IDE agent workflows

Plan, act, observe, and review

Access and cost preflight

Follow the course safety baseline.

1. Check the current GitHub docs and your organization's rules.
2. Check the repository, data, and tools.
3. Use a small task with tests.
4. Set a usage limit for metered work.

The agent loop

```
Plan → Act → Observe → Review → Revise or stop
```

Use the loop to review the work. Check current controls and behavior before a live demonstration.

Task contract

```
## Goal
Add one small, testable behavior.

## Constraints
- Approved files and data only
- Required tests
- No unapproved dependencies

## Completion
- Tests and review evidence are available
- A human approves the change
```

Review checkpoints

- Inspect the proposed file changes.
- Review every command before it is run.
- Check test results and unexpected side effects.
- Stop when scope, data, policy, or the agreed stop rule is unclear.

Required access

Check that Agent mode works before the lab starts. If it does not, stop the lab.
Manual editing does not demonstrate
tool approval, steering, or agent review.

Agenda and time plan

Time	Topic	Trainer move
0:00	Assistant-to-agent shift	Contrast task shapes
0:08	Agent loop and tools	Trace one iteration
0:18	Task contracts and trust	Define boundaries
0:28	Multi-file live demo	Pause at checkpoints
0:43	Chat vs. agent mode	Apply decision framework
0:50	Intervention and TODO delegation	Show safe handoff
0:57	Lab handoff and Q&A	Check stop rules

Transition: "Autocomplete predicts text; an agent pursues a goal through tools."

From autocomplete to agent

Mode	Developer supplies	Surface returns
Inline	Local code context	A continuation
Chat	A question and references	Guidance or a proposed edit
Agent	Goal, constraints, and repository	A multi-step change with a diff and check results

More autonomy needs clear file limits, progress checks, and review.

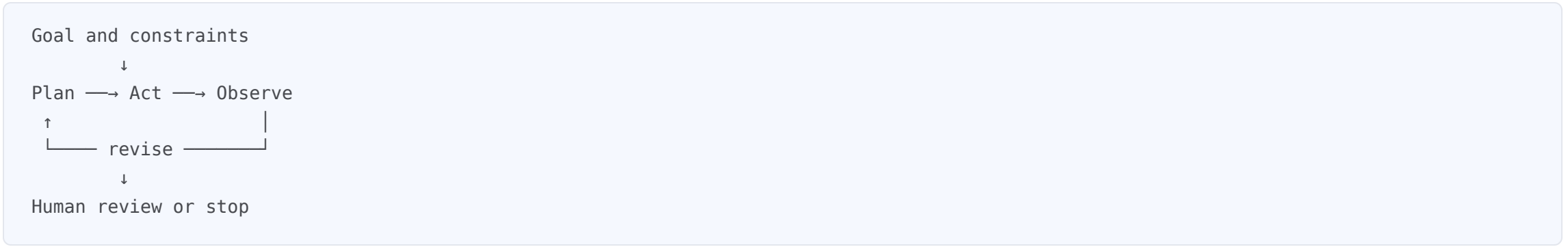
What agent mode can do

Subject to current product support and policy, agent mode may:

- inspect repository files and conventions;
- plan a sequence of changes;
- edit multiple related files;
- run approved terminal commands and tests;
- observe failures and revise;
- use approved tools, including configured MCP tools.

Product capability does not grant approval. Check what policy allows.

One loop, many checkpoints



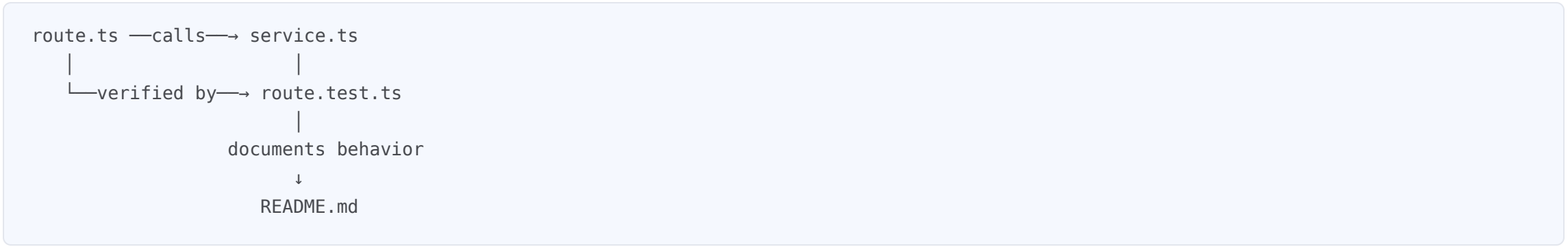
Trainer cue: Pause after each arrow and ask which plan, diff, or test result learners should see.

Tool use and trust boundaries

Tool action	Review question
Read a file	Is this path and data approved?
Edit a file	Is it required by the task?
Run a command	Is it understood, reversible, and scoped?
Add a dependency	Is it necessary, approved, and reviewed?
Search the web	May repository context leave the environment?
Call an MCP tool	Are server identity and permissions approved?

Stop rather than infer authorization.

Multi-file work is a graph



Name the behavior you want to change. Review every changed file and check that its callers and tests still fit.

Demo brief: one feature, three files

```
Add `completed` filtering to the task list endpoint.

Acceptance criteria:
- `GET /tasks?completed=true` returns completed tasks only.
- Invalid values return HTTP 400.
- Existing behavior without the parameter is unchanged.
- Focused tests cover true, false, and invalid values.

Constraints: no new dependencies; edit route, service, and tests only.
```

Concrete demo code

```
const value = req.query.completed;
if (value !== undefined && value !== "true" && value !== "false") {
  return res.status(400).json({ error: "completed must be true or false" });
}

const completed = value === undefined ? undefined : value === "true";
return res.json(taskService.list({ completed }));
```

Ask the agent to implement and test the smallest valid change, then explain it.

Narrate the demo

1. Show a clean working tree and the focused test command.
2. Paste the task contract and request a plan before edits.
3. Check that the plan names route, service, and tests.
4. Approve only commands needed for the task.
5. When a test fails, ask the agent to explain before revising.
6. Inspect the final diff and run tests independently.

Transition: "A passing test checks one behavior. Review the rest of the diff too."

Intervene, redirect, stop

Use a correction that preserves the original contract:

```
Stop. Do not change the response schema or add dependencies.
Keep the existing service API and add only the optional filter.
Show the revised plan before editing.
```

Stop for unexpected file scope, destructive commands, restricted data, unapproved network access, or repeated changes without new test results or an explained diagnosis.

Same task: Chat or agent mode?

Use Chat when...	Use agent mode when...
You need explanation or options	The outcome spans related files
You want one surgical edit	Tests can drive iteration
Requirements are still forming	Requirements and constraints are clear
You need to retain manual control	Tool output helps you check the work

Implement manually when policy, risk, or ambiguity makes an agent unsuitable.

TODO code actions

A TODO can define a short task to delegate:

```
// TODO: reject duplicate task titles within the same project.
```

Before delegating:

- convert the comment into acceptance criteria;
- check the target repository and branch;
- remove sensitive context;
- define tests and reviewer ownership;
- check current support and policy for the selected handoff.

Demo debrief

Ask learners:

- Which plan step gave the earliest useful signal?
- Which command required the most scrutiny?
- Did the agent touch exactly the expected files?
- Which acceptance criterion was easiest to miss?
- Would Chat have been safer or faster for this task?

Use the answers to discuss task selection.

Lab handoff

You will scaffold, implement, test, and refactor features with clear limits while:

1. observing tool calls and multi-file edits;
2. redirecting the agent after an intentional scope change;
3. comparing manual and agent-mode implementations;
4. trying a TODO delegation only when access is approved;
5. recording the checks used to accept or reject output.

Build one Todo API, add ownership and due dates, then redirect its repository refactor.

Final deliverable: The tested API, plan, comparison, course correction, and human decision.

Remember

1. Agent mode fits clearly defined, testable, multi-step work.
2. Write the task limits before the agent starts.
3. Tool approval and file scope are trust decisions.
4. Redirect or stop the agent when needed.
5. A human reviewer decides whether to accept the work.

Questions and lab readiness

Where should a reviewer intervene in your team's workflow?

- What is your stop condition?
- Which test will you run independently?
- Is the starter repository and access ready?