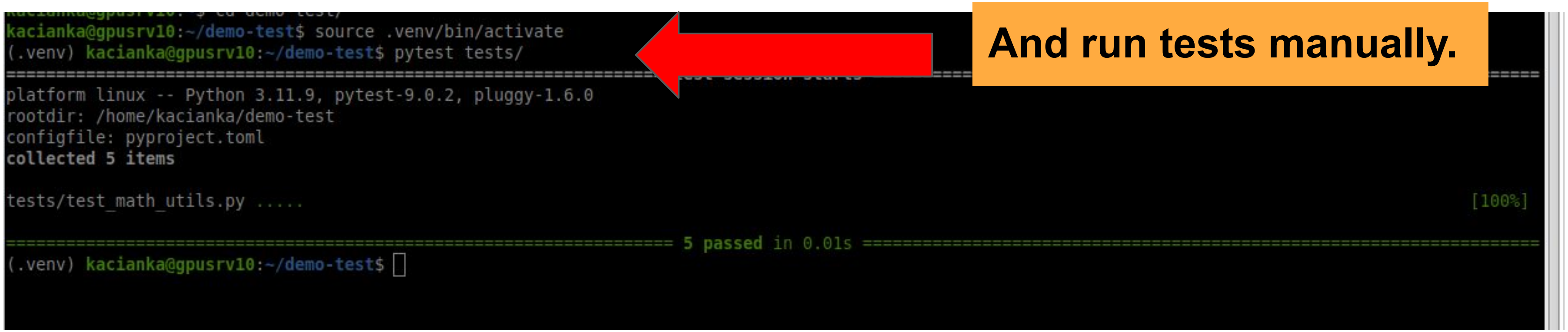


```
> how would I use the library and how would I run the tests?

● Using the library:

  from mylib.math_utils import add

  result = add(2, 3)  # 5

  Running the tests:

  .venv/bin/pytest tests/

  Or if your virtualenv is activated (source .venv/bin/activate):

  pytest tests/

>

  ⏵⏵ accept edits on (shift+tab to cycle)
```



```
how would I active the venv?

● source .venv/bin/activate

  To deactivate later: deactivate

────────────────────────────────────────────────────────────────
❯
────────────────────────────────────────────────────────────────
  ›› accept edits on (shift+tab to cycle)
```

And we can also ask ‘stupid’ questions.



```
kacianka@gpusrv10:~$ cd demo-test/
kacianka@gpusrv10:~/demo-test$ source .venv/bin/activate
(.venv) kacianka@gpusrv10:~/demo-test$ pytest tests/
================================================== test session starts ==================================================
platform linux -- Python 3.11.9, pytest-9.0.2, pluggy-1.6.0
rootdir: /home/kacianka/demo-test
configfile: pyproject.toml
collected 5 items

tests/test_math_utils.py .....                                                                                   [100%]

=================================================== 5 passed in 0.01s ===================================================
(.venv) kacianka@gpusrv10:~/demo-test$ 
```

And run tests manually.

Key Insight

- You define correctness
- Claude fills in implementation
- Tests are what make automation safe
- Necessary for using multiple agents in an automated manner!

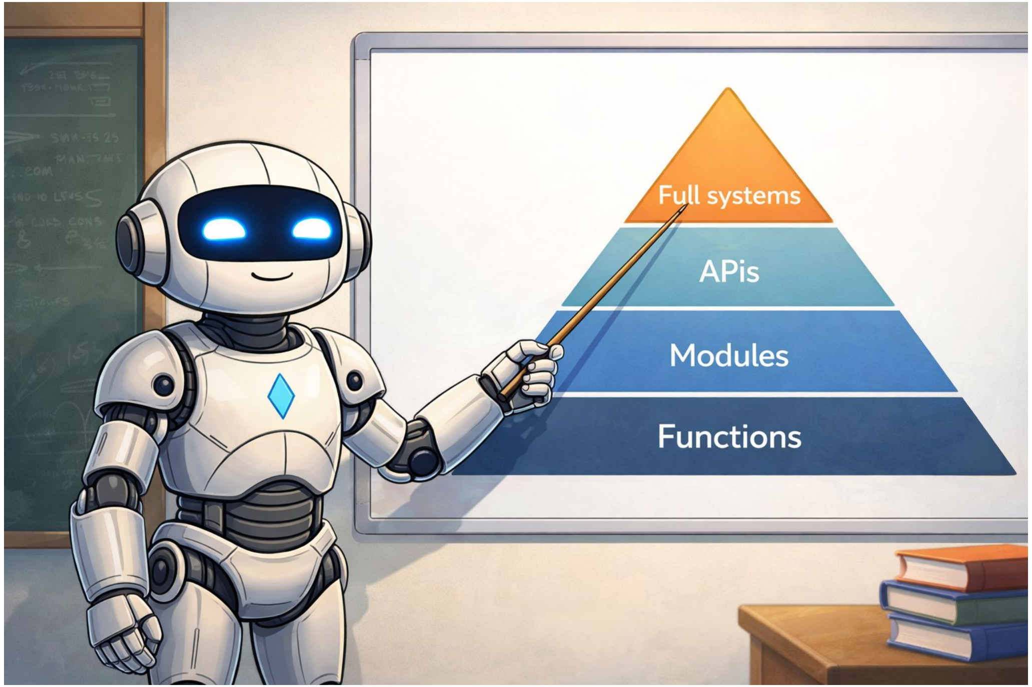
"Tests become the specification."

Code Generation

From Small to Large

Claude can generate:

- Functions
- Modules
- APIs
- Full systems



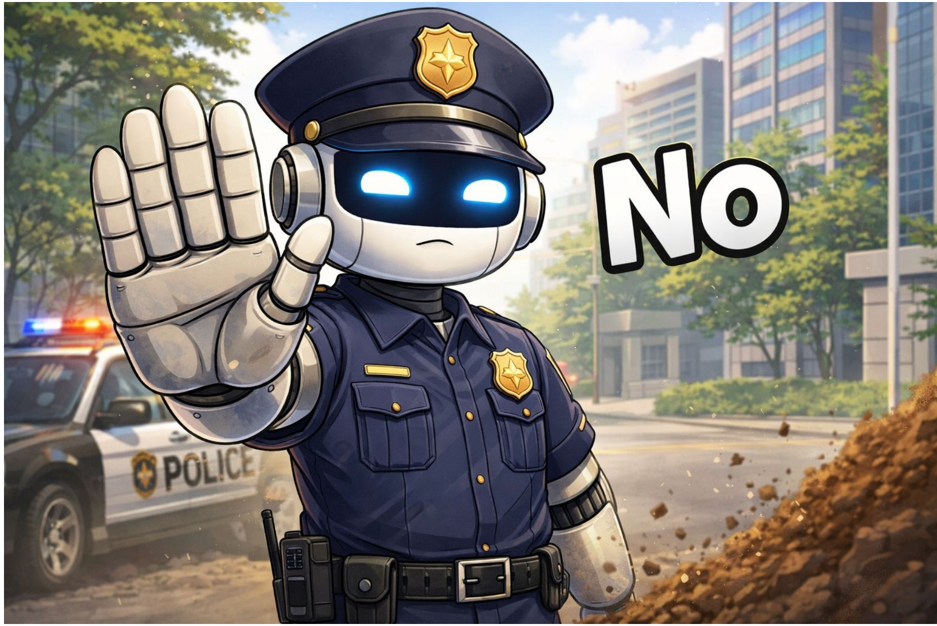
Best Practices

- Be explicit in prompts
- Break tasks into steps
- Very important:
  Ask Claude to create a plan first!
- Review the plan
- Validate each output

When in doubt: type roughly what you want and let Claude work. You might be surprised how good it even works with ambiguous prompts (but then carefully review what Claude did!)

Anti-Patterns

- Asking for everything at once
- Blindly trusting output
- Skipping review
- Longer prompts are not automatically better prompts



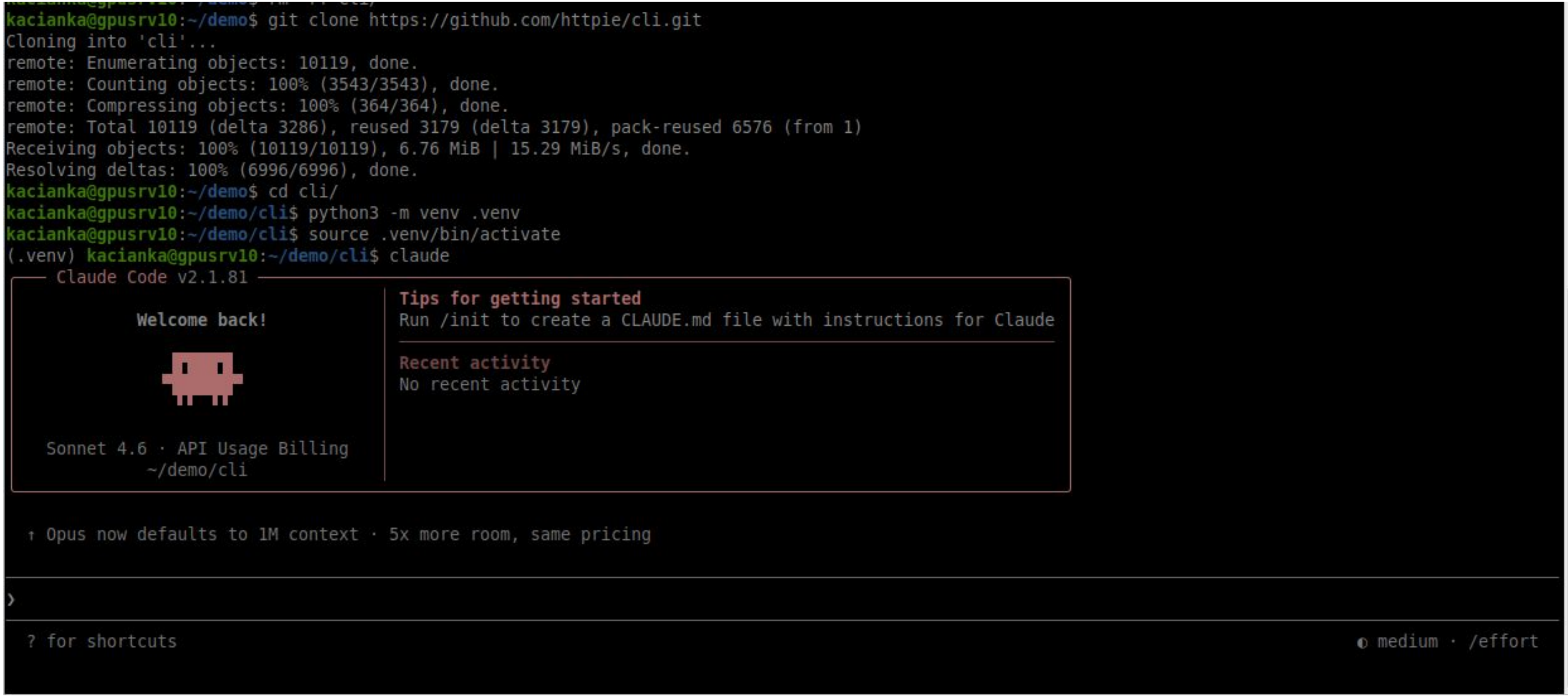
Example Workflow

- Define feature
- Generate tests
- Generate code
- Run CI
- Commit code
- Review + refine

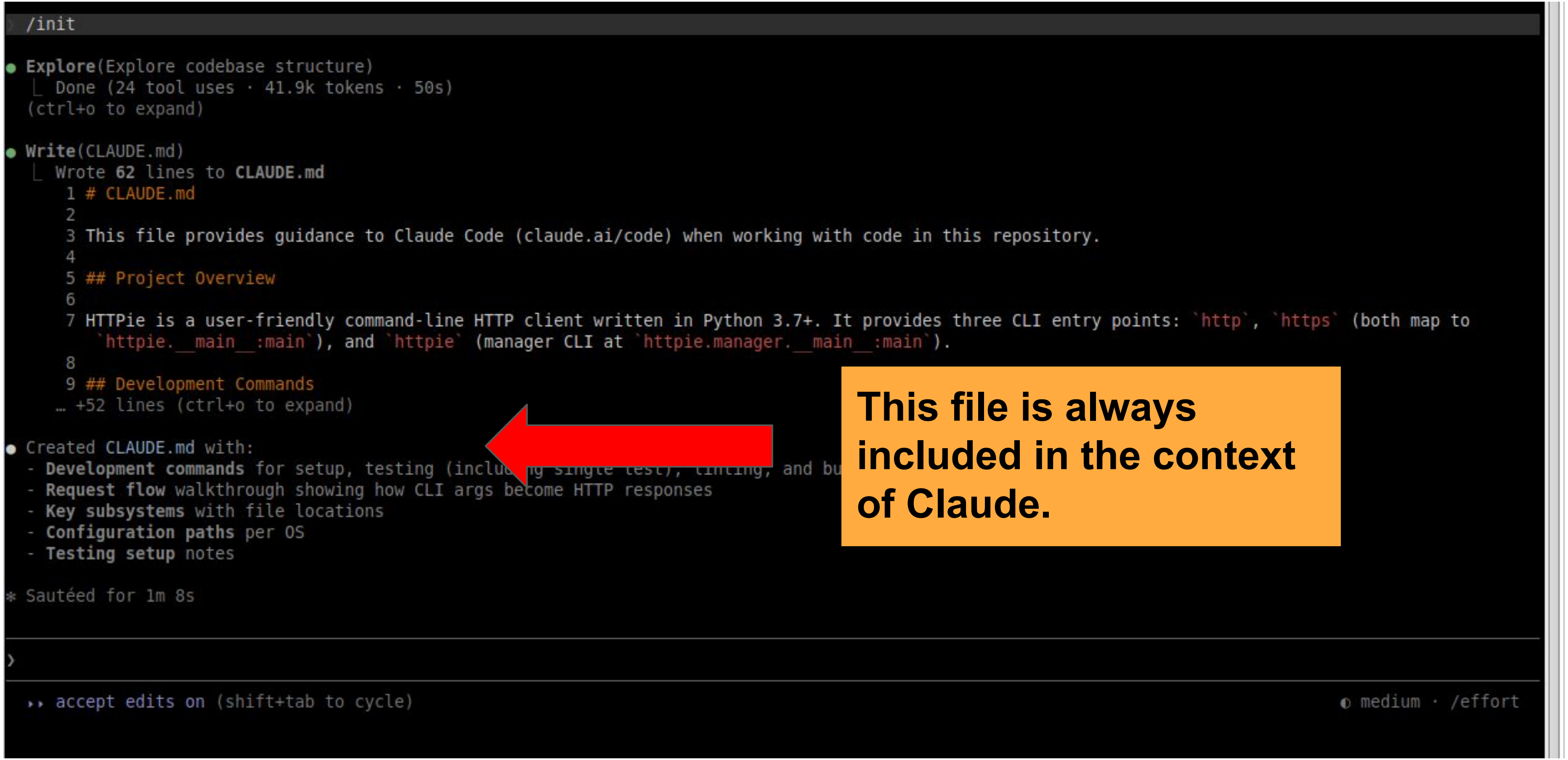
Example in a bigger project

Claude in real projects

- We use an existing open-source project: https://github.com/httpie/cli
- Use Claude to understand an unknown repository
- Create a feature that outputs the HTML website as JSON
- Then expand into creating a CLI-based web browser

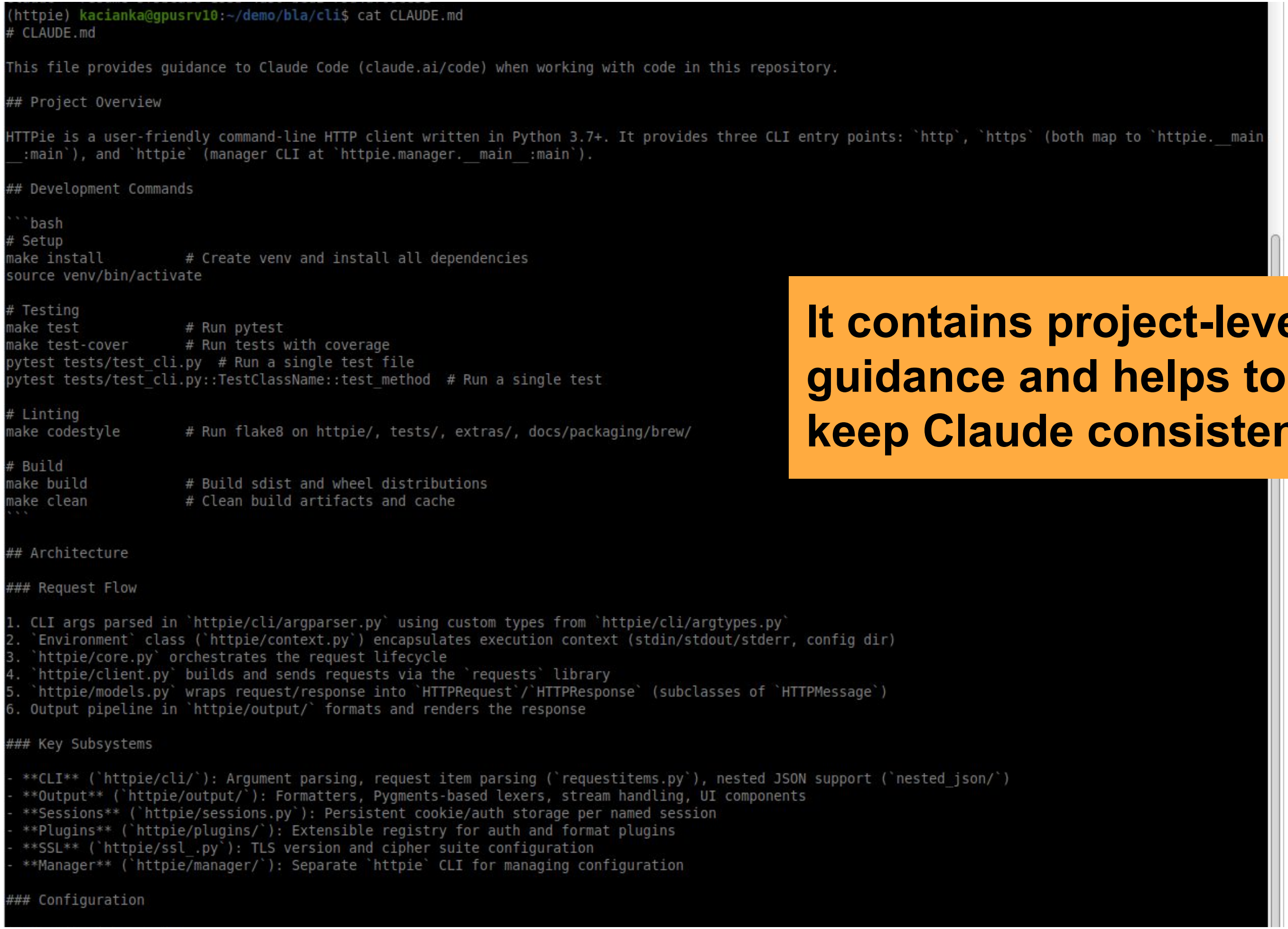


```
kacianka@gpusrv10:~/demo$ git clone https://github.com/httpie/cli.git
Cloning into 'cli'...
remote: Enumerating objects: 10119, done.
remote: Counting objects: 100% (3543/3543), done.
remote: Compressing objects: 100% (364/364), done.
remote: Total 10119 (delta 3286), reused 3179 (delta 3179), pack-reused 6576 (from 1)
Receiving objects: 100% (10119/10119), 6.76 MiB | 15.29 MiB/s, done.
Resolving deltas: 100% (6996/6996), done.
kacianka@gpusrv10:~/demo$ cd cli/
kacianka@gpusrv10:~/demo/cli$ python3 -m venv .venv
kacianka@gpusrv10:~/demo/cli$ source .venv/bin/activate
(.venv) kacianka@gpusrv10:~/demo/cli$ claude
 ── Claude Code v2.1.81 ──────────────────────────────────────────────────────────────
│                                    │ Tips for getting started                         │
│            Welcome back!           │ Run /init to create a CLAUDE.md file with instructions for Claude │
│                                    │ ──────────────────────────────────────────────── │
│                                    │ Recent activity                                  │
│                                    │ No recent activity                               │
│                                    │                                                  │
│                                    │                                                  │
│   Sonnet 4.6 · API Usage Billing   │                                                  │
│             ~/demo/cli             │                                                  │
 ─────────────────────────────────────────────────────────────────────────────────────

  ↑ Opus now defaults to 1M context · 5x more room, same pricing

────────────────────────────────────────────────────────────────────────────────────────────
❯
────────────────────────────────────────────────────────────────────────────────────────────
  ? for shortcuts                                                          ◐ medium · /effort
```



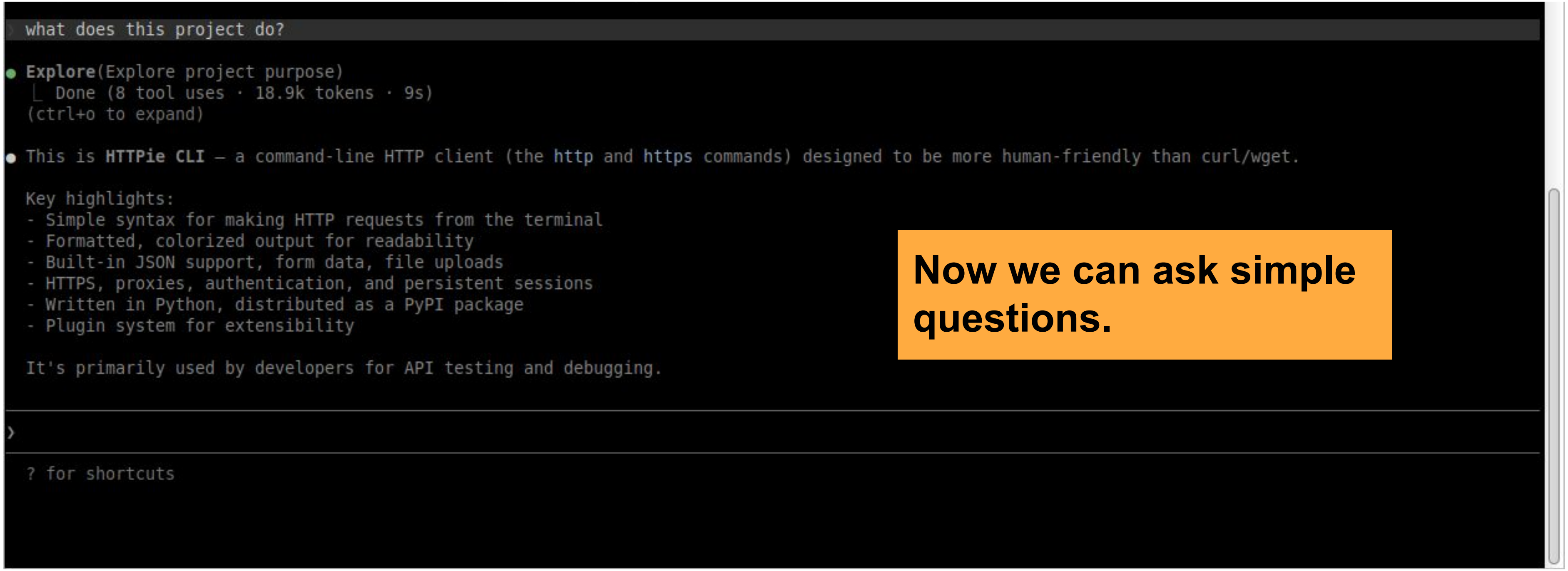
```
> /init

● Explore(Explore codebase structure)
  ⎿  Done (24 tool uses · 41.9k tokens · 50s)
  (ctrl+o to expand)

● Write(CLAUDE.md)
  ⎿  Wrote 62 lines to CLAUDE.md
      1 # CLAUDE.md
      2
      3 This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.
      4
      5 ## Project Overview
      6
      7 HTTPie is a user-friendly command-line HTTP client written in Python 3.7+. It provides three CLI entry points: `http`, `https` (both map to
         `httpie.__main__:main`), and `httpie` (manager CLI at `httpie.manager.__main__:main`).
      8
      9 ## Development Commands
     … +52 lines (ctrl+o to expand)

● Created CLAUDE.md with:
  - Development commands for setup, testing (inclu ng single test), linting, and bu
  - Request flow walkthrough showing how CLI args become HTTP responses
  - Key subsystems with file locations
  - Configuration paths per OS
  - Testing setup notes

* Sautéed for 1m 8s

>

  ⏵⏵ accept edits on (shift+tab to cycle)                                                                           ◐ medium · /effort
```

This file is always included in the context of Claude.



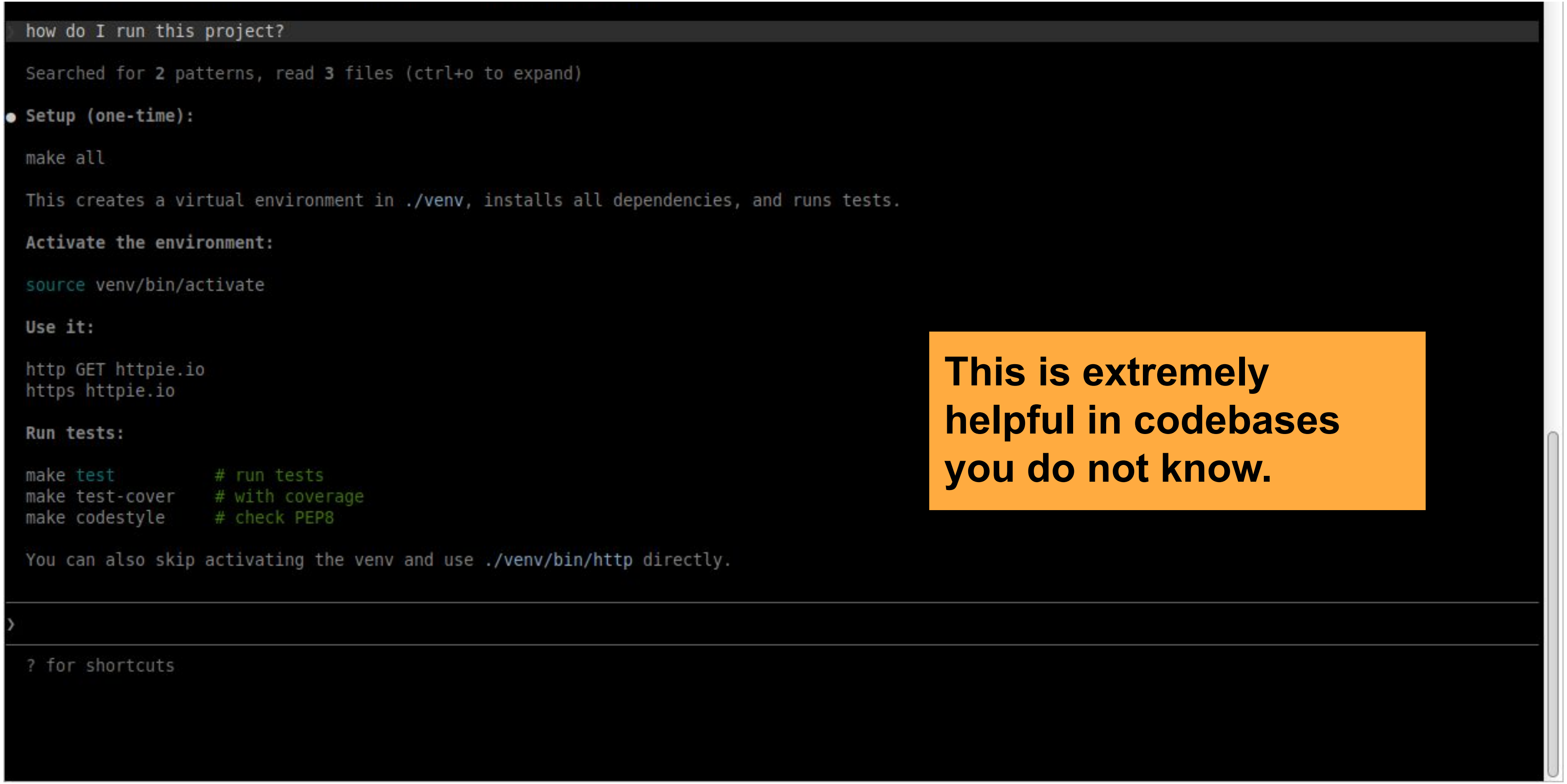
~~~
(httpie) kacianka@gpusrv10:~/demo/bla/cli$ cat CLAUDE.md
# CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

## Project Overview

HTTPie is a user-friendly command-line HTTP client written in Python 3.7+. It provides three CLI entry points: `http`, `https` (both map to `httpie.__main
__:main`), and `httpie` (manager CLI at `httpie.manager.__main__:main`).

## Development Commands

```bash
# Setup
make install          # Create venv and install all dependencies
source venv/bin/activate

# Testing
make test             # Run pytest
make test-cover       # Run tests with coverage
pytest tests/test_cli.py  # Run a single test file
pytest tests/test_cli.py::TestClassName::test_method  # Run a single test

# Linting
make codestyle        # Run flake8 on httpie/, tests/, extras/, docs/packaging/brew/

# Build
make build            # Build sdist and wheel distributions
make clean            # Clean build artifacts and cache
```

## Architecture

### Request Flow

1. CLI args parsed in `httpie/cli/argparser.py` using custom types from `httpie/cli/argtypes.py`
2. `Environment` class (`httpie/context.py`) encapsulates execution context (stdin/stdout/stderr, config dir)
3. `httpie/core.py` orchestrates the request lifecycle
4. `httpie/client.py` builds and sends requests via the `requests` library
5. `httpie/models.py` wraps request/response into `HTTPRequest`/`HTTPResponse` (subclasses of `HTTPMessage`)
6. Output pipeline in `httpie/output/` formats and renders the response

### Key Subsystems

- **CLI** (`httpie/cli/`): Argument parsing, request item parsing (`requestitems.py`), nested JSON support (`nested_json/`)
- **Output** (`httpie/output/`): Formatters, Pygments-based lexers, stream handling, UI components
- **Sessions** (`httpie/sessions.py`): Persistent cookie/auth storage per named session
- **Plugins** (`httpie/plugins/`): Extensible registry for auth and format plugins
- **SSL** (`httpie/ssl_.py`): TLS version and cipher suite configuration
- **Manager** (`httpie/manager/`): Separate `httpie` CLI for managing configuration

### Configuration
~~~

It contains project-level guidance and helps to keep Claude consistent



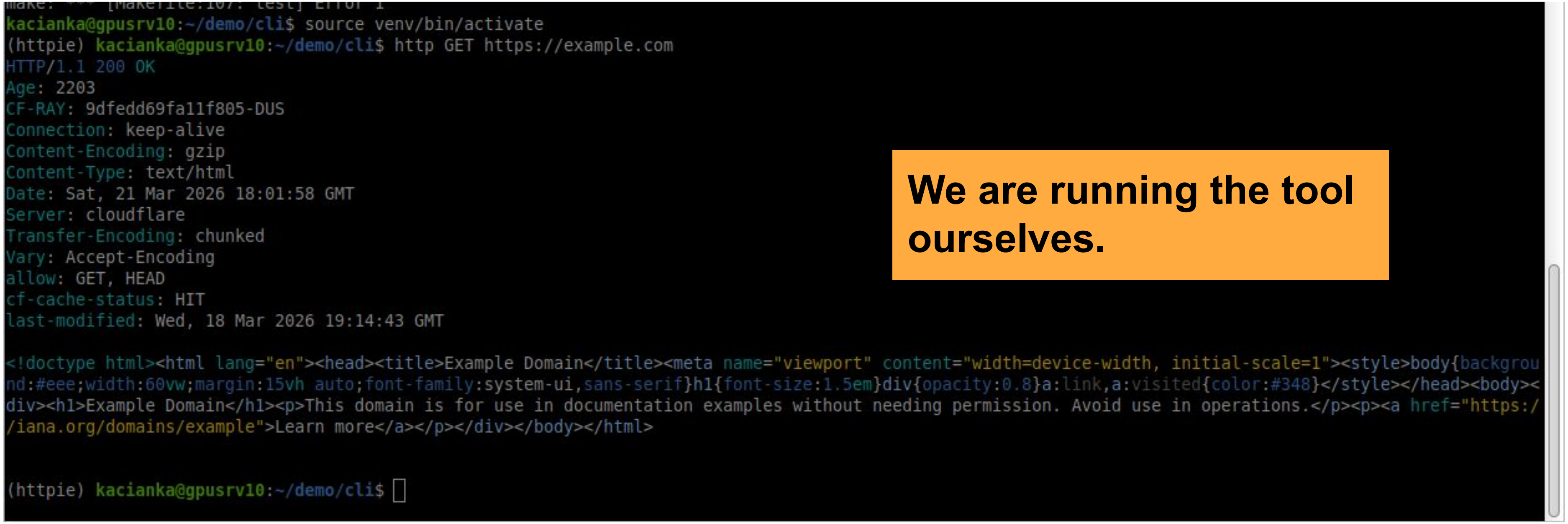
```
> what does this project do?

● Explore(Explore project purpose)
  └ Done (8 tool uses · 18.9k tokens · 9s)
(ctrl+o to expand)

● This is HTTPie CLI — a command-line HTTP client (the http and https commands) designed to be more human-friendly than curl/wget.

  Key highlights:
  - Simple syntax for making HTTP requests from the terminal
  - Formatted, colorized output for readability
  - Built-in JSON support, form data, file uploads
  - HTTPS, proxies, authentication, and persistent sessions
  - Written in Python, distributed as a PyPI package
  - Plugin system for extensibility

  It's primarily used by developers for API testing and debugging.

>

  ? for shortcuts
```

Now we can ask simple questions.



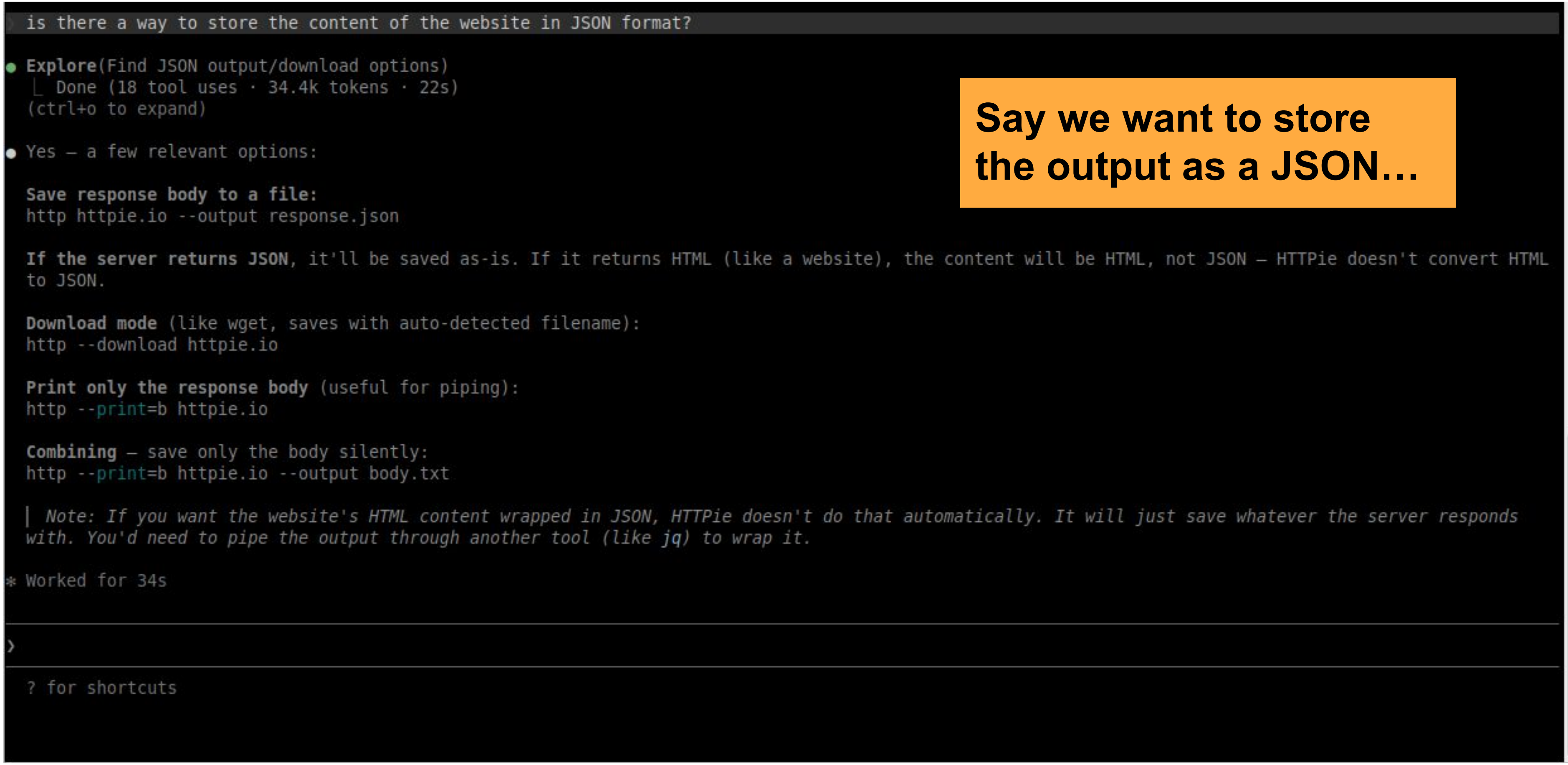
```
> how do I run this project?

  Searched for 2 patterns, read 3 files (ctrl+o to expand)

● Setup (one-time):

  make all

  This creates a virtual environment in ./venv, installs all dependencies, and runs tests.

  Activate the environment:

  source venv/bin/activate

  Use it:

  http GET httpie.io
  https httpie.io

  Run tests:

  make test           # run tests
  make test-cover     # with coverage
  make codestyle      # check PEP8

  You can also skip activating the venv and use ./venv/bin/http directly.

>

  ? for shortcuts
```

This is extremely helpful in codebases you do not know.



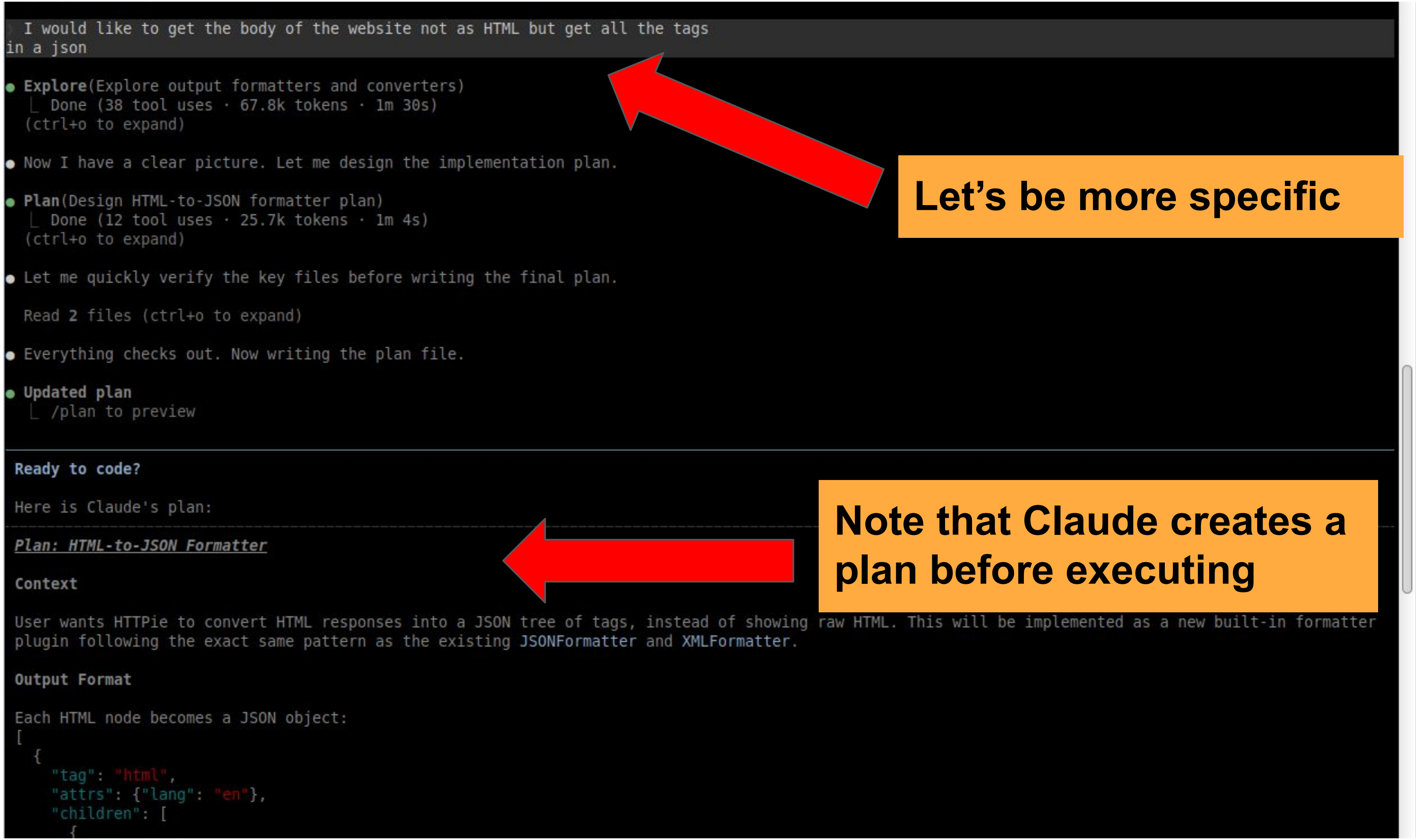
```
make: *** [Makefile:107: test] Error 1
kacianka@gpusrv10:~/demo/cli$ source venv/bin/activate
(httpie) kacianka@gpusrv10:~/demo/cli$ http GET https://example.com
HTTP/1.1 200 OK
Age: 2203
CF-RAY: 9dfedd69fa11f805-DUS
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html
Date: Sat, 21 Mar 2026 18:01:58 GMT
Server: cloudflare
Transfer-Encoding: chunked
Vary: Accept-Encoding
allow: GET, HEAD
cf-cache-status: HIT
last-modified: Wed, 18 Mar 2026 19:14:43 GMT

<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{backgrou
nd:#eee;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}</style></head><body><
div><h1>Example Domain</h1><p>This domain is for use in documentation examples without needing permission. Avoid use in operations.</p><p><a href="https:/
/iana.org/domains/example">Learn more</a></p></div></body></html>


(httpie) kacianka@gpusrv10:~/demo/cli$ 
```

We are running the tool ourselves.



```
> is there a way to store the content of the website in JSON format?

● Explore(Find JSON output/download options)
  ⎿  Done (18 tool uses · 34.4k tokens · 22s)
  (ctrl+o to expand)

● Yes — a few relevant options:

  Save response body to a file:
  http httpie.io --output response.json

  If the server returns JSON, it'll be saved as-is. If it returns HTML (like a website), the content will be HTML, not JSON — HTTPie doesn't convert HTML
  to JSON.

  Download mode (like wget, saves with auto-detected filename):
  http --download httpie.io

  Print only the response body (useful for piping):
  http --print=b httpie.io

  Combining — save only the body silently:
  http --print=b httpie.io --output body.txt

  | Note: If you want the website's HTML content wrapped in JSON, HTTPie doesn't do that automatically. It will just save whatever the server responds
  with. You'd need to pipe the output through another tool (like jq) to wrap it.

* Worked for 34s

────────────────────────────────────────────────────────────────────────────────
>
────────────────────────────────────────────────────────────────────────────────
  ? for shortcuts
```

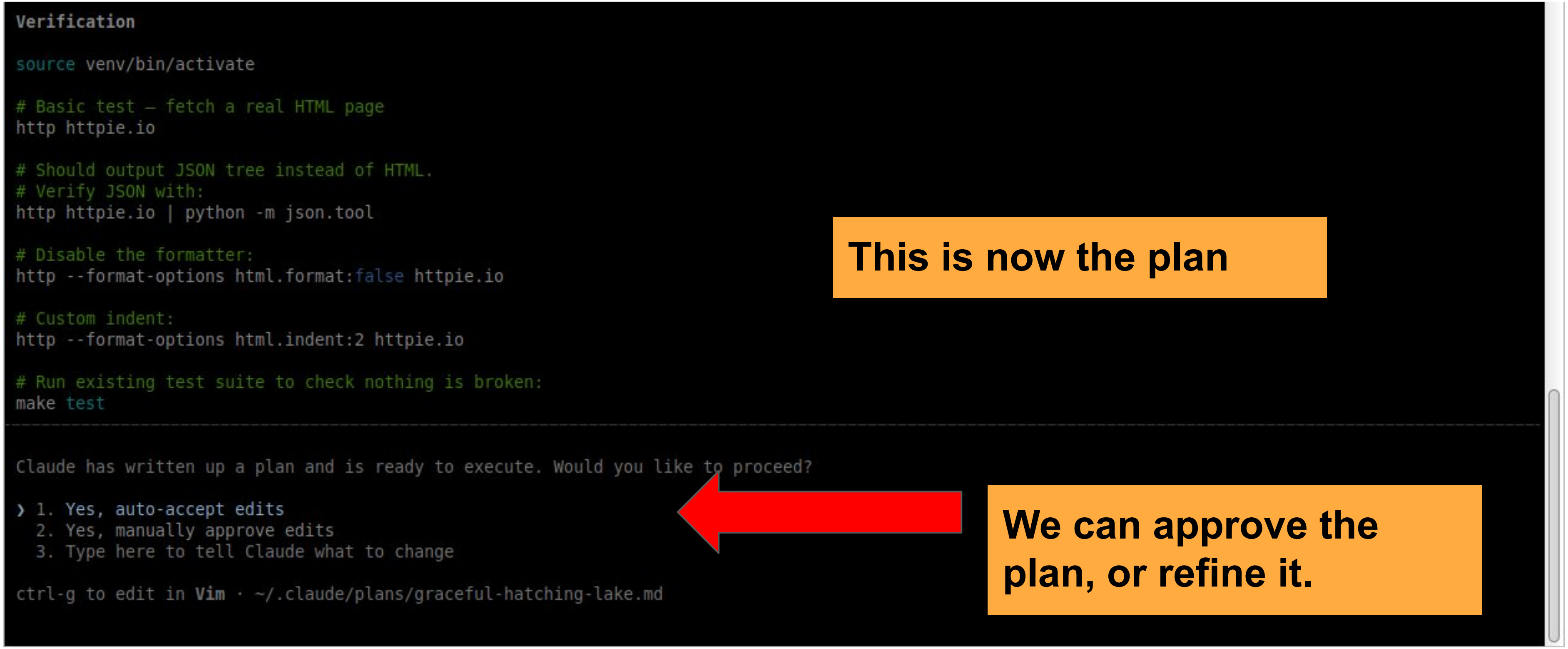
Say we want to store the output as a JSON…



```
> I would like to get the body of the website not as HTML but get all the tags
in a json

● Explore(Explore output formatters and converters)
  ⎿ Done (38 tool uses · 67.8k tokens · 1m 30s)
  (ctrl+o to expand)

● Now I have a clear picture. Let me design the implementation plan.

● Plan(Design HTML-to-JSON formatter plan)
  ⎿ Done (12 tool uses · 25.7k tokens · 1m 4s)
  (ctrl+o to expand)

● Let me quickly verify the key files before writing the final plan.

  Read 2 files (ctrl+o to expand)

● Everything checks out. Now writing the plan file.

● Updated plan
  ⎿ /plan to preview

 Ready to code?

 Here is Claude's plan:

 Plan: HTML-to-JSON Formatter

 Context

 User wants HTTPie to convert HTML responses into a JSON tree of tags, instead of showing raw HTML. This will be implemented as a new built-in formatter
 plugin following the exact same pattern as the existing JSONFormatter and XMLFormatter.

 Output Format

 Each HTML node becomes a JSON object:
 [
   {
     "tag": "html",
     "attrs": {"lang": "en"},
     "children": [
       {
```

Let’s be more specific

Note that Claude creates a plan before executing

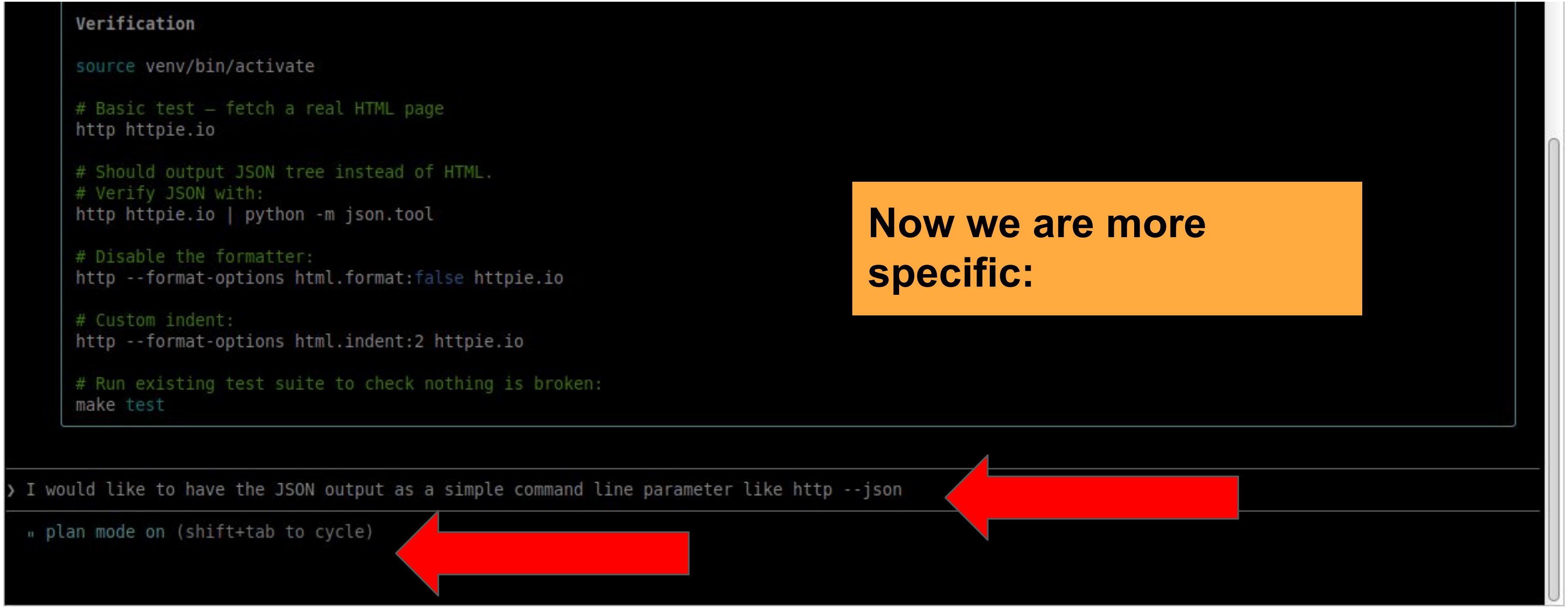


```
Verification

source venv/bin/activate

# Basic test — fetch a real HTML page
http httpie.io

# Should output JSON tree instead of HTML.
# Verify JSON with:
http httpie.io | python -m json.tool

# Disable the formatter:
http --format-options html.format:false httpie.io

# Custom indent:
http --format-options html.indent:2 httpie.io

# Run existing test suite to check nothing is broken:
make test
--------------------------------------------------------------------------------

Claude has written up a plan and is ready to execute. Would you like to proceed?

❯ 1. Yes, auto-accept edits
  2. Yes, manually approve edits
  3. Type here to tell Claude what to change

ctrl-g to edit in Vim · ~/.claude/plans/graceful-hatching-lake.md
```

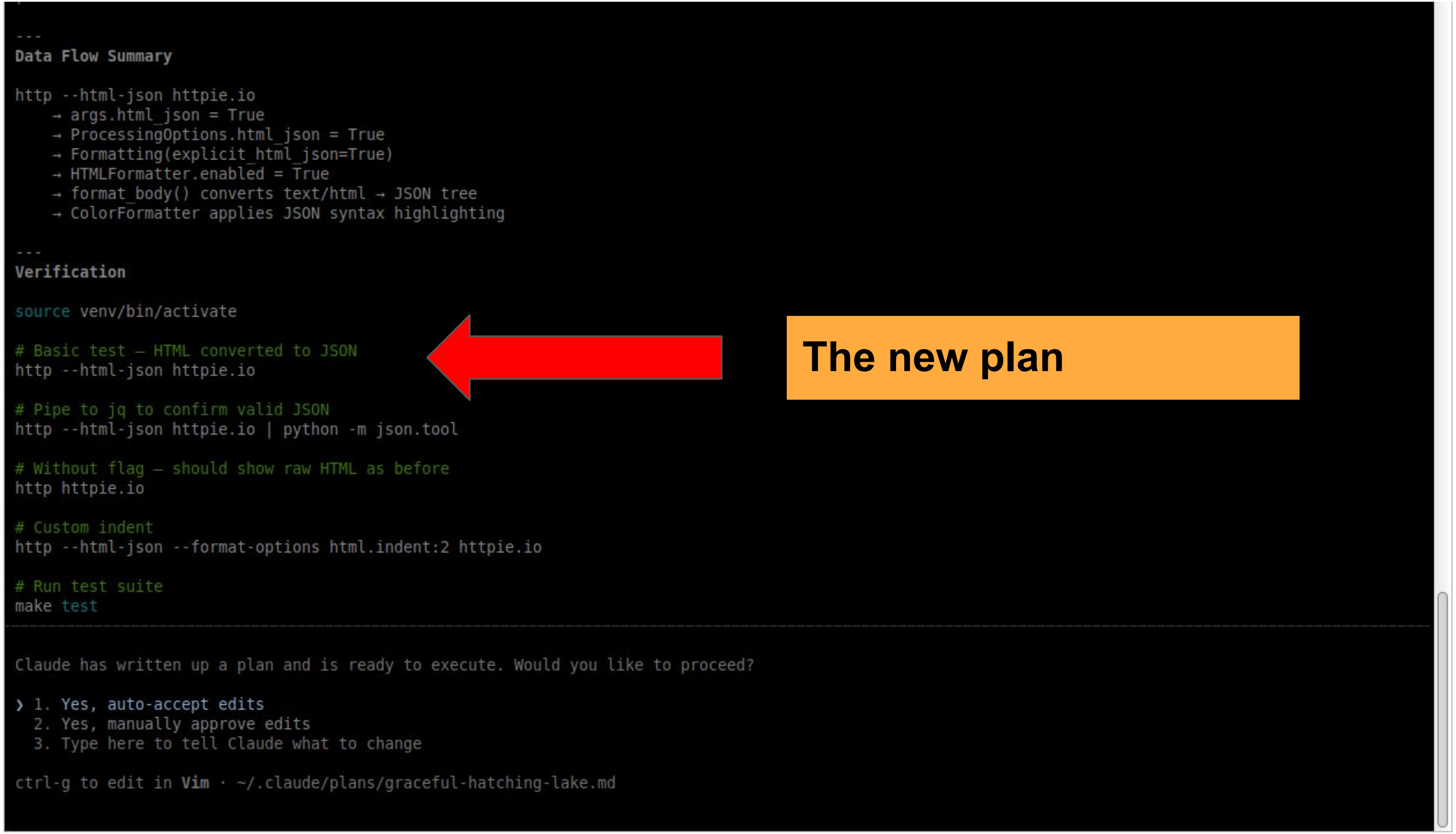
This is now the plan

We can approve the plan, or refine it.

Now we are more specific:

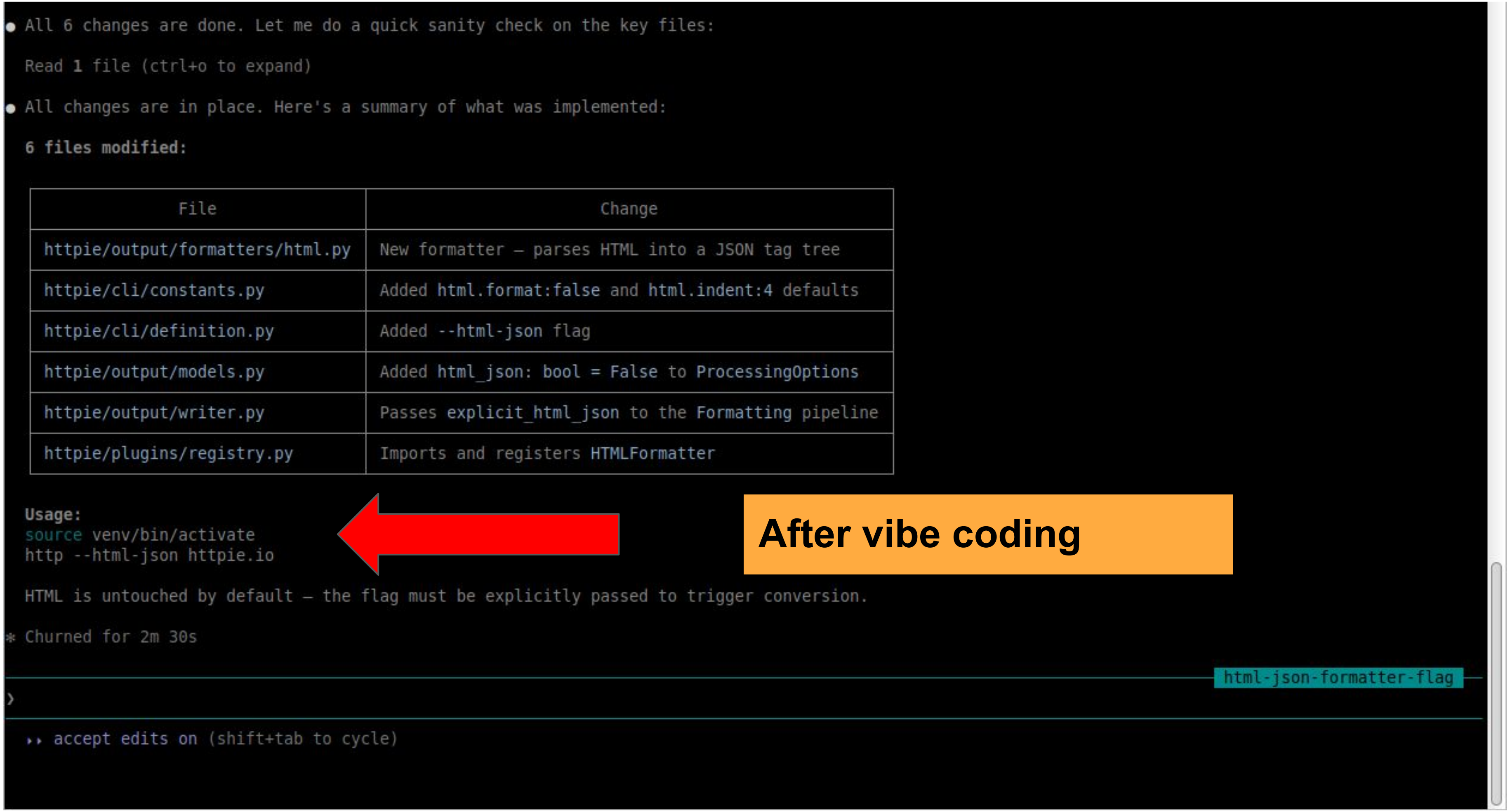


```
Verification

source venv/bin/activate

# Basic test — fetch a real HTML page
http httpie.io

# Should output JSON tree instead of HTML.
# Verify JSON with:
http httpie.io | python -m json.tool

# Disable the formatter:
http --format-options html.format:false httpie.io

# Custom indent:
http --format-options html.indent:2 httpie.io

# Run existing test suite to check nothing is broken:
make test

› I would like to have the JSON output as a simple command line parameter like http --json

  ⏸ plan mode on (shift+tab to cycle)
```



```
---
Data Flow Summary

http --html-json httpie.io
    → args.html_json = True
    → ProcessingOptions.html_json = True
    → Formatting(explicit_html_json=True)
    → HTMLFormatter.enabled = True
    → format_body() converts text/html → JSON tree
    → ColorFormatter applies JSON syntax highlighting

---
Verification

source venv/bin/activate

# Basic test — HTML converted to JSON
http --html-json httpie.io

# Pipe to jq to confirm valid JSON
http --html-json httpie.io | python -m json.tool

# Without flag — should show raw HTML as before
http httpie.io

# Custom indent
http --html-json --format-options html.indent:2 httpie.io

# Run test suite
make test

Claude has written up a plan and is ready to execute. Would you like to proceed?

❯ 1. Yes, auto-accept edits
  2. Yes, manually approve edits
  3. Type here to tell Claude what to change

ctrl-g to edit in Vim · ~/.claude/plans/graceful-hatching-lake.md
```

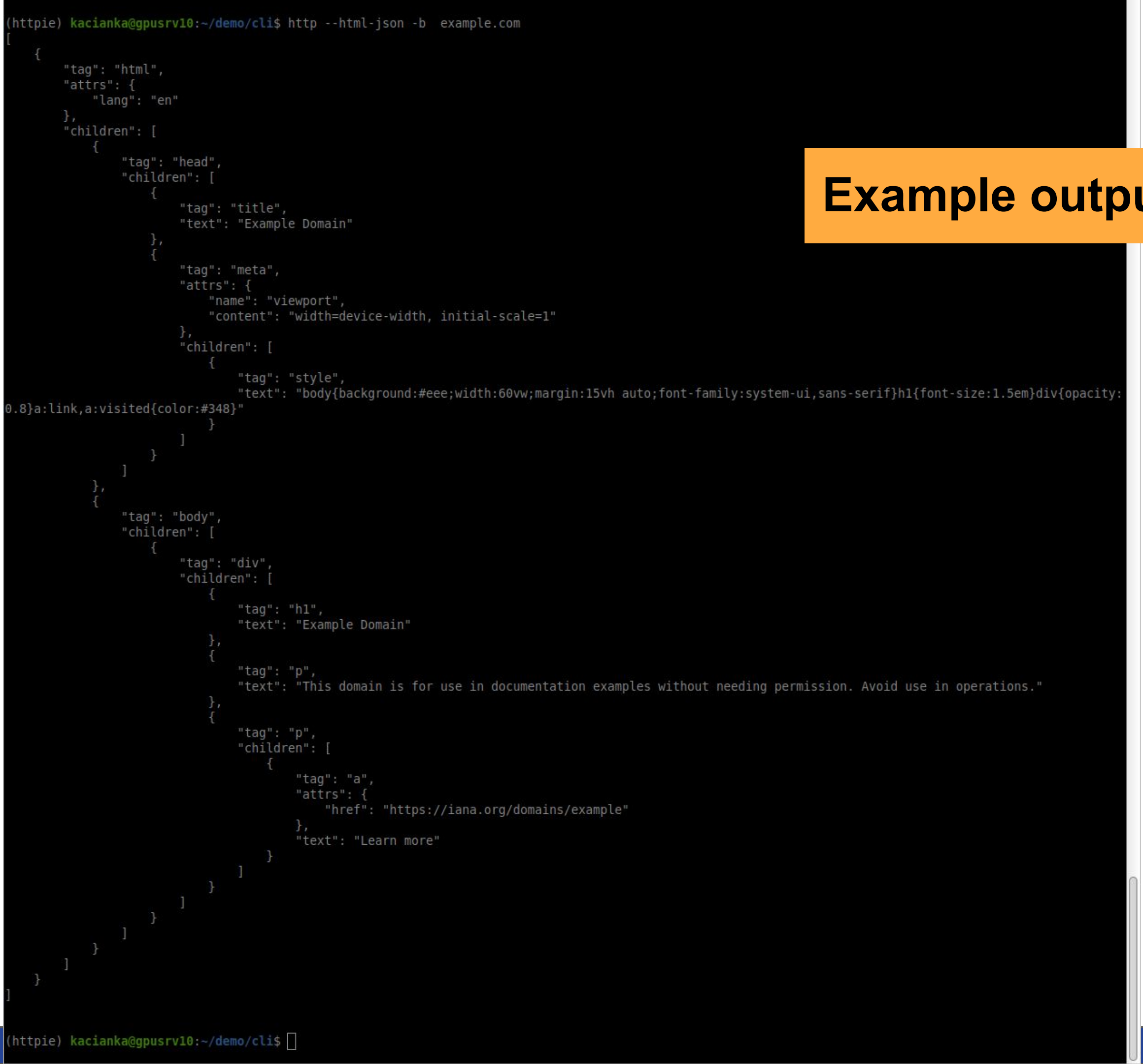
The new plan



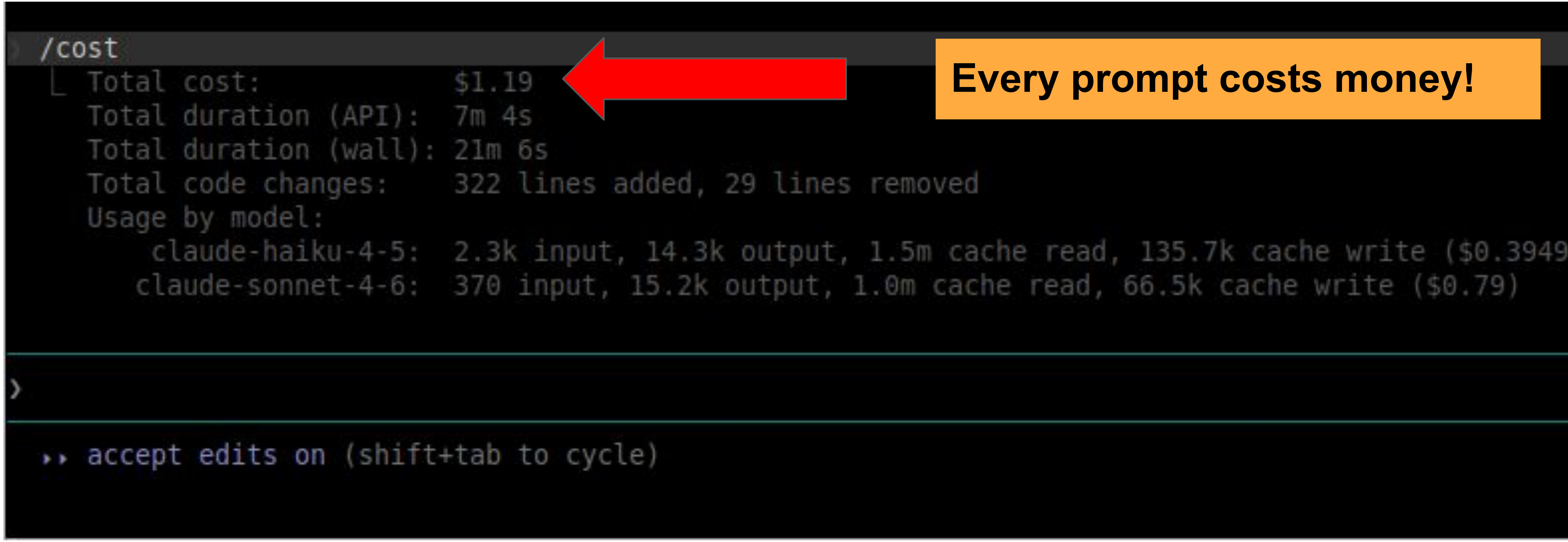
```
● All 6 changes are done. Let me do a quick sanity check on the key files:

  Read 1 file (ctrl+o to expand)

● All changes are in place. Here's a summary of what was implemented:

  6 files modified:

  ┌─────────────────────────────────┬──────────────────────────────────────────────────────┐
  │              File               │                        Change                        │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/output/formatters/html.py │ New formatter — parses HTML into a JSON tag tree     │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/cli/constants.py         │ Added html.format:false and html.indent:4 defaults   │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/cli/definition.py        │ Added --html-json flag                               │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/output/models.py         │ Added html_json: bool = False to ProcessingOptions   │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/output/writer.py         │ Passes explicit_html_json to the Formatting pipeline │
  ├─────────────────────────────────┼──────────────────────────────────────────────────────┤
  │ httpie/plugins/registry.py      │ Imports and registers HTMLFormatter                  │
  └─────────────────────────────────┴──────────────────────────────────────────────────────┘

  Usage:
  source venv/bin/activate
  http --html-json httpie.io

  HTML is untouched by default — the flag must be explicitly passed to trigger conversion.

* Churned for 2m 30s

──────────────────────────────────────────────────────────────── html-json-formatter-flag ──
›
────────────────────────────────────────────────────────────────────────────────────────────
  ⏵⏵ accept edits on (shift+tab to cycle)
```

After vibe coding

Example output



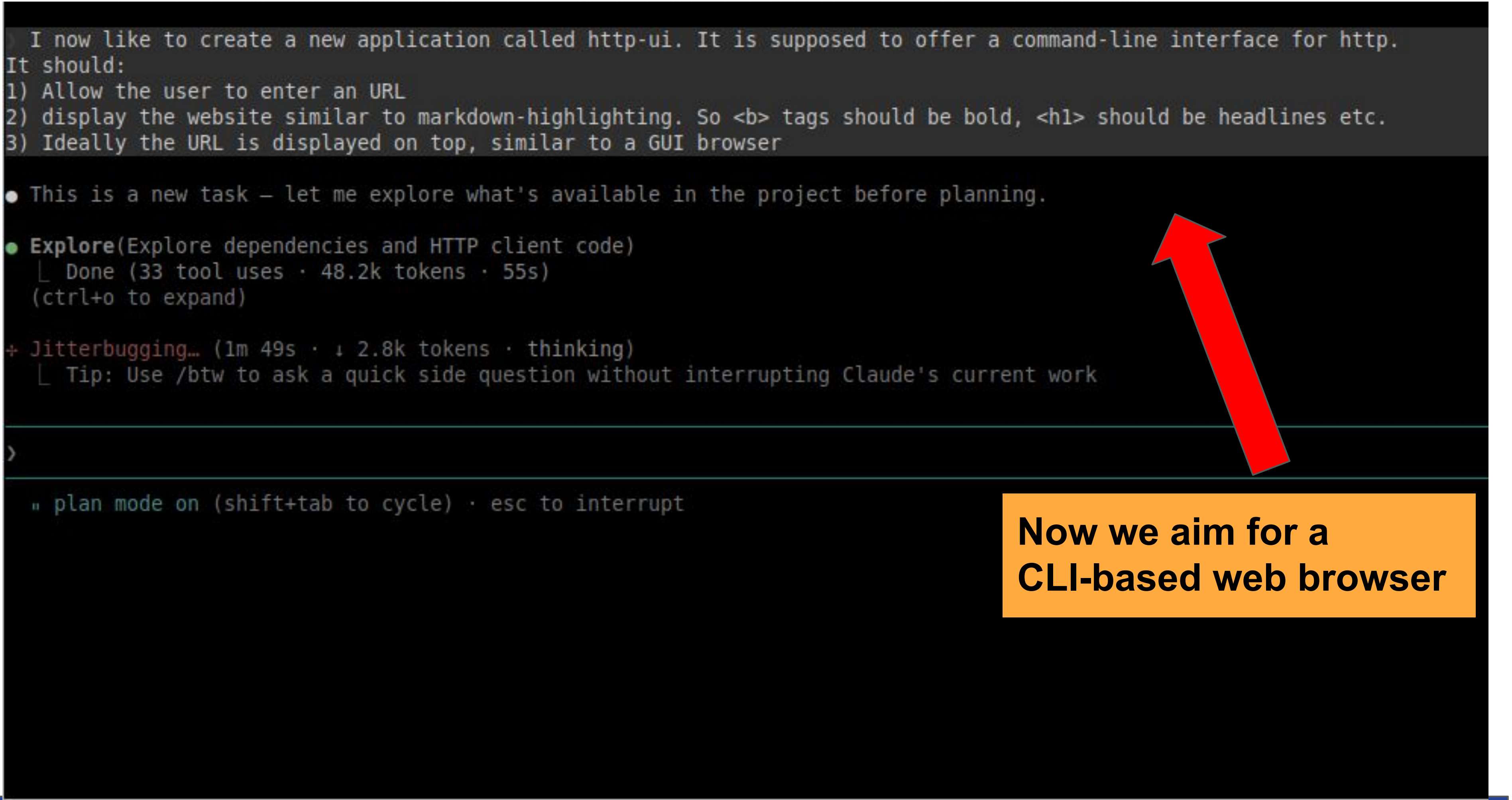
```
(httpie) kacianka@gpusrv10:~/demo/cli$ http --html-json -b  example.com
[
    {
        "tag": "html",
        "attrs": {
            "lang": "en"
        },
        "children": [
            {
                "tag": "head",
                "children": [
                    {
                        "tag": "title",
                        "text": "Example Domain"
                    },
                    {
                        "tag": "meta",
                        "attrs": {
                            "name": "viewport",
                            "content": "width=device-width, initial-scale=1"
                        },
                        "children": [
                            {
                                "tag": "style",
                                "text": "body{background:#eee;width:60vw;margin:15vh auto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:
0.8}a:link,a:visited{color:#348}"
                            }
                        ]
                    }
                ]
            },
            {
                "tag": "body",
                "children": [
                    {
                        "tag": "div",
                        "children": [
                            {
                                "tag": "h1",
                                "text": "Example Domain"
                            },
                            {
                                "tag": "p",
                                "text": "This domain is for use in documentation examples without needing permission. Avoid use in operations."
                            },
                            {
                                "tag": "p",
                                "children": [
                                    {
                                        "tag": "a",
                                        "attrs": {
                                            "href": "https://iana.org/domains/example"
                                        },
                                        "text": "Learn more"
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
    }
]

(httpie) kacianka@gpusrv10:~/demo/cli$ 
```



```
> /cost
  └ Total cost:            $1.19
    Total duration (API):  7m 4s
    Total duration (wall): 21m 6s
    Total code changes:    322 lines added, 29 lines removed
    Usage by model:
        claude-haiku-4-5:  2.3k input, 14.3k output, 1.5m cache read, 135.7k cache write ($0.3949
       claude-sonnet-4-6:  370 input, 15.2k output, 1.0m cache read, 66.5k cache write ($0.79)

>

  ▸▸ accept edits on (shift+tab to cycle)
```

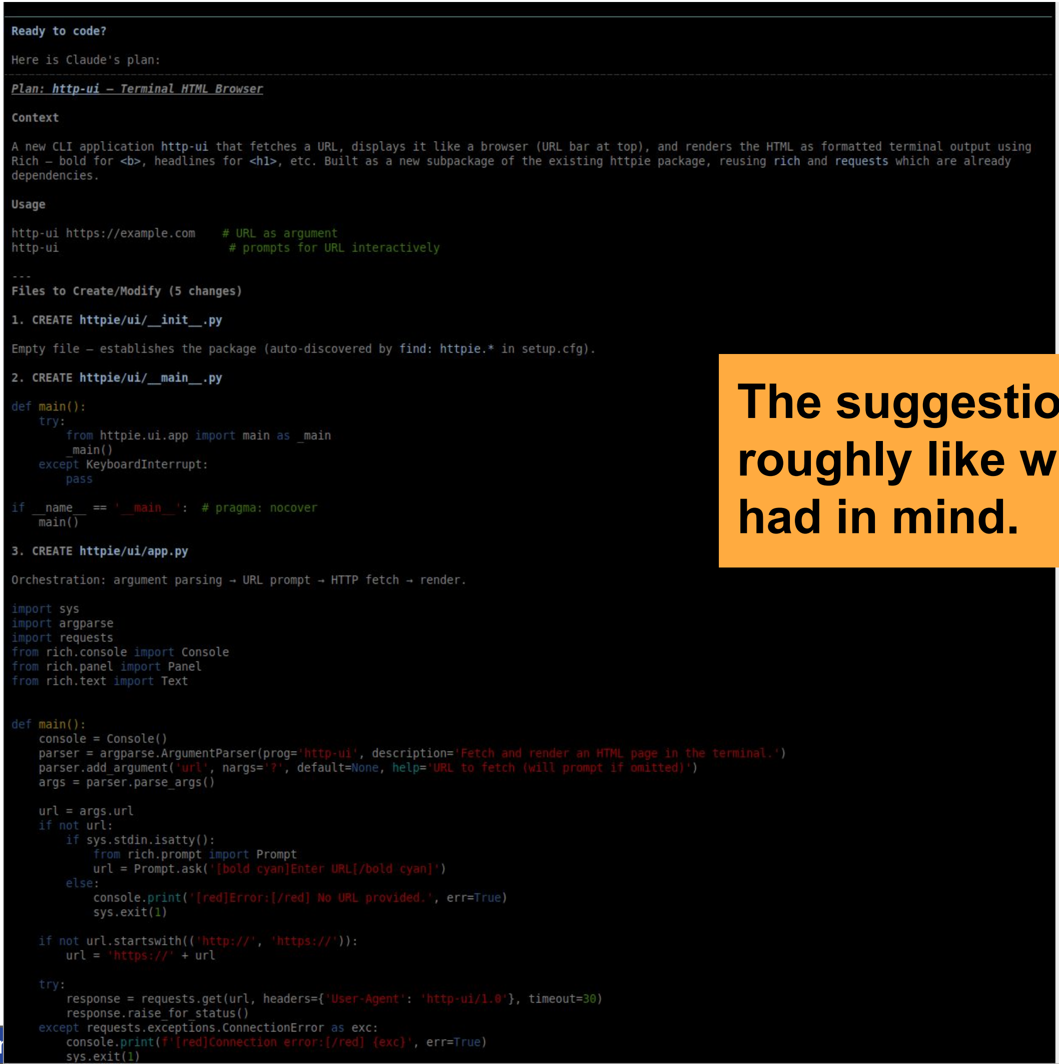
Every prompt costs money!

Vibecode a CLI-GUI



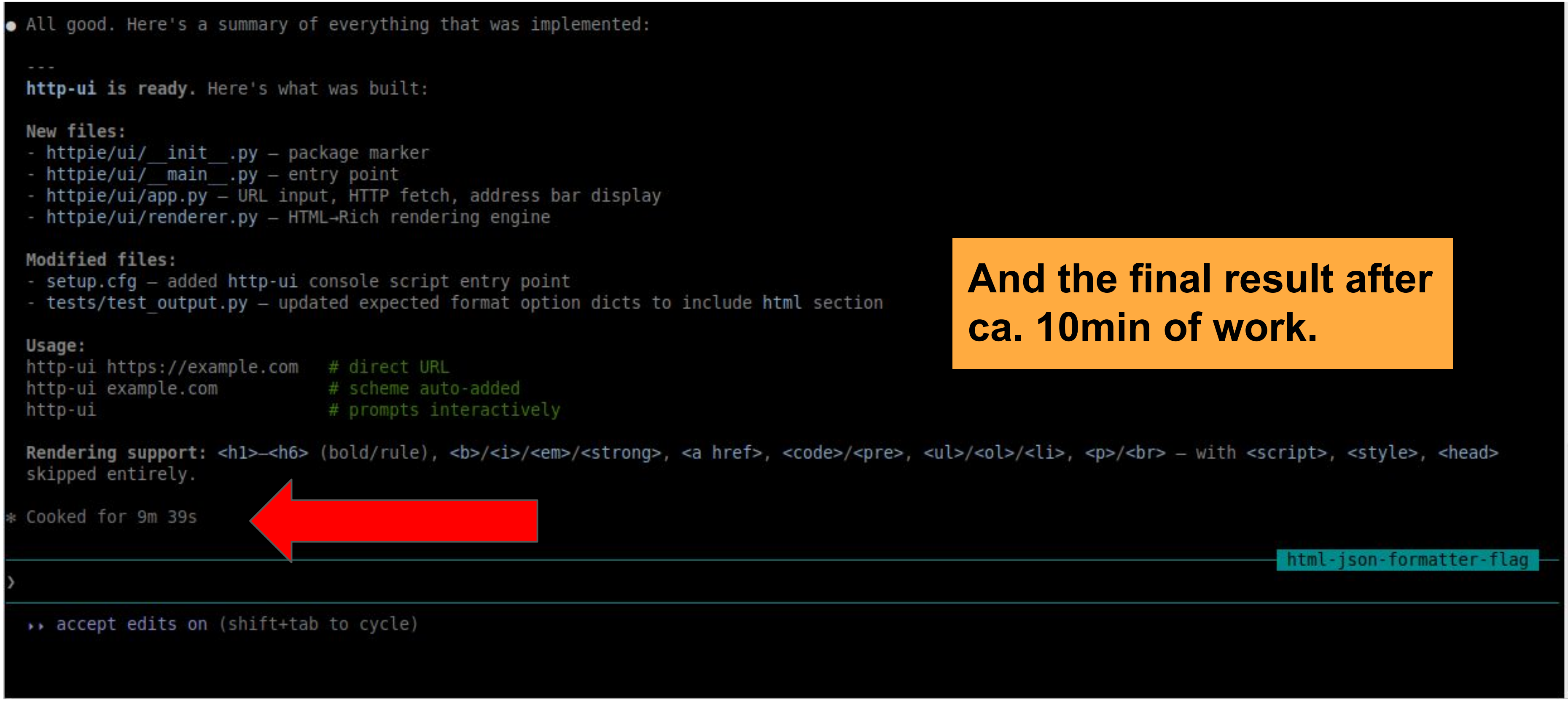
```
❯ I now like to create a new application called http-ui. It is supposed to offer a command-line interface for http.
It should:
1) Allow the user to enter an URL
2) display the website similar to markdown-highlighting. So <b> tags should be bold, <h1> should be headlines etc.
3) Ideally the URL is displayed on top, similar to a GUI browser

● This is a new task — let me explore what's available in the project before planning.

● Explore(Explore dependencies and HTTP client code)
  ⎿  Done (33 tool uses · 48.2k tokens · 55s)
  (ctrl+o to expand)

✢ Jitterbugging… (1m 49s · ↓ 2.8k tokens · thinking)
  ⎿  Tip: Use /btw to ask a quick side question without interrupting Claude's current work

❯

  ⏸ plan mode on (shift+tab to cycle) · esc to interrupt
```

Now we aim for a CLI-based web browser



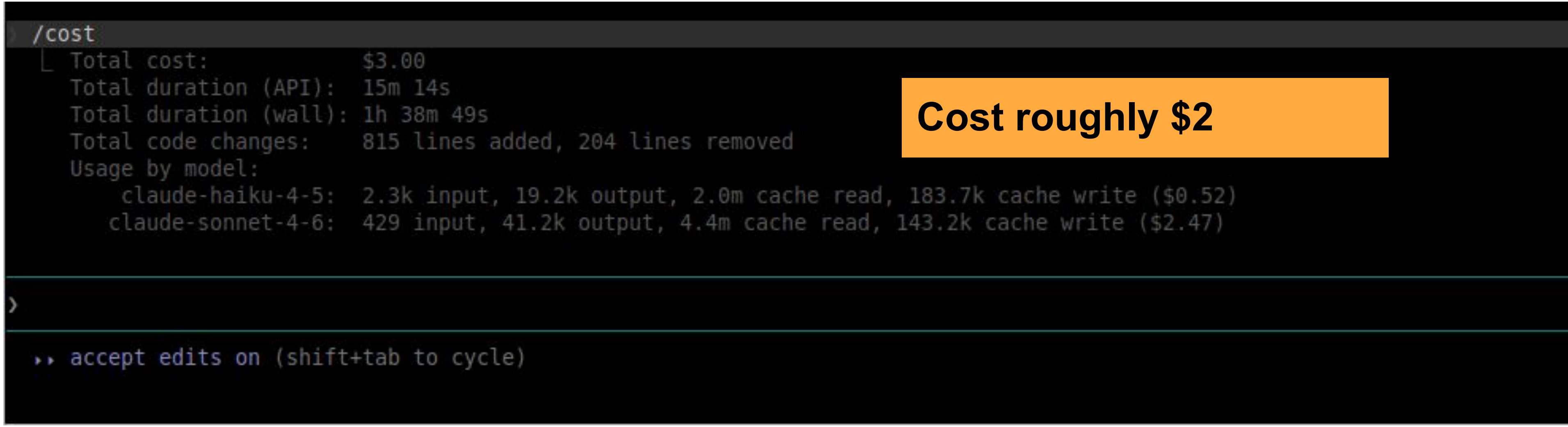
```
Ready to code?

Here is Claude's plan:
------------------------------------------------------------------------------------------------------------------------
Plan: http-ui – Terminal HTML Browser

Context

A new CLI application http-ui that fetches a URL, displays it like a browser (URL bar at top), and renders the HTML as formatted terminal output using
Rich – bold for <b>, headlines for <h1>, etc. Built as a new subpackage of the existing httpie package, reusing rich and requests which are already
dependencies.

Usage

http-ui https://example.com    # URL as argument
http-ui                         # prompts for URL interactively

---
Files to Create/Modify (5 changes)

1. CREATE httpie/ui/__init__.py

Empty file – establishes the package (auto-discovered by find: httpie.* in setup.cfg).

2. CREATE httpie/ui/__main__.py

def main():
    try:
        from httpie.ui.app import main as _main
        _main()
    except KeyboardInterrupt:
        pass

if __name__ == '__main__':  # pragma: nocover
    main()

3. CREATE httpie/ui/app.py

Orchestration: argument parsing → URL prompt → HTTP fetch → render.

import sys
import argparse
import requests
from rich.console import Console
from rich.panel import Panel
from rich.text import Text


def main():
    console = Console()
    parser = argparse.ArgumentParser(prog='http-ui', description='Fetch and render an HTML page in the terminal.')
    parser.add_argument('url', nargs='?', default=None, help='URL to fetch (will prompt if omitted)')
    args = parser.parse_args()

    url = args.url
    if not url:
        if sys.stdin.isatty():
            from rich.prompt import Prompt
            url = Prompt.ask('[bold cyan]Enter URL[/bold cyan]')
        else:
            console.print('[red]Error:[/red] No URL provided.', err=True)
            sys.exit(1)

    if not url.startswith(('http://', 'https://')):
        url = 'https://' + url

    try:
        response = requests.get(url, headers={'User-Agent': 'http-ui/1.0'}, timeout=30)
        response.raise_for_status()
    except requests.exceptions.ConnectionError as exc:
        console.print(f'[red]Connection error:[/red] {exc}', err=True)
        sys.exit(1)
```

The suggestion looks roughly like what we had in mind.



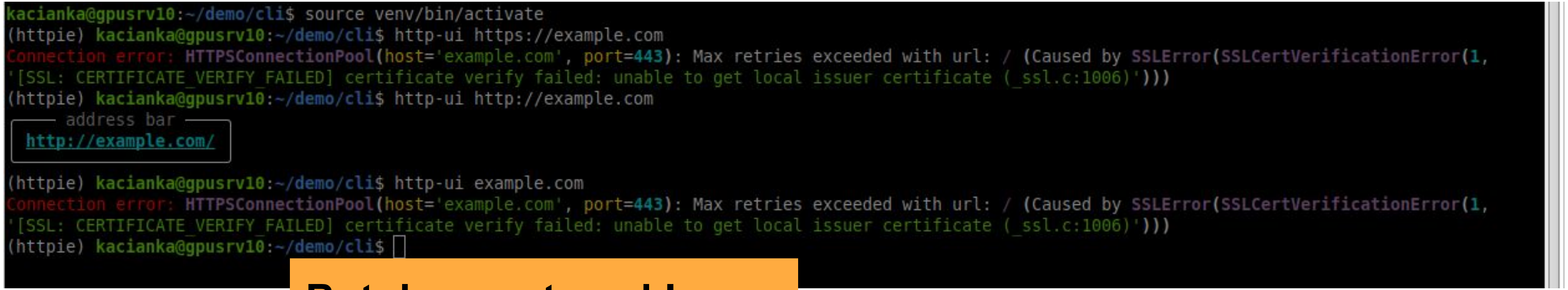
```
● All good. Here's a summary of everything that was implemented:

  ---
  http-ui is ready. Here's what was built:

  New files:
  - httpie/ui/__init__.py — package marker
  - httpie/ui/__main__.py — entry point
  - httpie/ui/app.py — URL input, HTTP fetch, address bar display
  - httpie/ui/renderer.py — HTML→Rich rendering engine

  Modified files:
  - setup.cfg — added http-ui console script entry point
  - tests/test_output.py — updated expected format option dicts to include html section

  Usage:
  http-ui https://example.com   # direct URL
  http-ui example.com           # scheme auto-added
  http-ui                       # prompts interactively

  Rendering support: <h1>–<h6> (bold/rule), <b>/<i>/<em>/<strong>, <a href>, <code>/<pre>, <ul>/<ol>/<li>, <p>/<br> — with <script>, <style>, <head>
  skipped entirely.

✻ Cooked for 9m 39s

──────────────────────────────────────────────────────────────────────────────────────── html-json-formatter-flag ──
❯
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
  ⏵⏵ accept edits on (shift+tab to cycle)
```

And the final result after ca. 10min of work.



```
> /cost
  └ Total cost:            $3.00
    Total duration (API):  15m 14s
    Total duration (wall): 1h 38m 49s
    Total code changes:    815 lines added, 204 lines removed
    Usage by model:
        claude-haiku-4-5:  2.3k input, 19.2k output, 2.0m cache read, 183.7k cache write ($0.52)
       claude-sonnet-4-6:  429 input, 41.2k output, 4.4m cache read, 143.2k cache write ($2.47)

>

  ▸▸ accept edits on (shift+tab to cycle)
```

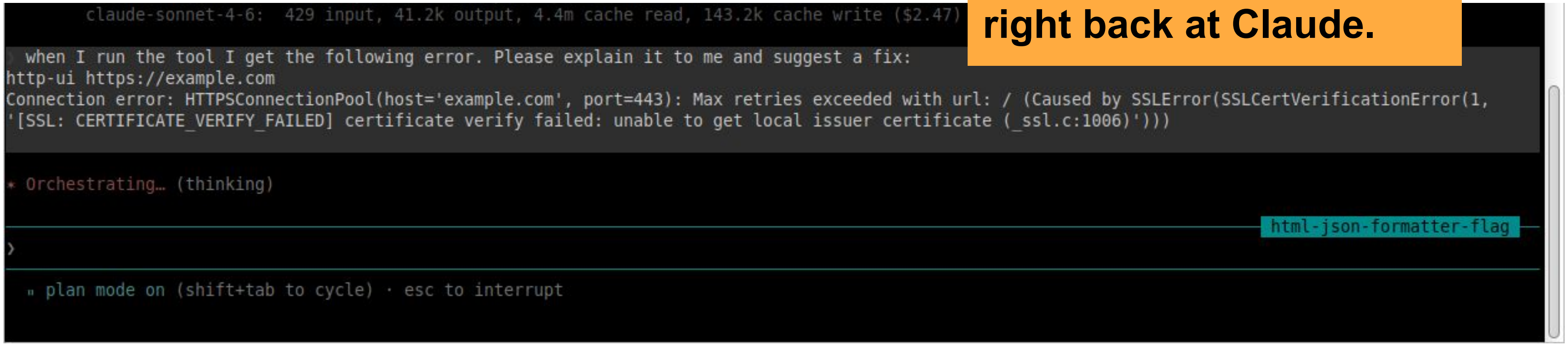
Cost roughly $2



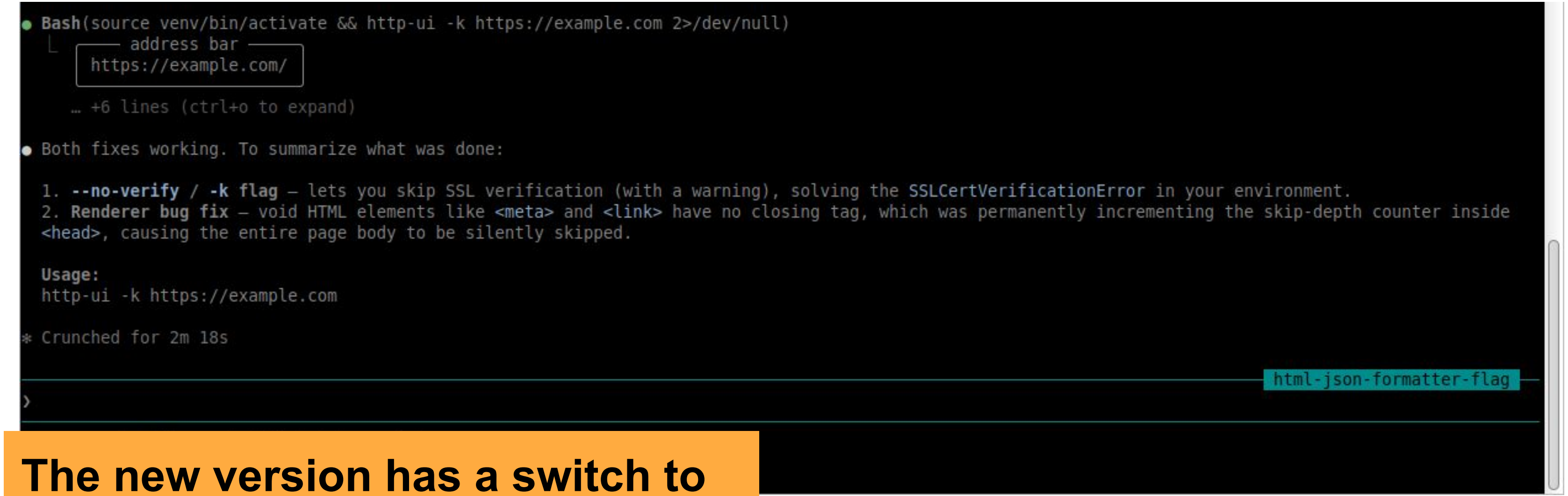
```
kacianka@gpusrv10:~/demo/cli$ source venv/bin/activate
(httpie) kacianka@gpusrv10:~/demo/cli$ http-ui https://example.com
Connection error: HTTPSConnectionPool(host='example.com', port=443): Max retries exceeded with url: / (Caused by SSLError(SSLCertVerificationError(1,
'[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1006)')))
(httpie) kacianka@gpusrv10:~/demo/cli$ http-ui http://example.com
┌─── address bar ───┐
│ http://example.com/ │
└───────────────────┘

(httpie) kacianka@gpusrv10:~/demo/cli$ http-ui example.com
Connection error: HTTPSConnectionPool(host='example.com', port=443): Max retries exceeded with url: / (Caused by SSLError(SSLCertVerificationError(1,
'[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1006)')))
(httpie) kacianka@gpusrv10:~/demo/cli$ 
```

But does not work!

So we throw the error right back at Claude.

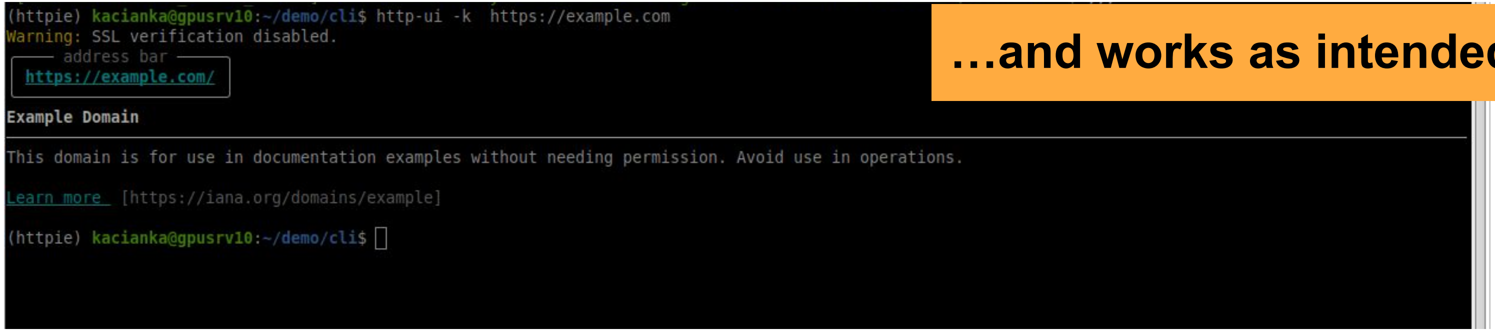


```
      claude-sonnet-4-6:  429 input, 41.2k output, 4.4m cache read, 143.2k cache write ($2.47)

> when I run the tool I get the following error. Please explain it to me and suggest a fix:
http-ui https://example.com
Connection error: HTTPSConnectionPool(host='example.com', port=443): Max retries exceeded with url: / (Caused by SSLError(SSLCertVerificationError(1,
'[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1006)')))

* Orchestrating… (thinking)

──────────────────────────────────────────────────────────────────────────────── html-json-formatter-flag ─
>
───────────────────────────────────────────────────────────────────────────────────────────────────────────
  ⏸ plan mode on (shift+tab to cycle) · esc to interrupt
```



```
● Bash(source venv/bin/activate && http-ui -k https://example.com 2>/dev/null)
  └ ┌─────── address bar ───────┐
    │ https://example.com/       │
    └────────────────────────────┘
    … +6 lines (ctrl+o to expand)

● Both fixes working. To summarize what was done:

  1. --no-verify / -k flag — lets you skip SSL verification (with a warning), solving the SSLCertVerificationError in your environment.
  2. Renderer bug fix — void HTML elements like <meta> and <link> have no closing tag, which was permanently incrementing the skip-depth counter inside
  <head>, causing the entire page body to be silently skipped.

  Usage:
  http-ui -k https://example.com

✻ Crunched for 2m 18s

──────────────────────────────────────────────────────────────────────────────── html-json-formatter-flag ──
❯
```

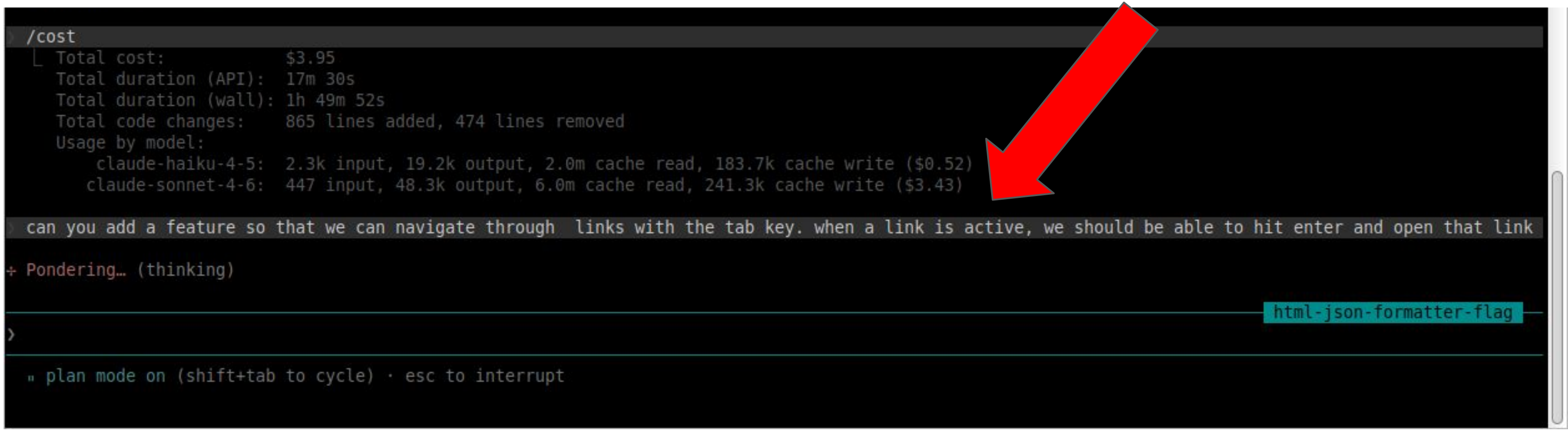
The new version has a switch to skip SSL checks…



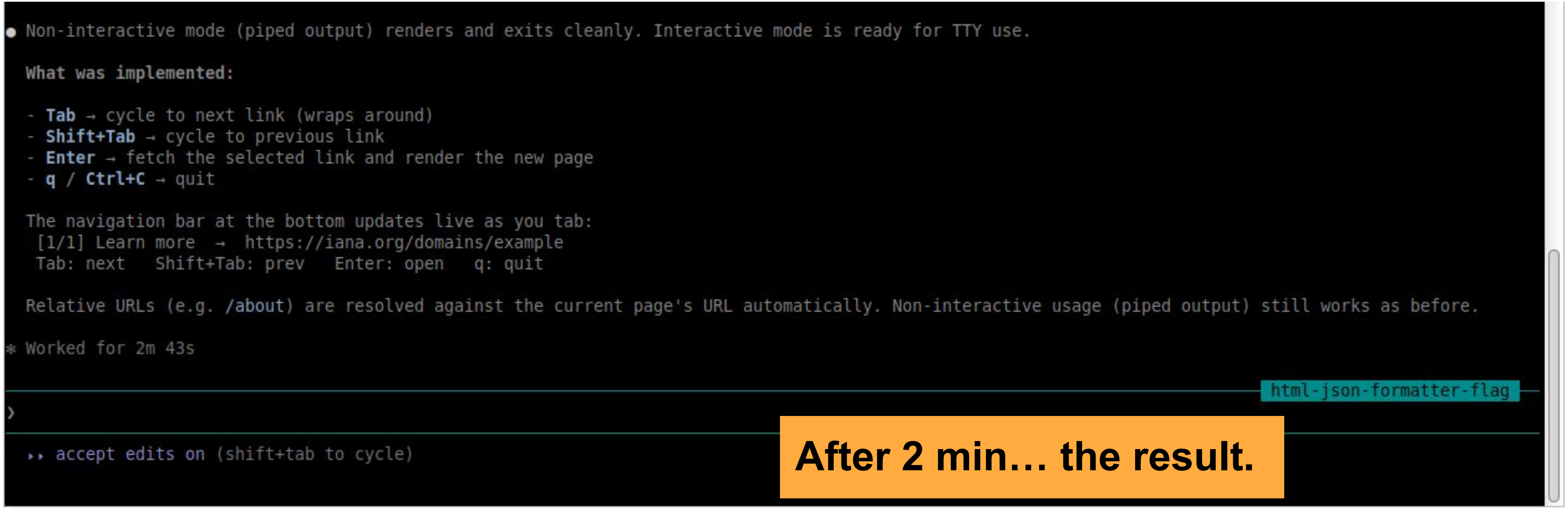
```
(httpie) kacianka@gpusrv10:~/demo/cli$ http-ui -k  https://example.com
Warning: SSL verification disabled.
 ┌─── address bar ───┐
 │ https://example.com/ │
 └──────────────────────┘
Example Domain
─────────────────────────────────────────────────────────────────────────────
This domain is for use in documentation examples without needing permission. Avoid use in operations.

Learn more  [https://iana.org/domains/example]

(httpie) kacianka@gpusrv10:~/demo/cli$ 
```

…and works as intended.

Why not turn this into a browser?

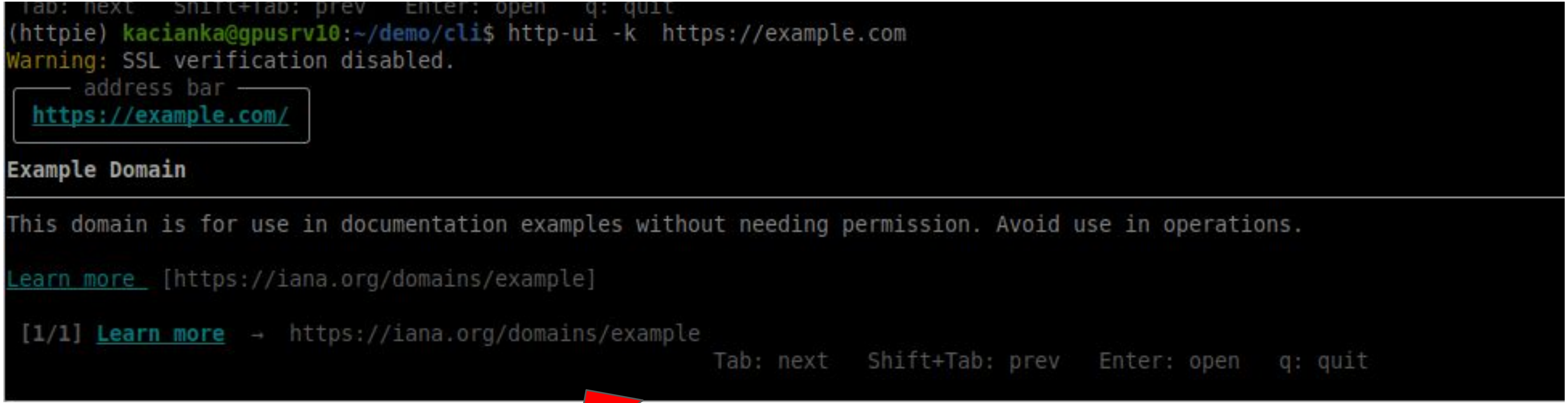


```
› /cost
  ⎿ Total cost:            $3.95
    Total duration (API):  17m 30s
    Total duration (wall): 1h 49m 52s
    Total code changes:    865 lines added, 474 lines removed
    Usage by model:
        claude-haiku-4-5:  2.3k input, 19.2k output, 2.0m cache read, 183.7k cache write ($0.52)
       claude-sonnet-4-6:  447 input, 48.3k output, 6.0m cache read, 241.3k cache write ($3.43)

› can you add a feature so that we can navigate through  links with the tab key. when a link is active, we should be able to hit enter and open that link

+ Pondering… (thinking)

──────────────────────────────────────────────────────────────────────────── html-json-formatter-flag ──
›
────────────────────────────────────────────────────────────────────────────────────────────────────────
  ⏸ plan mode on (shift+tab to cycle) · esc to interrupt
```



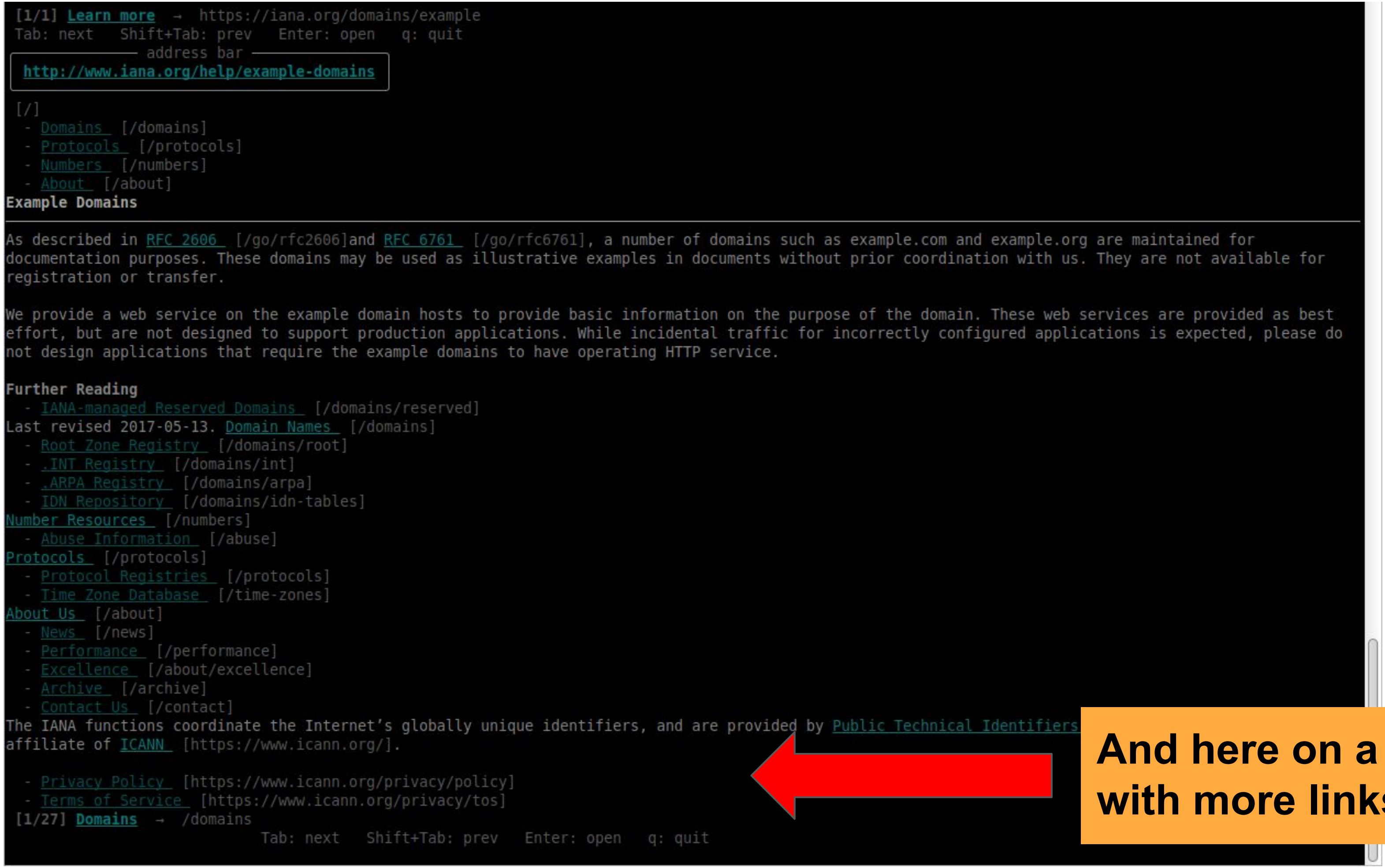
```
● Non-interactive mode (piped output) renders and exits cleanly. Interactive mode is ready for TTY use.

  What was implemented:

  - Tab → cycle to next link (wraps around)
  - Shift+Tab → cycle to previous link
  - Enter → fetch the selected link and render the new page
  - q / Ctrl+C → quit

  The navigation bar at the bottom updates live as you tab:
   [1/1] Learn more  →  https://iana.org/domains/example
   Tab: next   Shift+Tab: prev   Enter: open   q: quit

  Relative URLs (e.g. /about) are resolved against the current page's URL automatically. Non-interactive usage (piped output) still works as before.

* Worked for 2m 43s

──────────────────────────────────────────────────────────────────────── html-json-formatter-flag ─
❯
──────────────────────────────────────────────────────────────────────────────────────────────────
  ›› accept edits on (shift+tab to cycle)
```

After 2 min… the result.



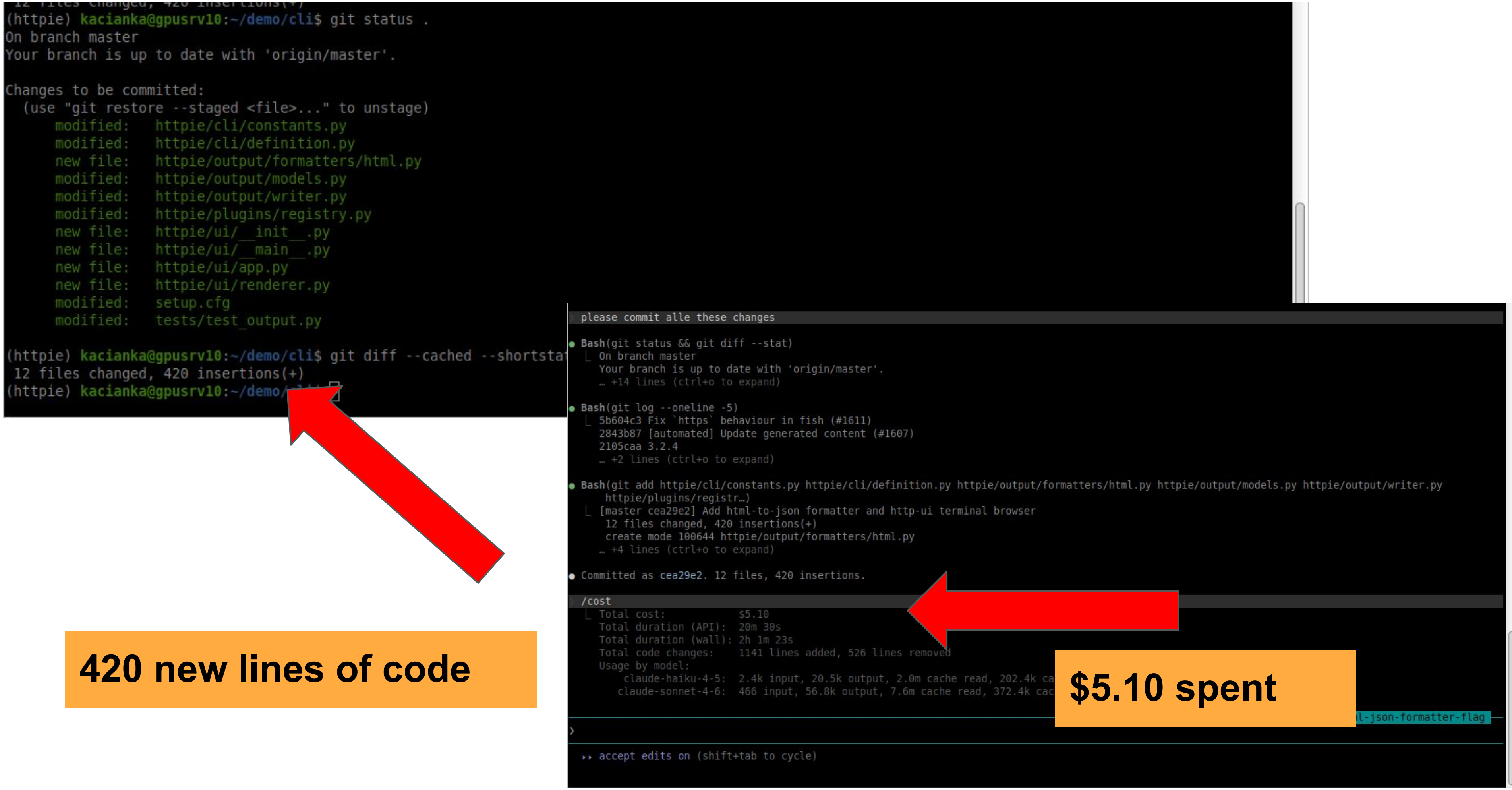
```
Tab: next   Shift+Tab: prev   Enter: open   q: quit
(httpie) kacianka@gpusrv10:~/demo/cli$ http-ui -k  https://example.com
Warning: SSL verification disabled.
┌──── address bar ─────────┐
│ https://example.com/     │
└──────────────────────────┘

Example Domain
────────────────────────────────────────────────────────────────────────────────────────
This domain is for use in documentation examples without needing permission. Avoid use in operations.

Learn more  [https://iana.org/domains/example]

 [1/1] Learn more  →  https://iana.org/domains/example
                                             Tab: next   Shift+Tab: prev   Enter: open   q: quit
```

Our UI



```
[1/1] Learn more  →  https://iana.org/domains/example
Tab: next    Shift+Tab: prev    Enter: open    q: quit
─────────────── address bar ───────────────
│ http://www.iana.org/help/example-domains │

[/]
  - Domains  [/domains]
  - Protocols  [/protocols]
  - Numbers  [/numbers]
  - About  [/about]
Example Domains
──────────────────────────────────────────────────────────────────────────────

As described in RFC 2606  [/go/rfc2606]and RFC 6761  [/go/rfc6761], a number of domains such as example.com and example.org are maintained for
documentation purposes. These domains may be used as illustrative examples in documents without prior coordination with us. They are not available for
registration or transfer.

We provide a web service on the example domain hosts to provide basic information on the purpose of the domain. These web services are provided as best
effort, but are not designed to support production applications. While incidental traffic for incorrectly configured applications is expected, please do
not design applications that require the example domains to have operating HTTP service.

Further Reading
  - IANA-managed Reserved Domains  [/domains/reserved]
Last revised 2017-05-13. Domain Names  [/domains]
  - Root Zone Registry  [/domains/root]
  - .INT Registry  [/domains/int]
  - .ARPA Registry  [/domains/arpa]
  - IDN Repository  [/domains/idn-tables]
Number Resources  [/numbers]
  - Abuse Information  [/abuse]
Protocols  [/protocols]
  - Protocol Registries  [/protocols]
  - Time Zone Database  [/time-zones]
About Us  [/about]
  - News  [/news]
  - Performance  [/performance]
  - Excellence  [/about/excellence]
  - Archive  [/archive]
  - Contact Us  [/contact]
The IANA functions coordinate the Internet's globally unique identifiers, and are provided by Public Technical Identifiers
affiliate of ICANN  [https://www.icann.org/].

  - Privacy Policy  [https://www.icann.org/privacy/policy]
  - Terms of Service  [https://www.icann.org/privacy/tos]
 [1/27] Domains  →  /domains
                              Tab: next    Shift+Tab: prev    Enter: open    q: quit
```

And here on a website with more links



```
 12 files changed, 420 insertions(+)
(httpie) kacianka@gpusrv10:~/demo/cli$ git status .
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   httpie/cli/constants.py
        modified:   httpie/cli/definition.py
        new file:   httpie/output/formatters/html.py
        modified:   httpie/output/models.py
        modified:   httpie/output/writer.py
        modified:   httpie/plugins/registry.py
        new file:   httpie/ui/__init__.py
        new file:   httpie/ui/__main__.py
        new file:   httpie/ui/app.py
        new file:   httpie/ui/renderer.py
        modified:   setup.cfg
        modified:   tests/test_output.py

(httpie) kacianka@gpusrv10:~/demo/cli$ git diff --cached --shortstat
 12 files changed, 420 insertions(+)
(httpie) kacianka@gpusrv10:~/demo/
```

```
 please commit alle these changes

● Bash(git status && git diff --stat)
  ⎿  On branch master
     Your branch is up to date with 'origin/master'.
     … +14 lines (ctrl+o to expand)

● Bash(git log --oneline -5)
  ⎿  5b604c3 Fix `https` behaviour in fish (#1611)
     2843b87 [automated] Update generated content (#1607)
     2105caa 3.2.4
     … +2 lines (ctrl+o to expand)

● Bash(git add httpie/cli/constants.py httpie/cli/definition.py httpie/output/formatters/html.py httpie/output/models.py httpie/output/writer.py
      httpie/plugins/registr…)
  ⎿  [master cea29e2] Add html-to-json formatter and http-ui terminal browser
      12 files changed, 420 insertions(+)
      create mode 100644 httpie/output/formatters/html.py
     … +4 lines (ctrl+o to expand)

● Committed as cea29e2. 12 files, 420 insertions.

 /cost
  ⎿  Total cost:            $5.10
     Total duration (API):  20m 30s
     Total duration (wall): 2h 1m 23s
     Total code changes:    1141 lines added, 526 lines removed
     Usage by model:
         claude-haiku-4-5:  2.4k input, 20.5k output, 2.0m cache read, 202.4k ca
        claude-sonnet-4-6:  466 input, 56.8k output, 7.6m cache read, 372.4k cac
                                                                                  l-json-formatter-flag
›

  ›› accept edits on (shift+tab to cycle)
```

420 new lines of code

$5.10 spent

Summary

Summary

- 420 lines added in ~2 hours
- $5.10 spent
- Key takeaways:
  - Small steps beat big prompts.
  - Tests define correctness.
  - Fast generation does not remove the need for review.

Break

Access to Claude

Connect to the server

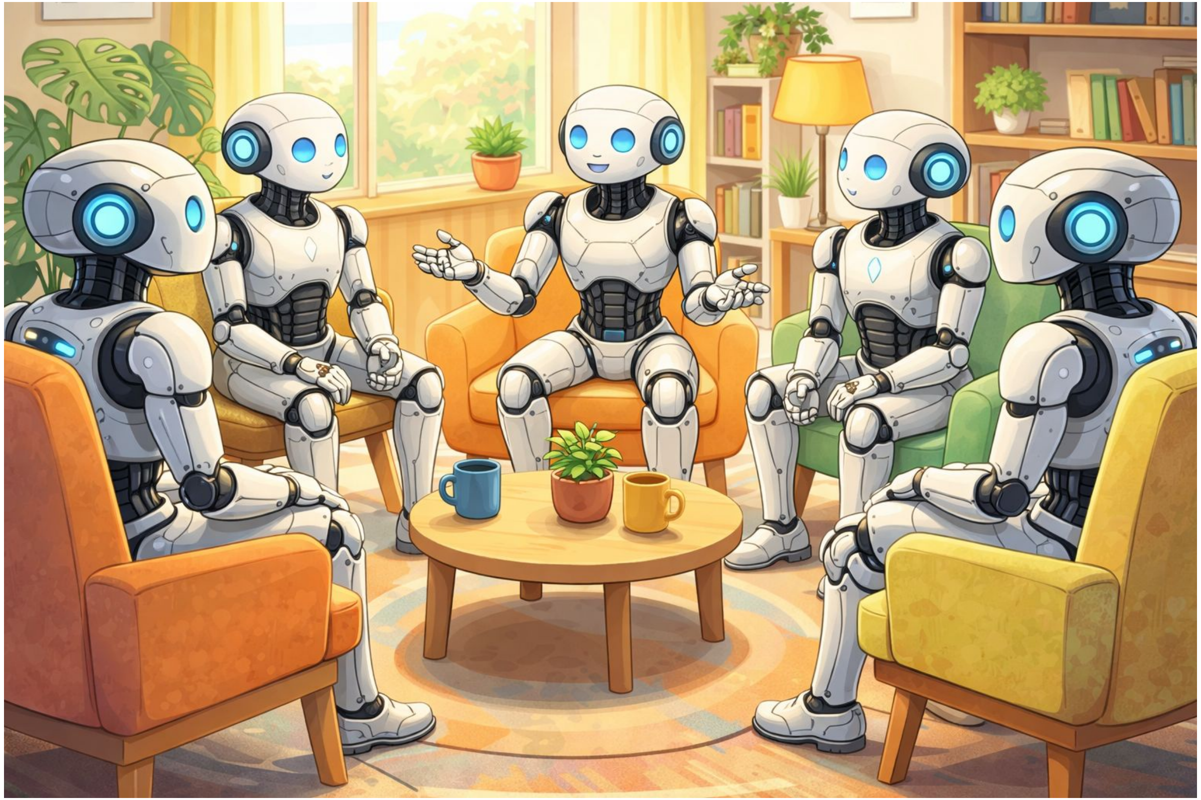
- Use SSH or PuTTY
- Server: aideation.fortiss-demo.org
- User: ihk
- Password:
- Port: 2025
- ssh ihk@aideation.fortiss-demo.org -p 2025

Start Claude

1. mkdir <your-name>
2. cd <your-name>
3. git clone https://github.com/httpie/cli.git
4. Alternative: clone or scp any other repo to the machine
5. cd cli
6. claude
7. start coding!

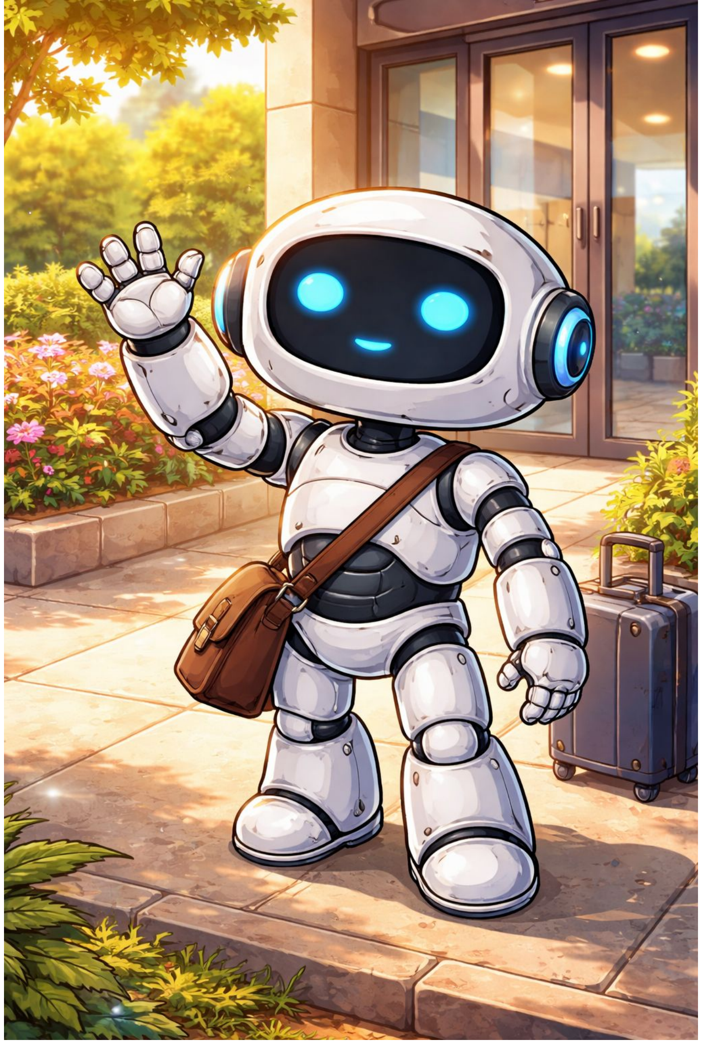
Wrap-Up

Discussion



Take-home Message

- AI coding agents are powerful.
- They are especially useful for prototyping and debugging.
  - Explore an idea or create a specific converter.
  - Find and explain obscure errors.
- Tests are essential.
  - The resulting code must be tested and reviewed.
  - Security is an especially important aspect!
- How to embed them into software engineering processes is still an open problem.
  - Ensure quality.
  - Ensure maintainability.



Thank you!

Questions?

Or anytime via e-mail:
kacianka@fortiss.org