TUTORIAL

Your first ticket, end to end

About 20 minutes. No prior Symphony knowledge assumed; you need a terminal, Python and git.

한국어 Cheatsheet PDF github.com/cskwork/oh-my-symphony

1 What Symphony is

Symphony is a local orchestrator: it polls a Kanban board, hands each ticket to the AI coding agent you chose, and runs those agents concurrently, each inside its own isolated `git worktree`. You watch progress in a terminal Kanban (the TUI) or in the built-in web board on `http://127.0.0.1:9999/`. Tickets are Markdown files next to your code: no SaaS, no API key, no signup.

`install` → `project create / add` → `doctor` → `tui / service` → `ticket` → `agent runs` → `Human Review` → `Done`

It is for solo devs running unattended overnight refactors, teams parallelizing bug fixes across several agents, researchers comparing the eight agents on one task, and anyone who hit the "one chat window per agent" ceiling.

2 Prerequisites

Python 3.12 or newer, git, and one agent CLI on `$PATH`, already logged in:

`agent.kind`	CLI and the sign-in the README gives
`codex`	`codex` (with the `app-server` subcommand)
`claude`	`claude` (Claude Code)
`gemini`	`gemini` (Gemini CLI)
`agy`	`agy` (Antigravity CLI, from Google Antigravity)
`kiro`	`kiro-cli` (`https://cli.kiro.dev/install`); `kiro-cli login` or `KIRO_API_KEY`
`opencode`	`npm install -g opencode-ai` ; then `opencode auth login`
`pi`	`npm i -g @earendil-works/pi-coding-agent` ; sign in once via `pi` → `/login`
`prime-agent`	`curl -fsSL https://app.primeintellect.ai/prime-agent/install.sh | sh` ; sign in once via `prime-agent` → `/login`

What you should see: `which claude` (or your CLI) prints a path.

3 Install

Goal: a virtualenv with the `symphony` command.

```
git clone https://github.com/cskwork/oh-my-symphony.git
cd oh-my-symphony
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -e ".[dev]"
symphony --version
```

What you should see: `symphony 0.23.0`.

4

Create or register a project

Goal: a project directory of your own. The `oh-my-symphony` checkout stays the control plane; doctor and startup refuse a `WORKFLOW.md` inside Symphony's own repository, so agents can never edit the orchestrator that launched them.

```
symphony project create "My App" --path ../my-app
# or adopt an existing directory; Git and missing Symphony files are added
safely
symphony project add /path/to/existing-project --name "My App"
cd ../my-app
```

What you should see: a new Git repository for a missing path; an existing non-Git directory is initialized in place; an existing Git repository keeps its metadata and files. Symphony adds only missing operator files and never stages unrelated changes.

> **Existing non-Git directories:** Symphony commits only the files it adds. Your product files remain untracked, so review and commit them before dispatching tickets or the worker worktrees will not include them.

5

Read the generated WORKFLOW.md

Goal: understand the four blocks that matter for first-run sanity. Open `WORKFLOW.md` in your project (the shipped file-tracker template is `WORKFLOW.file.example.md`).

```
tracker:
  kind: file
  board_root: ./kanban
  active_states: [Todo, "In Progress", Verify, Document]
  terminal_states: ["Human Review", Done, Blocked, Archive]

workspace:
  root: ~/symphony_workspaces

hooks:
  # Each ticket gets its own workspace at workspace.root/<ID>, attached as a
  # git worktree of the host repo on a symphony/<ID> branch.
  after_create: |
    bash "$SYMPHONY_WORKFLOW_DIR/scripts/symphony-setup-worktree.sh"

agent:
  kind: codex          # codex | claude | gemini | agy | kiro | opencode | pi |
prime-agent
  max_concurrent_agents: 1

server:
  port: 9999
```

```
prompts:
  base: ./docs/symphony-prompts/file/base.md
  stages:
    Todo: ./docs/symphony-prompts/file/stages/todo.md
    "In Progress": ./docs/symphony-prompts/file/stages/in-progress.md
    Verify: ./docs/symphony-prompts/file/stages/verify.md
    Document: ./docs/symphony-prompts/file/stages/document.md
    Done: ./docs/symphony-prompts/file/stages/done.md
```

`tracker` says where tickets live and which columns exist. `hooks` run around each agent turn; the default attaches the workspace as a worktree so the host working tree is never disturbed. `agent.kind` picks the backend: change it to `claude`, `pi`, or any other value from the comment, and only the matching block (`claude:`, `pi:`, …) is used at runtime. `prompts` sends `base` plus only the file for the ticket's current state.

What you should see: a `kanban/` directory next to `WORKFLOW.md`, and an `agent.kind` that matches the CLI you installed.

6

Preflight with `symphony doctor`

Goal: catch the common first-run failures before anything launches.

```
symphony doctor ./WORKFLOW.md
```

What you should see, one line per check:

```
PASS  server.port=9999              127.0.0.1:9999 is free
PASS  agent.kind=claude             claude → /usr/local/bin/claude
FAIL  hooks.after_create            contains placeholder 'my-org/my-repo' —
every dispatch will fail with rc=128. Switch to the worktree default or replace
with a real clone / `: noop`.
PASS  workspace.root=~/symphony_workspaces  exists and is writable
PASS  tracker.board_root            ./kanban (3 tickets)
```

Exit code `0` means every check passed, `1` means at least one FAIL, `2` means `WORKFLOW.md` itself could not be loaded. Doctor catches port collisions, a missing CLI on `$PATH`, the placeholder clone URL, an unwritable workspace and a missing board directory. Fix what a FAIL line names and run it again until it is clean.

7

Optional: try it without an agent

Goal: watch a card come alive with the bundled mock backend. This step is optional; skip it if your real CLI is ready. The mock speaks Codex's protocol but only simulates turns and token ticks.

```
symphony project create "Symphony Demo" --path ../symphony-demo
cd ../symphony-demo

cat > WORKFLOW.md <<'YAML'
---
tracker: { kind: file, board_root: ./kanban,
           active_states: [Todo, "In Progress", Verify, Document],
           terminal_states: ["Human Review", Done, Blocked, Archive] }
polling: { interval_ms: 5000 }
workspace: { root: ~/symphony_workspaces }
hooks:
  after_create: ": noop"
```

```
  before_run:   ": noop"
  after_run:    "echo done"
agent:  { kind: codex, max_concurrent_agents: 1, max_turns: 4, max_total_turns:
60 }
codex:  { command: python -m symphony.mock_codex }
server: { port: 9999 }
---
You are picking up ticket {{ issue.identifier }}: {{ issue.title }}.
YAML

symphony board init ./kanban
symphony board new TASK-1 "smoke test"
symphony tui ./WORKFLOW.md
```

What you should see: within about 5 seconds TASK-1 grows a green ● in the Todo column with a climbing turn counter and token total. Quit with `Ctrl-C`.

> **Cards stay in Todo under the mock.** Only a real agent rewrites `kanban/TASK-1.md` to move the card. The mock proves the orchestrator → backend → workspace → hooks pipeline without an LLM call.

8

Add a ticket, three ways

Goal: one Markdown file at `kanban/TASK-1.md`. Back in your real project:

```
symphony board new TASK-1 "Fix flaky pagination test" \
  --priority 2 \
  --labels backend,test \
  --description "tests/test_pagination.py::test_cursor_advance is flaky on CI."
# → created kanban/TASK-1.md

symphony board ls                    # all tickets
symphony board show TASK-1           # full body
```

The second way is the TUI: press `n` on the board to register a new ticket with a multiline body (file board only), `e` edits the focused one. The third is the web board: the Board page creates, edits and deletes issues through the same validation as the CLI.

What you should see: `board new` validates before writing. The id must match `^[A-Za-z][A-Za-z0-9_-]{0,63}$`, the state must be in `tracker.active_states` / `terminal_states`, and every `--blocked-by` target must exist. Add `--agent-kind codex` to pin a different backend for just this ticket.

9

Launch the TUI and read the board

Goal: see your ticket dispatched.

```
symphony tui ./WORKFLOW.md
```

What you should see: columns are your tracker states (`active_states` first, then `terminal_states`). Within one poll tick (`polling.interval_ms`, default 30 s) a worker is dispatched and the card grows a green ● with turn counter and token totals; ↻ yellow = retry queue, ✓ = completed this session.

Key	Try it now

`?`	Grouped help modal with every binding (`esc` closes)
`tab` / `enter`	Focus the next card / open its full-detail modal
`p`	Show or hide the detail pane; `]` focuses it, `[` returns
`P`	Pause / resume the focused running worker
`/`	Filter cards by id, title or label; `esc` clears

> **The ● does not move the card.** Columns follow the ticket file's `state` field, which only the agent writes; a running ticket stays in Todo until then. The TUI needs a real TTY.

10

Or run it as a service and open the web board

Goal: the same orchestrator in the background with a browser UI. `service start` runs `symphony doctor` first and records the workflow under `.symphony/run/<workflow-hash>.json`, so the same `WORKFLOW.md` cannot be started twice on different ports by accident.

```
symphony service start ./WORKFLOW.md --port 9999
symphony service status ./WORKFLOW.md
symphony service logs ./WORKFLOW.md
# later: symphony service restart ./WORKFLOW.md  |  symphony service stop
./WORKFLOW.md
```

What you should see: open `http://127.0.0.1:9999/`. Running cards carry **Pause / Resume** buttons, the header refresh button triggers a `poll + reconcile`, and the header shows real local git branch dropdowns for `agent.feature_base_branch` (where new feature branches start) and `agent.auto_merge_target_branch` (where the Done merge lands). The board is loopback-only.

> **Optional bearer token.** Set `export SYMPHONY_API_TOKEN="$(python -c 'import secrets; print(secrets.token_urlsafe(32))')"` before launching; every operator `/api/` request then needs `Authorization: Bearer`, and the web app prompts after its first `401`. Keep the value out of `WORKFLOW.md`, shell history and logs.

11

Follow the ticket through the states

Goal: understand what each column means while you wait.

State	What the agent does
Todo	Triage; route to In Progress. A ticket with a body and an Acceptance Criteria section moves on without spending a model turn.
In Progress	Plan + TDD implementation + self-critique, on the `symphony/<ID>` branch inside its worktree.
Verify	Review + QA + Merge Gate. The feature branch must merge into the target branch before the ticket can move to Document.
Document	Docs + wiki write-back; Done unless intervention is needed.
Human Review	Manual intervention or explicit review before Done. The agent stops here; you decide.

Done	Verified complete.

What you should see at Human Review: the card parked in a terminal column, no ● indicator. Focus it and press `c` in the TUI to confirm it as Done, or move it to Done from the web board. Symphony never confirms Done for you.

12

Inspect the result

Goal: find the code, the notes and the run history.

```
symphony board show TASK-1              # the agent's ## Resolution lives in
the body
ls ~/symphony_workspaces/TASK-1         # workspace it operated in
symphony runs ./WORKFLOW.md --limit 5   # recent run-registry attempts
git branch --list 'symphony/*'          # the ticket's feature branch
```

What you should see: a `## Resolution` section appended to the ticket, committed artefacts under `docs/<TICKET-ID>/<stage>/` on the ticket branch, and reviewer files (screenshots, reports) copied to `.symphony/artifacts/<TICKET-ID>/` on the host, listed under `## Artifacts` on the ticket and shown in the web ticket drawer. In headless mode (`symphony ./WORKFLOW.md` without `tui`) a live `WORKFLOW-PROGRESS.md` is rewritten next to the workflow on every tick; it is read-only output, do not edit it by hand.

13

Everyday operations

Task	How
Pause / resume a worker	`P` on the focused running card, or the web buttons
Retry	Automatic: a failed worker enters the ↻ retry queue with exponential backoff up to `agent.max_retries` ; `r` forces a refresh + re-poll
Archive	`a` on a Done card; `tracker.archive_after_days` also sweeps idle terminal tickets
Skip Document	`S` on the focused card for tickets that need no wiki write-back
Move a ticket by hand	`symphony board mv TASK-1 Blocked` or `symphony board update TASK-1 --state Blocked --add-blocked-by BUG-7` ; the orchestrator re-evaluates on the next poll tick

What you should see: manual moves are for unsticking. Normally the agent transitions tickets itself, following the stage prompt files in `WORKFLOW.md` .

14

Multiple projects

Goal: run several boards from one control plane.

```
symphony project list
symphony project start my-app
symphony project status
symphony project stop my-app
symphony hub                            # central UI for every registered
project
```

What you should see: the Hub and every project board show the canonical repository, workflow and "Issues are stored here" paths. Selecting another project starts its service when needed; services and workers for the previous project keep running against their original repository, so switching cannot retarget active jobs.

15

Troubleshooting and where to go next

Symptom	Fix
doctor prints FAIL	Fix what the line names and re-run. Most common: the placeholder clone URL; switch to the worktree default or `: noop`. Exit `2` = the YAML did not load.
Worker exited, card shows ↻	Retries back off up to `agent.max_retries`, then the ticket is escalated and parked. Last error: `symphony runs ./WORKFLOW.md --limit 5`, `symphony service logs ./WORKFLOW.md`.
Agent silent for minutes	No progress event inside `stall_timeout_ms` cancels the worker. Widen only the heavy lane: `agent.stall_timeout_ms_by_state: {Verify: 900000}`. Logs go to stderr: `symphony tui ./WORKFLOW.md 2>> log/symphony.log`.
Port in use	doctor reports it. Override with `--port N`, or check `symphony service status ./WORKFLOW.md`.
TUI exits silently	Needs a real TTY: use a foreground terminal, or go headless (`symphony ./WORKFLOW.md`) and follow `WORKFLOW-PROGRESS.md`.

Next: the README (lane presets, chat intake, JSON API), the CHANGELOG, Issues, and the cheatsheet.

oh-my-symphony 0.23.0 · Apache-2.0 · generated from README on the same commit