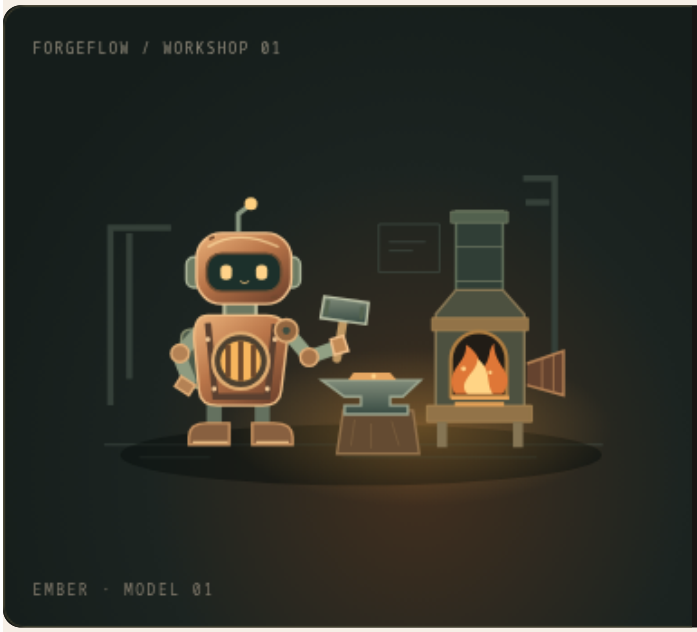
ForgeFlow

FIELD GUIDE / EDITION 01

Your first idea. Your next finished feature.

A visual user guide to ForgeFlow, from installation to a reviewed, validated handoff.



MEET EMBER

Make the work visible.

ForgeFlow gives your coding assistant a repeatable way to understand a task, involve specialists, build carefully, and check the result. Ember shows the workflow's reported activity.

01	02	03	04	05	06	07
Discuss	Research	Plan	Consult	Implement	Review	Ship

Who this is for

A developer or project owner using **Claude Code or Codex**, comfortable opening a terminal and a Git repository. You do not need to know ForgeFlow's agents or commands.

What you will finish

A working installation, one small reviewed change, an understandable dashboard, and a repeatable path to shipping the next change.

> **How to use this guide**
>
> Follow chapters 1–6 for your first useful run. Chapters 7–15 explain the complete delivery cycle. Keep chapters 16–18 nearby as a desk reference. In the HTML edition, use the host selector, copy buttons, and checklist; the PDF shows both hosts.

Workflow reference verified against ForgeFlow commit `1353184` · 6 September 2026
Visual identity and workshop screenshot refreshed 22 September 2026. The workshop screenshot uses illustrative example data. Ember gallery images use its labeled preview controls. Example project prompts describe a teaching scenario, not a completed project.

YOUR READING PATH

Inside the guide

Start at installation, or jump to the chapter you need.

Numbers identify chapters. In a PDF reader, use these links to jump to a chapter. Each chapter begins on a new page.

02 / THE MAP

A team, with a clear handoff.

You explain the outcome. ForgeFlow organizes the work around it.

YOU

Set the destination

Explain the problem, constraints, success criteria, and what may be changed.

FORGEFLOW

Coordinate the work

Prepare context, choose useful specialists, preserve decisions, and run checks.

YOU + EVIDENCE

Decide when to ship

Inspect the result and validation. Give explicit instructions for remote actions such as pushing or publishing.

Agent	What they contribute
Builder	Backend structure, data, naming, and code quality.
Guardian	Security, validation, system boundaries, and reuse.
Designer	User experience, accessibility, frontend quality, and connectivity.
Coordinator	Scope, coordination, project memory, and handoff notes.
Architect	Combines specialist input into a brief or technical verdict.
Product Lead	Checks requirements, tests, plan adherence, and product intent.
Verifier	Checks high-risk findings using visible evidence.

Choose the smallest useful entry point

Small, clear change	Quick → validate
Feature with a known goal	Consult → implement → review → ship
Uncertain product or architecture	Discuss → research → plan → consult → implement → review → ship

Not every task needs every agent or every phase. A route can be **skip, thin, full, or deep**, depending on the change. Ask for the reason when the route is unclear.

Brief: the implementation contract. **Artifact:** a saved local result, such as that brief. **Host:** Claude Code or Codex, where you talk to the assistant.

03 / INSTALL

Install ForgeFlow into your host.

Use the same computer and shell environment in which your coding assistant runs.

Before you begin | Git | Node.js + npm | Bash | A working Claude Code or Codex session

The repository's CI validates on **Node.js 24**. These terminal examples use Bash; Windows users need a compatible Bash environment, such as WSL, with the host and files in that same environment. ForgeFlow uses your host's available model access and permissions.

Terminal · Bash

```
git --version
node --version
npm --version
git clone https://github.com/BrandedTamarasu-glitch/ForgeFlow.git
cd ForgeFlow
```

Choose one host below. Preview the destinations, then install. This runs from the ForgeFlow checkout, not from your application repository.

Terminal · Codex installation

```
node scripts/forgeflow/install-template.js --target codex --dry-run --json
node scripts/forgeflow/install-template.js --target codex
FF_RUNTIME="${CODEX_HOME:-$HOME/.codex}/forgeflow"
```

Terminal · Claude Code installation

```
node scripts/forgeflow/install-template.js --target claude --dry-run --json
node scripts/forgeflow/install-template.js --target claude
FF_RUNTIME="$HOME/.claude/forgeflow"
```

To install both, use `--target both`. The template installer copies managed files and preserves unrelated configuration; it does not merge your host settings.

Enable the optional local dashboard

Terminal · Bash

```
npm install --prefix "$FF_RUNTIME/services/dashboard" --ignore-scripts
npm install --prefix "$FF_RUNTIME/services/agent-chat" --ignore-scripts
```

Checkpoint

The install reports copied agents, skills or commands, and runtime helpers. Restart your host so it discovers them. Keep this shell open: later commands use `FF_RUNTIME`. Automatic dashboard startup never installs these dependencies for you.

04 / CONNECT

Confirm the host can use it.

Files on disk and a host that has loaded them are two different checks.

CODEX

Use the ForgeFlow skill name

After restarting, invoke `$quick` or `$consult`. For ForgeFlow review, use **`$forge-review`**. Codex's `/review` is its built-in review, not this workflow.

Agents are copied into `~/.codex/agents/`; skills into `~/.codex/skills/`. A custom `CODEX_HOME` changes those roots. Do not replace your existing config with the repository sample.

CLAUDE CODE

Load commands and wire hooks

After restarting, commands such as `/quick` and `/consult` should be available. The template installer leaves `~/.claude/settings.json` for you to merge manually.

Hooks provide phase entry, telemetry, and context guidance. Preserve existing settings and avoid registering a hook twice if another installation already provides it.

Claude Code hook wiring checklist

Host setting	Installed script under ~/.claude/hooks/
SessionStart and UserPromptSubmit	`forgeflow-lean-activate.js`
PostToolUse	`forgeflow-context-monitor.js`, `forgeflow-gate.js`, and `forgeflow-telemetry.js`
statusLine	`forgeflow-statusline.js`

Use the repository's manual settings guidance as the source for the merge. A command should use an expanded home path, such as `node "$HOME/.claude/hooks/forgeflow-telemetry.js"`. Ask the assistant to inspect the existing settings and present the exact additions before applying them.

Claude Code · setup prompt

```
Inspect my existing Claude Code settings and ForgeFlow's post-install hook guidance. Show the additions
needed for hooks and the status line, preserving unrelated settings and avoiding duplicate
registrations.
```

Validate the resulting JSON, restart Claude Code, and run `/forgeflow-health`. See Settings and Recovery for the detailed wiring and recovery reference.

If a model cannot be used

An installed agent may name a model unavailable to your account or host. Have the assistant identify the affected agent and propose an available model before continuing. Do not interpret an unsupported-model error as completed specialist work.

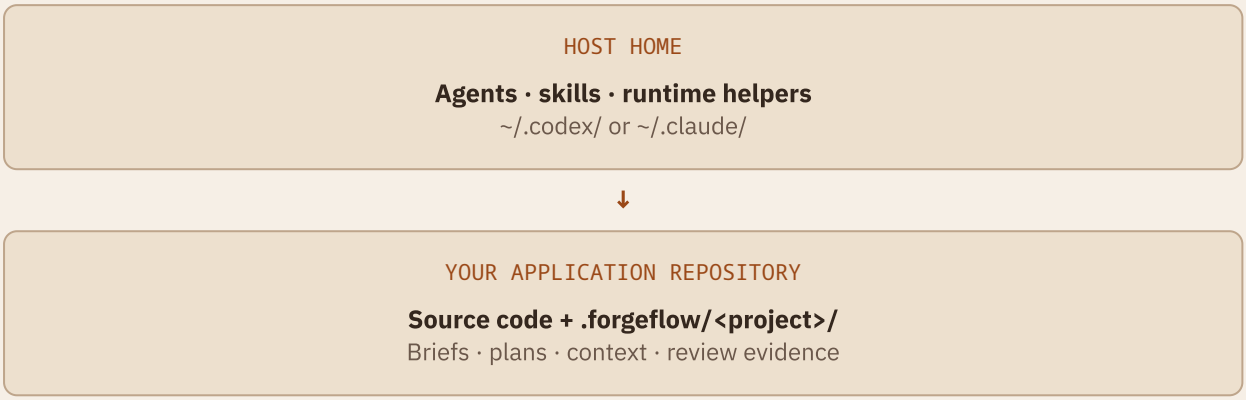
Checkpoint

The host recognizes a ForgeFlow command or skill, can start its selected agents, and reports any dependency, permissions, or model problem plainly.

05 / YOUR PROJECT

Move from the tool to your code.

ForgeFlow is installed once per host. Its working notes belong to each project.



Open the repository you actually want to improve. Replace the example path below, confirm the Git state, and choose a new branch name.

Terminal · Bash

```
cd /path/to/your-project
git status --short
git switch -c feature/forgeflow-first-task
bash "$FF_RUNTIME/scripts/forgeflow/ensure-forgeflow-state.sh"
node "$FF_RUNTIME/scripts/forgeflow/seed-budget-config.js" --json
```

The first helper creates local workflow folders and ignores `.forgeflow/`. The budget helper seeds a config without overwriting an existing one. Inspect any repository changes before committing.

Give the assistant a bounded orientation task

Codex · type in the assistant

```
$quick inspect this repository. Identify how to
run it, its relevant tests, and a small first
task. Read only; do not edit files.
```

Claude Code · type in the assistant

```
/quick inspect this repository. Identify how to
run it, its relevant tests, and a small first
task. Read only; do not edit files.
```

> **Keep the scope visible**
>
> Name the project, the branch, and the files or feature area. Ask the assistant to preserve existing uncommitted work. Do not run application work in the ForgeFlow source checkout unless ForgeFlow itself is your project.

Optional deeper orientation

Ask for the first-run guide, a project code map, or a project operating model if the repository is unfamiliar. Missing optional reports are normal before you generate them. The general health helper includes Claude installation assumptions; use Codex file discovery and its actual skills to verify a Codex installation rather than treating every Claude inventory item as required.

06 / FIRST USEFUL RUN

Build one small thing, end to end.

Worked example: an existing task-list app should explain its empty state.

TEACHING SCENARIO

“No tasks yet” → a clear next action

When the task list is empty, show helpful text and a button that opens the app’s existing task form. Keep the existing design and task-creation behavior.

Codex · type in the assistant

```
$consult produce an implementation brief for an
empty task-list state. Show “No tasks yet” and a
Create task button using the existing form.
Cover keyboard access, loading, and errors. Keep
the change local.
```

Claude Code · type in the assistant

```
/consult produce an implementation brief for an
empty task-list state. Show “No tasks yet” and a
Create task button using the existing form.
Cover keyboard access, loading, and errors. Keep
the change local.
```

1. **Read the brief**
   Confirm where the change belongs, what is out of scope, and which checks will prove it works. Correct misunderstandings now.

2. **Implement the brief**
   Run the implementation command below. The assistant should name the files changed and explain validation.

3. **Try the feature**
   Open the app, visit an empty list, and activate Create task with both mouse and keyboard. Check loading and error states too.

4. **Request a review**
   Run the review command. Ask for concrete findings tied to this change. Resolve required corrections and rerun affected checks.

5. **Prepare the handoff**
   Run Ship to prepare a summary. Inspect the diff and test results before requesting a commit, push, or deployment.

Codex · type in the assistant

```
$implement execute the current brief
$forge-review review the current changes against
the brief
$ship prepare the branch and summarize
validation
```

Claude Code · type in the assistant

```
/implement execute the current brief
/review review the current changes against the
brief
/ship prepare the branch and summarize
validation
```

Run one prompt at a time

Wait for each phase, read its output, and steer it before the next command. A useful first run ends with observable behavior and validation evidence, not just a confident summary.

Make uncertainty explicit.

Use these phases when the destination or the approach is not yet clear.

DISCUSS

What should change?

State the user problem, who is affected, constraints, accessibility needs, and success criteria. The useful output is a shared problem statement and a list of open questions.

Codex · type in the assistant

```
$discuss users cannot tell whether an empty task
list means no tasks, loading, or an error. Frame
the behavior and acceptance criteria.
$research compare ways to reuse the current task
form from the empty state. Inspect existing
patterns first.
```

RESEARCH

Which approach fits?

Compare the codebase’s existing patterns, plausible alternatives, integration risks, and tradeoffs. The useful output is evidence and a reasoned choice, including what remains uncertain.

Claude Code · type in the assistant

```
/discuss users cannot tell whether an empty task
list means no tasks, loading, or an error. Frame
the behavior and acceptance criteria.
/research compare ways to reuse the current task
form from the empty state. Inspect existing
patterns first.
```

Three questions to answer before planning

01

What is success?

A user can understand the current state and create a task without learning a new interaction.

02

What must stay true?

Use the current form, preserve permissions, and keep loading and error behavior distinct.

03

What is uncertain?

Does the existing form work from this route? Which focus behavior should be reused?

When research branches

Focused research automatically chooses normal or divergent research and explains the route. Use `--no-diverge` for a focused investigation, or `--diverge` when you deliberately want independent approaches to a consequential decision.

Codex · type in the assistant

```
$research --no-diverge investigate why the
current task form loses keyboard focus
```

Claude Code · type in the assistant

```
/research --no-diverge investigate why the
current task form loses keyboard focus
```

Checkpoint

You can state the problem, the chosen approach, and the evidence still needed. If the request was already clear and small, enter at Consult or Quick instead.

Turn the choice into a buildable brief.

A plan sequences the work. A consultation defines how the implementation should fit together.

PLAN	IMPLEMENTATION BRIEF	IMPLEMENTATION
Scope → phases → dependencies → checks	**Behavior → interfaces → ownership → proof**	**Small changes, checked against the contract**

Codex · type in the assistant

```
$plan create a phased plan for the empty-state
change, including keyboard validation and scope
boundaries.
$consult turn the plan into a concrete
implementation brief using existing components.
```

Claude Code · type in the assistant

```
/plan create a phased plan for the empty-state
change, including keyboard validation and scope
boundaries.
/consult turn the plan into a concrete
implementation brief using existing components.
```

Look for in the brief	Example for the task-list change
User-visible behavior	Empty list has explanatory text and a working action. Loading and failure have their own states.
Scope boundaries	Reuse the form; do not redesign the whole task page.
Interfaces and ownership	Identify the list component, form entry point, and who owns each change.
Accessibility	Keyboard activation, visible focus, and predictable focus after opening or closing the form.
Validation	Check empty, populated, loading, and error views; confirm the existing creation path still works.
Known uncertainties	Call out any component or behavior not verified in the codebase.

The usual local files are `.forgeflow/<project>/current-plan.md` and `current-brief.md`. The implementation workflow expects the current brief, so check that it describes this task rather than a previous one.

> **If the brief is wrong**
>
> Give a concrete correction: “Keep the existing form. Remove the proposed new modal. Add a keyboard check.” Have the assistant update the brief before continuing.

09 / IMPLEMENT

Keep intent and evidence together.

The brief is the reference point as work becomes code.

Codex · type in the assistant

```
$implement execute the current brief. Preserve
unrelated edits, run relevant checks, and report
any scope change before broadening the work.
```

Claude Code · type in the assistant

```
/implement execute the current brief. Preserve
unrelated edits, run relevant checks, and report
any scope change before broadening the work.
```

1. **Confirm the starting point**
   The assistant reads the current brief, identifies relevant files, and uses compact context and ownership notes when available.

2. **Build in focused pieces**
   Specialists work where useful. A shared interface or overlapping file ownership should be resolved before parallel edits cause conflicts.

3. **Validate the behavior**
   Product Lead focuses on the required checks; integration review checks that the pieces fit. The exact commands depend on your application.

4. **Record decisions and gaps**
   Implementation notes preserve choices, tradeoffs, deviations, follow-ups, and validation evidence for the next phase.

Useful steering while the assistant works

Either host · natural-language steering

```
Keep the change limited to the task-list view and its tests.
Explain the test failure before changing the implementation.
Show what changed from the brief and why.
Pause before any database migration or deployment.
```

Read the final implementation report

It should explain what changed, why, how it was checked, and what is still unverified. A passing test count alone does not tell you whether the requested behavior was covered.

Inspect the app yourself

For visible changes, use the application. Look at mobile sizing, keyboard navigation, empty/loading/error states, and the original flow that should still work.

> **When a check fails**
>
> Ask for the failing command, the relevant output, and the smallest proposed correction. Keep raw output available if a compact failure digest says the raw evidence is required. A failure state in Ember is a signal to investigate, not a reason to hide the failed check.

Understand the verdict before moving on.

Review compares the change with its requirements and the evidence in the code.

Codex · type in the assistant

```
$forge-review review this branch against main
and the current brief. Prioritize concrete
correctness, security, and accessibility
findings.
```

Claude Code · type in the assistant

```
/review review this branch against main and the
current brief. Prioritize concrete correctness,
security, and accessibility findings.
```

Specialist findings → Verifier when needed → Architect verdict → Product Lead check

Architect decision	What to do next
APPROVE	No blocking change is requested by this review. Still inspect validation and any remaining shipping requirements.
CONDITIONAL APPROVE	Read and satisfy the stated conditions. Do not assume they are optional.
REVISE	Correct the identified issues, run affected checks, and request follow-up review.
BLOCK	Resolve the blocking risk or missing evidence before proceeding.

Product Lead can **CONFIRM** or **CHALLENGE** the verdict. Verifier can confirm, reject, or block a high-risk finding from visible evidence. These are different roles, not interchangeable approval labels.

What a useful finding contains

A location, a concrete trigger, an explanation of the effect, and enough evidence to judge the proposed correction. Ask for clarification if a finding is only a preference or a vague prediction.

Either host · follow-up prompt

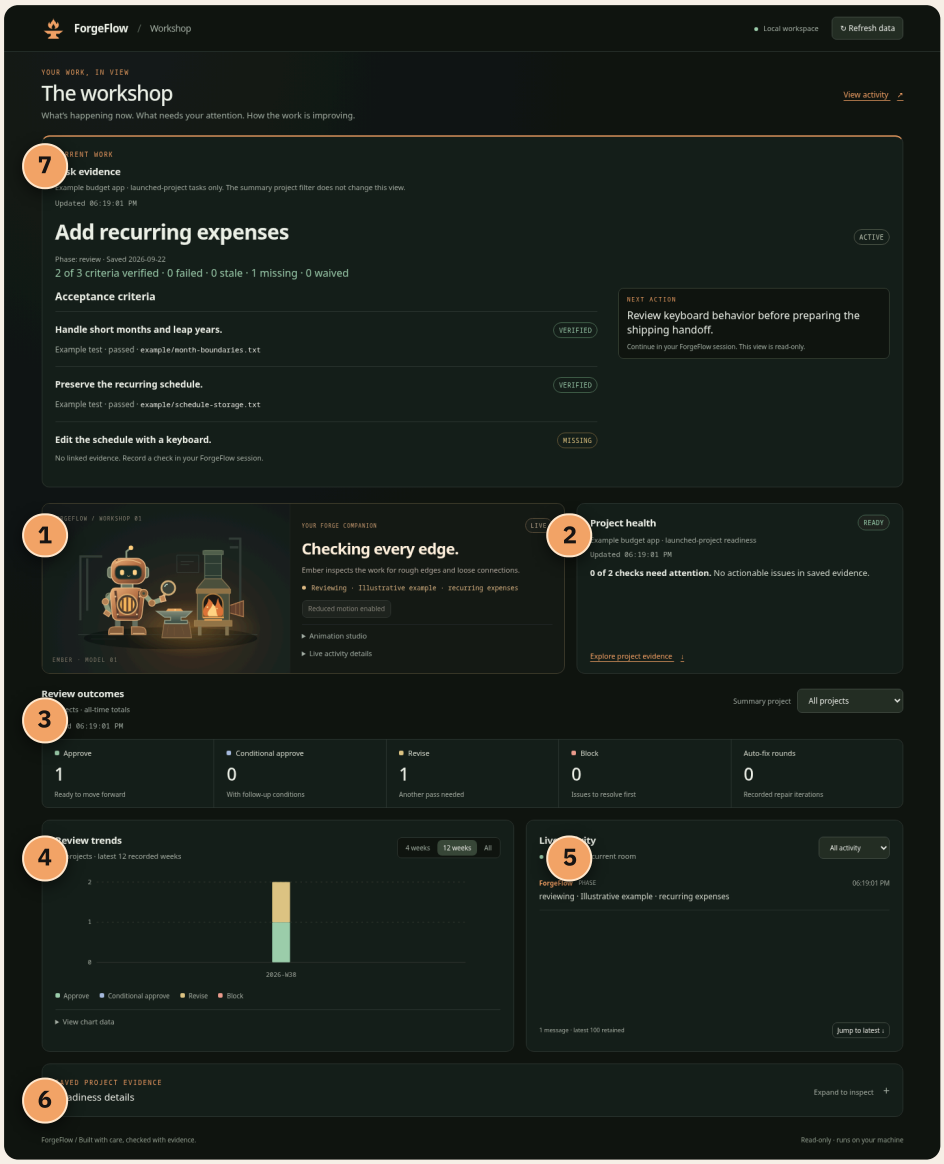
```
Fix the confirmed findings from this review. Keep the patch scoped, rerun the affected checks, and
request a follow-up review.
```

> **Review metrics are recorded decisions**
>
> The workshop charts count saved verdicts. Codex's ForgeFlow review and implementation skills explicitly record actual decisions with saved evidence and a stable event ID. A retry should not double-count an outcome. An unrecorded historical review will not appear automatically.

For a deeper examination of a subsystem, use `/audit` or `$audit`. For conservative automated repair in Claude Code, `/review-auto` has additional rules and limits. There is no matching installed Codex `$review-auto` skill in this edition.

Read the dashboard at a glance.

Open http://127.0.0.1:4003/ on the machine running ForgeFlow.



Current workshop UI with illustrative example data. Counts, task evidence, and review decisions demonstrate the interface; they are not live project results.

7. **Task evidence.** The active task, acceptance criteria, saved evidence, and next action for the launched project.
2. **Project health.** Readiness for the repository that launched this dashboard. Copy its next action into your assistant.
4. **Review trends.** All projects, using the latest 4, 12, or all recorded ISO weeks.
6. **Saved evidence.** Expand for individual readiness reports and the Lean Prime checklist.
1. **Ember.** The current room's reported work, preview studio, and motion controls.
3. **Review outcomes.** All-time totals for the selected summary project, with conditional approvals shown separately.
5. **Live activity.** Workflow phase updates and agent messages, with filters and recent history.

Automatic opening

The first eligible workflow invocation in a desktop session starts or reuses the dashboard and activity service, reports its phase, and opens the browser once. Later invocations reuse the services. Closing the tab does not make the next invocation reopen it.

A companion, not a progress estimate.

Ember reflects explicit reports. Time passing does not imply success.

Idle

Connected; no work is reported.

Planning

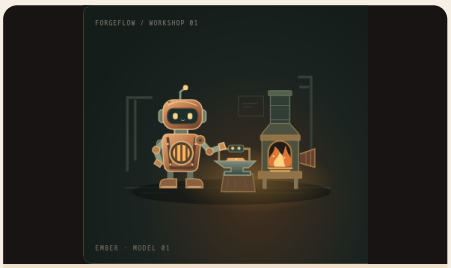
Preparing a blueprint or plan.

Implementing

Building the requested change.

Reviewing

Inspecting the result and its edges.



Testing

Running reported validation.

Complete

The workflow explicitly reports completion.

Gallery captured using Animation studio previews. The live text label is the authoritative description, including for screen readers.

Other state	How to read it
Researching	The workflow is investigating and gathering evidence.
Waiting	Work needs input, or an active report is over 90 seconds old. Read the label to distinguish the two.
Failed	A task or check explicitly reported failure. Inspect the actual result.
Offline	The activity connection is unavailable. It does not tell you whether the underlying task succeeded.

Try a pose

Open **Animation studio**. A preview is visibly labeled and only changes the local pose. Return to live to follow reported work.

Reduce motion

Use **Pause motion** or your system's reduced-motion preference. Pausing the animation does not pause activity updates or the workflow.

When several agents report, fresh failures and waiting states take priority over active work; active work takes priority over completion. Ember may therefore show an issue even while another agent has finished.

Empty does not always mean broken.

Know which source each panel reads before trying to fix it.

Panel	Source and scope	When it is empty
Project health	Saved artifacts for the launched repository. It is not an overall diagnosis of every installed tool.	Some reports have not been generated yet. Expand the details.
Review outcomes	Saved verdict telemetry from local Claude and Codex project roots. Summary filter selects all-time totals.	No verdicts were recorded in the selected scope. Planning, tests, and activity do not create approvals.
Review trends	Weekly saved verdict buckets across all projects. The window means recorded weeks, not every calendar week.	There may be no saved verdicts, or no data in the selected recorded window.
Live activity	The activity service's current room, independent of the summary project filter.	No current messages or states have been reported, or the selected filter hides them.

Readiness has two distinct kinds of information

Actionable warning

A budget violation, missing project guidance, actual saved blocker, or corrupt evidence needs investigation. Use the summary and suggested command.

Informational evidence

Unrecorded benchmarks, cross-host verification, release snapshots, or failure digests may be expected. No failure digest is needed until there is a real failure to capture.

Watch with zero actionable warnings can mean optional evidence is still incomplete. It does not mean those optional activities were performed or passed. Lean injection can remain unavailable when evidence is too thin; the underlying status stays visible.

Refresh and freshness

Refresh data fetches metrics and readiness separately. A failed refresh keeps the previous snapshot and labels it stale. Initial errors say Unavailable. The live WebSocket connects independently. Refresh the browser after installing a new dashboard version.

Changing projects

The summary filter changes review totals, not readiness or Ember's current room. A reused dashboard retains its launched repository. Ask the assistant to identify the existing service and deliberately restart it from the intended project if that scope needs to change.

Let useful context survive the session.

Local evidence should help the next task without turning old assumptions into rules.

Current task + code
Scope, requirements, changed files

Compact packets + brief
Selected evidence for the agents

Implementation + review notes
Decisions, checks, findings, outcomes

Project learnings
Useful patterns for the next task

Most local artifacts live under `.forgeflow/<project>/`. Treat them as advisory context. Current instructions, code, and test evidence take precedence.

When the context budget warns

Ask the assistant to identify the oversized packet, narrow the file list, and trim unrelated memory or split the work into review waves. Do not increase a budget just to turn a warning green. The default compact-token limit is 16,000 unless project configuration changes it.

Terminal · Bash

```
node "$FF_RUNTIME/scripts/forgeflow/check-context-budget.js" --root .forgeflow --warn-only --json
node "$FF_RUNTIME/scripts/forgeflow/advise-context.js" --root .forgeflow --record --json
```

The advisor can record local trend history. Token figures are estimates; they are not a bill or proof of a model's actual usage.

Preferences and resuming

Tell ForgeFlow how you want it to work, such as concise updates and explicit validation status. Ask to record only preferences you intentionally choose. Keep project-specific design preferences local to that project.

Either host · handoff prompt

```
Before we stop, summarize the current goal, decisions, changed files, checks completed, unresolved
issues, and the next concrete step. Save this as local handoff context.
```

> **Local-first is not offline AI**
>
> The dashboard and saved telemetry are local. Your AI host may still send task context to its model provider; GitHub operations and dependency installs use the network. Review local notes, packets, telemetry, and support bundles before sharing them.

Finish with a reviewable handoff.

Preparing to ship, pushing code, and deploying a product are separate actions.

Codex · type in the assistant

```
$ship prepare this branch for handoff. Summarize
the behavior change, validation, known limits,
and proposed PR description.
```

Claude Code · type in the assistant

```
/ship prepare this branch for handoff. Summarize
the behavior change, validation, known limits,
and proposed PR description.
```

01

Prepare

Summary, presentation, and PR body.

02

Inspect

Diff, tests, review conditions, remaining risk.

03

Authorize

Specify commit, push, PR, or deployment.

04

Verify

Check the remote run and resulting behavior.

What Ship prepares

Under `.forgeflow/<project>/ship/`, inspect `ship-summary.json`, `ship-presentation.html`, and `pr-body.md`.

Ask the assistant to refine generated summaries against the final change. A report assembled from old notes can be incomplete; read it before publishing.

Example authorization after inspection

Either host · remote-action prompt

```
Commit these reviewed changes, push this branch, create a draft PR to main, and monitor its CI. Fix
failures caused by this change. Do not deploy.
```

For an already agreed main-branch workflow, state that target explicitly instead. The assistant should respect repository policy and your intended release process.

Evidence to check	Question to answer
Behavior	Does the result satisfy the original acceptance criteria?
Review	Were required corrections and approval conditions resolved?
Validation	What ran, what passed, and what was not verified?
Remote CI	Does the successful run belong to the exact pushed commit?
Release	Is tagging or deployment required by this project, and was it actually requested?

> **Your project's pipeline decides the build**
>
> A push only triggers jobs configured in that repository. ForgeFlow's own repository runs deterministic validation on pushes to main; that does not automatically give your application a CI or deployment pipeline. Check its configuration rather than assuming a build ran.

Start with the observed symptom.

Use a specific check before reinstalling, widening scope, or rerunning everything.

What you see	What to check or do
Skill or command missing	Restart the host. Confirm the selected installation home and rerun its dry-run installer. In Codex, use `$forge-review`.
Agent model unavailable	Identify the agent and model. Choose an available model explicitly; rerun the failed role.
Dashboard never opens	Check dashboard dependencies and a stable host session ID. CI, SSH, headless Linux, or `FORGEFLOW_DASHBOARD_AUTO_OPEN=off` skip automatic startup. Try the local URL or manual startup below.
Ember Offline	Start agent-chat with `$agent-chat-on` in Codex or `/agent-chat:on` in Claude Code. Inspect any startup error.
Ember Waiting	Read the label. It can mean input is needed or the last active report is stale. Ask for the actual task status.
Live Activity empty	Select All activity. Confirm the connection and current room. The host must report phases or messages; opening the page alone creates neither.
Outcomes or trends empty	Complete and record a real review. Then use Refresh data and check the project filter. Do not create fake history to fill the chart.
Readiness needs attention	Expand the affected report. Distinguish missing optional evidence from saved failures or unreadable artifacts. Run the stated correction in the assistant, then refresh.
Port already occupied	Identify the owning process. Reuse the expected service or deliberately restart it. Do not kill an unknown process merely to free a port.

Manual dashboard startup

From the intended project, keep this terminal process running; then open the URL in a browser on the same machine.

```
Terminal · Bash

node "$FF_RUNTIME/services/dashboard/server.js"
```

Update or repair

Codex: in a clean ForgeFlow source checkout, run `git pull --ff-only`, rerun `install-template.js --target codex`, and restart Codex. Resolve local source edits before pulling.

Claude Code: use `/update-forgeflow`; use `--repair` for missing managed files. The updater's `--rollback` restores its previous managed-file snapshot when one exists. Restart and verify. That rollback mechanism is not a general Codex rollback.

The commands you will reach for.

Choose a host command for the workflow; use a terminal helper for a specific local report.

Goal	Claude Code	Codex
Small task or orientation	`/quick`	`$quick`
Frame the problem	`/discuss`	`$discuss`
Compare approaches	`/research`	`$research`
Sequence the work	`/plan`	`$plan`
Write the implementation brief	`/consult`	`$consult`
Execute the brief	`/implement`	`$implement`
Review current changes	`/review`	`$forge-review`
Deep subsystem analysis	`/audit`	`$audit`
Prepare shipping artifacts	`/ship`	`$ship`
Start / stop activity service	`/agent-chat:on` `/agent-chat:off`	`$agent-chat-on` `$agent-chat-off`

Local report helpers for either host

Run from your project with the `FF_RUNTIME` set during installation. Read each result before deciding on the next action.

Terminal · Bash

```
node "$FF_RUNTIME/scripts/forgeflow/build-project-operating-model.js" --json
node "$FF_RUNTIME/scripts/forgeflow/show-project-trends.js" --refresh --json
node "$FF_RUNTIME/scripts/forgeflow/render-lean-prime.js" --json
node "$FF_RUNTIME/scripts/forgeflow/render-release-readiness.js" --plan-only --json
```

The model and trend helpers write or refresh local guidance. Lean Prime here reports readiness; release readiness with `--plan-only` previews checks rather than running them. None of these examples asks for a commit or deployment.

Explore advanced paths after a first useful run

UI iteration: measured browser and accessibility checks. **Fleet:** independent work in isolated worktrees. **Review-auto:** conservative repair with its own evidence rules. **Lean and context waves:** control context volume. **Team adoption:** collect real outcomes before making quality or efficiency claims.

Availability differs by host. The Claude command catalog is larger than the installed Codex skill set; do not assume every slash command has a dollar-prefixed equivalent.

Keep this checklist.

A repeatable first run is more useful than memorizing the command catalog.

- [ ] My host has loaded the ForgeFlow installation and can use its selected agents.
- [ ] I am working in the correct project and branch, with existing edits understood.
- [ ] I can describe one small outcome and how to verify it.
- [ ] The current brief matches that outcome and its scope.
- [ ] The implementation report names the changes, checks, and remaining limits.
- [ ] I tried the relevant application behavior, including keyboard and failure states.
- [ ] Required review corrections are resolved and the real verdict is recorded.
- [ ] I understand the dashboard's project scope, optional evidence, and live activity.
- [ ] Any requested push or release has been verified against its actual remote result.
- [ ] The next session has local notes with decisions, unresolved issues, and a next step.

A good next request

"Use ForgeFlow to help me improve [specific behavior]. Keep the first change small. Start by inspecting the relevant code, propose a brief, then implement and validate it."

Go deeper when you need to

Codex first run
Installation, discovery, and skill names.

Settings and recovery
Claude hooks, repair, and rollback.

User paths
Choose a path for a specific outcome.

Workflow reference
The complete command catalog.

Local data and sharing
What is stored and what to review before sharing.

Evidence readiness
Benchmarks, host verification, and release claims.

Guide sources: repository README, template installer and manifest, workflow skills, dashboard server/readiness/Ember implementations, and the linked documentation at commit `1353184`. Workflow instructions document that implementation; later versions may differ. The visual identity and workshop screenshot were refreshed on 22 September 2026.

ONE CLEAR OUTCOME. ONE WELL-FORGED CHANGE.