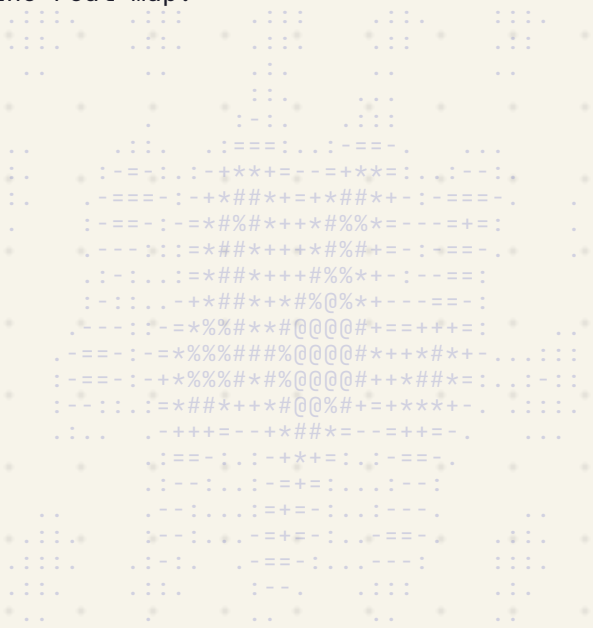
// SECTION 01 · FIELD GUIDE

CLAUDE CONNECTORS

Perplexity, Firecrawl, and Playwright are the three connectors worth wiring into Claude. Only one of them installs the way you expect. This is which door each one goes through, and why the other two refuse the obvious one.

```
> boot opusjake_os
> resource: claude-connectors
> version: v1.0 · 2026
> status: ready_
```

Everyone tells you to paste a server URL into the Connectors panel. That works for exactly one of the three tools you actually want. The other two hit a wall nobody mentions until you are already stuck. Here is the real map.



THREE DOORS

Claude is not one product. It is a web app, a desktop app, and a terminal, and they do not accept connectors the same way. Pick the wrong door and the tool simply will not appear.

DOOR	WHAT IT ACCEPTS	WHERE IT RUNS
claude.ai + mobile	Remote servers with OAuth	Anthropic's cloud
Claude Desktop	Remote servers + local extensions	Your machine
Claude Code	Anything, any transport	Your machine

// THE OAUTH WALL

This is the fact that costs people an afternoon. A custom connector on claude.ai **must authenticate with OAuth**. There is no field for a static API key header. If a tool's docs hand you `Authorization: Bearer sk-...`, that tool does not go in the web Connectors panel, no matter how many tutorials show you pasting the URL.

Add a custom connector at **Customize → Connectors → + → Add custom connector**. On Team and Enterprise it moves to **Organization settings → Connectors** and only an Owner can add it. Free plans get one custom connector, total.

// WHERE OUR THREE LAND

- **Firecrawl** is a native directory connector. One click, real OAuth, works on every surface including mobile.
- **Perplexity** is API-key only. Claude Code, Desktop config, or the API. Not the web panel.
- **Playwright** runs a browser on your machine. It can never be a remote connector, by design.

That is the whole guide in three lines. The rest is how to wire each one without wasting a key.

01 / FIRECRAWL

The extraction layer, and the only one of the three that installs the way the marketing says. Point it at a URL, get clean markdown instead of nav bars and cookie banners. Point it at a domain, get every page.

// THE ONE-CLICK PATH

Firecrawl is in the Claude connector directory. You do not paste a URL and you do not touch Advanced settings.

Open the **+** menu in a conversation, go to **Connectors**, enable Firecrawl. Claude opens Firecrawl in your browser, you sign in, pick a team, approve. Done. Requests bill to the credits of the team you picked, so pick deliberately if you belong to more than one.

// THE KEY-FREE PATH

Before you sign up for anything, try the keyless endpoint. It is a real URL that works with no account, rate limited by IP.

```
https://mcp.firecrawl.dev/v2/mcp
```

Scrape, search, and parse run keyless. Crawl, map, and agent need a key. That covers most first weeks. For Claude Code with a key, put it in a header, never in the path:

```
claude mcp add --transport http firecrawl \
  https://mcp.firecrawl.dev/v2/mcp \
  --header "Authorization: Bearer fc-your-key"
```

Older guides show the key baked into the URL as a path segment. Firecrawl now says explicitly not to do that. Keys in URLs end up in logs, history, and screenshots.

// THE FIVE TOOLS THAT MATTER

The server exposes twenty-five. You will use these.

- `firecrawl_scrape` takes one URL, returns clean markdown. Most of your calls.
- `firecrawl_map` lists every indexed URL on a domain. Run this before any crawl.
- `firecrawl_crawl` walks a site. Async, so it pairs with `firecrawl_check_crawl_status`.
- `firecrawl_parse` handles a PDF, DOCX, or XLSX and returns the same clean output.
- `firecrawl_monitor_create` watches a page and reports diffs on a schedule. Nobody uses this. They should.

// USE IT LIKE THIS

```
Use firecrawl_map on [DOMAIN] first and show me the URL list
before you scrape anything.

Then scrape only the ones matching [PATTERN]. Skip tag pages,
author pages, and anything paginated past page 2.

Return one table: URL, page title, the claim they lead with,
and the price if there is one. Flag any page where the price
is gated behind a demo request instead of published.
```

Show me the list before the scrape. A bad pattern caught for free beats a credit bill you find out about later.

02 / PERPLEXITY

The answer layer. It searches, reads, and returns a written answer with citations attached, so Claude spends its context on your problem instead of on twelve pages of search results.

// WHY IT IS NOT IN YOUR CONNECTORS PANEL

Perplexity's remote server authenticates with a bearer token. Claude.ai custom connectors require OAuth and cannot send a static API key header. Perplexity's own docs say it outright: use Claude Code, Claude Desktop, or the API instead.

So stop trying to paste the URL into the web app. It is not a bug in your setup.

// WIRE IT WHERE IT WORKS

Claude Code, one line:

```
claude mcp add --transport http perplexity \
  https://api.perplexity.ai/mcp \
  --header "Authorization: Bearer pplx-your-key"
```

Get the key from the Perplexity API console. Tool calls bill to that key at normal API rates. This one is not free and never has been.

// FOUR TOOLS, THREE PRICE POINTS

The difference between them is depth, which means the difference is money and time.

- `perplexity_search` returns ranked results with titles, URLs, snippets. The cheap one.
- `perplexity_ask` is a fast conversational answer with live search behind it. The default.
- `perplexity_research` runs deep and comes back with a cited report. Slow, expensive, worth it.
- `perplexity_reason` sits in the middle, for problems that need logic more than lookups.

Name the tool in the prompt. Left alone Claude reaches for `ask` nearly every time, and you will either pay for depth you did not ask for or wait on research you did not need.

// USE IT LIKE THIS

```
Use perplexity_research to build me a brief on [TOPIC].

I need: what changed in the last 90 days, who the three
credible sources are, what the strongest counterargument is,
and what is still unsettled.

Cite every claim with a link. If sources disagree, show me the
disagreement instead of averaging it. If something is a vendor
blog post rather than independent reporting, say so.
```

That last line earns its place. Search results are full of marketing dressed as analysis, and asking for the distinction up front is cheaper than catching it after you have quoted it.

03 / PLAYWRIGHT

The hands. It clicks, types, logs in, and works the pages the other two cannot touch, because it drives a real browser holding your real sessions.

// WHY IT WILL NEVER BE A CONNECTOR

A remote connector runs on someone else's servers, reached through Anthropic's infrastructure. The entire value of Playwright is that the browser is on **your** machine with **your** logins in it. There is nothing to host. Any guide offering you a hosted Playwright connector URL is offering you somebody else's browser.

Claude Code, one line, no key, no account:

```
claude mcp add playwright npx @playwright/mcp@latest
```

Microsoft also ships it as an official Claude Code plugin, roughly 320,000 installs. Same server, installed from the plugin directory instead of the command line.

// GETTING IT INTO CLAUDE DESKTOP

Desktop takes local servers through extensions. An `.mcpb` file is a zipped local MCP server plus a manifest, installed like a browser extension: double-click it, drag it onto the window, or go **Settings → Extensions → Advanced settings → Install Extension…**

No OAuth, runs offline, bundles its own Node runtime. This is the supported path for anything that needs your filesystem, your VPN, or your logged-in browser.

// THE TOOLS, AND THE ONE THAT MATTERS

- `browser_navigate` opens the page.
- `browser_snapshot` reads it as a structured accessibility tree. This is the important one.
- `browser_click`, `browser_type`, and `browser_fill_form` do the work.
- `browser_take_screenshot` is for showing you, not for the model to read.
- `browser_network_requests` catches the API call behind a page that will not scrape.

Snapshot is why this beats screenshot-driven automation. The model gets the real element tree, so a click lands on the actual button instead of a pixel guess.

Flags worth knowing: `--isolated` for a throwaway profile, `--headless` to hide the window, `--storage-state` to load saved cookies, `--extension` to drive tabs in a browser you already have open, and `--port 8931` to run it as a local HTTP server. That last one still will not reach claude.ai, because the web app cannot see your localhost.

// USE IT LIKE THIS

```
Open [URL] with the browser, take a snapshot, and tell me what
is on the page before you touch anything.

Then [TASK]. Snapshot again after each step so you are reading
the real page and not assuming it worked.

Do not enter payment details, do not create accounts, and stop
before any submit, send, or confirm button. Show me the filled
form and I will click it.
```

Snapshot, act, snapshot again. Automation that assumes a click worked is how you end up three steps deep on a page that never loaded.

THE HANDOFF

Each one alone is useful. The reason to run all three is the handoffs. Once they are wired, Claude makes them on its own if you tell it the shape of the job.

```
Three-part job. Use the right tool for each part.

1. perplexity_research: who are the real competitors to
   [COMPANY] right now, and what changed in this market last
   quarter? Cite everything.

2. firecrawl_map then firecrawl_scrape: for each competitor you
   named, pull the pricing page, the homepage, and the last 5
   blog posts. Clean markdown.

3. Browser: for any competitor whose pricing is gated, open the
   page, snapshot it, and tell me exactly what they ask for
   before showing a number. Do not fill anything in.

Then one table: positioning, price, who each says it is for.
Where step 1 and step 2 disagree, trust step 2 and tell me it
happened.
```

That last instruction is the point of the whole guide. Perplexity tells you what the internet says about a company. Firecrawl tells you what the company says about itself. When they disagree the primary source wins, and knowing which tool produced which claim is what lets you settle it.

BEFORE YOU CONNECT ANYTHING

- **Match the tool to the door.** OAuth server, web panel. API key, Claude Code. Local browser, extension or Code. Nothing else works, and the error messages will not tell you why.
- **Start Firecrawl keyless.** Scrape, search, and parse run with no account. Find out whether you need a key before you buy one.
- **Never put a key in a URL.** Headers only. URLs land in logs, shell history, and screen shares.
- **Name the tool in the prompt.** `perplexity_research` and `perplexity_ask` cost different money and take different time.
- **Map before you crawl.** A blind crawl on a large site spends credits on archives nobody asked for.
- **Only connect servers you trust.** A connector can serve instructions to Claude through the content it returns. Anthropic's own guidance is to connect servers built by organizations you trust and watch for tools whose behavior changes.
- **Stop before submit.** The browser holds your real sessions. Payments, account creation, and anything irreversible stays on your hands.
- **Three is the number.** Every extra connector is context spent and one more thing for the model to pick wrong.

Wire Firecrawl today, it takes a minute. Add Perplexity when you get tired of Claude guessing at the current state of something. Add the browser the first time you hit a login. Follow opusjake.ai for the rest of the stack.