AIT 1.2.0 / Collaboration guide

Agent Teams: collaboration and review

Roles, default agent review, parallel Tasks, correction handoffs and remote recovery, with practical 1.2.0 examples.

Roles, independent review, parallel Tasks and recovery. Examples follow released 1.2.0; this is not the complete CLI reference.

https://ait-native.dev/development/1.2.0/agent-teams/
2026-11-02

Contents

1.2.0 · Reference 3

From a request to independently reviewed work 3

Default agent review and the gates that remain 3

Initialize once and preserve existing conversations 3

Example: one requirement, one Task, one reviewed result 4

Example: a review finding and its correction 4

Example: two Developers working in parallel 5

Local completion, remote sealing and retries 5

Read contacts without changing sessions 5

Prepare an existing member 6

Role delivery and continuing work 6

Authority and version boundary 7

1.2.0 · Reference

This 1.2.0 reference keeps the original /development/ URLs for existing links. Use Quickstart for installation and Release Status for each channel's actual availability and qualification scope. The 1.1.1 technical manuals and PDFs remain archived references.

From a request to independently reviewed work

Agent Teams connects your intent, the assigned conversations, isolated Task worktrees, exact review candidates and native completion. You choose the outcome and acceptance criteria. AIT records and checks the workflow; the agents perform planning, implementation and review.

Role	Responsibility	Handoff
Product Manager	Clarifies user scenarios, scope, priorities and measurable acceptance.	Settled requirements and necessary decisions to the Project Manager.
Project Manager	Prepares the executable card, dependencies, local or authorized remote scope, Developer and independent Reviewer.	One complete assignment; coordinates scope changes and blockers.
Developer	Implements in the returned edit_root, runs relevant checks and freezes a Snapshot.	READY directly to the assigned Reviewer; corrects findings in the same Task.
Reviewer	Independently examines the frozen candidate and evidence, checks required gates, and performs the applicable finish.	CHANGES_REQUIRED directly to the Developer, or COMPLETED to the Project Manager after native closeout.

The Reviewer does not author the candidate being reviewed. If a check fails, the Reviewer investigates enough to give an actionable finding: affected location, observed problem, required result and verification. Forwarding a failed CI result alone does not establish a review conclusion.

Default agent review and the gates that remain

A fresh team candidate uses `ai_only_experimental` as its effective default: independent agent code review without an extra human code-review approval step. The setting has a precise boundary:

- A valid explicit author mode takes precedence.
- Enabling Teams preserves the saved solo author-mode default; that saved value is distinct from the effective default of a new team candidate.
- A sealed candidate keeps its original mode and identity.
- Real CI, Task approval and mandatory Policy still apply. An agent message or an idle session cannot substitute for these gates.

You continue to set product intent and acceptance. The default removes an extra human code-review step; it does not promise that generated code is correct or that every candidate is automatically accepted.

Initialize once and preserve existing conversations

Fresh `ait init` enables Teams and generates the repository workflow instructions. A supported agent host must still prepare usable conversations. Team setup, sprint binding and Local/Remote scope are independent choices; local initialization does not require a running Server.

For an existing repository, initialization preserves its settings and session bindings. Reinitializing a repository that already has bindings, or enabling Teams there, synchronizes the applicable configuration and instruction assets without replacing or reloading those conversations. An archived, missing or uncertain observation is not permission to discard a stored binding. A first setup with no bindings follows the initial bootstrap path.

Explicit session preparation remains available when needed. Preparing a session is not dispatching a Task. Use the init/config reference and the session preparation section below for exact prerequisites.

Example: one requirement, one Task, one reviewed result

1. **Plan.** The Project Manager prepares a sprint card with scope, acceptance, dependencies, a stable Plan reference, an exact unchecked item, the Developer and the independent Reviewer.
2. **Start.** The Developer starts from that exact item and works only in the returned edit_root. Task start synchronizes and binds the initial card.
3. **Implement and check.** The Developer reviews `.aitignore` before the first build or capture, implements the change, runs the relevant checks and freezes a Snapshot.
4. **Review.** The Developer sends READY directly to the Reviewer. The Reviewer inspects that exact Snapshot and performs the independent and required checks.
5. **Finish.** Once the applicable gates pass, the Reviewer performs native finish. Successful output reports Task completion, worktree cleanup and bound-item closeout where applicable.

The following illustrates a sprint-bound local assignment. Use the commands generated by the installed version and substitute the returned identities; these placeholders are not runnable IDs.

```
ait task start --from docs/sprints/shop.md#shop/discount --intent "Discount rule" --local
# In the returned edit_root, after implementation and checks:
ait snapshot create <TASK_ID> --message "Discount rule ready for review"
```

The direct handoff is short plain text. Detailed evidence stays with the native results and relevant checks; there is no extra JSON messaging envelope.

```
READY task=<TASK_ID> round=R01 snapshot=<SNAPSHOT_ID>
```

For a clean reviewed local candidate, the Reviewer follows the applicable generated finish route, such as `ait task finish <TASK_ID> --local`. A successful Snapshot capture, CI job or chat response alone is not Task completion. Do not manually check the sprint item to simulate closeout.

Example: a review finding and its correction

Suppose the discount implementation rounds twice and produces an incorrect total. The Reviewer returns a concrete finding directly to the Developer:

```
CHANGES_REQUIRED task=<TASK_ID> round=R01 snapshot=<REVIEWED_SNAPSHOT>
F01 | src/price.rs:42 | rounds twice | round final total once | test boundary
```

Initial Task work and its first review create no Change. Before actual revision edits, the Developer creates a meaningfully titled correction Change from the exact reviewed Snapshot, if no correction Change exists:

```
ait change create <TASK_ID> --title "Fix rounding" --fork-snapshot <REVIEWED_SNAPSHOT> --local
```

The Developer reuses that Change for corrections within its scope, fixes the implementation, reruns affected checks and captures a replacement Snapshot. The replacement goes back to the same Reviewer:

```
READY task=<TASK_ID> round=R02 snapshot=<REPLACEMENT_SNAPSHOT> resolves=R01/F01
```

The Reviewer verifies the correction and closes the finding. A new review round does not require a new Task, another card or routine Project Manager approval. Reuse prior evidence only while the candidate differences, inputs and coverage justify it. Scope or ownership changes return to the Project Manager.

Example: two Developers working in parallel

Developer A changes a price calculation; Developer B changes the help text. Each has a separate Task, worktree and, with sprint binding enabled, its own requirement. Implementation and validation can overlap.

1. A reaches a reviewed result and completes first.
2. Before B is applied, AIT rechecks B against the now-current shared target. Compatible changes are reconciled with the applicable validation.
3. If both Tasks changed the same logic incompatibly, completion stops. The responsible Developer corrects the conflict and returns the replacement for review.
4. Each successful finish cleans up its Task worktree and closes its bound item where applicable.

Task isolation does not eliminate every shared write. The `docs/` directory is shared with the repository root, and shared targets or workspaces still require coordination. Worktrees are workflow boundaries, not operating-system sandboxes. Final verification must use the reviewed candidate; writers pause for finish, which may remove the worktree.

Local completion, remote sealing and retries

Local finish and remote acceptance are separate outcomes. A local review does not itself authorize remote preparation or publication. One assigned Reviewer owns the authorized remote aggregate, its readiness, CI and guarded finish.

`ait workflow ready` checks readiness without changing it by default; `--apply` prepares the candidate. Preserve the exact identity returned by the native command and follow its next actions. A different aggregate needs its own applicable review and validation.

Situation	Required behavior
A later Local Task finishes while remote candidate S1 is being checked.	S1 stays sealed. The later Local finish does not rewrite the remote candidate or change what its CI reviewed.
The Server restarts before formal acceptance.	Reuse the original complete expected identity and operation context for admitted ready apply. Verify a new Job with the same sealed content and original mode; obtain fresh CI and required review. Missing or contradictory identity stops recovery.
The Server accepted finish, but the client missed the response.	The formal retry reuses the original acceptance. Preserve successful work and resume only the failed phase; do not repeat apply.

Use the 1.2.0 workflow guide for ready/finish details. The native result is completion evidence; a session being idle or a single CI job passing is not.

Read contacts without changing sessions

```
ait team contacts --json
```

This reads local bindings and native availability. It does not create, replace or reload sessions. When Teams is disabled, `team_enabled: false` and empty `members` do not mean the retained roster was deleted.

Field	Meaning and limit
`planned_seat_count`	Baseline seats, not a count of usable or running sessions. The four planned roles cover Project Manager, Product Manager, Developer and independent Reviewer.
`actual_member_count` / `bound_session_count`	Listed members / nonempty session bindings. Neither counts active work.
`agent_index` / `session_id`	Member identity and its native session binding. Match the assigned session and host IDs when querying desktop runtime.
Role path/revision, `loaded_role_revision_index`, `role_reload_status`, `role_state`	Assigned role and delivery observations. Loaded-role metadata records delivery, not retained context.
`availability` / `host_status`	Native availability/host observations. `available` can include `notLoaded`; neither proves running work.

Unbound, archived, missing, mismatch and unknown observations do not prove a usable session. Use the desktop `list_threads` tool separately for running/idle status and match both session and host IDs. Contacts cannot authoritatively describe another desktop host's execution. Filter unrelated metadata. Limited listings can omit a session; omission, failed observations and `notLoaded` stay unknown. Idle does not prove Task completion; native Task/workflow finish does.

Prepare an existing member

The following is syntax; replace placeholders with the assigned member and numbered role. Do not run ensure merely to check runtime status.

```
ait team session ensure --agent <AGENT> --json
ait team session ensure --agent <AGENT> --role <ROLE> --json
```

Ensure creates, reuses or recovers the existing member's session and delivers its numbered `docs/roles/` Plan. An additional member needs an explicit role. Reuse valid bindings. Replacement requires proven absence in the original native store; uncertain host evidence retains the binding and reports a blocker. An active session is not evidence of absence. Preparing a Reviewer requires server registration.

When native output offers activation, perform its required app actions in that same conversation, then acknowledge only that offered token:

```
ait team session acknowledge --activation <ACTIVATION> --json
```

Acknowledgment verifies title, project, binding and role receipt. It assigns no work. Preparation returns `work_started: false`; the PM must separately assign intent, scope, card and Reviewer. A prepared seat proves neither running work nor Task completion.

Role delivery and continuing work

Plan sync records a role edit without automatically reloading sessions. Use guarded preparation when changed-role delivery is needed. A new Task alone needs no reload. After compaction, recover full applicable guidance and the current work order. Disabling Teams preserves sessions, bindings, role history and unfinished Tasks; their existing ownership and gates remain until explicit handoff.

Historical workflow-mode and team-size presets are not current commands. Follow generated `AGENTS.md` for Task start, independent review and finish, and use the init/config reference for the independent Teams, sprint and scope settings.

Authority and version boundary

This guide follows the released 1.2.0 source and the reviewed Agent Teams explanation. Its PDF is generated from this same page source. Historical manuals retain their original versions.

This reference follows Core’s reviewed Agent Teams, init, config, Task and workflow references, and Native’s current distribution and release SOP. Use Release Status for the current release and channel record. The dated 1.1.4 distribution source records the historical 1.1.4 baseline. Archived technical manuals and PDFs retain their original version and evidence.