Built-in tools

Lobster

Typed workflow runtime for OpenClaw — composable pipelines with approval gates.

Lobster is a workflow shell that lets OpenClaw run multi-step tool sequences as a single, deterministic operation with explicit approval checkpoints.

Hook

Your assistant can build the tools that manage itself. Ask for a workflow, and 30 minutes later you have a CLI plus pipelines that run as one call. Lobster is the missing piece: deterministic pipelines, explicit approvals, and resumable state.

Why

Today, complex workflows require many back-and-forth tool calls. Each call costs tokens, and the LLM has to orchestrate every step. Lobster moves that orchestration into a typed runtime:

- **One call instead of many**: OpenClaw runs one Lobster tool call and gets a structured result.
- **Approvals built in**: Side effects (send email, post comment) halt the workflow until explicitly approved.

Resumable: Halted workflows return a token; approve and resume without re-running everything.

›

Why a DSL instead of plain programs?

Lobster is intentionally small. The goal is not “a new language,” it’s a predictable, AI-friendly pipeline spec with first-class approvals and resume tokens.

- **Approve/resume is built in**: A normal program can prompt a human, but it can’t *pause and resume* with a durable token without you inventing that runtime yourself.
- **Determinism + auditability**: Pipelines are data, so they’re easy to log, diff, replay, and review.
- **Constrained surface for AI**: A tiny grammar + JSON piping reduces “creative” code paths and makes validation realistic.
- **Safety policy baked in**: Timeouts, output caps, sandbox checks, and allowlists are enforced by the runtime, not each script.
- **Still programmable**: Each step can call any CLI or script. If you want JS/TS, generate `.lobster` files from code.

How it works

OpenClaw launches the local `lobster` CLI in **tool mode** and parses a JSON envelope from stdout. If the pipeline pauses for approval, the tool returns a `resumeToken` so you can continue later.

Pattern: small CLI + JSON pipes + approvals

Build tiny commands that speak JSON, then chain them into a single Lobster call. (Example command names below — swap in your own.)

```
inbox list --json
inbox categorize --json
inbox apply --json
```

```
{
  "action": "run",
  "pipeline": "exec --json --shell 'inbox list --json' | exec --stdin json --shel
  "timeoutMs": 30000
}
```

If the pipeline requests approval, resume with the token:

```
{
  "action": "resume",
  "token": "<resumeToken>",
  "approve": true
}
```

AI triggers the workflow; Lobster executes the steps. Approval gates keep side effects explicit and auditable.

Example: map input items into tool calls:

```
gog.gmail.search --query 'newer_than:1d' \
  | openclaw.invoke --tool message --action send --each --item-key message --args
```

JSON-only LLM steps (llm-task)

For workflows that need a **structured LLM step**, enable the optional `llm-task` plugin tool and call it from Lobster. This keeps the workflow deterministic while still letting you classify/summarize/draft with a model.

Enable the tool:

```
{
  "plugins": {
    "entries": {
      "llm-task": { "enabled": true }
    }
  },
  "agents": {
    "list": [
      {
        "id": "main",
        "tools": { "allow": ["llm-task"] }
      }
    ]
  }
}
```

Use it in a pipeline:

```
openclaw.invoke --tool llm-task --action json --args-json '{
  "prompt": "Given the input email, return intent and draft.",
  "input": { "subject": "Hello", "body": "Can you help?" },
  "schema": {
    "type": "object",
    "properties": {
      "intent": { "type": "string" },
      "draft": { "type": "string" }
    },
    "required": ["intent", "draft"],
    "additionalProperties": false
  }
}'
```

See   for details and configuration options.

Workflow files (.lobster)

Lobster can run YAML/JSON workflow files with `name` , `args` , `steps` , `env` , `condition` , and `approval` fields. In OpenClaw tool calls, set `pipeline` to the file path.

```yaml
name: inbox-triage
args:
  tag:
    default: "family"
steps:
  - id: collect
    command: inbox list --json
  - id: categorize
    command: inbox categorize --json
    stdin: $collect.stdout
  - id: approve
    command: inbox apply --approve
    stdin: $categorize.stdout
    approval: required
  - id: execute
    command: inbox apply --execute
    stdin: $categorize.stdout
    condition: $approve.approved
```

Notes:

- `stdin: $step.stdout` and `stdin: $step.json` pass a prior step's output.
- `condition` (or `when`) can gate steps on `$step.approved` .

Install Lobster

Install the Lobster CLI on the **same host** that runs the OpenClaw Gateway (see the), and ensure `lobster` is on `PATH` . If you want to use a custom binary location, pass an **absolute** `lobsterPath` in the tool call.

Enable the tool

Lobster is an **optional** plugin tool (not enabled by default).

Recommended (additive, safe):

```
{
  "tools": {
    "alsoAllow": ["lobster"]
  }
}
```

Or per-agent:

```
{
  "agents": {
    "list": [
      {
        "id": "main",
        "tools": {
          "alsoAllow": ["lobster"]
        }
      }
    ]
  }
}
```

Avoid using `tools.allow: ["lobster"]` unless you intend to run in restrictive allowlist mode.

Note: allowlists are opt-in for optional plugins. If your allowlist only names plugin tools (like `lobster`), OpenClaw keeps core tools enabled. To restrict core tools, include the core tools or groups you want in the allowlist too.

Example: Email triage

Without Lobster:

```
User: "Check my email and draft replies"
→ openclaw calls gmail.list
→ LLM summarizes
→ User: "draft replies to #2 and #5"
→ LLM drafts
→ User: "send #2"
→ openclaw calls gmail.send
(repeat daily, no memory of what was triaged)
```

With Lobster:

```json
{
  "action": "run",
  "pipeline": "email.triage --limit 20",
  "timeoutMs": 30000
}
```

Returns a JSON envelope (truncated):

```json
{
  "ok": true,
  "status": "needs_approval",
  "output": [{ "summary": "5 need replies, 2 need action" }],
  "requiresApproval": {
    "type": "approval_request",
    "prompt": "Send 2 draft replies?",
    "items": [],
    "resumeToken": "..."
  }
}
```

User approves → resume:

```
{
  "action": "resume",
  "token": "<resumeToken>",
  "approve": true
}
```

One workflow. Deterministic. Safe.

Tool parameters

run

Run a pipeline in tool mode.

```
{
  "action": "run",
  "pipeline": "gog.gmail.search --query 'newer_than:1d' | email.triage",
  "cwd": "/path/to/workspace",
  "timeoutMs": 30000,
  "maxStdoutBytes": 512000
}
```

Run a workflow file with args:

```
{
  "action": "run",
  "pipeline": "/path/to/inbox-triage.lobster",
  "argsJson": "{\"tag\":\"family\"}"
}
```

resume

Continue a halted workflow after approval.

```
{
  "action": "resume",
  "token": "<resumeToken>",
  "approve": true
}
```

Optional inputs

- `lobsterPath` : Absolute path to the Lobster binary (omit to use `PATH`).
- `cwd` : Working directory for the pipeline (defaults to the current process working directory).
- `timeoutMs` : Kill the subprocess if it exceeds this duration (default: 20000).
- `maxStdoutBytes` : Kill the subprocess if stdout exceeds this size (default: 512000).
- `argsJson` : JSON string passed to `lobster run --args-json` (workflow files only).

Output envelope

Lobster returns a JSON envelope with one of three statuses:

- `ok` → finished successfully
- `needs_approval` → paused; `requiresApproval.resumeToken` is required to resume
- `cancelled` → explicitly denied or cancelled

The tool surfaces the envelope in both `content` (pretty JSON) and `details` (raw object).

Approvals

If `requiresApproval` is present, inspect the prompt and decide:

- `approve: true` → resume and continue side effects
- `approve: false` → cancel and finalize the workflow

Use `approve --preview-from-stdin --limit N` to attach a JSON preview to approval requests without custom jq/heredoc glue. Resume tokens are now compact: Lobster stores workflow resume state under its state dir and hands back a small token key.

OpenProse

OpenProse pairs well with Lobster: use `/prose` to orchestrate multi-agent prep, then run a Lobster pipeline for deterministic approvals. If a Prose program needs Lobster, allow the `lobster` tool for sub-agents via `tools.subagents.tools` . See OpenProse.

Safety

- **Local subprocess only** — no network calls from the plugin itself.
- **No secrets** — Lobster doesn't manage OAuth; it calls OpenClaw tools that do.
- **Sandbox-aware** — disabled when the tool context is sandboxed.
- **Hardened** — `lobsterPath` must be absolute if specified; timeouts and output caps enforced.

Troubleshooting

- `lobster subprocess timed out` → increase `timeoutMs` , or split a long pipeline.

- `lobster output exceeded maxStdoutBytes` → raise `maxStdoutBytes` or reduce output size.
- `lobster returned invalid JSON` → ensure the pipeline runs in tool mode and prints only JSON.
- `lobster failed (code …)` → run the same pipeline in a terminal to inspect stderr.

Learn more

- Plugins
- Plugin tool authoring

Case study: community workflows

One public example: a “second brain” CLI + Lobster pipelines that manage three Markdown vaults (personal, partner, shared). The CLI emits JSON for stats, inbox listings, and stale scans; Lobster chains those commands into workflows like `weekly-review`, `inbox-triage`, `memory-consolidation`, and `shared-task-sync`, each with approval gates. AI handles judgment (categorization) when available and falls back to deterministic rules when not.

- Thread: https://x.com/plattenschieber/status/2014508656335770033
- Repo: https://github.com/bloomedai/brain-cli

‹ Tools LLM Task ›

Powered by mintlify