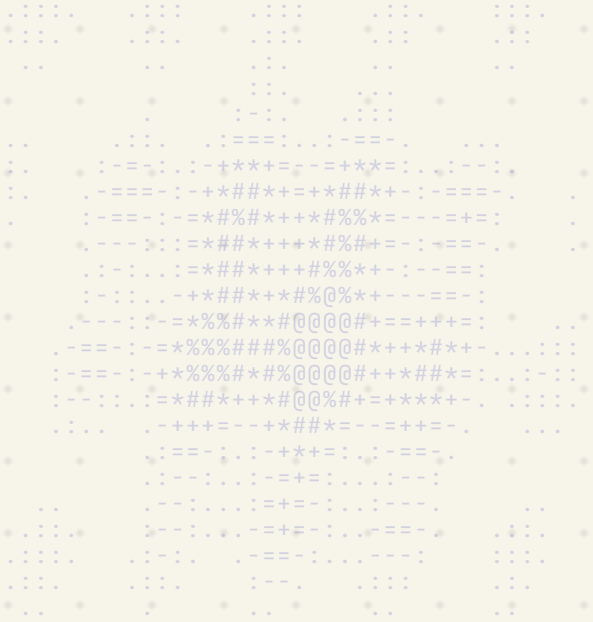
// SECTION 01 · FIELD GUIDE

THE MCP BIG THREE

Most people install twelve MCP servers and use two. These are the three that cover almost everything. Perplexity for cited answers, Firecrawl for clean data, a real browser for the rest.

```
> boot opusjake_os
> resource: mcp-big-three
> version: v1.0 · 2026
> status: ready_
```

Claude out of the box cannot see today's web. Three connectors fix that, and each one does a different job: one answers questions with sources, one turns any site into clean data, one drives a real browser with your logins already in it. Install these three and stop.



THE THREE JOBS

Most people install twelve MCP servers, use two, and slow Claude down with the other ten. Every tool you connect eats context and gives the model one more thing to pick wrong. The fix is not more servers. It is three servers that do not overlap.

Here is the division of labor. Learn it and you will always know which one to name in a prompt.

JOB	CONNECTOR	REACH FOR IT WHEN
Answer a question	Perplexity	You want a sourced answer, not a pile of pages
Turn a site into data	Firecrawl	You need the content itself, clean, at volume
Drive a real browser	Playwright	The page needs a login, a click, or a form

The failure mode is using the wrong one. Asking Firecrawl a question gets you raw markdown you still have to read. Asking Perplexity for all 400 pricing pages gets you a summary of four. Asking either one to log into your dashboard gets you nothing, because neither has a browser.

// THE ONE-LINE TEST

Ask yourself what you want back. **A conclusion** is Perplexity. **A dataset** is Firecrawl. **An action** is the browser. That is the whole decision.

01 / PERPLEXITY

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

// INSTALL IT

```
claude mcp add perplexity --env PERPLEXITY_API_KEY="pplx-your-key" \
  -- npx -y @perplexity-ai/mcp-server
```

Or connect the hosted server, no install, which also works as a Connector in the Claude apps:

```
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, so this one is not free.

// WHAT YOU ACTUALLY GET

Four tools, and the difference between them is depth, which means the difference is cost and time.

- `perplexity_search` gives ranked results with titles, URLs, and 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. The slow, expensive, worth-it one.
- `perplexity_reason` sits in the middle for problems that need logic more than lookups.

Name the tool you want in the prompt. Left to itself Claude reaches for `ask` almost every time, and you will pay for depth you never asked 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 the 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 matters more than it looks. Search results are full of marketing pages dressed as analysis, and asking for the distinction up front is cheaper than catching it later.

02 / FIRECRAWL

The extraction layer. Point it at a URL and it returns clean markdown instead of a wall of nav bars, cookie banners, and script tags. Point it at a domain and it does that for every page on it.

// INSTALL IT

```
claude mcp add firecrawl --env FIRECRAWL_API_KEY="fc-your-key" \
  -- npx -y firecrawl-mcp
```

The hosted endpoint is `https://mcp.firecrawl.dev/{YOUR_KEY}/v2/mcp`, which drops straight into the Claude apps as a Connector. There is also a keyless URL at `https://mcp.firecrawl.dev/v2/mcp` where scrape, search, and interact work with no account at all. Start there before you pay for anything.

// WHAT YOU ACTUALLY GET

The tool list is long. These five are the ones you will use.

- `firecrawl_scrape` takes one URL and returns clean markdown. Ninety percent of your calls.
- `firecrawl_map` lists every URL on a domain. Run this first, always, before any crawl.
- `firecrawl_crawl` walks a whole site. Asynchronous, so it pairs with `firecrawl_check_crawl_status`.
- `firecrawl_extract` pulls structured fields out of pages using a schema you define.
- `firecrawl_parse` handles a local PDF, DOCX, or XLSX and returns the same clean output.

Map before you crawl. A crawl launched blind on a large site burns credits on tag pages, paginated archives, and author bios you did not want, and you will not know until the bill lands.

// USE IT LIKE THIS

```
Use firecrawl_map on [DOMAIN] first and show me the URL list.

Then scrape only the ones matching [PATTERN, e.g. /pricing,
/docs/*, /blog posts from 2026]. 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.
```

Two moves make this reliable. Show me the URL list before the scrape, so you catch a bad pattern for free. Tell it what to skip, because the junk is always the same junk.

03 / THE BROWSER

The hands. This is the one that clicks, types, logs in, and works on the pages the other two cannot touch, because it drives an actual browser on your actual machine.

// READ THIS BEFORE YOU INSTALL ANYTHING

Almost every guide still tells you to install the Puppeteer MCP server. **Do not.** The Model Context Protocol team archived it on May 29, 2025. The repo is read-only, it ships no security guarantees, and it sits on dependencies nobody is patching.

The replacement is Microsoft's Playwright MCP. Same job, actively maintained, and better at the job because it reads the page's accessibility tree instead of guessing at screenshots. No API key, nothing to sign up for, free.

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

One catch worth knowing up front: this one runs locally, so it is a Claude Code and desktop-config tool. There is no hosted URL to paste into the Connectors panel, because the whole point is that the browser is on your machine with your sessions in it.

// WHAT YOU ACTUALLY GET

Two dozen tools. The shape of every session is the same.

- `browser_navigate` opens the page.
- `browser_snapshot` reads it as structured text. 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.

Useful flags: `--isolated` runs a throwaway profile with nothing persisted, `--headless` hides the window, and `--storage-state` loads saved cookies so you are not logging in every run.

// 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, e.g. filter to last 30 days and pull every row].
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 CHAIN

Each one alone is useful. The reason to run all three is that the handoffs are where the real work happens, and once they are installed Claude will make those handoffs on its own if you tell it the shape of the job.

// THE COMPETITIVE TEARDOWN

```
Three-part job, use the right tool for each part.

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

2. firecrawl_map then firecrawl_scrape: for each competitor
   you named, pull their pricing page, their homepage, and
   their most recent 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 give me one table comparing positioning, price, and who
each one 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 those two disagree, the primary source wins, and knowing which tool produced which claim is what lets you resolve it.

BEFORE YOU RUN IT

- **Three is the number.** Every extra connector is context spent and one more wrong choice available. Add a fourth only when you have hit a wall these three cannot climb.
- **Start Firecrawl keyless.** Scrape, search, and interact run with no account. Find out whether you need it before you pay for it.
- **Name the tool in the prompt.** `perplexity_research` and `perplexity_ask` cost different money and take different time. Left alone, Claude picks for you.
- **Map before you crawl.** Always. A blind crawl on a big site spends credits on archives nobody asked for.
- **The browser is not a scraper.** If a page is public and static, Firecrawl is faster and cheaper. Reach for the browser when there is a login, a click, or a form in the way.
- **Stop before submit.** The browser has your real sessions. Payment details, account creation, and anything irreversible stays on your hands, not the model's.
- **Anything Puppeteer is stale.** If a tutorial recommends it, that tutorial has not been updated since May 2025, which tells you what else in it to trust.

Install the two hosted ones first and give them a week. When you hit the first page that needs a login, add the browser. Follow opusjake.ai for the rest of the stack.