Agent Browser Setup Guide

Install Vercel's Agent Browser and wire it into Claude Code or Codex so your agent drives a real browser with element refs instead of screenshots.

Why I stopped letting agents look at screenshots

Every browser automation tool I tried before this one worked the same way. The agent takes a screenshot, sends the image to the model, the model describes what it sees, then it guesses a pixel coordinate and clicks. That loop is slow, it costs a lot per step, and it misses buttons that moved three pixels.

Agent Browser from Vercel Labs does it differently. It reads the page as an accessibility tree and gives every interactive element a short reference like **@e1**. Your agent gets a few lines of text instead of an image, and it clicks a reference instead of a coordinate. Run a snapshot on example.com and the whole page state comes back as one line.

By the end of this guide you will have the CLI installed, a browser downloaded, Agent Browser wired into Claude Code or Codex, and a first run you can watch happen on your own screen. It is open source under Apache-2.0, it needs no API key for the browser work, and it sits at roughly 50,300 GitHub stars as of November 2026.

01

How it actually works

1. You open a page. The CLI talks to a background daemon over the Chrome DevTools Protocol, and that browser stays open between commands, so each step your agent takes is one short shell call rather than a fresh session.
2. You snapshot the page. Agent Browser returns the accessibility tree as compact text, and every interactive element carries a reference such as [ref=e1]. On example.com that is a single line: - link "Learn more" [ref=e1].
3. Your agent acts on a reference. It runs click @e1 or fill @e4 "some text", and the action lands on that exact element. No image is sent, no coordinate is guessed, and the same command works again next run.

02

The one thing almost everyone gets wrong

Refs belong to the snapshot that produced them. People take one snapshot at the start of a task, then keep passing @e3 around for the next eight steps, and wonder why the agent starts clicking the wrong thing or erroring out. Once the page navigates, re-renders, or opens a modal, the old refs no longer describe what is on screen.

WATCH OUT

Re-run snapshot after anything that changes the page: a navigation, a form submit, a tab switch, a dialog. Treat refs as single use. If your agent keeps clicking stale elements, this is almost always why.

The second half of the same mistake is loading every tool at once. The MCP server defaults to the **core** profile on purpose, because core keeps the context small. People set **--tools all** on day one, pull in the entire CLI parity surface, and hand back the token savings they installed this thing for.

---

03 The files and the config

SOURCE CODE

The full repo, the README, the command reference and the benchmarks directory:

https://github.com/vercel-labs/agent-browser

THE CORE LOOP (PASTE THIS IN YOUR TERMINAL FIRST)

```
agent-browser open example.com
agent-browser snapshot -i       # interactive elements, each with a ref
agent-browser click @e1         # act on a ref from that snapshot
agent-browser get url           # check the result
agent-browser close
```

WHAT SNAPSHOT -I GIVES YOU BACK

```
- link "Learn more" [ref=e1]
```

MCP.JSON (MCP CLIENT CONFIG, FROM THE DOCS)

```
{
  "mcpServers": {
    "agent-browser": {
      "command": "agent-browser",
      "args": ["mcp"]
    }
  }
}
```

AGENTS.MD (THE NO-MCP ROUTE, ADD THESE TWO LINES)

```
Use `agent-browser` for browser automation. Before the first use, run
`agent-browser skills get core` and follow it.
```

WIDENING THE TOOL PROFILES WHEN YOU NEED MORE

```
agent-browser mcp                              # default: the core profile
agent-browser mcp --tools core,network,react   # combine profiles with commas
agent-browser mcp --tools all                  # the full typed CLI surface
```

The profiles are **core**, **network**, **state**, **debug**, **tabs**, **react**, **mobile** and **all**. Core covers navigation, snapshots, interaction, waits, reads, screenshots, JavaScript eval, close, tab basics and profile discovery. In your client the tools arrive prefixed, so you will see agent_browser_open, agent_browser_snapshot, agent_browser_click, agent_browser_fill and agent_browser_eval.

SITES SHOWN IN THE VIDEO

- vercel-labs/agent-browser on GitHub
  https://github.com/vercel-labs/agent-browser

Set it up in Claude Code or Codex

1. Install the CLI globally: **npm install -g agent-browser**. On a Mac you can use **brew install agent-browser** instead. If you only want to try it once, skip the install and run **npx agent-browser open example.com**.
2. Download the browser: **agent-browser install**. This pulls Chrome for Testing and only runs the first time. On Linux use **agent-browser install --with-deps** so it also installs the browser's system libraries.
3. Clone the repo if you want to read the source or the skills that ship with it: **git clone https://github.com/vercel-labs/agent-browser**. You do not need the clone to use the tool, only to poke at it.
4. Check the install worked before you involve an agent. Run the five core loop commands above by hand. A successful first run prints a one line snapshot with [ref=e1] in it, then **agent-browser get url** prints the URL you landed on after the click.
5. Pick your wiring. Route A is MCP: drop the mcp.json block above into your client's MCP config, then restart the client. In Claude Code the server shows up as **agent-browser** with the core tools. Route B is the plain CLI: add the two AGENTS.md lines above to the project, or run **npx skills add vercel-labs/agent-browser** to install the discovery skill into your agent harness.
6. Point the agent at the skill before its first browser task. The repo ships its own instructions and the agent reads them with **agent-browser skills get core**, which explains the loop, refs, waiting and the common workflows. Add **--full** for the complete command reference, or run **agent-browser skills list** to see the specialized ones.
7. Only if you want the built in chat command: ask Claude to add **AI_GATEWAY_API_KEY** to your shell profile, then open a new terminal so it loads. Nothing else in Agent Browser needs a key, so skip this step unless you plan to use **agent-browser chat**.
8. Run a real task. Tell your agent to open a page you actually use, snapshot it, and read back what it sees. If it describes the page in element names rather than in pixels, it is working.

On Codex the folder convention is the same idea in a different place: either keep the instructions in the project folder or put a skill under **.codex/skills/<name>/** with a pointer in AGENTS.md. For Claude Code skills, the folder lives at **~/.claude/skills/<name>/** and the folder name has to match the **name:** field in its SKILL.md, or it will not load.

> 💡 **QUICK TIPS**
>
> - Start on the core profile and widen only when a task actually fails for a missing tool. That is the whole token argument.
> - Use **agent-browser screenshot --if-changed** in loops. The docs recommend it because it skips writing an image when the page has not moved.
> - Each **--session** gets its own daemon and its own browser, so you can keep a logged-in session separate from a scratch one.
> - Agent Browser handles iframes during the snapshot and inlines them under the iframe element, and the refs carry frame context. You can fill a payment iframe without switching frames.

04

The commands I actually run

FIRST RUN, TO PROVE THE LOOP

```
Use agent-browser. Run `agent-browser skills get core` first and follow it. Then open [URL], take an interactive snapshot, and tell me every element you can act on with its ref. Do not click anything yet.
```

LOG IN AND GET TO A PAGE BEHIND AUTH

```
Use agent-browser with --session [SESSION NAME]. Open [LOGIN URL], snapshot it, fill the email and password fields from [WHERE THE CREDENTIALS LIVE], submit, then snapshot again and confirm you are on [EXPECTED PAGE]. Save the session state so I don't have to log in again.
```

FILL A LONG FORM WITHOUT ME WATCHING

```
Use agent-browser. Open [FORM URL] and snapshot it. For each field, fill it from this list: [FIELD: VALUE pairs]. Re-snapshot after every page or step change before you use any ref. Submit, then show me the confirmation text.
```

PULL THE NUMBERS OFF A DASHBOARD

```
Use agent-browser on [DASHBOARD URL] with --session [SESSION NAME]. Read the page, pull out [THE METRICS I WANT], and give them to me as a markdown table. Use `read` rather than screenshots so we don't burn tokens.
```

DEBUG MY OWN APP IN THE BROWSER

```
Use agent-browser. Open [LOCALHOST URL], click through [THE FLOW THAT IS BROKEN], then show me `console`, `errors` and `network` output for that flow. Tell me which request failed and what the page did about it.
```

When it breaks

ERROR	FIX
agent-browser: command not found	The global npm bin folder is not on your PATH. Run `npm prefix -g` and add its bin folder to your shell profile, then open a new terminal. Or skip the install and use `npx agent-browser ...`.
No browser / browser fails to launch	You skipped the one time download. Run `agent-browser install`. On Linux run `agent-browser install --with-deps` so the system libraries go in too.
Linux: browser starts then dies instantly	Missing shared libraries. `agent-browser install --with-deps` is the fix. Run `agent-browser doctor` to see what it thinks is wrong.
The agent clicks the wrong element, or a ref is not found	Stale refs. Something navigated or re-rendered since the last snapshot. Re-run `snapshot -i` and use the new refs.
The MCP tools never show up in my client	The client did not restart, or the config went in the wrong file. Confirm the mcp.json block, restart the client, then check the tool list for agent_browser_open.
A tool I expected is missing from MCP	You are on the default core profile. Add the profile you need, for example `agent-browser mcp --tools core,network`, or use `--tools all`.
Tool call fails with an invalid params error naming unknown keys	The argument is not in that tool's schema. The error lists the tool's real arguments. Use those names rather than CLI flag spellings.
The browser is gone after I left it for a while	A headless daemon exits after one hour idle. It saves state first when session persistence is configured. Just open again, and use `--session` plus saved state so the login survives.
Commands time out on a slow page	Raise the default with the AGENT_BROWSER_DEFAULT_TIMEOUT environment variable, or wait for the thing you actually need with `agent-browser wait` before you act.
agent-browser chat says it needs a key	Chat is the one command that calls a model. Set AI_GATEWAY_API_KEY in your shell profile and open a new terminal. The default model is anthropic/claude-sonnet-4.6, overridable with --model or AI_GATEWAY_MODEL.
`record` produces nothing	Video recording needs ffmpeg on your PATH. Install it, then try again.

Ready to go deeper?

Want the videos made for you? Paste any link or idea into Content Engine and get a published video in under 3 minutes, with DM automation armed to capture leads.

Meet Content Engine →

https://builds.digicuratoragency.com/content-engine/