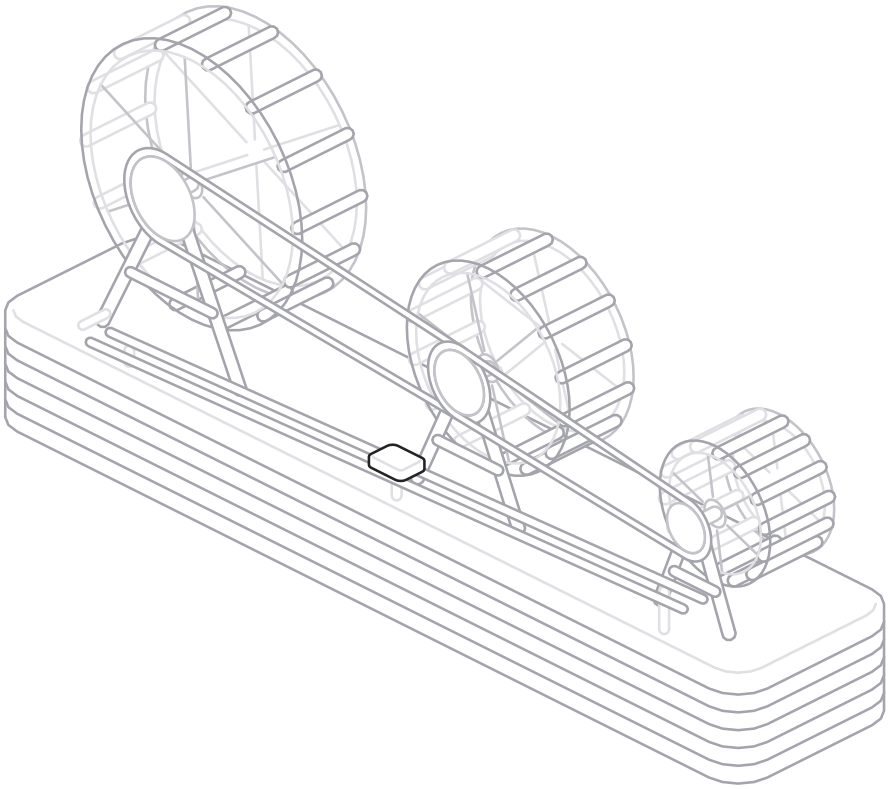
Hamster

GUIDE

Intent-driven development on Hamster

How a team goes from an empty workspace to agreed workstreams running in parallel.

For engineering leaders and tech leads whose teams already use Claude Code, Cursor, or Codex.



Eyal Toledano, Founder, Hamster

October 5, 2026

tryhamster.com/guides/intent-driven-development

Contents

	In short	3
01	What is intent-driven development?	4
02	How do you set up the workspace and connect your repositories?	5
03	How do you connect Claude Code, Cursor, and Codex to Hamster?	7
04	How does a team write its first Brief together?	9
05	How do alignment votes work?	11
06	How do you turn an agreed Brief into a Plan and Tasks?	12
07	How do agents deliver the Plan, in the cloud or in your own tools?	13
08	How do sessions write decisions back as they work?	15
09	How do you review the PR and close the loop?	17
10	What should run on its own?	18
11	Where do Goals and Initiatives fit above Briefs?	20
12	A week-one and month-one rollout	21
13	How can you tell it is working?	22
	Common questions	23

Web version, with the templates to copy: tryhamster.com/guides/intent-driven-development

In short

Intent-driven development is a way of working in which a team runs direction, discovery, and delivery as connected loops on one shared record of its intent, decisions, and knowledge, so agents build what the team agreed and each person can run several workstreams at once at the highest level of abstraction.

- In Hamster, direction lives in Goals and Initiatives, discovery in Briefs, research, and alignment votes, and delivery in Plans, agents, and pull requests, with the Context Graph, Blueprints, and Skills and Methods underneath, and each loop's output feeds the next.
- Teams connect Claude Code, Cursor, or Codex by installing the Hamster plugin, running its setup skill once per repository, and signing in to the MCP server at https://tryhamster.com/mcp.
- Each person whose work a Brief changes votes Ready or Not yet, and because Hamster's pre-delivery checks do not include votes, the team sets its own rule for waiting on agreement.
- Generate Plan builds parent Tasks with subtasks and acceptance criteria from the Brief, its Spec, the team's Blueprints and Methods, and the connected code.
- One Brief runs one delivery at a time, so parallel work comes from running several Briefs at once in Hamster Cloud, Cursor Agents, or the team's own tools.

Intent-driven development is a way of working for teams whose agents do the building. It runs three loops: direction decides what matters, discovery agrees on what to build, and delivery turns that agreement into merged work. Under all three sits one shared record of the team's intent, decisions, and knowledge, so agents build from what the team agreed and each person can direct several workstreams at once, at the highest level of abstraction.

It is a loop of loops. Each loop's output feeds the next, and every turn writes back to the record, so the next turn starts from more of what the team already knows. Hamster is built around it. "Before the agents start" explains why the practice matters; this guide sets it up in Hamster as a short sequence of steps, from an empty workspace to agreed workstreams running in parallel.

Every step has something to click or run and a result you can check on screen, and each one leaves the team a decision that Hamster keeps on record.

01

What is intent-driven development?

Intent-driven development is a way of working in which a team runs direction, discovery, and delivery as connected loops on one shared record of its intent, decisions, and knowledge, so agents build what the team agreed and each person can run several workstreams at once at the highest level of abstraction. Every agent session reads that record and writes its decisions back, so the next piece of work starts from everything the team already decided.

- **Direction** decides what matters. Goals hold what the team wants to move, and Initiatives hold the commitments that move it. In Hamster this is Goals and the Initiatives under them.
- **Discovery** works out what to build. Research, prototypes, and the Brief's own conversation turn an Initiative into a Brief, and alignment votes record who agreed. In Hamster this is Briefs, research agents, and Ready or Not yet votes.
- **Delivery** turns an agreed Brief into merged work. Hamster generates the Plan and its Tasks, agents build them in Hamster Cloud or in your own tools, and the team reviews the pull request against the Brief. In Hamster this is Plans and Cloud Agents.
- **Knowledge** is the context under all three. Briefs, decisions, votes, conversations, and merged changes land in the Context Graph, beside the Blueprints that describe the product and the Skills and Methods that describe how the team works. In Hamster this is the Context Graph, Blueprints, and Skills and Methods.

Each loop feeds the next. An Initiative becomes the Briefs that deliver it, an agreed Brief becomes a Plan, and the merged work flows back into the record with the decisions made along the way. The next Initiative and the next Brief start from that record, so the team stops re-arguing settled decisions and each turn of the loop goes further than the last.

Hamster names each part of the work.

- **Brief.** The team's shared statement of what to build and why. Several people shape it, vote on it, and argue in it, and agents execute from it. A Brief is a multiplayer conduit for intent, not a PRD.
- **Alignment vote.** A teammate's Ready or Not yet on a Brief. A Not yet vote needs a written explanation. The latest vote from each person counts, and the Brief's Activity tab keeps the full history.
- **Technical Spec.** The engineering companion to a Brief, on its own Spec tab, where open technical questions get answered and decisions are kept.

- **Plan.** The breakdown Hamster generates from the Brief and its Spec. Parent Tasks carry subtasks with descriptions, scope boundaries, and acceptance criteria.
- **Delivery.** A run that works through the Plan in Hamster Cloud or in your own coding tool, and ends in a pull request or a pushed branch.
- **Context Graph.** The live, connected record behind every answer Hamster gives, linking Briefs, Tasks, conversations, decisions, votes, documents, and code.

The walkthrough below follows the loops. Setting up the workspace and connecting your agents builds the Knowledge layer. Writing a Brief and collecting votes is Discovery. Planning, delivering, and reviewing is Delivery, and writing decisions back feeds Knowledge again. Routines keep the loops turning on their own. Goals and Initiatives are Direction, and they come last here because a team can start delivering before it sets them up.

02

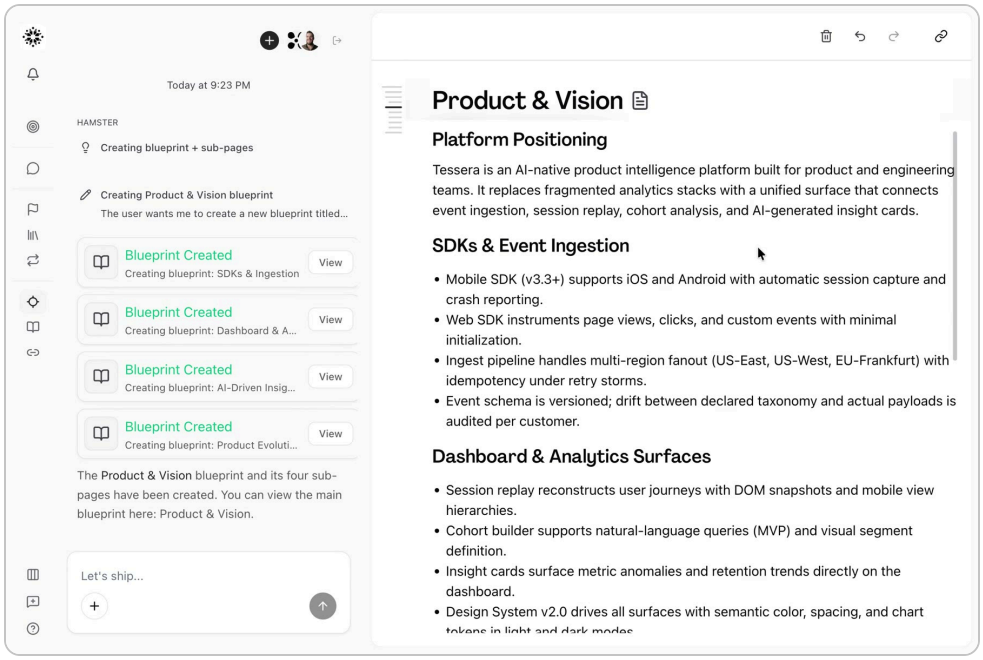
How do you set up the workspace and connect your repositories?

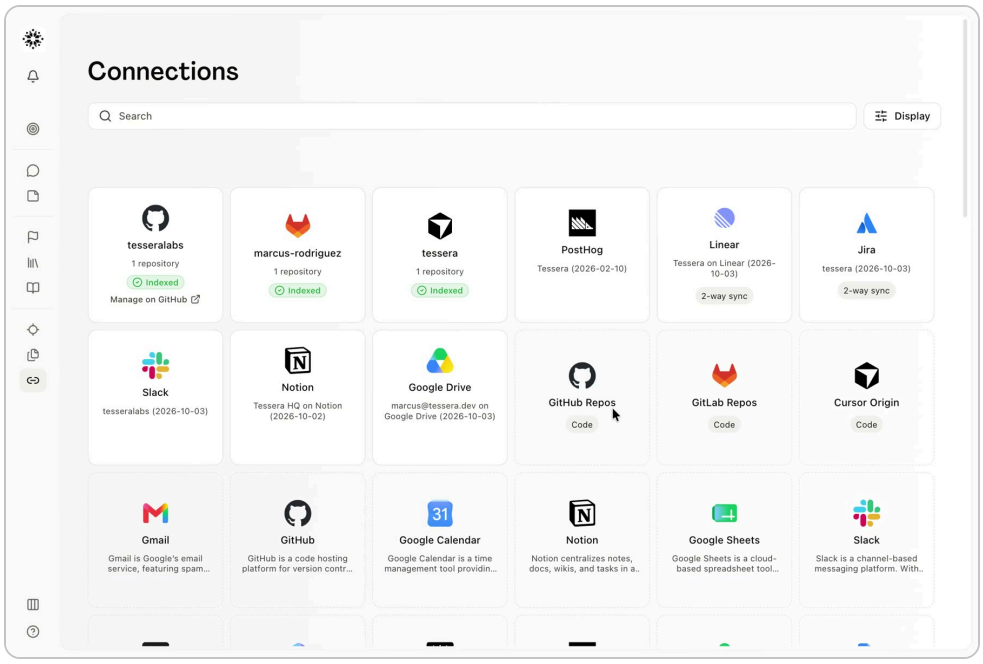
Sign up, walk through the onboarding wizard, then connect the repositories your team ships from under **Knowledge > Connections**. Hamster indexes each repository's tracked branch, so every Brief and Plan can read your real code.

The wizard shows only the steps that apply to you. In a new workspace, **Set your direction** takes your company website and any strategy decks, and Hamster drafts your first Blueprints from them while you finish. **Choose the methods Hamster will use** keeps the core Direction, Discovery, and Delivery methods on. Invite people at **How big is your team?** Reviewer seats are free and can vote on alignment. Invite everyone whose work your Briefs will change, including people who never open a coding tool.

Connect code next. Hamster supports three code hosts side by side, each as its own connection with its own repository list.

1. **GitHub.** Click the **GitHub Repos** tile and choose **Install GitHub App**. On GitHub, pick the organization and the repositories. Back in Hamster, choose the branch to track for each one.
2. **Cursor Origin.** Click the **Cursor Origin** tile and choose **Install Origin App**. Pick repositories on Origin's install page, and Hamster brings you back to its repository picker.
3. **GitLab.** Click the **GitLab Repos** tile and connect a gitlab.com project.





Each card shows its progress as **Indexing**, then **Embedding**, then **Indexed**. The free plan indexes one repository across all three hosts, and paid plans index up to 100. Track the branch you develop on, usually `main`, because Hamster refreshes the index after each push to that branch. The GitHub and Origin setup pages list the permissions each app requests.

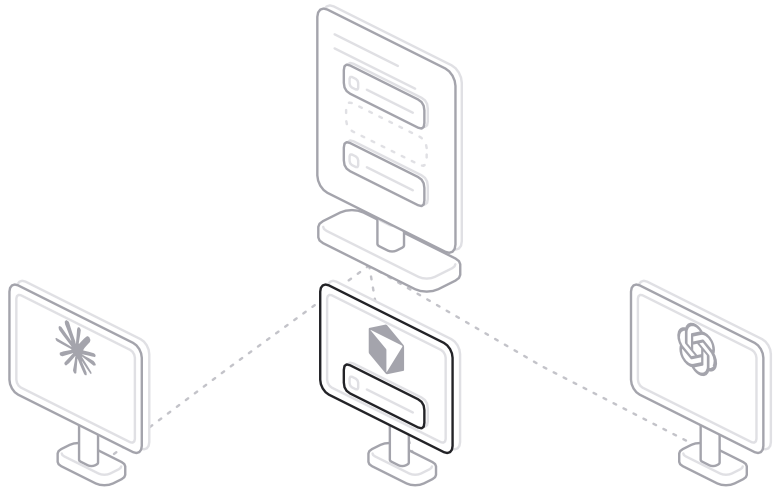
Decide which repository goes in first. Start with the one your first workstream touches. Add Slack and your issue tracker once people want to vote from Slack or keep Linear or Jira in step with Hamster.

Hamster works alongside Linear or Jira rather than replacing them. With two-way sync on, the Linear connection keeps Briefs in step with Linear projects and Tasks with Linear issues, in both directions, within seconds. The team can keep running standups in Linear while it plans in Hamster. A new Linear connection starts with sync off, so connect it for context first and turn sync on once the team trusts the mapping. The Jira connection offers the same optional two-way sync between Briefs, Tasks, and Jira issues and epics.

03

How do you connect Claude Code, Cursor, and Codex to Hamster?

Install the Hamster plugin in each coding tool your team uses, run its `setup` skill once per repository, and sign in to the MCP server. Product managers and tech leads plan the work in Hamster as a Brief and a Plan, and Claude Code, Cursor, and Codex pick up the Plan's Tasks through the plugin and write their decisions back.



The plugin connects your tool to Hamster's hosted MCP server at `https://tryhamster.com/mcp` and adds eight skills. Its `setup` skill installs the `hamster` CLI if it is missing and signs it in. It then runs `hamster init` to link the repository to your team, followed by the first sync. From then on the CLI keeps Briefs, Plans, Blueprints and Methods as Markdown in the repository's `.hamster/` folder, where the plugin's Ship skill reads them.

TEMPLATE **Install the Hamster plugin**

Claude Code (slash commands inside a session)
/plugin marketplace add gethamster/plugin
/plugin install hamster@hamster-plugins
/reload-plugins
/hamster:setup
Then open /mcp, select plugin:hamster:hamster, and sign in.

Codex (first two lines in your terminal)
codex plugin marketplace add gethamster/plugin
codex plugin add hamster@hamster-plugins
$hamster:setup
codex mcp login hamster

Cursor
Customize > Add Marketplace > Import from GitHub
Paste https://github.com/gethamster/plugin at User scope and select Import.
Open the Personal tab and select Add next to Hamster.
/setup
Follow Cursor's sign-in prompt for Hamster.

Any tool, one prompt
Install Hamster in this repository. Fetch and follow https://tryhamster.com/plugin/install

Check the result in your terminal
hamster status

`hamster status` should report that the CLI is signed in and synced, and the repository should now have a `.hamster/` folder. Skills start as `/hamster:<skill>` in Claude Code, `/<skill>` in Cursor, and `$hamster:<skill>` in Codex. The plugin page has the steps for Antigravity, Copilot CLI, Grok Build and Pi.

A tool that does not load plugins can connect the MCP server alone. It gets the MCP tools, including `ask_hamster`, without the plugin skills or the synced `.hamster/` folder. During a long session, run `hamster sync --watch` in a side terminal, and the `.hamster/` folder stays current as teammates edit.

The Claude Code page shows the same flow from an engineer's seat. The decision here is who installs the plugin. Install it for every engineer, including those who never ship from their own machine, because the plugin also lets a session ask Hamster which decisions apply to its branch.

```
Claude Code v2.1.281
Opus 5.5 (1M context) · Claude Max
~/tessera-app

Get to finished work sooner with Opus 5.5. Switch anytime with /model.
1 more notice hidden

> /plugin
  └ ✓ Installed Hamster. Plugin is now active.

> /plugin

                                                  ◐ medium · /effort
>
▸▸ auto mode on (shift+tab to cycle) · ← for agents
```

Try it with your AI

Your coding agent can run this guide with you. Paste one line into Claude Code, Cursor, or Codex, in the repo your team works in.

```
Read https://tryhamster.com/guides/intent-driven-development/skill.md and run it with me.
```

1. It connects Hamster and the plugin in your coding tool. You sign in once in the browser.
2. It checks the repository connection and adds the session instructions to AGENTS.md or CLAUDE.md.
3. It drafts the first Brief from your repo, creates it with a decision log, and tells you who needs to vote.
4. It writes decisions back as it works, and plans or delivers only when you ask.

04

How does a team write its first Brief together?

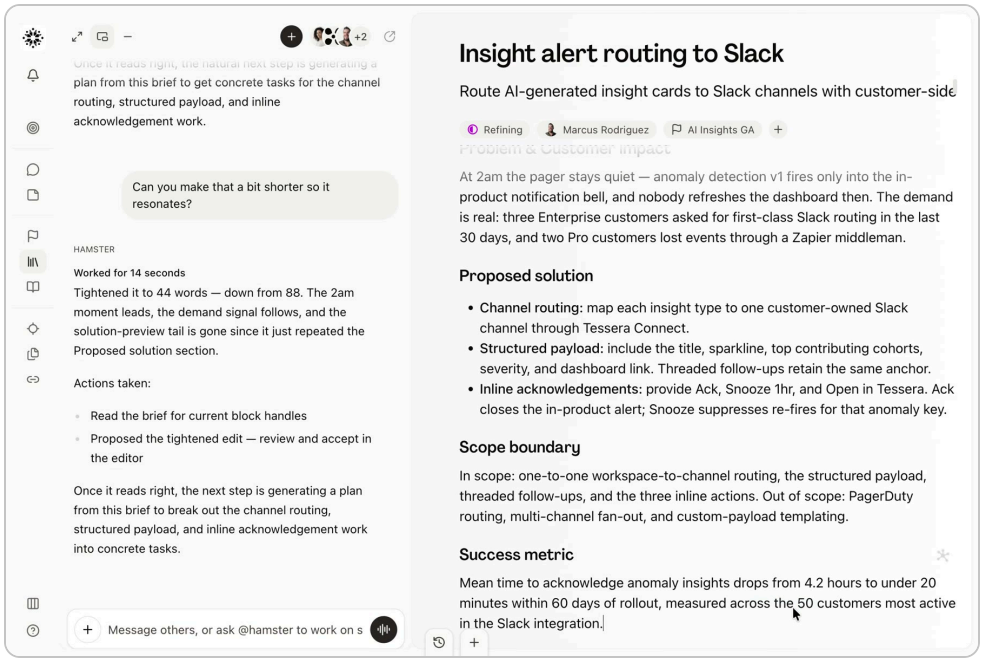
Open Hamster Chat and answer its first question, “What are you trying to achieve?”, in your own words. Hamster captures the Goal, shapes an Initiative for it, and drafts the first Brief in a thread of its own, which the team then edits together.

You can also click **New Brief** in the Briefs list and start from a title and a one-line description. Either way, the editor opens with the Brief's chat docked beside it. The chat can read everything in the Brief, and every change saves as you type.

A Brief that Hamster drafts uses the five sections from the first guide's template: Context, Goals, Phases / Approach, Scope, and Next Steps. Keep the Brief about what and why. Architecture, data choices, and stack decisions go on the Spec tab, which keeps the Brief readable by everyone whose work it changes.

Hamster's own sizing guidance is a Brief of half a day to a day of work, enough to generate a Plan and ship one pull request. When a draft implies many deliverables or more than a few days, the chat proposes an Initiative with smaller child Briefs.

Bring people in while the Brief is still rough. Real-time editing shows each person's cursor and changes live, and an @mention pulls a teammate into a section. Hamster's own edits arrive as suggestions you accept or reject. Paste in the customer call, the Figma file, and the Slack thread before you ask for a draft, because Hamster grounds the draft in what is attached.

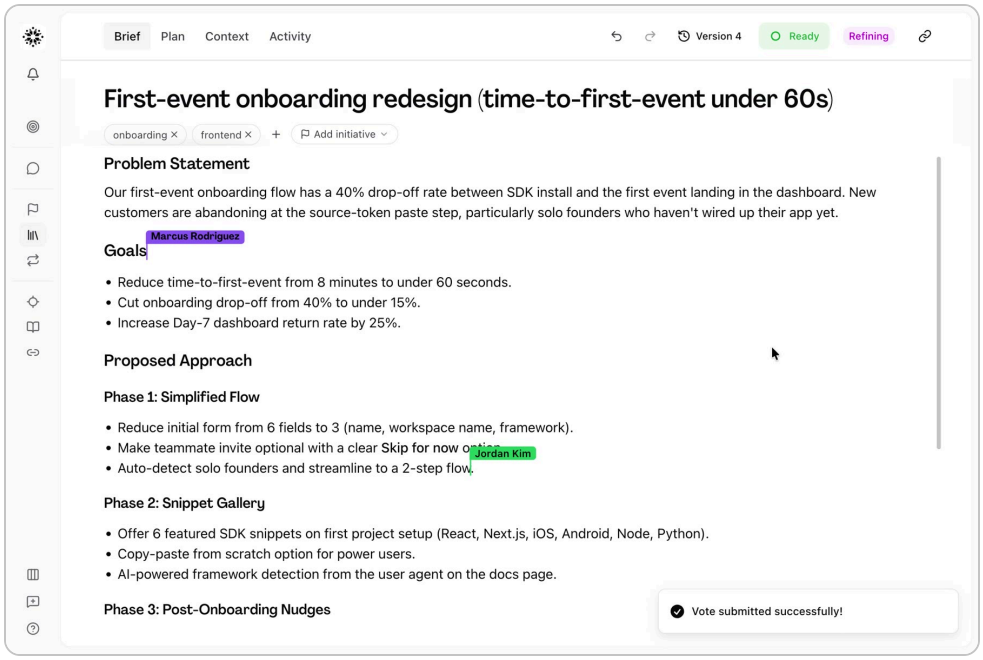


Two decisions belong to this step. Name one owner, the person who answers for the outcome after agents deliver. Then name the people who must vote, which means everyone whose work the Brief changes.

05

How do alignment votes work?

Each person whose work the Brief changes opens it and votes **Ready** or **Not yet** from the **Ready?** control in the header. A Not yet vote needs a short explanation, and the Brief's Activity tab lists every vote, concern, and status change. Each person's latest vote counts. A vote is not tied to a version, so editing the Brief does not reset anyone's vote.



A lead reads alignment on the Activity tab. The current version sits at the top with a circular indicator, green when everyone is aligned and yellow while concerns are outstanding. The Concerns section lists who flagged what, in their own words. Older versions stay collapsed below with the votes and concerns recorded while each was current, so you can check whether a concern raised during version 3 was settled by the edit that made version 4.

People do not have to open Hamster to vote. With the Slack connection, Brief cards in Slack let teammates mark a Brief Ready or Not yet, and each vote lands on the same Activity timeline.

Hamster's pre-delivery checks cover the Plan, the repository, the Spec, and any delivery already running. Votes are not among them, so the team sets its own rule for waiting on agreement and keeps it. Write the rule down once and pin it where the team works.

TEMPLATE **Alignment rules**

Who votes: everyone whose work this Brief changes, plus the owner.
Reviewers vote too. Reviewer seats are free.
A Not yet vote always says what would change it.
The owner answers each concern in the Brief chat or with an edit.
After an edit to Goals or Scope, the owner asks every voter to vote again.
Nobody generates the Plan until every named voter is Ready.
If the owner overrules a concern, the Spec records who decided and why.

06

How do you turn an agreed Brief into a Plan and Tasks?

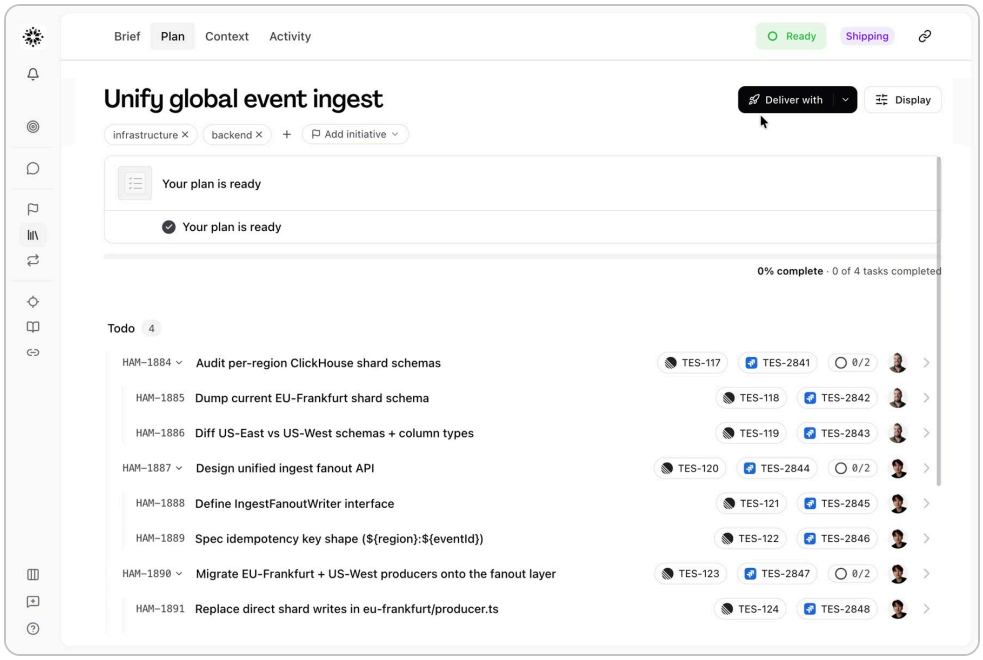
Settle the Spec, then click **Generate Plan** on the Brief's Plan tab. Hamster reads the Brief, its Spec, your Blueprints and Methods, and the connected code, then builds parent Tasks and elaborates each one into subtasks with acceptance criteria.

Start on the Spec tab. Hamster lists the open technical questions it found and the decisions that need direction before the work can be planned. Answer them there, because the Spec carries its settled direction into the Plan. The Spec can also reuse decisions settled in related work instead of asking the same questions again. Brief owners get a notification when a Spec review needs answers.

Then generate the Plan. A progress card shows each phase as it runs: **Analyzing Brief**, **Generating Tasks**, **Processing Tasks**, and **Generating subtasks**. Parent Tasks appear first, and subtasks fill in as Hamster elaborates each parent in parallel. A complex Plan can take a few minutes, and you can leave the page while it builds.

Read the Plan before anyone delivers it. The Brief's chat stays docked beside the Plan tab. Ask it to split a Task, add a missing subtask, or rewrite an acceptance criterion, and Hamster edits the Plan in place. Drag Tasks to change their order, and delivery follows the new order. **Regenerate** clears every Task and builds a fresh Plan from the current Brief, so save it for a real change of intent.

The decision at this step is whether the Plan says what the team agreed. If a Task appears that nobody voted for, fix the Brief and regenerate, and leave the Plan's scope to follow the Brief's.



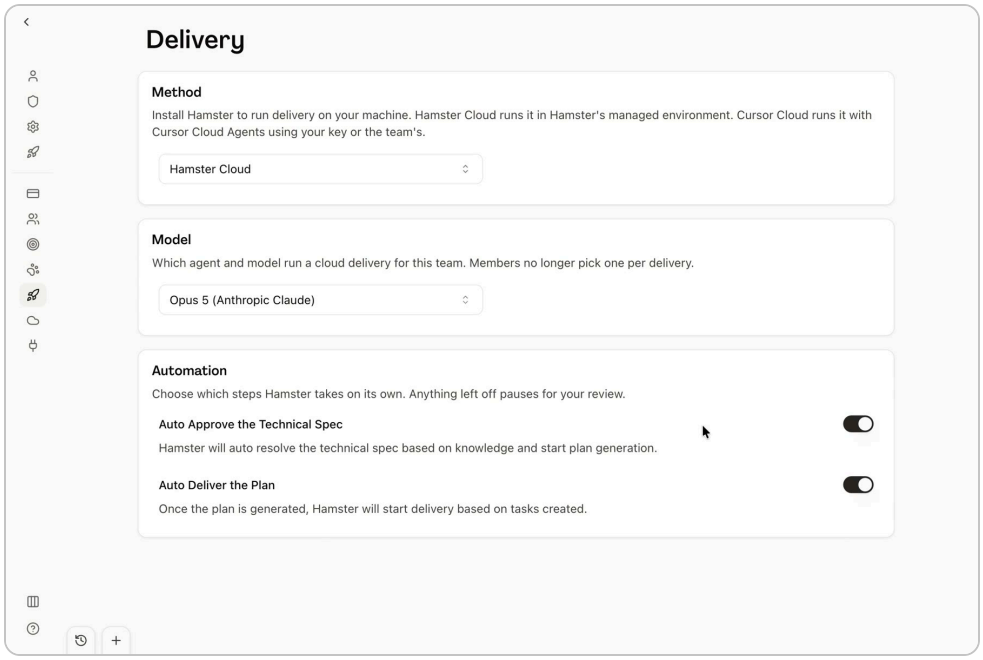
07

How do agents deliver the Plan, in the cloud or in your own tools?

Click **Deliver** on the Brief, or open the chevron beside it to choose a method: the Hamster plugin in your own checkout, Hamster Cloud, or Cursor Agents. Every method works from the same Brief, Spec and Plan.

The first time anyone on the team clicks **Deliver**, Hamster asks how the team delivers. Owners and admins can change the method, the model, and the automation later in **Settings > Delivery**. Two automation switches matter most. **Auto Approve the Technical Spec** answers open Spec questions with the recommended options, and **Auto Deliver the Plan** starts delivery without a pause to review the Plan. Turn both off while the team is learning, so each pause becomes a decision someone makes on purpose.

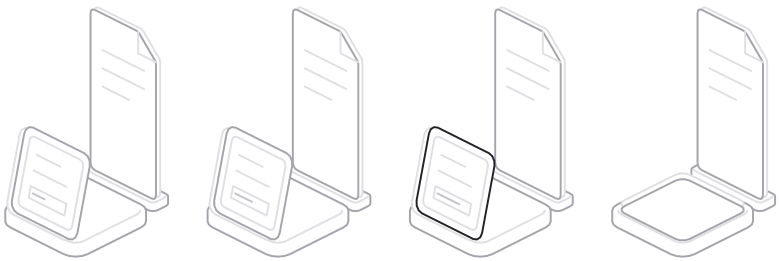
Hamster Cloud runs in a saved environment called a Cloud Agent. Open **Cloud Agents** in your team settings, click **New**, attach repositories, and click **Set up environment**. Hamster works out the Install, Build, Test, and Run commands and checks them in a fresh sandbox. If it needs a credential, it pauses and asks. When the Cloud Agent shows **Active**, anyone who can manage documents can deliver into it. Cloud Agents come with the Team and Enterprise plans, and the Cloud Agents docs cover environment variables and resets. A Hamster Cloud delivery opens a draft pull request on GitHub or Origin, or a merge request on GitLab, and links it to the Brief.



Your own agents run the plugin's Ship skill. Type `/hamster:ship <brief>` in Claude Code, `/ship <brief>` in Cursor, or `$hamster:ship <brief>` in Codex. Ship groups parent Tasks that do not overlap into waves that run in parallel. Each parent Task gets its own commits, and a reviewer pass checks each wave's combined changes. Ship asks before every push and before it opens a pull request.

Cursor Agents hands the delivery to Cursor's agents through a connected Cursor API key, for repositories on GitHub. For a repository hosted on Origin, use Hamster Cloud.

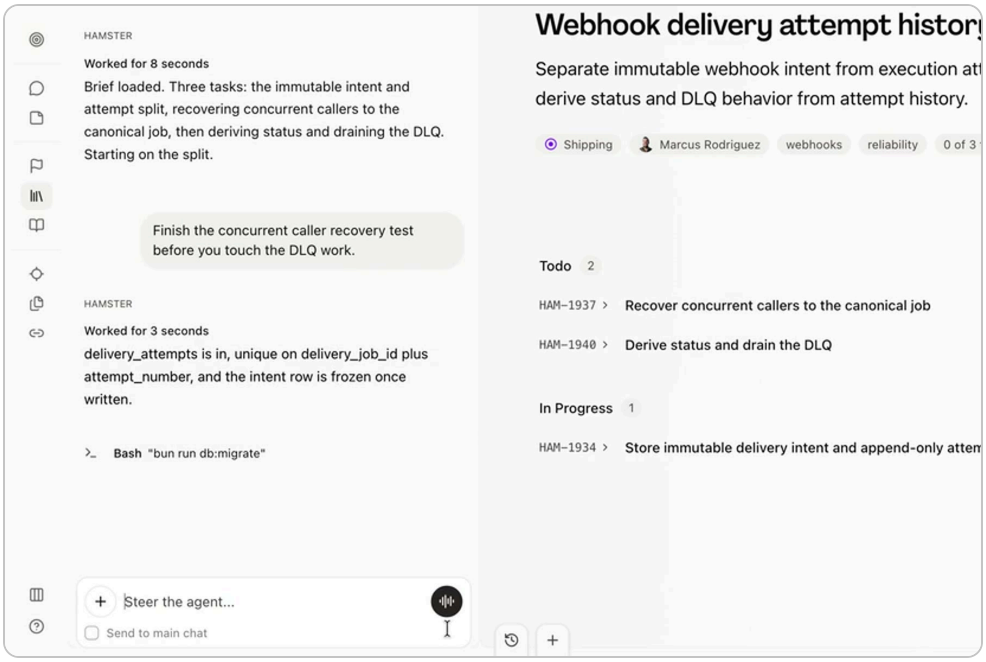
Before a hosted delivery starts, Hamster checks that the Plan has Tasks and that a repository is connected. It also checks that the Spec meets your approval settings and that no other delivery of this Brief is running. One Brief runs one delivery at a time, so parallel work comes from running several Briefs at once. Two Briefs aimed at different Cloud Agents run side by side, and two Briefs aimed at the same Cloud Agent each get their own sandbox.



While a delivery runs, its thread streams every step as it completes, from environment setup to files read, code written, and checks run. Send a message in that thread to steer the work.

Hamster holds the message until the current turn finishes, then passes it into the coding session at a safe boundary, and the delivery continues without restarting. **Stop** ends the run, and a new message resumes it. If the Plan includes a Person or Tool Task, delivery pauses there and resumes once that Task is done.

The team still has to choose which work goes to the cloud. Hamster's guidance keeps engineer-driven work in the IDE with the plugin, where an engineer wants to follow the agent's choices as they happen. Work a product manager starts, work a Routine starts, and parallel work across the team go to Hamster Cloud.



08

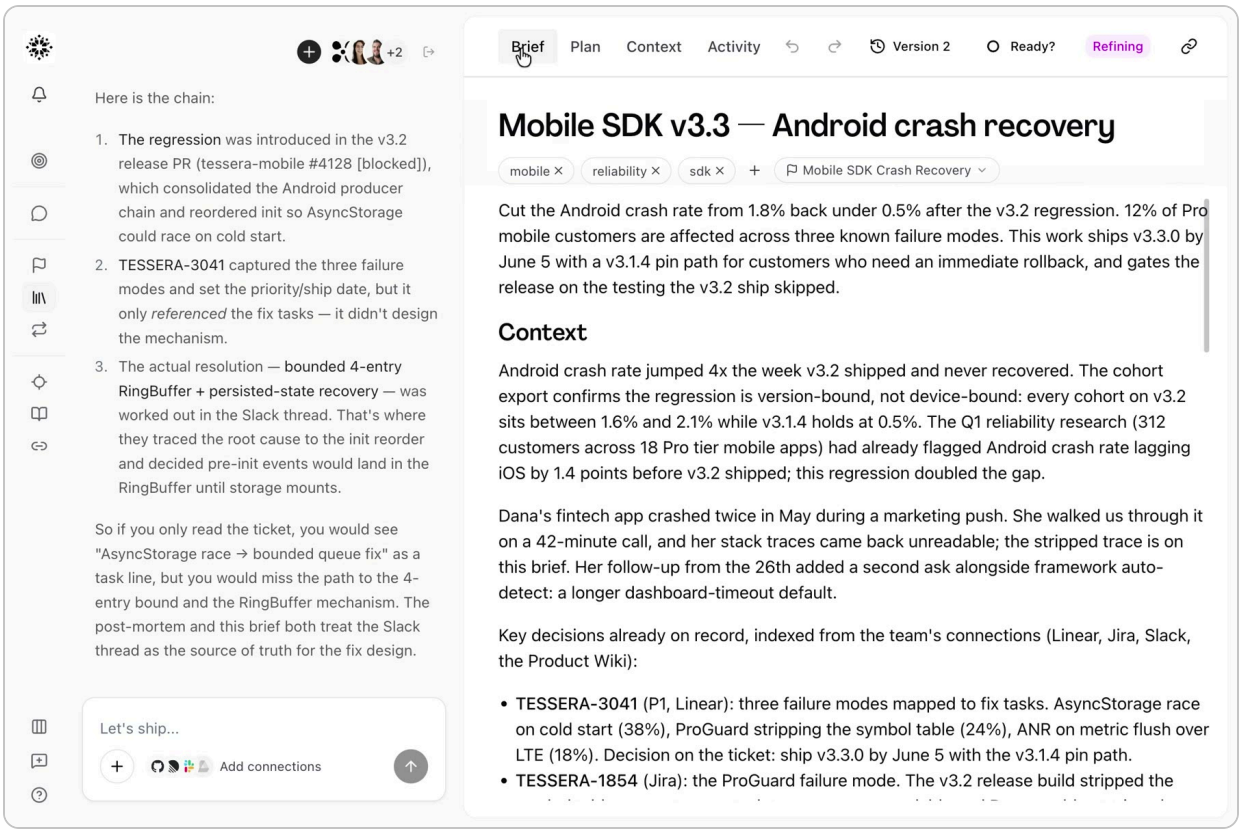
How do sessions write decisions back as they work?

Tell every session to read the current Brief before each step and to record each decision in Hamster through the MCP tools. Hamster captures its own actions automatically, and session instructions capture the decisions a coding agent would otherwise lose when its session ends.

Much of the record fills itself. The Context Graph links each Brief to its Initiative, its Tasks, its votes, and the conversations that produced its decisions. It updates within seconds of an edit, and every note, Brief edit, and Task change a session writes flows into it. A delivery's transcript stays readable after the run, as a record of what was built and why. Hosted deliveries update Task statuses and link their own pull requests, and Hamster Cloud pull requests show where the work departed from the Plan.

Sessions in your own tools need instructions, because a coding agent keeps no memory of its decisions after the session ends. The MCP server gives it the tools.

- `get_brief` reads the current version of a Brief.
- `update_task_status` moves a Task to `todo`, `in_progress`, or `done`.
- `create_note` and `update_note` keep a decision log as a note.
- `update_brief` changes a Brief's content, status, or owner.
- `link_pull_request` ties a pull request opened from a laptop to its Brief.
- `ask_hamster` asks Hamster a question with the branch as context, for example "Which decisions apply to this branch?"



Put the block below in the repository's `AGENTS.md` or `CLAUDE.md`, and every session in Claude Code, Cursor, and Codex starts with it.

The decision log note uses the columns from the first guide: date, decision, alternatives rejected, decided by, reversible, and workstreams affected. Keep one note per Brief, titled `Decision log: <Brief title>`, so every session can find it by name.

TEMPLATE **Session instructions for AGENTS.md or CLAUDE.md**

Working from Hamster

This repository is delivered from Hamster Briefs. Hamster is connected through MCP.

- Before each step, read the Brief again with get_brief. Build from the current version.
- Before you start a Task, move it to in_progress with update_task_status.
- When a Task is finished, move it to done.
- When you or the user make a decision, add a row to the Brief's decision log note with update_note.
- Each row names the alternatives you rejected and the person who confirmed the decision.
- If you made a decision alone, ask the user to confirm it before you log it.
- If a decision changes the purpose or scope, tell the user. Do not edit Goals or Scope yourself.
- Put technical decisions on the Brief's Spec, not in the Brief body.
- When you open a pull request, link it to the Brief with link_pull_request.
- If you are unsure which earlier decisions apply, ask with ask_hamster before you build.

The lead decides how much a session may change on its own. These instructions let a session log decisions and move Tasks, and they stop it before it changes Goals or Scope. Loosen them only after the team trusts what its sessions write.

09

How do you review the PR and close the loop?

Open the pull request from the Brief's header, review it on the Brief's Review tab, and vote again. On the Review tab, **Ready** approves the pull request on GitHub and **Not yet** requests changes, so the review decision and the alignment vote are one act.

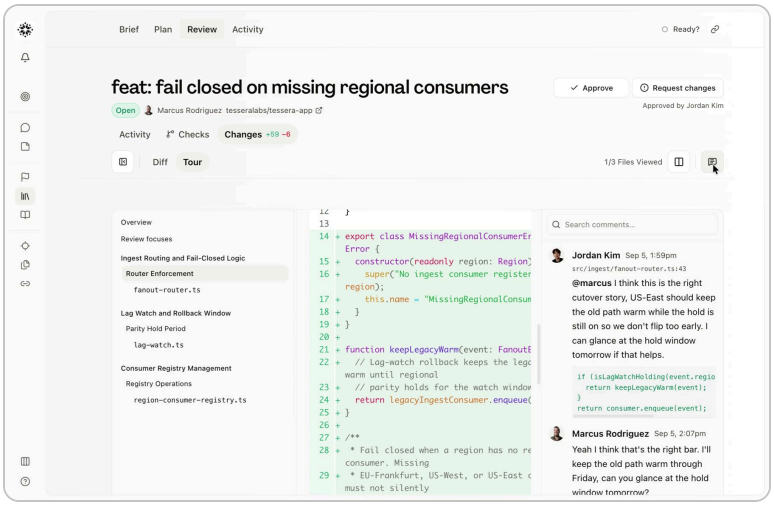
The pull request control in the Brief header shows the linked pull request's state: **Draft**, **Open**, **Merged**, or **Closed**. Hamster Cloud links its pull requests automatically and opens them as drafts, so mark one ready when you want reviewers on it. For work shipped from a laptop, link the pull request yourself. Click **+** in the Brief header, choose **Pull Request**, and paste the URL. A session can do the same with `link_pull_request`, and the CLI with `hamster brief link-pr`.

The Review tab opens on a Tour, a walkthrough Hamster writes for the pull request. It groups the changed files into named sections, each with a short summary and the real diff. Select code to comment, and tick **Send to Hamster** on any comment you want Hamster to act on in a delivery. Sign in with GitHub or GitLab first, so your comments and decisions reach the code host under

your name. On Origin, comments post through the Hamster Origin App, and your vote stays in Hamster.

The Brief's status follows the work. It moves to Shipping while a delivery runs, and to Delivered when a Hamster Cloud delivery succeeds or its linked pull request merges. Move it to Done yourself once someone has checked the outcome against the Goals the team agreed.

The decision at this step is whether the shipped work matches the Brief. Read the Goals first and the diff second. If they disagree, vote Not yet and say what should change, and the next delivery can act on it.



10

What should run on its own?

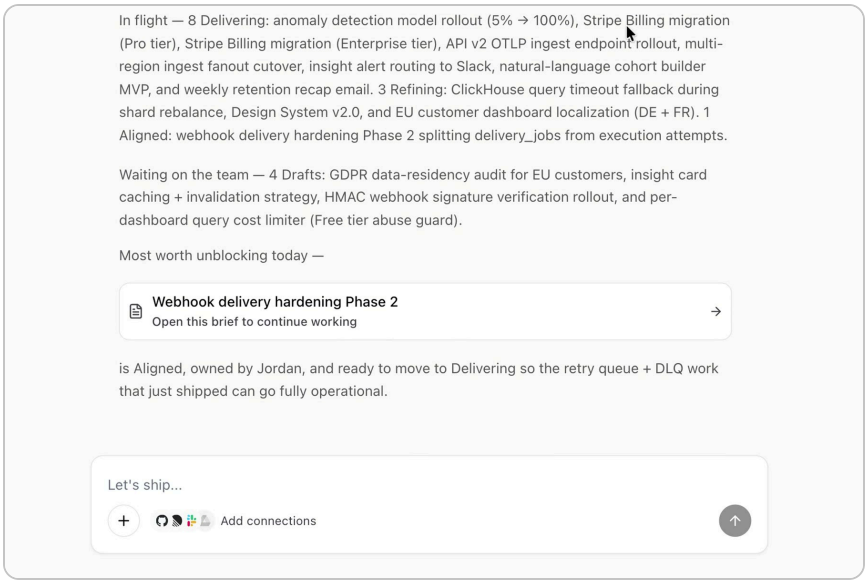
Give recurring checks to Routines, and ask Hamster in chat to tell you when something you care about changes. A Routine is an instruction written in plain English, plus the events or schedule that start it.

Open **Methods**, go to **Routines**, and select **New routine**. Write the **Instructions**, then add **Triggers**. Event triggers cover Briefs, Plans, Tasks, Initiatives, deliveries, and the connected repositories. A Routine can start when an alignment vote is submitted, when a Brief's status changes, when a delivery completes or fails, or when a pull request merges. Scheduled triggers run hourly, daily, on weekdays, weekly, or on a custom interval. Use **Test run** before you switch a Routine on, and read its **Run History** to see exactly what each run did.

The **Library** under Methods holds Routines that Hamster maintains, and each one switches on with a toggle. Fork one when you want to change it.

For a lighter watch, ask in any chat, for example “Tell me when this Brief is Delivered” or “Every weekday at nine, list the Briefs waiting on votes.” Hamster sets up a Routine that belongs to that conversation and posts each result back into it. A run with nothing new to report stays quiet.

Choose the first recurring check carefully. Start with one that reports and changes nothing. Creating a Routine needs the Manage Routines permission, and Hamster switches off any Routine that fails five times in a row, so a broken one cannot keep firing.



TEMPLATE **Three starter Routines**

Trigger: alignment vote submitted
When the vote is Not yet, post in the Brief chat. Quote the concern, name the owner, and ask the owner to answer it there.

Trigger: pull request merged
Find the Brief linked to this pull request. Compare the merged change with the Brief's Goals and Scope. Post in the Brief chat which Goals the change meets, which it misses, and anything it changed outside Scope.

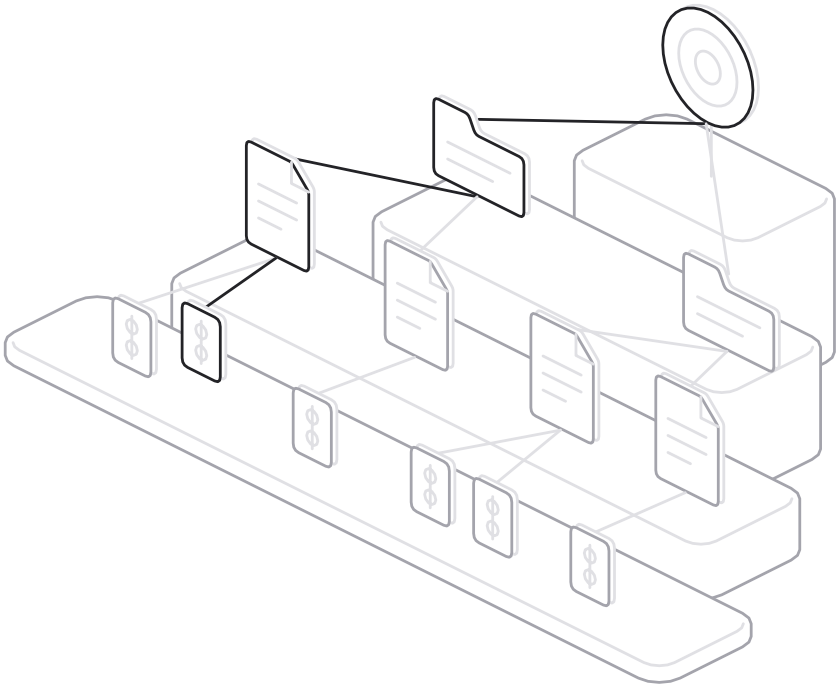
Trigger: schedule, weekly on Monday morning
List every Brief in Refining or Shipping, grouped by Initiative. Show its owner, its Ready votes, and its pull request state. Name each Brief that has waited more than a week on a vote or a review.

11

Where do Goals and Initiatives fit above Briefs?

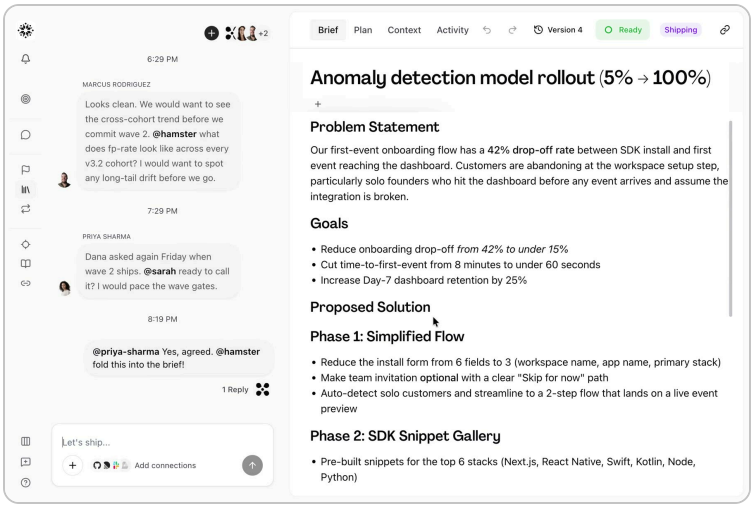
Goals hold what the team wants to move, Initiatives hold the commitments that move it, and Briefs are the deliverable pieces inside each Initiative. The chain runs from Goal to Initiative to Brief to Plan to Task to pull request, so a lead can trace a merged change up to the outcome it serves.

Set Goals up in the framework the team already uses. Hamster ships with OKR, OGSM, V2MOM, AARRR, HEART, and North Star. The chat can set one up from a sentence such as “Set up our Q3 OKRs. We’re focused on activation and revenue.” A measurable Goal takes a metric, a baseline, and a target, and each period records a target and an actual.



An Initiative names the outcome, an owner, the Goals it serves, and the Briefs that deliver it. It carries a state, from Draft through Shipping to Done, and a separate health, from Not Started through On Track and At Risk to Off Track. A Shipping Initiative can still be At Risk, so read the two together. Hamster’s sizing guidance is an Initiative of a week or two, made of Briefs of half a day to a day, which keeps every level closable.

At this level the team decides what it will not do this quarter. An Initiative with no Goal behind it, or a Goal with no Initiative under it, is worth a conversation before agents start on anything there.



12

A week-one and month-one rollout

Run one workstream all the way through in the first week. Add the next workstream only when the first one runs from agreed intent without help, and use the rest of the month for Cloud Agents, Routines, and Goals.

Pick a first Brief that several people care about. A day of work with three opinions behind it tests the votes, the Spec, and the review far better than a quiet refactor. Keep both automation switches in **Settings > Delivery** off for the first week, so the team sees every pause and makes every call.

TEMPLATE **Week one and month one**

Week one

- Day 1: Create the workspace. Connect the one repository the first workstream touches. Invite the people it affects as Reviewers.
- Day 1: Install the plugin in every coding tool the team uses. Run setup in the repository. Confirm with hamster status.
- Day 2: Write the first Brief in chat. Name the owner and the voters. Add the session instructions to AGENTS.md or CLAUDE.md.
- Day 3: Collect Ready or Not yet from every voter. Answer each concern in the Brief chat.
- Day 4: Settle the Spec. Generate the Plan. Read it as a team.
- Day 5: Deliver. Review the pull request on the Review tab. Move the Brief to Done when its Goals are met.

Month one

- Week 2: Set up a Cloud Agent for the main repository. Start a second Brief in a different stage of work.
- Week 2: Turn on one Routine that reports and changes nothing.
- Week 3: Group the Briefs under an Initiative. Run two deliveries at the same time.
- Week 4: Add the team's Goals. Link each Initiative to the Goal it serves.
- Week 4: Read the signals. Decide whether to add a third workstream.

13

How can you tell it is working?

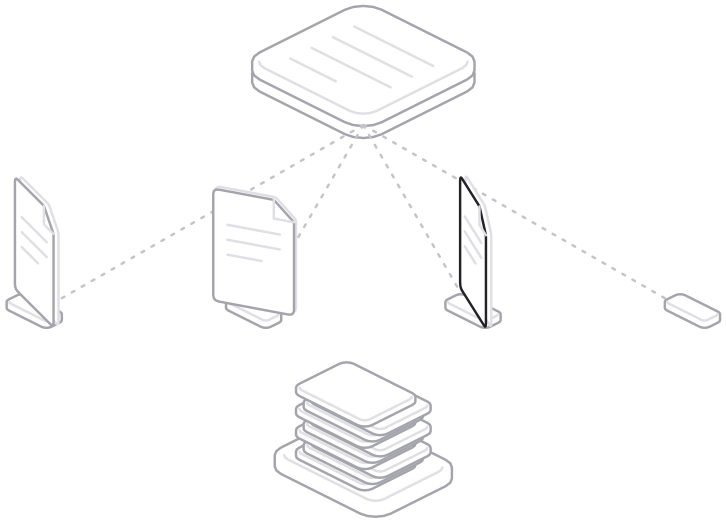
It is working when the record answers three questions without a meeting: what the team agreed, what it decided along the way, and what it shipped because of both. Hamster keeps each answer somewhere you can open.

TEMPLATE **Where to read the signals**

What the team agreed: the Activity tab of each Brief. Every voter's latest vote is Ready.
What it decided: the Spec and the Brief's decision log note. Each decision names who made it.
What it shipped: the pull request control in each Brief header. Every merged pull request traces to a Brief.
Where work waits: the Briefs list grouped by Status. Look for Briefs that sit in Refining with votes missing.
How Initiatives are tracking: the Initiatives list, with state and health read together.
Whether Routines help: each Routine's Run History. Switch off any Routine whose output nobody reads.

Read these monthly next to the leader’s scorecard from the first guide. Its five measures are concurrent workstreams per person, the share of agent work traced to agreed intent, rework, time from agreement to merge, and review time per merged pull request. The plugin’s `retro` skill writes an engineering retrospective from git history over a time window, which supplies the delivery numbers.

Warning signs show up in the same places. Watch for a Brief that ships with Not yet votes still open, a pull request with no Brief linked, or a decision log that stays empty while deliveries run. Each one means the record has fallen behind the work.



Fix that habit before you add the next workstream. Every workstream added on a thin record runs from a slightly different picture of what the team agreed, and the review queue pays for the difference.

Common questions

How do I connect Claude Code to Hamster?

In a Claude Code session, run /plugin marketplace add gethamster/plugin and /plugin install hamster@hamster-plugins, then /reload-plugins and /hamster:setup. Open /mcp, select plugin:hamster:hamster, and sign in, and hamster status should report that the CLI is signed in and synced.

Does Hamster replace Linear or Jira?

No. Hamster works alongside Linear or Jira, and optional two-way sync keeps Briefs and Tasks in step with Linear projects and issues or with Jira issues and epics. A new Linear connection starts with sync off.

How do alignment votes work in Hamster?

Each person whose work the Brief changes votes Ready or Not yet from the Ready? control in the Brief header, and a Not yet vote needs a short explanation. The Brief's Activity tab lists every vote, concern, and status change, and teammates can also vote from Brief cards in Slack.

Do alignment votes block delivery in Hamster?

No. Hamster's pre-delivery checks cover the Plan, the repository, the Spec, and any delivery already running, so the team writes down its own rule for waiting on agreement.

How do I run several coding agents in parallel on Hamster?

One Brief runs one delivery at a time, so run several Briefs at once. Two Briefs aimed at the same Cloud Agent each get their own sandbox, and in your own tools the Ship skill groups non-overlapping parent Tasks into waves that run in parallel.

How do agent sessions write decisions back to Hamster?

Through the MCP tools: get_brief reads the current Brief, update_task_status moves Tasks, update_note adds rows to the decision log, and link_pull_request ties a pull request to its Brief. Put the guide's session instructions in AGENTS.md or CLAUDE.md so every session starts with them.

What should a team do in its first week on Hamster?

Run one workstream all the way through. Connect one repository and install the plugin on day 1, write the first Brief on day 2, collect votes on day 3, settle the Spec and generate the Plan on day 4, and deliver and review on day 5.

Run this with your team.

Hamster is free for 10 Briefs a month, with unlimited viewers.

tryhamster.com/get-started

This guide on the web: tryhamster.com/guides/intent-driven-development