Setting up an agent team

@MINGBUILDS

How to get from one assistant to a working team of agents, then a fleet that was actually built that way and now runs outbound. The method is the same whether you are running Grok Bot, Hermes Bot, or anything else with named agents and handoff.

The method

01 Start with one agent, and make it the orchestrator

Do not create a dozen on day one. You end up with an org chart and no idea who does what.

Give it real context about you. Your work, what you are trying to achieve, what you are stuck on, what you want to be better at. Money, health, learning, whatever actually matters to you.

Then ask it: "*which agents should I create to help me achieve these goals?*"

> THE STEP ALMOST EVERYONE SKIPS
>
> Most people build the other way round and add an agent whenever something annoys them. What you get is a fleet nobody designed. It just accumulated. If you already have one, run this step as an audit instead: "*look at everything I have running and tell me what to merge, retire, or add.*"

02 One job per agent

The failure mode is the do-everything assistant that researches, writes, plans, books travel, manages email, and is mediocre at all of it.

Keep each one narrow. The orchestrator sits above them, so you never have to remember which agent you need.

```
You → orchestrator → specialist → result
```

A good test for a job boundary is whether you can write the job description in one sentence without using the word "and".

03 Split shared context from private context

Some things every agent should know. What you do, who you work with, your goals, your tools, how you like things done.

Other things belong to exactly one. Your writing voice sits with the content agent, financial detail with the finance agent, and the orchestrator will know things none of the others need.

> BOTH DIRECTIONS HURT
>
> Share too little and nobody knows anything unless you brief them again every morning. Share too much and every agent is carrying context it has no use for, which costs you tokens and dilutes what it pays attention to.

04 Connect real tools

The line between an assistant and a team is whether it *does* the work or tells you how to. Browsers, files, logged in sessions, APIs, whatever it needs.

What you want is the orchestrator taking a task, routing it to a specialist, and that agent opening what it needs, doing the work, saving the result and handing back. You see the finished output rather than the forty tabs it took.

05 Make the work visible somewhere that is not the chat

Once several agents are running at once, your chat history becomes the project manager, and it is a terrible one. Log the work into whatever tool you already use.

You want to be able to see what is in progress, who owns it, what finished, and what is stuck waiting on you. Without that you have employees and no management system.

06 Teach by demonstration rather than by prompt

Some workflows are miserable to write down. Click here, ignore that button, download it, rename it, upload it somewhere else. If your platform can watch you do it once and turn that into a reusable skill, use that instead of writing a long prompt.

This is aimed at the workflows nobody ever documented, because explaining them takes longer than just doing them. Those are the ones that never get automated.

> IF YOUR PLATFORM CANNOT RECORD
>
> Do the task alongside the agent once, correcting as you go, then have it write the procedure into a skill file itself.

07 Turn repeatable work into routines

An agent doing something once is useful. An agent that does it every morning without being asked is worth an order of magnitude more, which is why this step is where most of the value actually shows up.

Run it manually first. Correct what you do not like. Run it again. Only schedule it once it is producing something you would have sent yourself, because scheduling an uncorrected routine just automates a bad output on a timer.

> ROUTINES ROT
>
> Whatever a routine reads, a calendar, an inbox, a file, will eventually change shape. When that happens it keeps running and produces confident nonsense on schedule. Build in a periodic check that the inputs still look like what it expects.

08 Make the interface follow you

If the work runs server side rather than on your laptop, you can start something at your desk and pick it up on your phone with the same session and the same context.

The practical version of this is that your agents should be reachable where you already are, in whatever messaging app you actually check, rather than in a dashboard you have to remember to open.

09

Then, and only then, add specialists

Build the rest out once the first agent is genuinely useful. The orchestrator becomes the front door. You talk to one, it talks to the others.

> THE ACTUAL GOAL
>
> The point was never to end up with more agents. It was to end up with less that you personally have to manage. Twelve that all need supervising is worse than three that mostly run themselves, and it is an easy trap because building a new agent feels like progress.

A worked example: five agents that run outbound

Everything above is method. This is one fleet built that way, running now. Every choice in it maps back to a numbered step.

The job is business development outbound: find the right people, reach them where they are, keep the quality up at volume, and notice when someone is actually ready to talk. Five agents, one job each.

Each role below says what kind of access it needs, with what I use underneath as one example. Take the requirement, not the vendor. The shape worth copying is that **four of the five need access to something, and the fifth needs context.**

ROLE 01 · ORCHESTRATION

BD Lead

Owns the process end to end. Defines the brief, delegates to the other four, and produces the weekly report. Everything downstream is only as good as how well this one is briefed.

WHAT IT NEEDS

No tooling. The most complete written context you can give it: what you do, who you sell to, what you are selling, and what a good customer looks like with real examples rather than a description.

My context. *A long first conversation, not a config file. Everything downstream inherits it.*

ROLE 02 · SUPPLY

Prospecting Penguin

Finds leads against the ICP, writes them to the CRM, then goes back over them for enrichment.

WHAT IT NEEDS

ROLE 03 · CONTACT

Outreach Panda

Writes the copy and makes contact on Telegram, email or LinkedIn, whichever the enrichment turned up.

WHAT IT NEEDS

Two things: a data source it can query for enrichment, and authenticated access to wherever your buyers are actually visible. Without the second it is guessing at names.

My context. *The RootData API, plus my own LinkedIn through browser exec.*

Send access on each channel you actually use, and something that keeps the drafting from sounding generated. What goes out at volume should not read like it went out at volume.

My context. *A humanizer skill for the templates.*

ROLE 04 · CONTROL

Quality Capy

Checks the templates and the outgoing copy. Its only job is that quality does not drift once volume starts.

WHAT IT NEEDS

Nothing of its own, on purpose. The templates, the outgoing copy, and the brief to check them against. A checker with its own data sources starts having opinions about the strategy instead of the writing.

ROLE 05 · SIGNAL

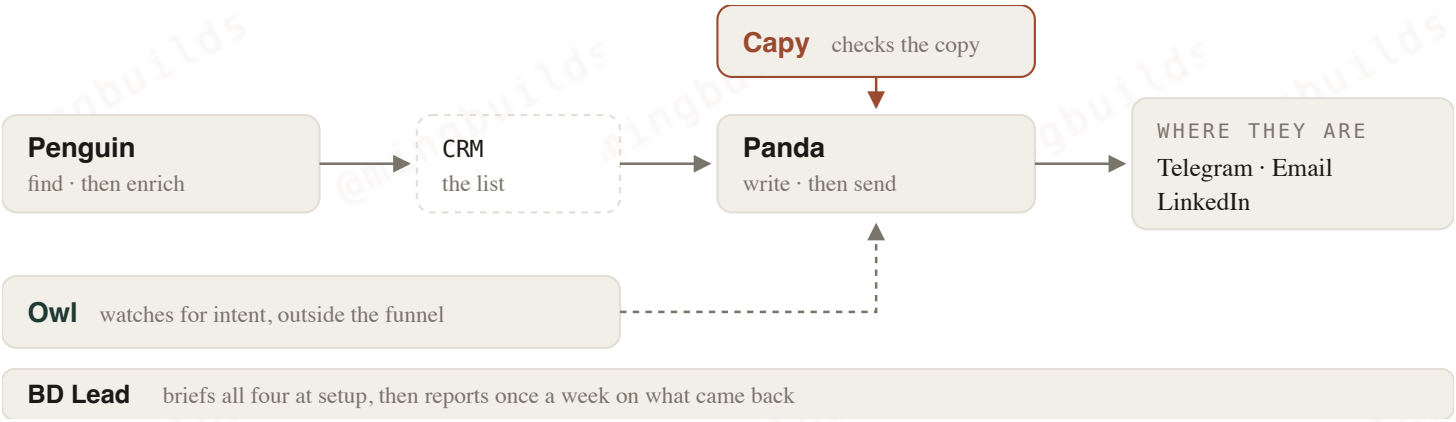
Buying Intent Owl

Always on, watching for buying intent across public channels. The only one that runs independently of the funnel.

WHAT IT NEEDS

Two feeds, and they are different in kind. One watching where your buyers post publicly, and one watching where they complain. Intent shows up as a signal in the first and as a pain point in the second.

My context. *Elfa's Auto with the skill attached, tracking X accounts across web3. Agent reach for specific subreddits.*



Setup, in this order

The order is doing the work. Step four is not optional.

1. **Define the ICP and what you are actually selling**, with the BD Lead, in as much detail as you can stand. Vague in, vague out. Every downstream agent inherits this brief.
2. **Have the BD Lead fully delegate and brief each agent.** You brief one; it briefs the rest.
3. **Get rough templates written by Panda, then verified by Capy before anything sends.** The template gets adapted per ICP, so a flaw here multiplies across every message.

4. **Run a full test pass before automating anything.** Watch one lead travel the whole way through.

The loop, once it is live

This runs without you. The weekly report is where you re-enter.

1. **Penguin** finds leads against the ICP and writes them to the CRM. Enrichment runs as a second pass, after the list exists.
2. **Panda** crafts the message from template plus lead, then reaches out on whichever channel the enrichment actually found.
3. **Capy** checks the template and the outgoing copy continuously. Volume is when quality slips, so this is not a one-time gate.
4. **Owl** watches for intent the whole time, independently of whatever the funnel is doing.
5. **BD Lead**, weekly: leads added, leads contacted, what came back, and what to change next week because of it.

> THE DETAIL PEOPLE GET WRONG
>
> **Prospecting first, enrichment second.** Enriching while you search means you go deep on leads that should never have made the list, and the cost lands before the filter does. Two passes is slower to describe and cheaper to run.

Where each choice came from

The fleet is the method applied.

THE CHOICE	STEP	WHY IT FOLLOWS
BD Lead is the only one you talk to	1 · 2	One front door. You never have to remember which agent you need.
Capy is separate from Panda	2	One job per agent, and **nothing marks its own homework well.** A writer that checks its own copy is one agent with two jobs and a conflict.
Leads live in a CRM, not in chat	5	The work has to be visible somewhere that is not the transcript. Otherwise chat history becomes your project manager.
Templates verified before any sending	7	Nothing gets scheduled until it has been run and corrected manually. A template flaw multiplies across every message.
Enrichment as a second pass	2 · 4	Different jobs, different tools. So, different passes.

Weekly report, not a dashboard	5 · 8	They reach you where you already are, and the report is where you come back in.

> WHAT THIS DOES NOT SOLVE
>
> It gets messages out reliably, and that is a real problem solved. **What it does not tell you is whether the person on the other end was ready to hear from you**, and volume without timing is just a faster way to get ignored. That is what the Owl is reaching for, and it is comfortably the hardest of the five, because everything upstream of it is mechanical and intent is not. **Expect the first four roles to work quickly and the fifth to take real effort.**

Platform notes

What differs between platforms is which capabilities you get for free and which you end up building.

Grok Bot (xAI)

Each bot gets a persistent cloud computer with a browser, filesystem and terminal. They sign into your tools with your own credentials and coordinate in a group chat. Access comes bundled into the existing subscription tiers. Strongest on computer use and on learning a task from watching you do it.

Hermes Agent (Nous Research)

Open source, out since February 2026, and the closest open equivalent. Bot Mode swaps the single session list for a roster of named bots. Each one is a full profile with its own isolated memory, its own skills and a pinned model, and they pass work between themselves by @mention and coordinate in group chats. It shipped bundled and default on in the v0.20.x line, and the latest release is v0.20.5, dated 19 August 2026. The v0.20.0 "Herald" release on 3 August added agent to agent communication (A2A v1.0), real time voice with streaming TTS, signed outbound webhooks, and grounded research with verifiable citations.

> VERIFY BEFORE DESIGNING AROUND IT
>
> Bot Mode's detailed behaviour is not yet fully documented in the public release notes. The memory isolation and model pinning described above come from the announcement and secondary coverage, with the full notes expected at v0.21.0. Check the repo before you build on any specific behaviour.

Because it is open source and self hostable, this is the one where pinning a model per agent actually pays for itself. Run the mechanical agents on something cheap and save the expensive model for the ones making judgement calls.

Claude Code and cron-based setups

Subagents give you specialisation, scheduled jobs give you routines, and a vault gives you shared memory. Two things are usually missing. There is no persistent per agent memory, so you reload context on every run, and scheduling is session scoped, so jobs die when the session does. Both are fixable with files and a boot routine, but they are the parts you build rather than configure.

If you already have a fleet, this is a migration

Which is a harder problem than the one above. You cannot start with one agent when eleven are already running against live dependencies.

1. Inventory everything first. For each job: what triggers it, what it reads, what it produces, and who uses the output.
2. Ask the orchestrator to critique that list rather than design a new one. Merge, retire, promote.
3. Look for the missing arbitration layer. Almost every accumulated fleet is short the same thing, something whose job is to look at the whole set and reason about its shape. Jobs get added. Nothing ever asks whether the set is still right.
4. Migrate the noisiest job first rather than the most important one. You want the fastest possible signal on whether the new structure is actually better.

> KEEP THIS RULE FROZEN WHILE YOU MIGRATE
>
> A routine that has never been corrected by hand does not get scheduled.

`@mingbuilds`

Platform details verified against primary sources on 23 August 2026. Release numbers move quickly, so re-check before relying on a specific version.