How to Add Your MCP Server as a Gemini CLI Extension

This process requires you to contribute your server's code directly to the `gemini-cli` open-source project.

Step 1: Before You Begin (Proposal & Legal)

Before writing any code, you must:

1. **Sign the Contributor License Agreement (CLA):** All contributions must be accompanied by a Google CLA. You can check your status or sign a new one at cla.developers.google.com.
2. **Open an Issue:** All Pull Requests (PRs) must be linked to an existing issue. You should first open a feature request on the gemini-cli GitHub tracker to propose adding your MCP server as an extension. A maintainer must approve this proposal before you start coding.
3. **Review Community Guidelines:** Familiarize yourself with Google's Open Source Community Guidelines.

Step 2: Set Up the Development Environment

Once your proposal is approved, you'll need to set up the `gemini-cli` project locally.

1. **Fork the Repository:** Create your own fork of the `google-gemini/gemini-cli` repository on GitHub.
2. **Clone Your Fork:**
   Bash

```
None

git clone https://github.com/YOUR_USERNAME/gemini-cli.git
cd gemini-cli
```

3.
4. **Install Dependencies:** The project uses Node.js (specifically version `~20.19.0` for development).
   Bash

```
None

npm install
```

5.
6. **Build the Project:** Build all the existing packages.
   Bash

```
None

npm run build
```

7.

Step 3: Add Your Server as a New Package

The `gemini-cli` project is a monorepo. Extensions are individual packages within the `packages/` directory (e.g., `vscode-ide-companion` is listed as an extension).

1. **Create a New Package:** Create a new directory for your server inside the `packages/` directory.

```
None

gemini-cli/
├── packages/
│   ├── cli/
│   ├── core/
│   ├── vscode-ide-companion/
│   └── mcp-server-extension/  <-- Your new package goes here
└── ...
```

2.
3. **Integrate Your Server:** Add your existing MCP server's code into this new `packages/mcp-server-extension` directory. You will need to adapt it to fit the project's structure, including adding a `package.json` file and conforming to the TypeScript and coding conventions of the monorepo.
4. **Follow Conventions:** Adhere to the coding styles and patterns used throughout the existing codebase.

Step 4: Validate Your Extension

Before submitting, you must ensure your new package passes all project-wide checks.

1. **Run Preflight Checks:** From the root `gemini-cli` directory, run the `preflight` script. This will run all tests, linting, and style checks for the *entire* project, including your new package.
   Bash

```
None

npm run preflight
```

2.
3. **Fix Errors:** If any checks fail, you must fix them before you can submit your code.

Step 5: Submit Your Contribution

Once all checks pass, you can submit your work for review.

1. **Commit and Push:** Commit your changes to your fork and push them to a new branch.
2. **Open a Pull Request (PR):** Open a PR from your branch to the `main` branch of the `google-gemini/gemini-cli` repository.
3. **Link Your Issue:** In the PR description, make sure to link the issue you created in Step 1 (e.g., `Fixes #123`).
4. **Code Review:** The project maintainers will review your submission. Be prepared to discuss your changes and make any requested adjustments.

After your PR is reviewed, approved, and merged, your MCP server will become part of the `gemini-cli` project and presumably be available through the extension system.