

Aurora Workflow Orchestration Adoption Guide

Shawn C. Wright, Aurora Research Initiative / Waveframe Labs Division

October 2025

AWO Adoption Guide

Version: v1.2.1 Maintainer: Waveframe Labs License: CC BY 4.0 (docs),
Apache 2.0 (code)

This guide shows how to adopt **Aurora Workflow Orchestration (AWO)** in real projects — from a solo, manual workflow to a team setup with CI. It assumes familiarity with Git and basic command-line usage.

AWO is a reproducibility protocol (method), not a software library. You can adopt it **manually** first, then automate enforcement later.

Table of Contents

1. Scope and Outcomes
 2. Prerequisites
 3. AWO in One Page
 4. Adoption Tiers (Minimum / Standard / Full)
 5. Your First AWO Project (Step by Step)
 6. Manual Orchestration (Before CRI-CORE)
 7. Solo Researcher Tips
 8. Directory Layout (Starter)
 9. Provenance Artifacts and Logs
 10. Decision Logging (ADRs) and Falsifiability Manifests
 11. Verification Checklist
 12. Example Projects Using AWO
 13. Troubleshooting and Common Pitfalls
 14. Cross-Links and References
 15. License
-

1. Scope and Outcomes

By the end of this guide you will:

- Initialize a project using AWO's file structure.
- Produce a **signed, reproducible run** with an auditable record under `/runs/`.
- Log decisions and assumptions using ADRs and falsifiability manifests.
- Validate the run against the **AWO Method Spec v1.2.1**.
- Know how to upgrade from manual orchestration to CI, and later to CRI-CORE.

2. Prerequisites

- Git, Python 3.10+.
- A Git hosting provider (e.g., GitHub).
- Optional: Pandoc for local PDF builds (otherwise use the repo's CI workflows).

3. AWO in One Page

- **What it is:** a procedural standard that makes AI-human research **falsifiable, auditable, and citable**.
- **What it isn't:** a monolithic tool or framework you must install before working.
- **How it works:** you structure your work, log decisions, run experiments, and produce **attested artifacts** (hashes, signatures, checksums) that prove exactly what happened.

You do not need CRI-CORE to use AWO. CRI-CORE later automates enforcement.

4. Adoption Tiers

Minimum (Individual)

- Manual logs under `/logs/`.
- ADRs for decisions under `/decisions/`.
- Falsifiability manifest per experiment.
- One signed run under `/runs/<timestamp>/` with:
 - `workflow_frozen.json` (what ran), `report.md` (what happened), `approval.json` (who signed).
- SHA256 checksums for public artifacts.

Standard (Small Team)

- Everything in Minimum, plus:
- CI workflows to build PDFs and guard docs (doc-guard).
- Required reviews for ADRs and releases.
- Release notes with artifact checksums.

Full (Institutional / Public)

- Everything in Standard, plus:
 - Automated attestation and schema validation via CRI-CORE.
 - Policy gates (e.g., “no run without a falsifiability manifest”).
 - Organization-level provenance registry and release automation.
-

5. Your First AWO Project (Step by Step)

If you have no data yet, use a synthetic or toy dataset; the process is the goal.

1) Create a new repository

```
my-awo-project/  
docs/  
decisions/  
figures/  
logs/  
runs/  
README.md
```

- 2) **Add a falsifiability manifest** Use the template at `/templates/falsifiability-manifest.md` to draft your hypothesis and disproof criteria. For each actual run, place the manifest in the run folder as `runs/<timestamp>/manifest.json` (or `runs_manifest.md`).

3) Create ADR-0001 Create `decisions/ADR-0001.md`:

```
Title: Scope and Baseline Assumptions  
Status: Accepted  
Context: Why this project exists; key assumptions.  
Decision: What we will do first.  
Consequences: Trade-offs and risks.
```

- 4) **Record a workflow plan** Create `docs/WORKFLOW_PLAN.md` with a short, numbered plan and a rough timing.
- 5) **Run a minimal AWO execution** If you cloned the AWO repo to borrow utilities, from its root:

```
python cli.py --init --demo
```

Otherwise, run your own script(s); capture all parameters to `runs/<timestamp>/workflow_frozen.json`. (Optional: store raw inputs under `runs/<timestamp>/inputs/.`)

6) Freeze and sign the run Create:

- `runs/<timestamp>/workflow_frozen.json`
- `runs/<timestamp>/report.md`

- `runs/<timestamp>/approval.json`

Include SHA256 checksums for any public outputs.

7) Tag a read-only snapshot

- Create a Git tag and a release description that lists:
 - Manifest version(s), ADRs referenced, and artifact checksums.
- 8) **Validate against the spec** Open `docs/AWO_Method_Spec_v1.2.md` and walk the conformance list:
 - Structure present, manifests present, run attested, decisions logged, signatures/checksums recorded.

6. Manual Orchestration (Before CRI-CORE)

Manual orchestration is a first-class path. Use it when CI is not set up yet.

- State clearly in your README: “This project uses **manual orchestration** under AWO v1.2; no automated CI.”
- Keep provenance local but signed.
- Use human approvals in `approval.json` and reference ADRs in every report.
- Keep `SHA256SUMS.txt` at the repo root when you publish artifacts.

You can later retrofit CI (Standard tier) and CRI-CORE (Full tier) without rewriting history.

7. Solo Researcher Tips

- Keep ADRs short (1 page) and frequent.
- Prefer toy datasets or deterministic tasks for early runs.
- Close the loop quickly: hypothesis → run → report → tag.
- Document failures explicitly; they count as evidence.

8. Directory Layout (Starter)

```
/docs/           # manifests, plans, method links
/decisions/      # ADRs (ADR-0001..)
/figures/        # diagrams
/logs/           # narrative logs or time-stamped notes
/runs/           # attested outputs per run
README.md
```

9. Provenance Artifacts and Logs

- `workflow_frozen.json` – exact steps and parameters executed.

- `report.md` – results, observations, failures.
- `approval.json` – signatures or approvals (human-in-the-loop).
- `SHA256SUMS.txt` – checksums of released artifacts.
- Optional: signed tarball of `/runs/<timestamp>/` for archival.

10. Decision Logging and Falsifiability Manifests

- **ADRs** capture *why* changes happened; one ADR per substantive decision.
- **Falsifiability manifests** define *what would disprove* the hypothesis.
 - Template lives in `/templates/falsifiability-manifest.md`.
 - The run’s live copy must be stored in `runs/<timestamp>/run_manifest.json` (or `run_manifest.md`) and referenced by `approval.json`.
- Cross-reference ADR IDs inside `report.md` and release notes.

11. Verification Checklist

- ☐ Repository follows AWO structure (docs, decisions, logs, runs).
- ☐ Falsifiability manifest exists and is referenced.
- ☐ At least one run is frozen and signed under `/runs/`.
- ☐ ADRs exist and are cited in the run report.
- ☐ SHA256 checksums published for outputs.
- ☐ Release/tag contains links to manifests and run artifacts.
- ☐ README links to AWO Method Spec, Whitepaper, and this guide.

12. Example Projects Using AWO

These are manual orchestration examples — built before CRI-CORE was available. They show how AWO principles work without automated CI. All provenance, audit, and decision artifacts were managed by hand and logged per AWO v1.2.

- Waveframe v4.0 (Cosmology) — AWO structure with falsifiability logs, ADRs, and reproducible documentation.
- Societal Simulator (Systems modeling) — Interactive sandbox applying AWO with manual provenance and audit.

Full runtime automation is demonstrated in the CRI-CORE repository and will be extended in template kits.

13. Troubleshooting and Common Pitfalls

- **No data yet:** Use toy data; focus on process, not results.
- **Messy parameters:** Write them to `workflow_frozen.json` at runtime.
- **Unclear decisions:** Create ADRs even for small choices; they prevent ambiguity later.

- **Non-determinism:** Pin seeds and constrain thread counts (e.g., `PYTHONHASHSEED=0`, `OMP_NUM_THREADS=1`).

14. Cross-Links and References

- Method Spec: `docs/AWO_Method_Spec_v1.2.1.md`
- Whitepaper: `docs/AWO_Whitepaper_v1.1.md`
- Template: <https://github.com/Waveframe-Labs/AWO-Template>
- Main README: `../README.md`

15. License

- Documentation: CC BY 4.0
- Code: Apache 2.0

© 2025 Waveframe Labs · Independent Open-Science Research Entity