Coveralls MCP Server v0.1 — User Guide

Ask an AI agent about your test coverage.
Coveralls Cloud (coveralls.io). November 2026.

1. What this is

Coveralls now includes an **MCP server**. MCP stands for Model Context Protocol. It is the open standard that AI agents (Claude, Cursor, and the agents behind gateways such as BiFrost) use to call a product's tools. Once your agent is connected, you can ask it questions about your coverage in plain language, and it answers from your Coveralls data:

- "Which owners are below our 80 percent target this week?"
- "Will pull request 1234 pass our coverage check?"
- "Which lines of `planning/route.py` lost coverage in the last build?"
- "Which folders of the `backend` subproject are weakest?"

The agent does the asking. You do not learn a query language, and you do not open Coveralls to read a table. The agent calls the server's tools, reads the numbers, and writes the answer, often as a table you can paste into a report.

What the server is not

- **It is read-only.** No tool changes anything in Coveralls. The agent cannot edit settings, delete builds or change a repository.
- **It never returns source code.** Coveralls does not store your source code, and the server does not fetch it. It reports line numbers and hit counts. Your agent already has the code in the engineer's checkout if it needs to read it.
- **It stores nothing about owners or targets.** You tell the agent who owns what and what the target is; the server computes the numbers and forgets the question.
- **It sees only what the token's user can see.** The agent acts as one Coveralls user and sees exactly that user's repositories.

2. Connecting, in one page

You connect your own agent in about two minutes. Nothing is installed; the server is part of coveralls.io.

1. **Create a Personal Access Token.** Sign in to Coveralls and open your account page, https://coveralls.io/account. Under **Personal Access Tokens**, create a token and copy it; it is shown once. The agent will see exactly the repositories your account can see. For a shared gateway, make the token under a service account instead of a personal one.

2. **Add one entry to your agent.** The server's address is `https://coveralls.io/mcp`, and the entry carries the header `Authorization: Bearer <your token>`. For Claude Code, this one command does it (type it in a terminal; keep the token out of chats and tickets):

   ```
   claude mcp add --transport http --scope user coveralls https://coveralls.io/mcp --head
   er "Authorization: Bearer <your token>"
   ```

   Cursor, Claude Desktop and gateways take the same address and header in their own MCP settings.

3. **Check it.** In Claude Code, type `/mcp`: `coveralls` should show as connected with seven tools. Then ask the agent "what can you tell me from Coveralls?" If it says it has no Coveralls tools, see section 7.

3. The seven tools, in plain words

You never call these yourself. The agent reads their descriptions from the server and picks the right one. Knowing what they answer helps you ask good questions.

Tool	The question it answers	Example question
`list_repos`	Which repositories can this token see, and what is each one's latest coverage?	"List our repos and their coverage."
`get_repo_coverage`	What is one repository's coverage on a branch, per branch and per subproject?	"What is `planner` at on `main`?"
`get_build`	Is this build done, what did it report, and how does it compare with the build it is measured against?	"Is build 1234 finished? Did coverage go up?"
`list_files`	Which files in a build are lowest, which changed, which left new lines uncovered?	"Show me the ten least-covered files in the latest build."
`list_directories`	Which folders in a build are weakest?	"Which folders under `planning/` are below 70 percent?"
`get_file_coverage`	Which lines of one file are covered, and which lines lost or gained coverage since the base build?	"Which lines of `planning/route.py` lost coverage?"
`get_owner_coverage`	Is each owner meeting its coverage target, across repositories?	"Are all owners above 80 percent?"

Six words the tools use:

- **Build**: the full coverage report for one commit.
- **Branch**, in a coverage count: one side of a decision in the code, such as the two outcomes of an `if`, each arm of a `case`, each side of a ternary. `covered_branches` is how many of those sides the tests reached. Not a git branch, and not a path through the program. The `branch` you pass to a tool is the git branch.
- **Relevant line**: a line that can be executed. Blank lines, comments and declarations are not counted, so a file's relevant lines are fewer than its lines.
- **Subproject**: one part of a build, uploaded by one parallel job and named by its **flag** (for example `backend`). A build with no flags has no subprojects.
- **Base build**: the build a build is compared with. For a branch build it is the previous build on that branch; for a pull request it is the build of the commit the pull request branched from.
- **Owner**: a named set of parts of your code, with an optional target percent. The word comes from GitHub's CODEOWNERS file and fits a team as well as one person.

4. A worked example: coverage per owner against a target

This is the question the server was built for. It takes one request and gives one table.

What you say

> Use our CODEOWNERS file for `github/acme/planner`. Target 80 percent. The docs owner's target is 50 percent. Also, `@acme/perception` owns all of `github/acme/perception`.

What the agent does

The agent reads the CODEOWNERS file from the repository:

```
*             @acme/platform
/planning/    @acme/planning
/perception/  @acme/perception
/docs/        @acme/docs
*.proto       @acme/platform @acme/perception
```

It turns each line into one **segment** per owner named on the line. A segment is "this repository, optionally this branch, optionally this pattern, optionally this subproject." A line with two owners gives both owners the same segment. Then it adds the thresholds you gave and the extra repository for `@acme/perception`, and sends one request:

```
owners:
  @acme/platform    target 80   planner "*"
                                planner "*.proto"
  @acme/planning    target 80   planner "/planning/"
  @acme/perception  target 80   planner "/perception/"
                                planner "*.proto"
                                perception (whole repository)
  @acme/docs        target 50   planner "/docs/"
```

Each repository is reported on its default branch unless you name another branch. The server uses the latest finished build on that branch.

What the agent shows you

Owner	Repository	Segment	Covered / relevant lines	Percent	Change	Pass
@acme/platform	planner	`*`	8,410 / 10,250	82.0		
		`*.proto`	120 / 160	75.0		
		repository total	**8,410 / 10,250**	**82.0**	**+0.4**	
		owner total	**8,410 / 10,250**	**82.0**	**+0.4**	**yes**
@acme/planning	planner	`/planning/`	2,980 / 3,400	87.6		
		owner total	**2,980 / 3,400**	**87.6**	**+1.1**	**yes**
@acme/perception	planner	`/perception/`	2,105 / 2,900	72.6		
		`*.proto`	120 / 160	75.0		
		repository total	**2,225 / 3,060**	**72.7**	**-0.8**	
	perception	whole repository	unavailable: no finished build on `main`			
		owner total	**2,225 / 3,060**	**72.7**		— (1 repository unavailable)
@acme/docs	planner	`/docs/`	310 / 540	57.4		
		owner total	**310 / 540**	**57.4**	**+2.3**	**yes**

How to read it

- **Covered / relevant lines, and branches.** Every row carries six counts: covered, missed and relevant lines, and covered, missed and relevant branches. The percent follows your repository's setting **"Include branch coverage in the coverage percentage"** (on by default): when it is on, percent = (covered lines + covered branches) ÷ (relevant lines + all branches), the same number coveralls.io shows; when it is off, percent = covered lines ÷ relevant lines. Every answer says which rule it used (`percent_includes_branches`). Builds and files report the percent Coveralls stored; folders, owner totals and per-file line blocks are computed from the counts under the same rule. In the rare case of a file whose branch counts are unknown (a very old build), the file's branch counts are `null`, its percent is lines only, and that block says `false`. The table above is for a repository with the setting off, so you can add the two counts yourself and get the same answers the server gets.
- **A file counts once per repository.** `@acme/platform` owns `*` and `*.proto`. The `.proto` files are already inside `*`, so the repository total is 10,250 relevant lines, not 10,410. Overlapping segments never double-count.
- **A file with two owners counts for both.** The `.proto` files appear under `@acme/platform` and under `@acme/perception`. The server reports coverage, not exclusive ownership, so GitHub's "last matching line wins" rule for choosing a reviewer is not applied here.
- **Change** is the difference in percentage points between this build and its base build, computed over the same segments. `+0.4` means the owner's code is 0.4 points better covered than in the previous build on that branch.
- **Pass** is `yes` or `no` against the threshold you gave. It is **empty (null)** when there is not enough data to judge: no threshold was given, no file matched, or one of the owner's repositories was unavailable. Null is not a failure. It says "part of this owner's code was not measured, so there is no verdict."
- **Unavailable** means that repository has no finished build on that branch yet (or the build is too large to report per owner). The owner's totals still show what could be measured, and the agent tells you how many repositories were left out. Push a build to that branch and ask again.

You can ask for the same table every week, or ask the agent to keep the CODEOWNERS mapping and the targets in a file in your repository so the question becomes "run the coverage report."

5. Asking about a pull request, a build, or a file

A pull request. “Will PR 1234 in `planner` pass our 80 percent check?” The agent fetches the pull request’s latest build, reads its total and its change against the base build, lists the changed lines that are still uncovered, and answers in sentences. (The patch coverage percent itself, the coverage of only the lines the pull request changed, is not returned yet; it is planned for a later release.) Follow up with “which changed lines are uncovered?” and it lists the files and line numbers that the pull request left untested.

A build. “Is build 1234 done? Which parallel job is missing?” The agent reports whether the build finished, which subprojects reported and which have not, when the build was created and when its calculation finished, and the totals so far.

A file. “Which lines of `planning/route.py` lost coverage in the latest build on `main`?” The agent gets the file’s lines for this build and for its base build and reports three lists: lines that **lost** coverage, lines that **gained** it, and lines **still uncovered**. The lists are computed by the server, not guessed by the agent.

A folder or a subproject. “Which folders of the `backend` subproject are below 60 percent?” The agent lists the directories of that one parallel job, lowest first.

6. Limits you should know

- **Line comparisons are by line number.** When a file’s text changed between two builds, an inserted or deleted line shifts every line below it. The “lost” and “gained” lists then describe positions, not the same code. The agent is told when a file’s text changed and should read the lists together with the file’s diff.
- **Lists come in pages of at most 200 rows.** The agent asks for the next page when it needs more. For a question like “the twenty least-covered files” it asks for exactly twenty.
- **One branch per repository per request** in an owner report. The same repository on two branches is two entries in the answer.
- **Numbers come from finished builds.** A build still processing is not counted. Ask again in a minute.
- **Reports are only as good as the mapping.** If a CODEOWNERS line is wrong, the owner’s numbers are wrong in the same way. The server reports what you asked for; it has no opinion about who should own what.
- **Two agent requests at a time per web server.** A third request gets a short “busy, try again” answer and agents retry it on their own. If you hit it often, tell us.

7. When something looks wrong

What you see	What it usually means	What to do
The agent says it has no Coveralls tools.	The gateway or client entry is missing or disabled.	Check the entry from section 2 (address and header), then restart the agent.
“401” or “unauthorized”.	The token is missing, mistyped or was deleted.	Create a new token (section 2) and update the entry.
“Repository not found, or not visible to this token.”	The token’s user cannot see that repository.	Add the repository to that user, or use a token from a user who can see it.
An owner’s repository is “unavailable.”	No finished build on that branch yet, or the build is too large for per-owner reporting.	Push a build to that branch; or ask on the branch that has builds.
“503, try again.”	More than two agent requests at once on one web server.	Nothing; agents retry. If it is frequent, tell us.
Numbers differ from the Coveralls page.	The agent and the page are looking at different builds or branches.	Ask the agent which build it used, and compare with the build the page shows.
A percent is a few points off, every time.	The repository’s “Include branch coverage in the coverage percentage” setting. Every answer says which rule it used (`percent_includes_branches`).	Compare with the same setting on; or turn the setting on, rerun the build, and ask again.

If none of these fits, send us the question you asked and the agent’s answer (support@coveralls.io), and we will look at it with you.

8. What comes next

v0.1 is the first release: management reporting and its foundation. Later releases add the troubleshooting workflows engineers asked for (why a build is slow or stuck, settings and history) and, further out, carefully scoped write actions. Tell us which question you most want the agent to answer next; that is how this list is ordered.