Co-Learner User Manual

This manual explains what the Co-Learner agent does, how to set it up, and how to use it for codebase understanding and refactoring.

What It Is

Co-Learner is a CLI agent that helps you: - map a repository quickly, - identify structural hotspots, - propose refactor steps, - and outline tests to reduce risk.

It uses a small set of tools (starting with `repo_scan`) and a ReAct-style loop. If you provide an LLM API key, it will draft more tailored answers and tool choices.

Quick Start

1) Install dependencies

```
cd extensions/colearner
npm install
```

2) Set an API key (optional but recommended)

```
export OPENAI_API_KEY="your-key"
# or
export ANTHROPIC_API_KEY="your-key"
```

3) Run the CLI

```
npm run dev
```

4) Ask a question

```
co-learner> map this repo
```

Kafka mode (optional):

```
COLEARNER_BUS=kafka AAFW_HOME=../aafw-home npm run dev
```

Tutorial: First Session

Goal: get a quick map of the repo and a refactor plan.

1) Start the CLI.
2) Ask for a map:

```
co-learner> give me a structure overview and key hotspots
```

3) You will see events:

```
[thinking] Using tool: repo_scan
[tool_start] repo_scan {"depth":1}
[tool_end] repo_scan (12ms)
```

```
answer:
...
```

4) Ask for a plan:

```
co-learner> propose a phased refactor plan for the core modules
```

Capabilities (Current)

- repo scan for high-level structure.
- tool summaries saved to `.colearner/context`.
- event stream for transparency.
- LLM-assisted answers (if API key is set).

Example Queries

- "Map this repo and highlight risky modules."
- "Give me a refactor plan for the API layer."
- "Which tests should I add before refactoring?"
- "Summarize architecture based on folder layout."

Didactics Commands

Use these commands in the CLI: - `learn <goal1, goal2>`: generate and store a learning plan. - `explain <topic>`: explain a concept at 3 levels. - `practice <topic>`: generate a safe exercise. - `assess <exercise>|<response>`: evaluate a response. - `refactor <topic>`: propose a refactor and teach the why/how. - `progress`: show learning state. - `lifecycle <init|plan|practice|review|done>`: record a session stage. - `history`: show lifecycle history for the current session. - `history <session>`: show lifecycle history for a specific session. - `evidence <path>|<note>`: send evidence with hash + snippet. - `request-evidence <path>|<reason>`: ask for evidence on a file or module. - `coach dashboard`: show aggregated progress for multiple students. - `student <id>`: set the student identifier for routing. - `role <coach|student>`: set the CLI role for filtering. - `sync`: poll new events for the current session.

How It Works (Short)

1) Co-Learner receives a query.
2) It chooses tools (LLM or heuristic).
3) Tool results are stored to disk.
4) It answers with a short plan and test guidance.

Teacher Section: How the Didactics Work

This tool is designed to behave like a practical instructor for onboarding. The didactics are built around evidence, scaffolding, and feedback:

1) **Goal intake**: The agent starts by capturing learner goals and skill level.
2) **Learning path**: It generates a minimal path focused on "what you need to contribute."
3) **Evidence grounding**: Explanations reference real files and code structure.
4) **Scaffolded teaching**: Concepts are explained in tiers (basic → advanced).
5) **Practice and feedback**: Exercises are small, safe, and reviewed with clear next steps.

As a teacher, you can use the CLI to guide onboarding sessions: - Start with `learn <goal1, goal2>` to define objectives. - Use `explain <topic>` to introduce concepts with concrete repo context. - Use `practice <topic>` for hands-on tasks and `assess <exercise>|<response>` for feedback. - Check `progress` to track milestones and confidence.

If you want to adapt the curriculum, edit `.colearner/learning.json` directly or rerun `learn` with new goals.

Student Section: How to Learn With Co-Learner

Use Co-Learner as your personal onboarding tutor. A good session looks like this:

1) **Set goals** with `learn <goal1, goal2>` (e.g., "understand data flow, ship first PR").
2) **Ask for explanations** using `explain <topic>` when you hit a concept you don't know.
3) **Practice** with `practice <topic>` and try the exercise in a local branch.
4) **Get feedback** using `assess <exercise>|<response>` to check your understanding.
5) **Track progress** using `progress` to see what you've mastered.

Tips: - Keep sessions focused on one objective at a time. - Use real file paths in your questions for sharper answers. - When you get a refactor proposal, read the "why" section first.

Coach Section: Recurring Pattern for Guiding Progress

Use this weekly loop to keep learners moving:

1) **Set the objective** (10–15 min)
   - Run `learn <goal1, goal2>` and pick 1–2 next steps.
2) **Explain with evidence** (20–30 min)
   - Use `explain <topic>` and walk the learner through real files.
3) **Practice task** (30–60 min)
   - Use `practice <topic>` and assign a small change.
4) **Review and assess** (15–20 min)
   - Use `assess <exercise>|<response>` and give precise feedback.
5) **Update progress** (5–10 min)

- Run `progress` and record confidence gains.

Repeat weekly, tightening scope as confidence grows.

Test Drive: End-to-End Walkthrough

This walkthrough exercises the full feature set (coach + student, plan, practice, feedback).

1) Start two terminals

- Terminal A (Coach)
- Terminal B (Student)

2) Set roles and a shared session

```
co-learner> role coach
co-learner> session onboarding-01
co-learner> student s-001
```

```
co-learner> role student
co-learner> session onboarding-01
co-learner> student s-001
```

3) Coach creates a plan

```
co-learner> learn repo overview, dependency map, safe refactors
co-learner> lifecycle plan
```

4) Student syncs and checks progress

```
co-learner> sync
co-learner> progress
```

5) Coach assigns an exercise

```
co-learner> assign dependency graph|Create a quick module map|npm test
```

6) Student syncs and submits

```
co-learner> sync
co-learner> submit ex-1|I mapped modules A->B->C and noted edges.
```

7) Coach gives feedback

```
co-learner> feedback pass|Missing edge A->D|Add that edge|0.1
```

7b) Coach requests evidence (optional)

```
co-learner> request-evidence src/core/router.ts|Verify routing logic
```

7c) Student sends evidence (optional)

```
co-learner> evidence src/core/router.ts|Reviewed switch branches
```

8) Student syncs and checks progress

```
co-learner> sync
co-learner> progress
```

9) Try didactics modes

```
co-learner> explain dependency graph
co-learner> practice module boundaries
co-learner> refactor extract interface
```

If you don't see events, ensure both terminals share the same `.colearner/` folder and run `sync`.

Security and Scope

For production use, set `COLEARNER_SCOPE_ROOT` to restrict file access:

```
export COLEARNER_SCOPE_ROOT="/path/to/repo"
```

This scope is enforced for `file_read` and evidence snapshots.

Testing

Use the full test plan in `docs/test-plan.md`. A quick check is: 1) `learn`, `explain`, `practice`, `assess`, `progress` 2) Coach/student `assign`, `submit`, `feedback`, `sync` 3) `lifecycle` and `history` 4) `request-evidence` + evidence snapshot 5) `coach dashboard`

Output Shape

Responses are structured in three parts: 1) Findings 2) Learning steps (phased) 3) Risks and tests

Troubleshooting

No LLM configured: - You will still get basic answers and tool usage. - Set `OPENAI_API_KEY` or `ANTHROPIC_API_KEY` for richer output.

Tool errors: - Check file permissions. - Ensure you are running from the repo you want to inspect.

Next Steps

If you want deeper analysis, add tools like: - file_read - symbol_index - dependency_graph - call_graph

See `extensions/colearner/docs/agent-spec.md` for the tool contract.