AIT-NATIVE / PUBLIC TECHNICAL REFERENCE

# Technical Documentation

Version-pinned concepts, workflows, command references, integrations, operations, and release authority for ait-native RC.5.

| VERSION | DOCUMENTATION |
| --- | --- |
| **RC.5 / 1.0.0-rc.5** | **Revision 1** |
| CONTENTS | CLI REFERENCE |
| **21 technical pages** | **11 public commands** |

ait-native.dev/technical/
Published 2026-08-14

# Contents

Use the linked page numbers to move through the PDF. Each chapter also links to its exact online route.

START

# Technical Documentation

The versioned entry point for ait-native concepts, workflows, commands, integrations, and operations.

**AUDIENCE Developers and operators**

https://ait-native.dev/technical/v1.0.0-rc.5/

## Use the shortest page that answers the question

This documentation is the technical companion to the basic Docs page. It is fixed to **ait-native 1.0.0-rc.5** and separates ordinary use, agent-owned workflow commands, integration APIs, and remote operations.

- Start with Getting Started when adding AIT to a repository.
- Read Core Concepts when an AIT record or identifier needs context.
- Use the CLI reference for exact command roles and bounded examples.
- Open Remote Infrastructure only when a server and runner solve a concrete coordination need.

## The operating model

The user initializes the repository once and describes a result. The coding agent reads the generated repository instructions, scopes governed work to an exact sprint item, implements inside the bound workspace, records a Snapshot, and completes the configured closeout path.

TEXT

```
request and constraints
  -> exact sprint item
  -> Task-bound implementation
  -> repository checks
  -> immutable Snapshot
  -> configured closeout
```

The local workflow does not require a running server. Remote authority and CI are explicit additions.

## Version boundary

Every page in this section is tied to the public RC.5 source tag. The version panel links to the exact source, Release, distribution contract, and CLI parser used to check this public documentation.

Development-main behavior can be newer than these pages. When a command from a different version disagrees with this documentation, use the documentation for that installed version.

## Where to go next

- Feature Workflow
- Regression Workflow
- Distribution and Release
- Troubleshooting

START

# Getting Started

Install RC.5, initialize one repository, make a request, and inspect the recorded result.

**AUDIENCE Developers**

https://ait-native.dev/technical/v1.0.0-rc.5/getting-started/

## 1. Install the current release candidate

Choose one verified RC.5 route from the Quickstart. The installed native command must report the expected product version before the repository is initialized.

BASH

```
ait --version
```

PyPI, npm, Homebrew, apt, and GitHub assets have different package envelopes, but they lead to the same repository workflow. The server remains inactive until it is explicitly configured and started.

## 2. Initialize one repository

Run initialization from the repository the coding agent will change.

BASH

```
cd <your-repository>
ait init
```

AIT creates repository authority and creates or refreshes the effective workflow block in `AGENTS.md`. It does not select a programming language, framework, build command, or test command.

## 3. Make a normal request

Describe the result and constraints instead of translating the work into AIT bookkeeping.

TEXT

```
Add an accessible password-visibility control.
Preserve the login API and keyboard behavior, and add the relevant tests.
```

The coding agent follows the repository's generated instructions. For governed work, that includes the sprint item, Task, bound workspace, repository checks, Snapshot, and configured closeout.

## 4. Inspect the result

Use read-only commands when you need direct evidence.

BASH

```
ait status --json
ait diff --stat
ait queue summary --all-changes
```

An empty workspace after successful closeout is expected. The completed Task and Snapshot retain the recorded result.

## Next pages

- Core Concepts
- `ait init`
- `ait status`

- Feature Workflow

START

# Core Concepts

Understand the repository objects that connect intent, isolated work, immutable state, evidence, and closeout.

**AUDIENCE Developers and integrators**

https://ait-native.dev/technical/v1.0.0-rc.5/concepts/

## Repository and Line

A **Repository** is the local AIT authority for one working directory. A **Line** is a named, movable pointer to an admitted Snapshot. `main` is the normal initial Line; a Task receives its own feature Line and bound workspace.

## Plan item

A **Plan** records versioned Markdown lineage. In sprint mode, one stable `[plan-ref: ...]` identifies the document and one unchecked checklist item with an exact `[ref: ...]` is the taskable unit.

MARKDOWN

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

The item states the outcome. It is not an instruction to execute several unrelated changes.

## Task, Change, and workspace

A **Task** is the bounded work unit tied to the Plan intent. A **Change** is its reviewable change lineage. The Task-bound workspace separates implementation from the target Line until the result is admitted.

The command that starts the Task prints the exact workspace path. The agent must edit there, not in the canonical repository root.

## Snapshot

A **Snapshot** is immutable repository state with authorship and parent history. It records the completed code state; it does not include an unrelated Markdown Plan edit merely to hide documentation lineage inside code history.

## Patchset and evidence

In a remote-backed workflow, a **Patchset** is the selected review candidate for one Change. CI, attestation, review, and policy evidence attach to that candidate before final closeout.

Local-only work does not require remote infrastructure. Its configured admission policy still determines what must be verified before completion.

## Closeout

Closeout advances the configured target Line, completes the Task, cleans the bound workspace, and applies the repository's sprint-checklist policy. Remote work may require the final Markdown checkbox update and Plan sync as a separate step after code closeout.

## Related reference

- `ait plan`
- `ait task`
- `ait snapshot`

- ait workflow

WORKFLOWS

# Feature Workflow

Follow a bounded feature request from sprint item through implementation, verification, and closeout.

**AUDIENCE Developers and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/workflows/feature/

## Scenario

The feature example matches the interactive Demo: add an accessible password-visibility control while preserving the login API and keyboard behavior.

The user supplies the request. The coding agent carries the repository workflow.

## Scope the work

The agent writes one sprint card with one exact taskable item.

MARKDOWN

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

It then starts the bound Task from that exact item.

BASH

```
ait task start \
  --from docs/sprints/password_visibility.md#login/password-visibility/implement \
  --intent "Add accessible password visibility control" \
  --base-line main
```

The output supplies the Task, Change, and workspace path.

## Implement in the bound workspace

The agent enters the printed path, updates the control and focused tests, and runs the repository-owned validation. AIT does not guess a framework or invent a test command.

TEXT

```
src/LoginForm.tsx
tests/LoginForm.test.tsx
3 tests passed
```

## Record the result

After the checks pass, the agent records the code state.

BASH

```
ait snapshot create --message "Add accessible password visibility control"
```

Remote-backed work then uses the guided readiness command until the selected Patchset has the required CI, attestation, review, and policy evidence.

BASH

```
ait workflow ready <change-id> --apply
```

## Complete and verify

The agent finishes with the Task command from the generated repository instructions, then verifies Task, workspace, Line, and sprint-item closeout.

BASH

```
ait task land <task-or-change-id>
ait task audit <task-id>
```

Use the Interactive Demo to watch the same flow, or open the `ait task` reference for command details.

WORKFLOWS

# Regression Workflow

Identify the responsible revision, isolate the repair, add a regression test, and close the new Task.

**AUDIENCE Developers and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/workflows/regression/

## Scenario

The existing password control no longer responds correctly to keyboard activation. The repair must preserve the intended behavior and add a test that fails on the responsible revision.

## Identify the responsible state

Before choosing the repair, inspect the relevant line or bounded range.

BASH

```
ait blame src/LoginForm.tsx --line 84
```

The result connects the source line to its responsible Snapshot and available Plan lineage. This evidence answers where the behavior entered history; it does not automatically decide the repair.

## Create a new repair Task

Record a new sprint item for the regression instead of reopening completed work.

MARKDOWN

```
# Password Keyboard Regression [plan-ref: login/password-keyboard-regression/root]

- [ ] Restore native keyboard activation and lock it with a regression test. [ref:
  login/password-keyboard-regression/repair]
```

BASH

```
ait task start \
  --from docs/sprints/password_keyboard_regression.md#login/password-keyboard-regression/repair
    \
  --intent "Restore password-control keyboard activation" \
  --base-line main
```

## Prove the repair

The agent changes only the bounded behavior, adds the regression test, and runs the repository checks in the Task workspace.

TEXT

```
4 tests passed
```

It records a new Snapshot and follows the same configured readiness and closeout path as any other governed change.

```bash
ait snapshot create --message "Restore password-control keyboard activation"
ait task land <task-or-change-id>
```

## Recovery rule

If blame evidence is incomplete or the repair grows beyond the stated scope, stop and re-scope the Task. Do not overwrite unrelated workspace changes or reinterpret a prior completed Task as active work.

See `ait blame` and Troubleshooting.

**CLI · USER AND INSPECTION**

# ait init

Create or reinitialize AIT repository authority and effective agent instructions.

**AUDIENCE Users**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/init/

## Role

Normally run by the **user** once from the repository that a coding agent will change.

## Purpose

`ait init` creates missing local repository authority and creates or refreshes the effective AIT workflow block in `AGENTS.md`. Reinitialization preserves valid existing authority instead of replacing it.

## Syntax

```bash
ait init
ait init --json
```

Important optional inputs include the initial repository name, default Line, policy profile, author mode, and a bounded repair flag for incomplete existing structure. Ordinary first use needs no option.

## Example

```bash
cd ~/src/example-app
ait init
```

After success, open `AGENTS.md` or run `ait config show` to inspect the effective workflow routing. Initialization does not start `ait-server` and does not infer a programming language or validation command.

## Output

Text output summarizes the created or preserved repository and guidance. JSON is the stable machine-readable result for automation.

## Failure and recovery

If the directory contains malformed or partial AIT state, keep the error and inspect the existing `.ait` structure before using a repair option. Never replace authority merely to make initialization succeed.

## Related

- Getting Started
- `ait config`
- `ait status`

CLI · USER AND INSPECTION

# ait config

Inspect effective workflow routing and apply explicit repository configuration changes.

**AUDIENCE Users and operators**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/config/

## Role

Normally read by the **user or coding agent**. Configuration changes should be made only when the repository owner has selected the new behavior.

## Purpose

`ait config` reports effective repository routing and applies explicit workflow presets or bounded overrides. A successful relevant change also refreshes the generated AIT block in `AGENTS.md`.

## Syntax

BASH

```
ait config show
ait config show --json
ait config set --workflow-mode solo_local
ait config set --sprint on
```

For a remote-backed individual workflow, the corresponding primary preset is `solo_remote`. Prefer a primary preset before adding a narrow override.

## Example

BASH

```
ait config show --json
ait config set --workflow-mode solo_local
ait config show
```

Inspect the generated guidance after the write. The printed task, Plan, and closeout commands are the repository-specific route.

## Output

Text output shows the effective values most useful for a decision. JSON provides the complete machine-readable configuration projection.

## Failure and recovery

Changing a default does not move an existing Task, Change, Plan binding, or workspace to the new scope. Complete or recover existing lineage under its recorded contract before assuming the new default applies.

## Related

- `ait init`

- `ait workflow`
- Remote Infrastructure

CLI · USER AND INSPECTION

# ait status

Inspect the current Line, repository state, workspace state, remotes, and worktree health.

**AUDIENCE Users and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/status/

## Role

Safe for the **user or coding agent** as the first repository inspection.

## Purpose

`ait status` summarizes the current Line, head Snapshot, workspace state, configured remotes, repository identity, and worktree health. It is read-only with respect to product lineage.

## Syntax

BASH

```
ait status
ait status --json
```

## Example

BASH

```
ait status --json
```

Use JSON when a coding agent must decide whether the canonical root or a Task-bound workspace is active. Use the compact text view for a quick human orientation.

## Output

Typical fields include the current Line, default remote, head Snapshot, workspace changed counts, repository name, and worktree hygiene. Operational health detail can change without creating a new Snapshot.

## Failure and recovery

If repository discovery fails, confirm the intended repository root and run `ait init` only when no valid authority exists. If status reports a dirty workspace, inspect `ait diff` before starting or recovering a Task.

## Related

- `ait diff`
- `ait queue`
- Troubleshooting

CLI · USER AND INSPECTION

# ait diff

Compare the current workspace with the current Line head using bounded path filters.

**AUDIENCE Users and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/diff/

## Role

Safe for the **user or coding agent** before a Snapshot, repair, or recovery decision.

## Purpose

`ait diff` compares the current workspace with the current Line head. It does not select arbitrary Snapshot ranges; use `ait snapshot diff` for immutable Snapshot comparison.

## Syntax

BASH

```
ait diff
ait diff --stat
ait diff --name-only
ait diff --path <path>
```

Path inputs are exact lexical filters, not shell-style patterns. `--stat` and `--name-only` are alternate summaries.

## Example

BASH

```
ait diff --path src/LoginForm.tsx
```

The result classifies modified, missing, and untracked paths and bounds large or binary content rather than printing an unlimited payload.

## Output

The normal text view emphasizes changed paths and useful content. `--json` provides the machine boundary when automation needs exact classifications. Sprint Markdown is tracked through Plan lineage and does not appear as code workspace content.

## Failure and recovery

If an expected file is absent, inspect the Task workspace and Line before restoring anything. Do not overwrite unrelated changes to force a clean diff.

## Related

- `ait status`
- `ait snapshot`
- Troubleshooting

CLI · USER AND INSPECTION

# ait blame

Trace a file or line range to the responsible Snapshot and Plan revision before a repair.

**AUDIENCE Users and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/blame/

## Role

Run by the **coding agent or user** before choosing a repair for an identified regression.

## Purpose

`ait blame` attributes a path or bounded line range to the responsible Snapshot and available Plan revision lineage. It connects code state with the recorded change history without mutating the workspace.

## Syntax

BASH

```
ait blame <path>
ait blame <path> --line <number>
ait blame <path> --start <line> --end <line>
```

## Example

BASH

```
ait blame src/LoginForm.tsx --line 84
```

Use the smallest range that answers the question. A whole-file query is useful when the relevant region is not yet known.

## Output

The result identifies the responsible Snapshot and bounded attribution evidence. Plan information is included when that source state is connected to versioned Markdown lineage.

## Failure and recovery

Missing attribution is evidence, not permission to guess. Confirm the path, Line, and imported history; then record the repair as a new bounded Task.

## Related

- Regression Workflow
- `ait snapshot`
- `ait task`

CLI · USER AND INSPECTION

# ait doctor

Discover focused diagnostic surfaces without changing repository authority.

**AUDIENCE Users and operators**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/doctor/

## Role

Used by the **user or operator** when normal status evidence identifies a storage, runtime, Plan-authority, or remote dependency problem.

## Purpose

`ait doctor` is a namespace of focused diagnostics. The available diagnostic topics are versioned, so begin with the installed command's help instead of copying an unrelated maintenance command.

## Syntax

BASH

```
ait doctor --help
```

Worktree-specific diagnosis is available under `ait worktree doctor`; it is separate from the top-level doctor topics.

## Example

BASH

```
ait status --json
ait doctor --help
ait worktree doctor --refresh --json
```

Select only the diagnostic that matches the status finding. Diagnostics should not become a speculative repair sequence.

## Output

Doctor output reports the inspected boundary, evidence, and any bounded next action. Preserve it when escalation is required.

## Failure and recovery

Do not run a destructive cleanup merely because a doctor reports stale or partial state. Resolve the exact owner, identifier, and path first, then use the recovery command supplied for that condition.

## Related

- `ait status`
- Troubleshooting

CLI · AGENT WORKFLOW

# ait plan

Record versioned Markdown lineage and synchronize exact sprint items in the configured scope.

**AUDIENCE Coding agents and maintainers**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/plan/

## Role

Normally run by the **coding agent** under the exact scope printed in the repository's generated instructions.

## Purpose

`ait plan` records versioned Markdown lineage. In sprint mode, one exact open checklist item becomes the Task binding; ordinary documentation can be synced without becoming a Task.

## Syntax

BASH

```
ait plan sync <markdown-file-or-dir> --local
ait plan sync <markdown-file-or-dir> --remote origin
```

Use only the scope that matches the effective repository configuration. The initial `ait task start --from` invocation owns exact-file synchronization for its bound sprint item.

## Example

MARKDOWN

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

BASH

```
ait plan sync docs/architecture.md --local
```

## Output

Sync reports the Plan identity, revision, action, and publication scope. Plan views can inspect items and revisions without changing them.

## Failure and recovery

Do not copy a Plan ID into `task start` or hide a Markdown checkbox update in a code Snapshot. When remote closeout leaves the checklist update separate, edit the exact bound item and sync that card in the configured remote scope.

## Related

- Core Concepts
- `ait task`
- Feature Workflow

CLI · AGENT WORKFLOW

# ait task

Start Plan-bound work, inspect readiness, and complete the Task through its configured scope.

**AUDIENCE Coding agents and maintainers**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/task/

## Role

Normally run by the **coding agent**. The user states the desired result and constraints rather than manually driving Task bookkeeping.

## Purpose

`ait task` binds sprint intent, Change lineage, a feature Line, a managed workspace, and final closeout. In sprint mode, start from one exact taskable Markdown item.

## Syntax

BASH

```
ait task start --from <sprint-card>#<exact-ref> --intent <intent> --base-line main
ait task audit <task-id>
ait task land <task-or-change-id>
```

## Example

BASH

```
ait task start \
  --from docs/sprints/password_visibility.md#login/password-visibility/implement \
  --intent "Add accessible password visibility control" \
  --base-line main
```

The final output includes the Task, Change, and exact `cd` command. All code work belongs in that printed workspace.

## Audit and closeout

`ait task audit` reads the Task's readiness and closeout state. `ait task land` uses the recorded workflow scope. A remote Task must already have a selected Patchset with completed readiness evidence.

## Failure and recovery

If startup partially succeeds, keep the printed identifiers and inspect the Task before retrying. If a workspace is missing, use the worktree recovery surface; do not create a second Task for the same open sprint item unless the original lineage is explicitly resolved.

## Related

- `ait plan`
- `ait snapshot`
- `ait workflow`

CLI · AGENT WORKFLOW

# ait snapshot

Record immutable repository state and inspect Snapshot content or ancestry.

**AUDIENCE Coding agents and maintainers**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/snapshot/

## Role

Normally created by the **coding agent** after the bounded implementation and repository checks are complete. Snapshot inspection is safe for users.

## Purpose

`ait snapshot` records immutable repository state and exposes content, comparison, ancestry, and recovery queries. A Snapshot keeps its authoring Line identity even when a later Change or closeout target moves.

## Syntax

BASH

```
ait snapshot create --message <message>
ait snapshot show <snapshot-id> --files
ait snapshot ancestry <snapshot-id> --ancestors
```

Snapshot comparison uses two immutable identifiers:

BASH

```
ait snapshot diff <old-snapshot-id> <new-snapshot-id>
```

## Example

BASH

```
ait snapshot create --message "Add accessible password visibility control"
```

Run this from the Task workspace after validation. A normal Snapshot records code state; separately authored sprint Markdown remains Plan lineage.

## Output

Create returns the new Snapshot and parent. Show can include the full file inventory, while ancestry queries are bounded unless complete output is explicitly requested.

## Failure and recovery

A dirty or misplaced workspace should be inspected with `ait status` and `ait diff` before retrying. Revert and replay operations are separate explicit surfaces; preview their effect before replacing workspace content.

## Related

- `ait diff`
- `ait task`
- Core Concepts

CLI · AGENT WORKFLOW

# ait queue

Read a compact inventory of active work, reviews, remotes, and worktree health.

**AUDIENCE Users and coding agents**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/queue/

## Role

Safe for the **user or coding agent** when orienting active local and remote work.

## Purpose

`ait queue` is a read-only inventory composed from Tasks, Changes, workspace, worktrees, reviews, and configured remote state. It does not own a separate lifecycle queue.

## Syntax

BASH

```
ait queue summary --all-changes
ait queue summary --all-changes --json
```

## Example

BASH

```
ait queue summary --all-changes
```

The compact text result reports shared Tasks, ready candidates, review inbox, local drafts, workspace changes, and worktree hygiene.

## Output

`--all-changes` enriches the active inventory with nonterminal remote Changes. Any ready indication is orientation evidence; use `ait task audit` or the guided workflow surface before mutation.

## Failure and recovery

A remote read error can appear beside otherwise valid local inventory. Keep the error visible, check the configured remote and server health, and avoid treating missing remote rows as completed work.

## Related

- `ait status`
- `ait task`
- `ait workflow`

CLI · AGENT WORKFLOW

# ait workflow

Classify bounded edits, prepare remote readiness evidence, and inspect closeout state.

**AUDIENCE Coding agents and operators**

https://ait-native.dev/technical/v1.0.0-rc.5/cli/workflow/

## Role

Normally run by the **coding agent**. Operators may use the same decision surfaces to inspect a remote gate without bypassing it.

## Purpose

`ait workflow` classifies bounded edits, prepares remote Patchset readiness, and inspects or resumes closeout. Effective scope and exact commands come from the generated repository instructions.

## Syntax

BASH

```
ait workflow tier --json
ait workflow ready <change-id> --apply
ait workflow land <change-id> --apply
```

`workflow ready` and `workflow land` are text-only decision surfaces. Do not append `--json` to them.

## Tier selection

Run `workflow tier --json` only after the edit is bounded. A Task workspace continues through its existing Task and Change lineage even if the changed file count is small.

## Remote readiness

`workflow ready --apply` can create or synchronize the selected Patchset and advance required CI, attestation, review, and policy state. It stops with one exact next action when a gate needs separate work.

Final Task closeout is normally performed with:

BASH

```
ait task land <task-or-change-id>
```

## Failure and recovery

Do not substitute local checks for required remote CI, append unsupported output flags, or switch authority to avoid a blocker. Re-run the decision surface after the owning evidence changes and preserve the same Change ID.

## Related

- `ait task`
- `ait queue`
- Remote Infrastructure

INFRASTRUCTURE AND INTEGRATIONS

# Remote Infrastructure

Operate the explicit ait-server and ait-runner boundary for shared authority and repository-owned CI.

**AUDIENCE Operators**

https://ait-native.dev/technical/v1.0.0-rc.5/remote-infrastructure/

## Activate remote authority only when needed

The normal local workflow requires no server. Remote infrastructure adds shared durable authority and repository-owned CI execution; it also adds operator responsibility for authentication, TLS, network exposure, backups, upgrades, and compatibility.

## Component boundary

- `ait-server` owns the repository registry, shared workflow state, policy, evidence, and Worker Job queue.
- `ait-runner` claims compatible typed jobs, materializes exact Snapshot content, executes the repository's declared CI entrypoint, and returns bounded results.

The runner does not become repository authority and does not choose a language-specific build system.

## Evaluate the RC.5 server locally

Bind the evaluation container to loopback unless a reviewed secure ingress is already available.

BASH

```
docker network create ait-native-rc
docker volume create ait-native-rc-data
docker run --detach \
  --name ait-server \
  --network ait-native-rc \
  --publish 127.0.0.1:8088:8088 \
  --restart unless-stopped \
  --volume ait-native-rc-data:/var/lib/ait \
  ghcr.io/weita2026/ait-server:1.0.0-rc.5

curl --fail http://127.0.0.1:8088/healthz
```

The public container index covers Linux `amd64` and `arm64`.

## Register and configure the repository

Use the server's repository registration result and the generated AIT guidance for the exact repository index and remote name. Confirm effective routing before starting governed work.

BASH

```
ait remote list --json
ait config show --json
```

Do not copy a repository index from another repository. The index is part of remote authority routing.

## Runner contract

The runner accepts version-compatible jobs for one registered repository and executes its exact `ci/run.sh` or `ci/run.ps1` entrypoint. Source roots, attempt roots, worker identity, and repository index are operator-selected values.

Keep attempt storage isolated, monitor failed deliveries, and verify that attempt-owned data is cleaned after terminal results.

## Readiness and closeout

Remote work publishes a selected Patchset and records completed CI, attestation, review, and policy evidence before final Task closeout.

```bash
ait workflow ready <change-id> --apply
ait task land <task-or-change-id>
```

The readiness commands report the exact next action when a gate is pending.

INFRASTRUCTURE AND INTEGRATIONS

# Python Integration

Use the direct in-process Python binding included in the ait-native PyPI distribution.

**AUDIENCE Python integrators**

https://ait-native.dev/technical/v1.0.0-rc.5/integrations/python/

## Package boundary

The PyPI project is `ait-native`. Its platform wheel includes the admitted native commands and the direct in-process Python binding from the same RC.5 compatibility family.

```bash
python -m pip install ait-native==1.0.0rc5
```

The binding does not download a runtime, launch an `ait` subprocess as its API transport, or provide a separate workflow implementation.

## Direct API

```python
from ait_python import AgentClient, NativeRuntime

runtime = NativeRuntime()
print(runtime.binding_info())

agent = AgentClient(runtime)
print(agent.capabilities().supported_transports)
```

`NativeRuntime` exposes versioned native exports in process. `AgentClient` provides typed access to supported agent capabilities and worker operations.

## Repository workflow

Installing the Python package does not initialize a repository. Run `ait init` once from the target repository, then let the coding agent follow the generated instructions.

```bash
ait init
ait status --json
```

Python projects do not receive a different AIT workflow. Test, lint, build, and packaging commands remain repository-authored inputs.

## Version check

Read binding metadata before combining it with another component. An independently rebuilt binding or mismatched core is not an admitted RC.5 family merely because its Python package imports.

See Distribution and Release for the family and license boundaries.

INFRASTRUCTURE AND INTEGRATIONS

# Node.js Integration

Use the direct Node-API JavaScript and TypeScript surface from the supported npm package.

**AUDIENCE Node.js integrators**

https://ait-native.dev/technical/v1.0.0-rc.5/integrations/node/

## Package boundary

The supported npm identity is `@wa120/ait-native`. It contains the portable JavaScript and TypeScript facade and selects the exact RC.5 platform Node-API addon.

BASH

```
npm install @wa120/ait-native@1.0.0-rc.5
```

The package does not bundle or start `ait-server`. Its API loads the package-owned addon directly and does not use a child process as the binding transport.

## Direct API

JS

```
import { NativeRuntime } from "@wa120/ait-native";

const ait = new NativeRuntime();
console.log(ait.bindingInfo());
const status = ait.runCli(["status", "--json"]);
console.log(status);
```

The packaged `ait` command calls the same native binding in process. The JavaScript API and command therefore share the admitted Rust behavior.

## Repository workflow

Node.js repositories use the same initialization and generated repository instructions as every other supported repository.

BASH

```
npx --no-install ait init
npx --no-install ait status --json
```

AIT does not inspect `package.json` to choose test commands or framework behavior. The repository remains responsible for its explicit validation.

## Platform packages

Target-specific addon packages are exact-version implementation dependencies, not separate products. Applications should depend on the supported top-level package and let normal npm resolution select the matching addon.

REFERENCE

# Distribution and Release

Understand the six-component compatibility family, install-channel roles, and exact RC.5 authority.

**AUDIENCE Developers, operators, and packagers**

https://ait-native.dev/technical/v1.0.0-rc.5/distribution/

## One compatibility family

ait-native 1.0.0-rc.5 admits six components together:

- `ait` - native repository and workflow CLI;
- `ait-agent` - native agent runtime;
- `ait-server` - remote protocol and durable authority;
- `ait-runner` - remote native execution plane;
- `ait-python` - direct in-process Python binding;
- `ait-node` - direct Node-API binding and npm packaging component.

Each component retains its source, Snapshot, artifact digest, and license identity. Matching version text alone does not replace release-family admission.

## Public install channels

- GitHub Release provides canonical native assets, checksums, validation, and receipt material.
- Homebrew installs the RC formula for macOS and Linux.
- PyPI publishes `ait-native==1.0.0rc5` platform wheels.
- npm publishes `@wa120/ait-native@1.0.0-rc.5` and exact-version addon dependencies.
- The signed apt testing repository publishes RC.5 for Debian and Ubuntu.
- GHCR publishes Linux `amd64` and `arm64` indexes for `ait-server` and `ait-runner`.

WinGet remains a validation-asset route for RC.5 and is not listed as a public installation command.

## Package composition

Homebrew, apt, and PyPI product bundles include the admitted native `ait` and `ait-server`; installation leaves the server inactive. PyPI also contains the Python binding. npm contains the Node.js binding and in-process command but not the server.

## License boundary

`ait`, `ait-agent`, `ait-runner`, `ait-python`, and `ait-node` use Apache-2.0. `ait-server` uses AGPL-3.0-only. A bundle does not merge these licenses or erase component notices.

## Exact release evidence

Use the page authority panel to open the immutable RC.5 Release and centralized distribution contract. Those sources own the artifact, endpoint, Snapshot, license, and verification record; this page is a navigable technical summary.

REFERENCE

# Troubleshooting

Use read-only evidence first, preserve recoverable state, and follow exact AIT recovery hints.

**AUDIENCE Developers and operators**

https://ait-native.dev/technical/v1.0.0-rc.5/troubleshooting/

## Start with read-only evidence

Do not guess whether the workspace, Task, or remote is healthy. Begin with the narrow read that matches the symptom.

BASH

```
ait status --json
ait diff --stat
ait queue summary --all-changes
ait worktree status --json
```

Keep the exact error and next-action text. AIT decision surfaces intentionally report the command that can safely advance or inspect the current state.

## The repository is not initialized

Run `ait init` from the intended repository root. If an `.ait` directory exists but is incomplete, do not replace it blindly; inspect the error and use the bounded repair option only when it matches the reported condition.

## A regression needs attribution

Use `ait blame` on the smallest useful source range before selecting a repair.

BASH

```
ait blame <path> --line <number>
```

Record the repair in a new sprint item and Task. Completed lineage remains historical evidence.

## A Task workspace is missing or stale

Inspect the Task and worktree before recreating materialization.

BASH

```
ait task audit <task-id>
ait worktree doctor --refresh --json
```

Follow the exact recovery command reported for that Task. Preserve unrelated workspace changes and avoid deleting a path that is not precisely identified as AIT-owned.

## Remote readiness is pending

Run the text-only readiness surface again and follow its stated gate.

BASH

```
ait workflow ready <change-id> --apply
```

Local test success does not substitute for required remote CI. A failed review, policy, attestation, or CI gate must remain visible until its owning evidence is corrected.

## The server or runner is unavailable

Check the configured remote, server health, repository index, runner compatibility, and repository-owned CI entrypoint. Do not switch a remote-backed Task to a different authority merely to bypass the failure.

## Escalate without destroying evidence

When recovery requires a new user decision, stop with the current Task, Snapshot, and worktree intact. Report the exact identifier, failing gate, command output, and last successful validation.

VERSION AUTHORITY

# RC.5 Source Authority

This PDF is a navigable public reference, not a separate product contract. Use the exact RC.5 source, Release, distribution contract, and CLI parser below when release authority matters.

| | |
|---|---|
| PUBLIC SOURCE | https://github.com/weita2026/ait-native/tree/v1.0.0-rc.5 |
| RELEASE | https://github.com/weita2026/ait-native/releases/tag/v1.0.0-rc.5 |
| DISTRIBUTION SHA-256 | `911d18a04826ce25d2cf1bddeb296ccda2e044f53cd5e90c7016564ce8d4972c` |
| CLI PARSER SHA-256 | `10879229571b305712402ac7fd638b7537b7641daacbfbedcaa48a115bb709e0` |
| AIT-CORE | `SNP-64B101AAE684` |
| AIT-SERVER | `SNP-0445E16F63EB` |
| AIT-RUNNER | `SNP-2458874F5737` |
| AIT-PYTHON | `SNP-9EE8E9FFF1D1` |
| AIT-NODE | `SNP-261F4DA754BE` |

For the current online edition, search, and route-specific updates, open ait-native.dev/technical/.